VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS INSTITUTAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Adaptyvus kvantinių schemų optimizavimas

Adaptive Quantum Circuit Optimization

Bakalauro baigiamasis darbas

Atliko:	Lukaš Michnevič	(parašas)
Darbo vadovas:	Lekt. Irus Grinis	(parašas)
Darbo recenzentas:	Prof. Dr. Saulius Gražulis	(parašas)

Vilnius – 2021

Santrauka

Kvantiniai kompiuteriai – dar pakankamai jauna technologija, kurioje nemažas skaičius problemų ir iššūkių atsiranda dėl klaidų, atsirandančių vykdant kvantinius skaičiavimus. Klaidos atsiranda dėl taip vadinamo kvantinio triukšmo – nepageidaujamų vidinių sąveikų bei išorinio poveikio kvantinei sistemai. Dėl šitos priežasties yra skiriama nemažai laiko ir pastangų ieškant vis efektyvesnių būdų valdyti neigiamus kvantinio triukšmo efektus ir mažinti atsirandančių klaidų skaičių.

Vienas iš akivaizdžiausių kovos su klaidomis būdų – kvantiniai klaidas taisantys kodai, kol kas negali būti efektyviai taikomi šiuolaikinėje kvantinių kompiuterių architektūroje, vadinamoje NISQ (ang. Noisy Intermediate-Scale Quantum). Tai yra dėl to, kad kvantiniai klaidas taisantys kodai labai padidina vartų skaičių galutiniame kvantiniame kode, kuris yra vykdomas realiojoje mašinoje, dėl ko kartais gaunasi atvirkštinis efektas – klaidų skaičius didėja [LB20].

Kitas, daug efektyvesnis kvantinių skaičiavimų kokybės padidinimo metodas yra algoritmų optimizacijos, skirtos sumažinti naudojamų vartų ir kubitų skaičių. Tokios optimizacijos remiasi faktu, kad kvantinės operacijos – tai matematinės operacijos su matricomis, ir jos yra apgręžiamos. Šitame darbe yra siūlomas adaptyvus kvantinių schemų optimizavimo metodas, kuris naudoja mašininio mokymosi metodus galimų optimizacijų paieškai. Sukurtas adaptyvus optimizatorius parodė universalaus kvantinių schemų analizatoriaus galimą naudą ne tik schemų optimizacijoje, bet ir kvantinių kompiliatorių tobulinime, leidžiant skirtingai reaguoti į pastebėtus šablonus, pavyzdžiui, generuoti įspėjimus arba siūlyti kodo pertvarkymus.

Raktiniai žodžiai: NISQ kvantiniai kompiuteriai, Apgręžiamas skaičiavimas, Kvantinių schemų optimizacija, Kvantinių schemų analizė, Qiskit

Abstract

Quantum computing, while being a young technology, is facing a lot of problems and challenges related to errors that occur during quantum computation. These errors appear mostly due to so called quantum noise – an unwanted impact on a closed quantum system from the inside and outside. A considerable amount of time and effort is invested into a search of ways of reducing the negative effects of quantum noise and reduce error rates in quantum circuits.

One of the most straightforward ways of quantum error handling – quantum error correction codes, cannot be used effectively in NISQ (Noisy Intermediate-Scale Quantum) based quantum computers. The reason is that quantum error correction codes generate a lot of additional gates that are susceptible to errors, therefore, we get a result that is contrary to what we tried to achieve – error rate is rising [LB20].

The effective way of dealing with quantum errors is to optimize quantum circuits by reducing the amount of gates. These optimizations rely on gate commutativity rules that use the fact that quantum gates are reversible. In this work, an adaptive quantum circuit optimization method is proposed, which employs machine learning to look for patterns in quantum circuits, that would mean that certain optimizations can be applied. The adaptive optimizer has shown that universal quantum circuit analyzer not only can be used to optimize circuits, but to improve quantum compilers, allowing to generate specific responses to certain patterns, for example, generate warnings or suggest code changes.

Keywords: NISQ quantum computers, Reversible computing, Quantum circuit optimization, Quantum circuit analysis, Qiskit

TURINYS

Įvadas

Kvantiniai kompiuteriai tapo labai populiaria tyrimų tema kai buvo sukurti algoritmai, sprendžiantys kai kurias problemas žymiai greičiau nei klasikiniai analogai. Vienas geriausių šių algoritmų pavyzdžių yra Šoro faktorizavimo algoritmas, kuris parodė, kad ideali kvantinė mašina galės panaikinti šiuolaikinę RSA kriptografiją. Laimei, ateitis, kurioje kvantiniai kompiuteriai gali nulaužti kiekvieną šifrą yra labai tolima. Tai yra dėl to, kad šiuolaikiniai NISQ tipo kvantiniai kompiuteriai yra silpnai apsaugoti nuo kvantinio triukšmo, o tai reiškia, kad yra generuojama labai daug klaidų kvantiniuose skaičiavimuose, dėl ko didžiąją dalį kvantinių algoritmų neįmanoma panaudoti.

Siūlomi kvantiniai klaidas taisantys kodai yra neefektyvūs NISQ kompiuteriuose, nes nėra jokios galimybės užtikrinti, kad kodas, taisantis kvantines klaidas, pats negeneruos klaidų. Dėl ko šiuo metu egzistuoja tik vienas būdas apsisaugoti nuo klaidų – paprasčiausiai sumažinti jų atsiradimo tikimybę, mažinant naudojamų operacijų kiekį remiantis tam tikrais matematiškai apibrėžtais schemų pertvarkymais. Tokios optimizacijos yra ypač reikalingos schemose, kuriose vartų ir operacijų skaičiai matuojami šimtais ir tūkstančiais. Pavyzdžiui, kvantinių sistemų simuliacijos, taikomos kvantinėje fizikoje, chemijoje ir biologijoje [NRS+18].

Optimizacijos, mažinančios vartų skaičių, generuoja papildomas problemas, tokias kaip poreikis verifikuoti optimizacijas – įsitikinti, kad optimizuotas kodas generuoja tas pačias kvantines būsenas, kaip ir pradinis kodas. Tokiems darbams reikalingi yra galingi kvantinių kompiuterių simuliatoriai arba klasikiniai algoritmai, gebantys įrodyti kad dvi kvantinių operacijų sekos generuoja tokią pat būseną. Tokios problemos patenka į NP-pilnų užduočių sąrašą, dėl ko realiose panaudos atvejose klasikiniai verifikatoriai yra naudojami retai.

Egzistuoja būdas konvertuoti kvantines schemas į aciklinius orientuotus grafus[NRS+18], kas leidžia turėti aiškiai klasikiniam kompiuteriui suprantamą duomenų struktūrą. Tai leistų ieškoti tam tikrų šablonų kvantinėse schemose, pagal kuriuos būtų galima žinoti, kokias optimizacijas reikėtų taikyti. Šablonus galima panaudoti neuroninių tinklų apmokymui. Neuroniniai tinklai galėtų padėti automatizuoti kvantinių schemų optimizavimą, generuojant pertvarkymų rinkinį, pritaikytą pasirinktai kvantinei schemai.

Šablonų kvantinėse schemose aptikimas leistų ne tik adaptyviai taikyti pertvarkymus ir optimizacijas, bet ir generuoti papildomus įspėjimus apie galimas problemas kvantiniame kode, kas savo ruožtu leistų pagerinti kvantinių kompiliatorių panaudojamumą, padedant programuotojui greičiau reaguoti į šitas problemas. Galų gale, universalus kvantinių schemų analizavimo mechanizmas dar sumažins kvantinių klaidų skaičių.

Darbo tikslas. Naudojant neuroninius tinklus bei jau egzistuojančius karkasus kvantiniam programavimui, tokius kaip Qiskit, sukurti adaptyvų kvantinių schemų optimizavimo būdą, bei įvertinti jo efektyvumą. Siekiama, kad sukurtą kodą būtų įmanoma naudoti realiuose panaudos atvejuose, naudojant jį kaip papildomą biblioteką Qiskit pakete.

Darbo uždaviniai.

1. Ištirti esminius NISQ kvantinių kompiuterių bruožus ir architektūrą.

2. Išanalizuoti Qiskit transliatoriaus struktūrą.
3. Apibrėžti optimizacijų verifikavimo būdą, kas leistų užtikrinti, kad taikomos optimizacijos nekeičia algoritmų rezultatų.
4. Apibrėžti taikomas optimizacijas naudojant tam tikrą matematinį aparatą.
5. Įgyvendinti šablonų kvantinėse schemose paiešką, kas leistų apmokyti neuroninius tinklus.
6. Atlikti optimizatoriaus bandymus, įvertinti optimizatoriaus efektyvumą.

1. Fizikinis kvantinio kompiuterio modelis

Kalbėdami apie kvantinius kompiuterius, turime žinoti teorinius jų veikimo principus. Šiame skyriuje bus aprašytos bazinės savokos iš matematikos ir kvantinės fizikos pasaulių, kurios yra reikalingos gilesniam kvantinio kompiuterio supratimui.

1.1. Kubitas

Kubitas (quantum bit - qubit) – tai minimalus kvantinės informacijos vienetas, aprašantis paprasčiausią kvantinės sistemos būseną. Matematiškai kubitas – tai vienetinis vektorius dvimatėje Hilberto erdvėje[YM08].

$$|\phi\rangle \in H, ||\phi|| = 1, \dim H = 2. \tag{1}$$

Pagrindinis skirtumas tarp kubito ir klasikinio bito yra tas, kad kubitas gali būti taip vadinamojoje superpozicijoje – būsenoje, kai negalima vienareikšmiškai nustatyti, kokią reikšmę turi kubitas.

Yra daug kubito įgyvendinimo būdų, žemiau pateikti keli realaus kubito pavyzdžiai:

1. Atomo elektronas, kuris gali būti dvejose skirtingose orbitalėse ir taip turėti dvi skirtingas būsenas.

2. Fotonas, kuri gali būti skirtingai poliarizuotas.

3. Subatominė dalelė, turinti du skirtingus sukinius (ang. *spin*)

1.1.1. Ortonormuota bazė

Bazė $\beta = \{v_0, v_1, ..., v_{n-1}\}$ skaliarinės sandaugos erdvėje V yra vadinama ortogonalia baze jei vektoriai yra tarpusavyje ortogonalūs. t.y. $\langle v_j, v_k \rangle$ = 0 kur j $\neq$ k. Ortogonali bazė vadinama **Ortonormuota baze** jei kiekvieno vektoriaus normalė yra 1[YM08].

1.1.2. Kubitų matavimas

Norėdami išgauti informaciją iš kubito mes turime pamatuoti jį. Tam mes turime parinkti ortonormuotą bazę kubito būsenų erdvei.

Po matavimo gauname vieną iš šitos bazės vektorių, ir sistema (subjektyviai tik mums) keičia savo būseną į tą vektorių.

Tikimybė, su kokia bazinis vektorius taps matavimo rezultatu nurodoma koeficientais prieš būseną aprašantį vektorių:

$$\begin{aligned} |\phi\rangle &= \alpha\,|0\rangle + \beta\,|1\rangle\,, \\ &\alpha, \beta \in \mathbb{C}, \\ |\alpha|^2 &+ |\beta|^2 = 1 \end{aligned} \tag{2}$$

$$P(|0\rangle) = |\alpha|^2 . P(|1\rangle) = |\beta|^2. \tag{3}$$

1.1.3. Kelių kubitų sistemos

Kelių sistemų būsena yra aprašoma vienetiniu vektoriumi Hilberto erdvėje:

Kubitas 1	Kubitas 2	Vektorius
$\|0\rangle$	$\|0\rangle$	$\|00\rangle$
$\|0\rangle$	$\|1\rangle$	$\|01\rangle$
$\|1\rangle$	$\|0\rangle$	$\|10\rangle$
$\|1\rangle$	$\|1\rangle$	$\|11\rangle$

1 lentelė. Kubitų sistemos

Žinodami, kad:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \tag{4}$$

Galime sudaryti sudėtinės sistemos bazinį vektorių naudodami tenzorinę sandaugą:

$$|01\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \otimes \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \\ 0 \cdot \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

1.1.4. Bloch'o sfera

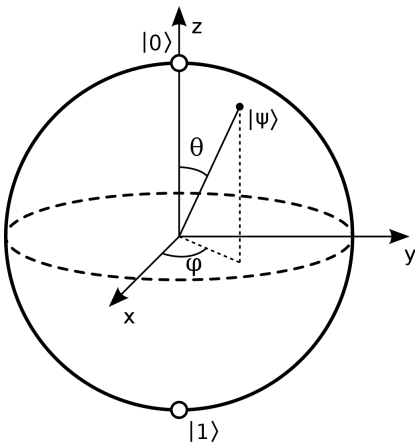
Kiekvienas kubitas gali būti atvaizduotas kaip vektorius vienetinėje sferoje. Tokia sfera yra vadinama Bloch'o sfera (pav.1). Tokiu atveju, kubito atvaizdavimui reikalingi du parametrai: platuma (Θ) ir ilguma (ϕ). Šiaurės sferos polis nurodo $|0\rangle$, o pietų – $|1\rangle$. Matuojamas kubitas susitraukia į vieną iš polių. Jeigu kubito vektorius yra ties sferos pusiauju, po matavimo kubitas susitrauks į vieną iš polių su 50% tikimybe[YM08].

Bloch'o sfera padeda kvantinio kompiuterio simuliacijoje. Jos dėka galime modeliuoti kubito būsenų koeficientus pasinaudodami esamomis polinėmis koordinatėmis.

1.1.5. Kvantinės būsenos modelis

Naudojant Bloch'o sferą, kiekviena kvantinė būsena gali būti aprašyta tokiu būdu:

$$|\phi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{j\phi} sin\frac{\theta}{2}|1\rangle \tag{6}$$



1 pav. Bloch'o sfera*

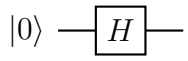
1.2. Kvantinės operacijos

Kalbėdami apie kvantinius kompiuterius, dar viena labai svarbi sąvoka yra kvantiniai vartai – taip vadinamos kvantinės operacijos. Jas paprasčiausiai galime interpretuoti kaip klasikinių aritmetinio – loginio vieneto vykdomų skaičiavimų analogus kvantiniame kompiuteryje.

Kvantiniai operatoriai yra reprezentuojami vienetinėmis matricomis. Tai reiškia kad algoritmas, apdorojantis kvantinės sistemos duomenis yra atvaizduotas kaip vienetinis operatorius būsenos erdvėje.

1.2.1. Paprasčiausi kvantiniai vartai

Paprasčiausi kvantiniai vartai yra tokie, kurie keičia tik vieno kubito būseną. Geriausias tokių vartų pavyzdys – Hadamardo operacija:



2 pav. Hadamardo vartų žymėjimas

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{7}$$

Hadamardo transformacija leidžia transformuoti bazes $|0\rangle$ ir $|1\rangle$ į Hadamardo bazes[YM08] $|+\rangle$ ir $|-\rangle$, kas vėliau leidžia operuoti kubitais superpozicijoje. Iš esmės, operacijos su kubitais superpozicijoje leidžia vykdyti skaičiavimus kiekvienai kvantinei būsenai vienu metu. Taigi, Hadamardo operatorius yra naudingiausias kada jis vėliau yra sujungiamas su kito tipo kvantiniais operatoriais.

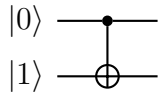
$$H\,|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = |+\rangle \tag{8}$$

*By Smite-Meister - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5829358

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\ 1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\ -1\end{bmatrix} = |-\rangle \tag{9}$$

1.2.2. Kelių kubitų kvantiniai vartai

Kitas kvantinių vartų tipas, apie kurį svarbu žinoti – kelių kubitų vartai, pavyzdžiui, *CNOT* (Controlled-NOT) vartai:



3 pav. CNOT vartų žymėjimas

Kelių kubitų vartai susideda iš dviejų tipų kubitų – kontrolės kubitų ir kontroliuojamų kubitų. Pavyzdžiui, 3 pav., kubitas $|0\rangle$ yra kontrolės kubitas, o $|1\rangle$ – kontroliuojamas kubitas. *CNOT* vartai apverčia kontroliuojamą kubitą tik tuo atveju, kai kontrolės kubito būsena yra $|1\rangle$.

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix} \tag{10}$$

1.2.3. Kvantinių vartų apgręžiamumas

Vienas labai svarbus kvantinių operacijų bruožas yra tai, kad visos kvantinės operacijos turi būti apgręžiamos. Kiekviena kvantinė operacija yra aprašoma unitarine matrica, t.y. tokia matrica U, su kuria teisinga lygtis:

$$U^*U = UU^* = I \tag{11}$$

Čia I – vienetinė matrica ir U^* – Ermito transponuota matrica. Viena iš svarbiausių apgręžiamumo priežasčių yra tai, kad kvantinės sistemos evoliucija yra apgręžiamasis procesas. Tai yra dėl to, kad visos transformacijos turi tiesiškai keisti sistemą, ir kvantinės sistemos būsenų tikimybės turi būti išsaugotos, t. y. informacija negali būti prarasta įvykus kvantinei transformacijai.

Svarbi problema klasikiniuose kompiuteriuose yra ta, kad vykdant skaičiavimus, įprastai yra prarandama dalis informacijos, ir išsiskiria energija. Šią savybę apibūdina Landauerio principas, teigiantis, kad kiekviena neapgręžiama loginė operacija būtinai padidins entropiją sistemos dalyse, kurios neneša jokios informacijos[Ben02]. Pavyzdžiui, energija išsiskirs šilumos pavidalu.

Tarkime, turėdami klasikinį sumatorių $S(A, B) \Rightarrow C$ mes negalime žinoti A, B reikšmių, turėdami tik C. Kvantiniai kompiuteriai veikia kitu principu, kuriame visi algoritmai turi būti kuriami taip, kad skaičiavimai būtų apgręžiami. Svarbu žinoti, kad apgręžiami skaičiavimai vis tiek generuoja šilumą, tačiau jos yra generuojama mažiau[LT07].

1.3. Kvantinio kompiuterio kokybė

Kvantiniai kompiuteriai pagal savo esmę yra panašūs į analogines, o ne skaitmenines skaičiavimo mašinas, todėl jų skaičiavimo rezultatai ne visada gali atitiktį idealų rezultatą. Dėl šitos priežasties yra naudojama taip vadinama kvantinė kokybė – reikšmė, kuri parodo atstumą tarp dviejų būsenų.

Tarkime, atlikę skaičiavimus kvantiniame kompiuteryje, dėl dekoherencijos gavome būseną σ, kuri nėra lygi būsenai ϕ, kurią tikėjomės gauti. Atstumą tarp σ ir ϕ, galime apskaičiuoti naudodami formulę

$$F(\phi, \sigma) = \|\sqrt{\phi}\sqrt{\sigma}\|_1 \tag{12}$$

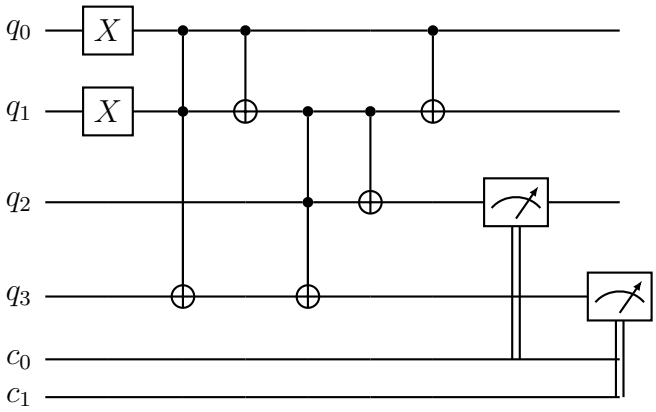
Dar vienas būdas apskaičiuoti kvantinio kompiuterio kokybę – atlikti tam tikrą kiekį skaičiavimų simuliatoriuje ir apskaičiuoti, kokią dalį sudaro rezultatai, kurie yra lygūs rezultatui, kuris yra gaunamas idealiuoju atveju.

2. Kvantinės schemos

Kvantinė schema – tai kvantinių skaičiavimų modelis, leidžiantis grafiškai pavaizduoti kvantinių skaičiavimų seką. Įprastai yra išskiriami trys kvantinės schemos komponentai, apibrėžiantys kvantinio algoritmo vykdymą:

1. Kubitų inicializacija.
2. Kvantinių vartų taikymas.
3. Kvantinis matavimas.

Kvantinio matavimo metu kubito reikšmė yra išsaugoma klasikiniame bite, todėl tokiose sistemose kaip **Qiskit** arba **IBM Quantum Experience** kvantinėse schemose papildomai vaizduojami ir klasikiniai bitai.

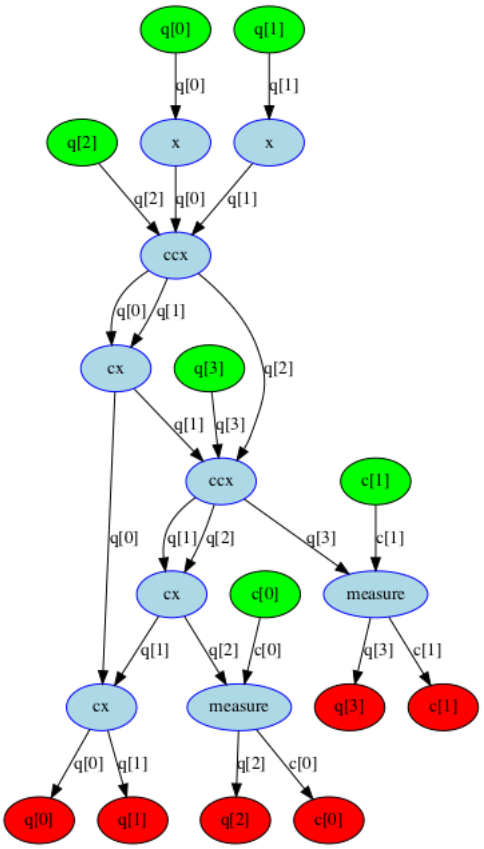


4 pav. Keturių kubitų sudėties algoritmas

Schemoje 4 kubitų 2 ir 3 matavimo rezultatai yra išsaugojami klasikiniuose bituose 0 ir 1 atitinkamai.

2.1. Kvantinių schemų vaizdavimas acikliniais orientuotais grafais

Kvantinės schemos dažnai turi būti apdorojamos klasikiniame kompiuteryje. Tam, kad būtų įmanoma atlikti tokią užduotį, schemos yra konvertuojamos į aciklinius orientuotus grafus. Tokia duomenų struktūra turi daug pranašumų, lyginant su daug paprastesniu metodu, kuriame kvantinės schemos yra paprasčiausiai konvertuojamos į sąrašą (ang. list). Kvantinių schemų kaip grafų interpretacija yra naudojama IBMQ kvantinių schemų optimizavimo sistemose, nes ji leidžia daug greičiau keisti schemą modifikuojant atskirus pografius neįtakojant pačios schemos semantikos (pav. 5).



5 pav. 4 schemos grafas sugeneruotas Qiskit

Aciklinio orientuoto grafo viršūnės vaizduoja kvantinius vartus, o briaunos – vartų pirmenybes, pavyzdžiui, kitus vartus, kuriuos reikia pritaikyti prieš taikant pasirinktus vartus. Jeigu tokios pirmenybės nėra, tai tokius pografius galima vykdyti nepriklausomai vienas nuo kito. [VDO+19] Schemos grafas yra generuojamas iš taip vadinamos virtualios, topologijai neadaptuotos schemos.

3. IBM kvantinių kompiuterių architektūra

NISQ (ang. Noisy Intermediate Scale Quantum) tipo kompiuteriai – tai naujausi ir šiuo metu labiausiai paplitę kvantiniai kompiuteriai. Turėdami iki 100 kubitų, NISQ tipo kvantiniai kompiuteriai leidžia dirbti ne tik su teoriniais bandymais, parodančiais galingesnių kvantinių kompiuterių galimybes, bet ir su realaus pasaulio užduotimis, pavyzdžiui, simuliuojant kvantines sistemas [Pre18]. Šiuolaikiniai IBMQ kompiuteriai, su kuriais galima dirbti **IBM Quantum Experience** sistemoje, irgi yra NISQ tipo.

Pagrindiniai NISQ kompiuterių bruožai yra tokie:

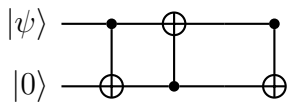
- Realūs kompiuteriai turi iki 100 kubitų.
- Palaikomos tik bazinės operacijos (*NOT*, *CNOT* ir fazių posūkiai). Visi kiti vartai yra konvertuojami į bazinių operacijų rinkinius.
- Sujungimai galimi tik tarp dviejų kubitų vienu metu. Jei taikomi trijų kubitų vartai (pvz. Toffoli), tai jie būtinai yra skaidomi į vieno ar dviejų kubitų vartų kombinacijas.
- Kubitai yra išdėstomi i tam tikras konstrukcijas (topologijas), kvantinių algoritmų schemos yra adaptuojamos topologijai prieš vykdant algoritmą kvantiniame kompiuteryje.

3.1. Kubitų topologija IBM kvantiniuose kompiuteriuose

IBM kvantiniuose procesoriuose kokybiškas surišimas yra įmanomas tarp kubitų, kurie yra betarpiškai sujungti, dėl ko yra labai sudėtinga gauti didelį kiekį efektyvių ir patikimų kvantinių surišimų tarp kubitų. Maža to, visi dabartiniai IBM kvantiniai procesoriai palaiko tik vieną operaciją keliems kubitams – *CNOT*. Dėl šitos priežasties atsiranda poreikis išdėstyti kubitus kvantiniame procesoriuje tokiu būdu, kad būtų įmanoma panaudoti kuo daugiau sudėtingų operacijų naudojant šalia esančius kubitus. Įprastai kubitų topologija yra vaizduojama naudojant grafus.

6 pav. IBMQ Santiago kubitų topologija

Topologijos schemoje 6 matome, kad surišimas yra įmanomas tik tarp kubitų (0, 1), (1,2), (2, 3) ir (3, 4). Surišimai tarp kubitų, kurie nėra šalia vienas kito irgi yra įmanomi, tačiau jų įgyvendinimui prireiks taikyti *SWAP* operaciją (schema 7). Sujungimo tarp kubitų spalva nurodo sujungimo klaidų dažnį, o kubitų spalva – taktinį dažnį. Kuo spalva tamsesnė, tuo dažnis yra mažesnis. IBM nerekomenduoja taikyti *CNOT* vartų kubitams, kurių sujungimai turi daug klaidų ir taikyti svarbiausius vartus kubitams, kurių dažnis yra mažiausias.



7 pav. *SWAP* Operacija

3.2. Qiskit sistema

Patogiam darbui su IBM kvantiniais kompiuteriais buvo sukurtas specialus kvantinio programavimo paketas Python kalbai, pavadintas Qiskit. Qiskit leidžia dirbti su kvantiniais kompiuteriais įprastoje programavimo aplinkoje, ir po to generuoti galutinį kvantinį kodą OpenQASM kalbos pavidalu, kuris jau yra vykdomas kvantinėje mašinoje. Tai leidžia greitai atlikti norimus bandymus ir gauti jų rezultatus, nenaudojant vizualaus programavimo metodo **IBMQ Quantum Experience** sistemoje.

Qiskit paketas susideda iš keturių svarbių komponentų:

- **Qiskit Terra**. Šitas komponentas yra Qiskit paketo pagrindas. Jis leidžia inicializuoti naujas kvantines schemas, vartus bei kubitus, transliuoti kvantinį kodą į topologijai adaptuotą kodą, bei vykdyti kvantinių schemų optimizacijas.
- **Qiskit Aer**. Tai yra galingas kvantinio kompiuterio simuliatorius, palaikantis kvantinio triukšmo simuliaciją.
- **Qiskit Ignis**. Šis komponentas leidžia dirbti kvantinio triukšmo charakterizavimu ir klaidas taisančiais kodais.
- **Qiskit Aqua**. Šitas komponentas turi visus svarbiausius kvantinius algoritmus, kuriuos galima iškart paleisti be papildomų pasiruošimų.

3.3. Kvantinių schemų adaptavimas kubitų topologijai Qiskit sistemoje

Kiekvienoje IBMQ sistemoje *SWAP* operacija yra skaidoma į tris *CNOT* operacijas. Kas yra labai problematiška klaidų atžvilgiu. Todėl reikalingi algoritmai, leidžiantys rasti minimalų *SWAP* operacijų skaičių norint pritaikyti pasirinktą algoritmą topologijai. Toks algoritmas patenka į NP–pilnų problemų aibę, todėl Qiskit naudoja skirtingus euristinius ir randomizuotus algoritmus. Vienas iš tokių algoritmų yra *Lookahead Swap* algoritmas:

3.3.1. „Lookahead Swap“ algoritmas

Algoritmo tikslas yra pakeisti pradinį virtualų algoritmą taip, kad jis atitiktų duotą kompiuterio kubitų topologiją. Algoritmas pereina pro galimas *SWAP* vartų kombinacijas naudojant supaprastintą geriausio–pirmo spindulio paieškos algoritmą – modifikuotą paieškos į plotį algoritmą.

Algoritmo žingsniai yra tokie[Qis21]:

1. Rasti vartus, kurie gali būti atlikti pateiktoje topologijoje ir juos pažymėti kaip paruoštus.

2. Kiekvienai galimai *SWAP* vartų kombinacijai, apskaičiuoti gautą algoritmo vartų išdėstymą ir išrikiuoti gautas kombinacijas pagal atstumus tarp susietų kubitų (kuo atstumai mažesni – tuo geriau).

3. Paimti keturis geriausius variantus ir kiekvienam pakartotinai apskaičiuoti naują algoritmo vartų išdėstymą.

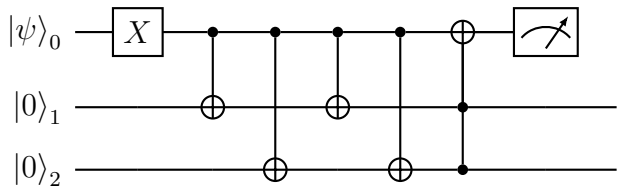
4. Kartoti šį procesą kol nebus apskaičiuoti 256 išdėstymai.

5. Paimti geriausią išdėstymą.

6. Kartoti algoritmą kol visi vartai nebus paruošti.

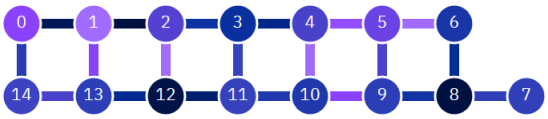
3.3.2. Schemos adaptavimas nenaudojant *SWAP* algoritmų

Žinodami, kad algoritmo adaptavimas generuoja nemažai papildomų operacijų ir turėdami topologijos aprašą, galime adaptuoti savo algoritmus rankiniu būdu ir taip žymiai sumažinti klaidų skaičių. Buvo atliktas bandymas, kurio metu trijų kubitų surišimo schema (8 pav.) buvo adaptuota IBMQ Melbourne kompiuterio topologijai rankiniu būdu, ir buvo palyginta skaičiavimų kokybė abiem atvejais.



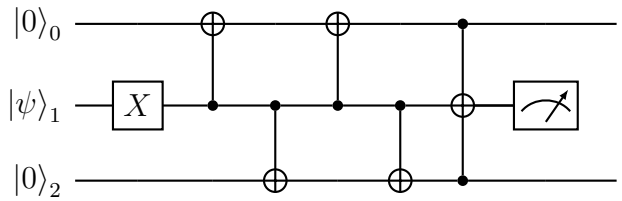
8 pav. Trijų kubitų bito apvertimo taisymo kodas

Įvykdžius 8192 bandymų IBM kvantiniame kompiuteryje, gauta kokybė buvo lygi 55.7%. Patikrinę Melbourne kompiuterio kubitų topologiją (pav. 9) matome, kad kubitai 0 ir 2 nėra tiesiogiai sujungti, dėl ko bus generuojamas papildomas kodas transliavimo metu.



9 pav. IBMQ Melbourne kubitų topologija

Galime pamėginti pakeisti algoritmą taip, kad koduojamas būtų kubitas 1 (pav. 10). Atlikus tokius pertvarkymus, algoritmo kokybė padidėjo iki 86%. Taigi, rankinis algoritmo schemos pertvarkymas yra labai efektyvus būdas padidinti skaičiavimo kokybę nekeičiant kubitų bei vartų skaičių, nors tai yra problematiška, dirbant su labai dideliais kvantiniais algoritmais.



10 pav. Pakeistas trijų kubitų bito apvertimo taisymo kodas

4. Klaidos kvantiniuose kompiuteriuose

NISQ tipo kvantiniai kompiuteriai yra silpnai apsaugoti nuo kvantinės dekoherencijos – išorinių poveikių, tokių kaip sąveika su kitomis dalelėmis, elektromagnetinės bangos bei temperatūros pokyčiai. Dekoherencija sukelia labai daug klaidų kvantiniuose skaičiavimuose, dėl ko kai kurie algoritmai šiuolaikiniuose kvantiniuose kompiuteriuose yra neįmanomi.

4.1. Kvantinio triukšmo modeliai

Dekoherencija kvantiniuose kompiuteriuose gali generuoti skirtingo tipo klaidas, todėl kvantiniai triukšmai yra klasifikuojami pagal jų sukeliamus efektus. Įprastai yra išskiriami du svarbiausi kvantinio triukšmo modeliai – amplitudės ir fazės slopinimas[AAG19].

Egzistuojančios kvantinio triukšmo analizės [AAG19] parodo, kad amplitudės ir fazės slopinimai gali būti modeliuojami kaip kvantiniai operatoriai, kurie aprašo kvantinės sistemos evoliuciją. Mums svarbu yra tinkamai modeliuoti triukšmą tam, kad galėtume efektyviai ieškoti būdų kovoti su jo sukeliamais neigiamais efektais, pavyzdžiui, kurti specialius klaidas taisančius kodus. Žinodami, kaip triukšmas keičia kubito būseną, žinosime, kaip ją atstatyti.

4.1.1. Amplitudės slopinimas

Amplitudės slopinimas – tai triukšmo tipas, kuris sukelia kvantinės sistemos energijos praradimą. Vienas iš pavyzdžių – elektrono, kuris yra būsenoje $|1\rangle$ spontaninė emisija, po kurios yra išskiriamas fotonas, ir elektronas keičia savo būseną į $|0\rangle$. Amplitudės slopinimo operatorius gali būti užrašomas kaip operatorius, naudojantis kintamąjį γ, nurodantį spontaninės emisijos tikimybę.[AAG19]:

$$L_{amp} = \sqrt{\gamma} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{13}$$

4.1.2. Fazės slopinimas

Fazės slopinimas aprašo kvantinės informacijos praradimą be sistemos energijos lygio pokyčių. Kvantinis operatorius aprašomas tokiu būdu[AAG19]:

$$L_{phase} = \sqrt{\lambda} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{14}$$

Reikšmės matricoje atitinka Pauli-Z operatorių. Taigi, fazės slopinimas - tai kubito posūkis aplink Z ašį Bloch'o sferoje tam tikru kampu, proporcingu skaičiui λ, kuris apibūdina modeliuojamo triukšmo stiprį.

4.2. Kvantinių klaidų įtaka algoritmų vykdymui

Klaidos kvantiniuose kompiuteriuose – labai svarbi problema, dėl kurios kai kurie algoritmai yra panaudotini vien tik teorijoje. Pavyzdžiui, vienas svarbiausių kvantinių algoritmų – Šoro algoritmas, neturi realios praktinės panaudos šiuolaikiniuose triukšminguose NISQ kvantiniuose kompiuteriuose. Šitame poskyryje bus atliktas Šoro faktorizavimo algoritmo bandymas Qiskit sistemoje, simuliuojant triukšmą, ir idealiosiomis sąlygomis.

Šoro algoritmo tikslas – surasti funkcijos

$$f_{a,N}(x) = a^x \text{mod} N \tag{15}$$

periodą, t.y mažiausią skaičių r su kuriuo bus teisinga lygybė [YM08]:

$$f_{a,N}(r) = a^r \text{mod} N = 1 \tag{16}$$

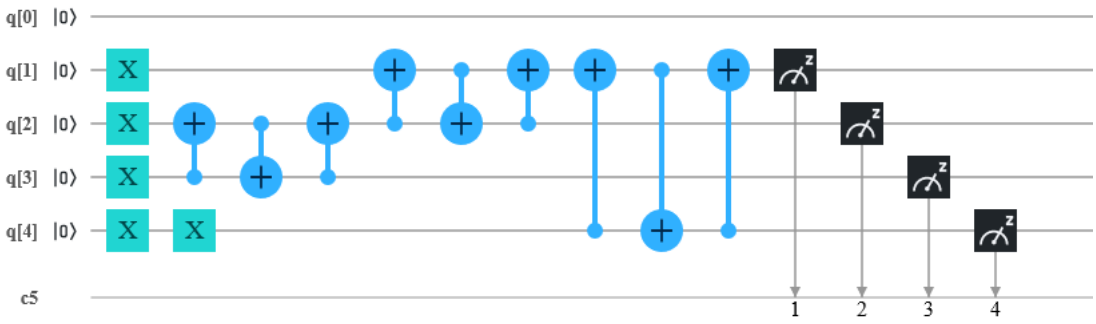
Čia N – faktorizuojamas skaičius, o a – atsitiktinis skaičius, neturintis bendro daliklio su N.

Bandymai buvo atlikti Qiskit Aer simuliatoriuje, naudojant Qiskit Aqua Šoro algoritmo implementaciją. Skaičiaus 15 faktorizavimui, Aqua generuoja kvantinę schemą, susidedančią iš komponentų, nurodytų lentelėje 2.

Operacija	Kiekis
Hadamardo vartai	8
X vartai	1
Kvantinė Furjė transformacija	1
1 mod 15	6
2 mod 15	1
4 mod 15	1

2 lentelė. Šoro algoritmo skaičiaus 15 faktorizavimui sudėtis

Funkcijos 1 mod 15 , 2 mod 15, 4 mod 15 yra periodo paieškos funkcijos. Šoro algoritmas bando atspėti periodą r, perrinkdamas galimus šio skaičiaus variantus. Testavimo metu skaičius a buvo lygus 2. Pav. (11) parodytas tokios periodo paieškos schemos pavyzdys. Ši schema buvo sugeneruota skaičiui $a = 7$ [MS12].



11 pav. IBM Q sugeneruota kvantinė schema periodo suradimui**

Testavimo metu buvo faktorizuojami skaičiai 15 ir 21 ir matuojamas vykdymo laikas naudojant Python kalbos *time* biblioteką. Triukšmo simuliacijai buvo paimti aktualūs *IBMQ Melbourne* 16 kubitų kompiuterio triukšmo bei kubitų greičio duomenys, paimti naudojant *IBMQ API*, taigi, simuliuojamas triukšmas atitinka realybę. Žemiau pateikti bandymo rezultatai:

Faktorizuojamas skaičius	Laikas, s.
15	2.84
21	13.57

3 lentelė. Faktorizavimo bandymo rezultatai idealiosiomis sąlygomis

Faktorizuojamas skaičius	Laikas, s.
15	8.83
21	127.2

4 lentelė. Faktorizavimo bandymo rezultatai simuliuojant triukšmą

Atliktas bandymas parodo neigiamą kvantinio triukšmo poveikį algoritmų efektyvumui. Be to, žinodami, kad realiam algoritmo panaudos atvejui – 1024 bitų RSA kodo dešifravimui mums prireiktų 2050 kubitų ir $4.81 * 10^{12}$ Toffoli vartų[HRS17], matome, kad platus Šoro algoritmo panaudojimas yra neįmanomas be aukštos kokybės, klaidoms atsparių kubitų.

**IBM Quantum Experience, https://quantum-computing.ibm.com/composer/docs/iqx/guide/shors-algorithm

4.3. Kvantinių klaidų taisymas

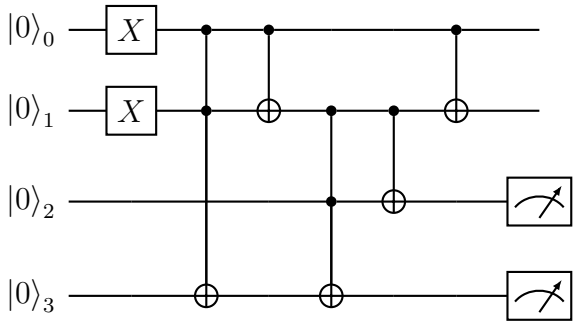
Neigiamų kvantinio triukšmo padarinių mažinimas šiuo metu yra viena didžiausių kvantinių kompiuterių problemų. Vienas iš galimų sprendimų yra kvantiniai klaidas taisantys kodai. Svarbiausia kvantinių klaidas taisančių kodų problema yra tai, kad jie negali remtis mums žinoma klasikine kodavimo teorija, nes dėl teoremos apie uždraustą klonavimą [WZ82] mes negalime kopijuoti kvantinių būsenų, dėl ko kodai, paprasčiausiai kartojantys pranešimą, kvantiniame pasaulyje yra neįmanomi, ir kvantiniai klaidas taisantys kodai turi būti įgyvendinti naudojant kvantinį surišimą.

Pirmą tokį kodą, kuris kokybiškai taisytų vieną atsitiktinę klaidą kubite, pasiūlė P. Šoras. Šoro kodas – tai devynių kubitų kodas, kuris užtikrintai taiso vieną atsitiktinę klaidą, bet tik tuo atveju, kai yra gerai žinoma, kur klaida atsiras. Jau egzistuoja ir naujesni Šoro kodo analogai. Mažiausias kodas, gebantis ištaisyti vieną atsitiktinę klaidą yra $[5, 1, 3]$ kodas[GYW+21].

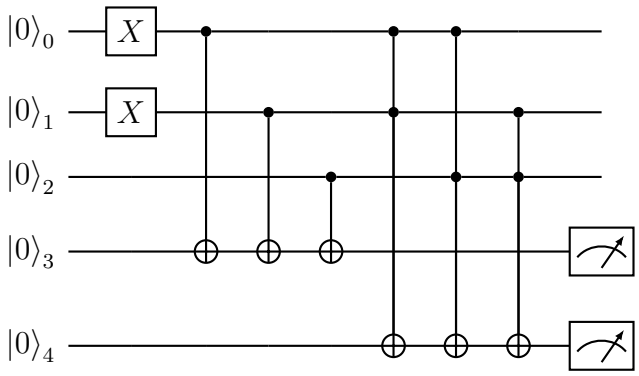
Nors kvantiniai klaidas taisantys kodai yra įmanomi ir mes galime įsitikinti tuo kad jie veikia teorijoje, realybėje, ypač NISQ architektūroje, kvantinių klaidas taisančių kodų panaudojimas programos lygmenyje atneša daugiau bėdų nei naudos. Priežastis yra labai paprasta: kodas, kuris turi taisyti klaidas, irgi gali būti paveiktas klaidų, dėl ko galutinė kvantinių skaičiavimų kokybė smarkiai mažėja. Tyrime [LB20] atlikti bandymai parodė, kad taikyti klaidas taisantys kodai vidutiniškai mažino skaičiavimo kokybę apie du kartus.

4.4. Kvantinių schemų optimizavimas mažinant vartų skaičių

Kadangi klaidas taisantys kodai nėra efektyvūs NISQ architektūroje dėl to, kad generuojama daug pašalinio triukšmingo kodo, yra logiška bandyti mažinti vartų skaičių siekiant padidinti kvantinių skaičiavimų kokybę. Galime paimti dvi skirtingas schemas, atliekančias tą pačią užduotį, bet turinčias skirtingą naudojamų kubitų ir vartų skaičių, ir palyginti šių schemų kokybes. Pavyzdžiui, yra dvi sudėties algoritmo schemos, naudojančios 4 ir 5 kubitus atitinkamai:



12 pav. Keturių kubitų sudėties algoritmas



13 pav. Penkių kubitų sudėties algoritmas

Šitos schemos sumuoja du vienetus, nes kubitai 0 ir 1 yra dėmenys, ir jie yra inicializuoti į reikšmę lygią 1. Tikėtinas 4 kubitų schemos rezultatas yra 01, o 5 kubitų – 10. Schemos buvo konvertuotos į *OpenQASM* kodą, kuris buvo paleistas *IBMQ Melbourne* kompiuteryje. Buvo atlikti 1024 bandymai. Kokybė bus lygi 100% tik tuo atveju, kai visada bus gaunamas teisingas atsakymas. Atlikus bandymus, buvo gauti tokie rezultatai:

Schema	Vartų skaičius	Kokybė, %
4 kubitų	57	56.1
5 kubitų	85	38

5 lentelė. Kokybės priklausomybė nuo vartų skaičiaus

Taigi, lentelė 5 parodo, kad kokybė yra atvirkščiai proporcinga vartų skaičiui, iš ko galime spręsti, kad geriausias būdas gauti aukštos kokybės skaičiavimus – mažinti naudojamų vartų skaičių.

5. Kvantinių vartų redukcija ir optimizacijos verifikavimas

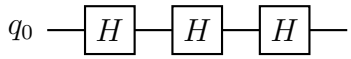
Įsitikinę, kad naudojamų vartų mažinimas padidina skaičiavimų kokybę, turime rasti būdą rasti ir išmesti nereikalingus vartus iš kvantinės schemos, nekeičiant galutinės skaičiavimo būsenos. Laimei, tai yra įmanoma dėl tam tikrų kvantinių kompiuterių savybių:

1. Kvantiniai vartai tai unitarinės Ermito matricos
2. Kvantiniai vartai yra apgręžiami
3. Kvantiniai vartai visada turi savo atvirkštines matricas

Minėtų savybių veikimu galime įsitikinti atlikę keletą trivialių matematinių operacijų.

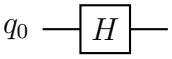
5.1. Kvantinių vartų komutatyvumas

Tarkime, turime tokią kvantinę schemą:



14 pav. Trys Hadamard operacijos

Remdamiesi aukščiau pateiktomis taisyklėmis, galime įsitikinti, kad schema 14 yra ekvivalenti tokiai schemai:



15 pav. Viena Hadamard operacija

Turime įsitikinti, kad tokia lygtis yra teisinga

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \equiv \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \tag{17}$$

Patikrinkime, ar Hadamardo operacija yra unitarinė. Sudauginę dvi Hadamard operacijos matricas gauname:

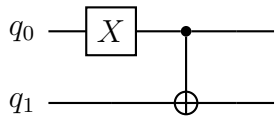
$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I \tag{18}$$

Gavome vienetinę matricą. Pagal apibrėžimą, kvadratinė matrica yra unitarinė jei $TT^{\dagger} = I$, taigi Hadamard matrica yra unitarinė ir yra pati sau atvirkštinė.

Taip pat žinome, kad $TI = T\ IT = T$, todėl lygtis 17 yra teisinga. Parodyta, kad Hadamard vartai yra tarpusavyje komutatyvūs. Jie panaikina vienas kitą, arba gali būti sukeista jų vykdymo tvarka nekeičiant algoritmo būsenos. Žinodami, kad visi kvantiniai vartai turi parodytas savybes, galime teigti, kad visi vienodi kvantiniai vartai yra tarpusavyje komutatyvūs. Šitą savybę galima naudoti optimizuojant kvantines schemas.

5.2. Vartų pertvarkymo operacijos

Kvantinių schemų mažinimas reikalauja vartų pertvarkymo. Turime įsitikinti, kokius vartus galime keisti vietomis nekeičiant algoritmo būsenos, o kokius – ne. Pavyzdžiui, nėra sudėtinga pastebėti, kad vartai, keičiantys skirtingus kubitus gali būti sukeisti, tačiau problemos atsiranda kai norime pakeisti vieno ir dviejų kubitų vartus. Pavyzdžiui, turime tokią schemą:



16 pav. X ir CNOT vartai

Šituo atveju galutinis sistemos būsenos vektorius atrodytų taip:

$$|\psi\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}0 \cdot \begin{bmatrix}0\\1\end{bmatrix}\\ 1 \cdot \begin{bmatrix}0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\1\end{bmatrix} \tag{19}$$

Tačiau jeigu X operacija būtų atlikta po *CNOT* vartų, būsenos vektorius būtų toks:

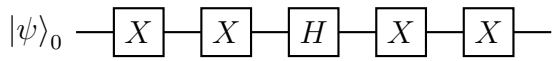
$$|\psi\rangle = \begin{bmatrix}0\\1\end{bmatrix} \otimes \begin{bmatrix}1\\0\end{bmatrix} = \begin{bmatrix}0 \cdot \begin{bmatrix}1\\0\end{bmatrix}\\ 1 \cdot \begin{bmatrix}1\\0\end{bmatrix}\end{bmatrix} = \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \tag{20}$$

Taigi, sukeisti vartus vietomis negalima, nes keičiasi algoritmo būsena. Reikia pabrėžti, kad toks ribojimas yra tik tuo atveju, jei keičiamas kontrolės kubitas. Remiantis tokiomis taisyklėmis, galime atlikti reikalingus schemos pertvarkymus ir rasti komutatyvius kubitus, kuriuos galima panaikinti.

5.3. Schemų optimizacijos verifikavimas

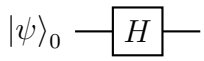
Kvantinių schemų optimizacija redukuojant pasikartojančius vartus remiasi matematinėmis taisyklėmis, kurios visada grąžina norimą rezultatą, todėl jas galime interpretuoti kaip aksiomas. Taigi, jeigu atlikti pertvarkymai atitinka vartų komutatyvumo taisykles, galime būti tikri kad optimizacija buvo atlikta teisingai, ir ji nekeičia algoritmo būsenos. Tačiau įmanomi atvejai, kada pertvarkymai buvo įgyvendinti netinkamai. Tam, kad apsisaugoti nuo tokių defektų, reikalingas būdas verifikuoti atliktą optimizaciją. Akivaizdus variantas yra palyginti neoptimizuotą ir optimizuotą schemą ir įsitikinti kad jos yra ekvivalenčios. Ši užduotis yra labai sunki, ji patenka į QMA-pilnų užduočių klasę – NP problemų analogą, apibrėžiantį kvantines užduotis[BRW20]. Reikalingas kitas verifikavimo metodas. Paprasčiausiai galime paleisti dvi schemas ir palyginti gautus būsenų vektorius. Toks sprendimas yra įmanomas tik simuliatoriuje, nes realioje kvantinėje mašinoje vienodas rezultatas neįmanomas dėl klaidų atsitiktinumo. Papildoma problema yra ta, kad toks sprendimas potencialiai yra labai lėtas, ypač jeigu verifikuojamos yra schemos, turinčios tūkstančius vartų.

Tyrime [BRW20] buvo parodytas verifikavimo metodas, kuris leidžia patikrinti, ar optimizacija buvo atlikta tinkamai, vykdant tik vieną skaičiavimo simuliatoriuje bandymą. Metodas remiasi tuo, kad kvantinės schemos irgi yra apgręžiamos, nes kiekviena operacija yra apgręžiama, t.y. $GG^\dagger = I$ kur G yra nagrinėjama kvantinė schema. Turime tokią schemą:



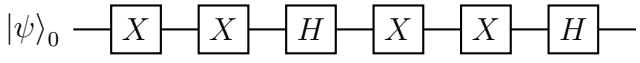
17 pav. Neoptimizuota kvantinė schema

Optimizuota, schema 17 atrodo taip:



18 pav. Optimizuota kvantinė schema

Galime sujungti dvi schemas ir gauti vieną didesnę schemą 19:



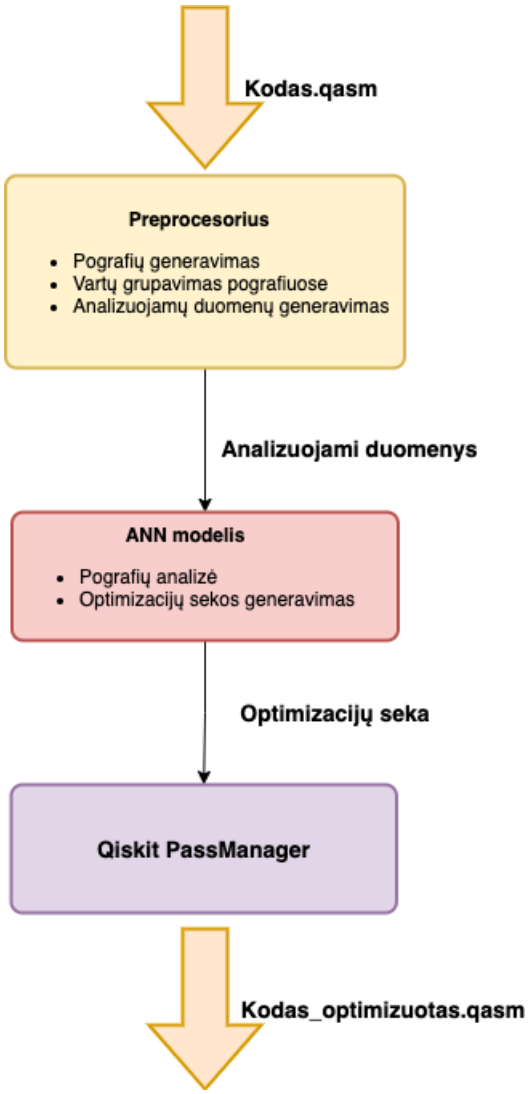
19 pav. Verifikavimo schema

Jeigu schemos 19 būsenų vektorius yra lygus $|0\rangle$, tai optimizacija yra verifikuota sėkmingai, nes abi schemos vienodai keičia kubitų būsenas.

6. Adaptyvus optimizatorius

Šiame darbe yra pristatomas kvantinių schemų optimizatorius, kuris remiasi aukščiau minėtomis taisyklėmis schemų pertvarkymams bei mašininio mokymosi metodais schemos šablonų paieškai. Optimizatorius yra sukurtas remiantis Qiskit klasėmis, jį galima lengvai įdėti į savo projektą ir naudoti schemų analizei bei optimizavimui. Šiame skyriuje yra aprašytas optimizatoriaus veikimo procesas bei svarbiausi algoritmai naudojami schemų pertvarkymams.

6.1. Optimizatoriaus eiga



20 pav. Optimizatoriaus eigos schema

Optimizatorius susideda iš dviejų svarbiausių komponentų – Preprocesoriaus ir dirbtinių neuronų tinklo (*Artificial neural network*, ANN) modelio. Qiskit PassManager naudojamas prieigai prie transliatoriaus gauti ir atlikti optimizacijas pagal nustatytą tvarką. Optimizatorius buvo kuriamas taip, kad jį būtų įmanoma greitai praplėsti naujomis optimizacijomis arba šablonais, kurie turi būti ieškomi ANN analizės metu.

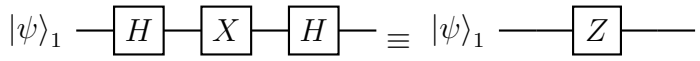
6.2. Pografių generavimo algoritmas

Optimizatoriaus preprocesosius iš gauto QASM failo generuoja kvantinės schemos grafą, kuris yra skaidomas į pografius naudojant tokį algoritmą:

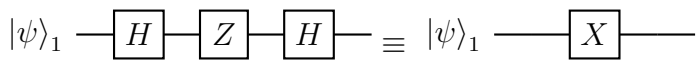
```
Given:
    dag_list: []

Declare:
    subdag_array: []
    current_array: []
    dag_size: integer

Main loop:
    dag_size = dag_list.size

    for(i = 0; i < dag_size; i++)
    {
        # Check if sub-dag elements can be interchanged
        if(can_be_interchanged(dag_list[i], dag_list[i+1]))
        {
            # Current element is placed in the current sub-dag array
            current_array.insert(dag_list[i])
        }
        else
        {
            # Elements can't be interchanged, saving current sub-dag and creating
                new sub-dag array

            current_array.insert(dag_list[i])
            subdag_array.insert(current_array)
            current_array = []
        }
    }

    # Checking if an unsaved sub-dag array exists

    if(current_array.size != 0)
    {
        subdag_array.insert(current_array)
    }
```

Sugeneruotuose pografiuose vienodi vartai yra grupuojami – tai yra įmanoma, nes pagal komutatyvumo taisykles visi pografio vartai gali būti sukeisti tarpusavyje. Taip pat sugeneruoto-

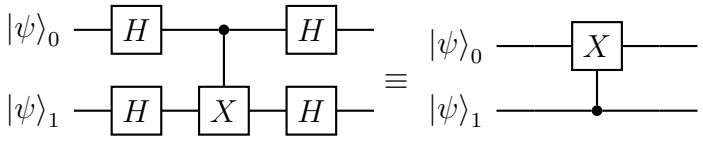
se pografiuose yra aptinkami tokie pografiai, kurie prasideda ir baigiasi Hadamard operacija tam pačiam kubitui. Tokius pografius svarbu aptikti nes jie gali būti optimizuoti pagal tokias taisykles[ASD14]:



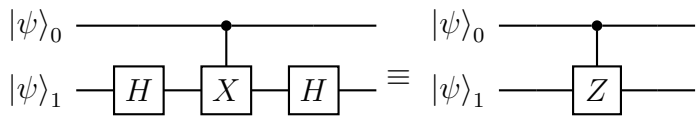
(21)



(22)



(23)



(24)

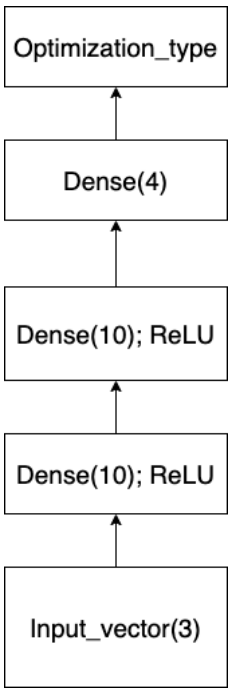
Pografiai yra pakartotinai apkarpomi, kad jų dydis būtų 3. Tai yra padaryta dėl to, kad dažnai užtenka trijų vartų šablono pastebėjimui. Jeigu gaunasi mažiau negu trys vartai pografyje, prie jo pridedamos vienetinės operacijos, kurios nekeičia kubito būsenos. Pografių duomenys yra išsaugomi csv faile, kuris yra paduodamas ANN modeliui, įgyvendintu naudojant *Tensorflow* paketą.

6.3. ANN modelis

Neuronų tinklo modelis buvo sukurtas naudojant *Tensorflow* ir *Keras*. Problema, sprendžiama optimizatoriuje yra paprasčiausia klasifikacijos užduotis, aprašyta darbe [DPR16]. Modelio tikslas – duotam duomenų masyvui nustatyti galimą optimizacijos tipą – komutatyvių vieno kubito vartų panaikinimą arba *CNOT* vartų panaikinimą. Modelio schema pavaizduota pav. 21

Tokia modelio konfigūracija buvo parinkta euristiškai, atlikus keletą ANN modelio apmokymo bandymų esant skirtingiems sluoksnių dydžiams. Gauti duomenys yra išsaugoti kodo repozitorijoje ir šito darbo prieduose (lentelė 8). Didesnis modelis šitam optimizatoriaus prototipui nebuvo reikalingas, nes buvo ieškoma tik keturių galimų būsenų (optimizacija nerasta, komutatyvių vartų panaikinimas, *CNOT* vartų panaikinimas ir vienodų fazės posūkių vartų sujungimas į vieną fazės posūkio operaciją). Aktyvavimo funkcija yra *ReLU*. Modelis yra apmokomas naudojant *SGD* algoritmą. Duomenys, skirti modelio apmokymui, yra pažymėti. Tam, kad modelis kokybiškai klasifikuotų optimizacijas, užteko 82 šablonų pavyzdžių modelio apmokymui bei 24 pavyzdžių modelio testavimui. Pografių operacijos yra unikaliai identifikuojamos *float* tipo skaičiais.

Modelis grąžina spėjimus, pagal kuriuos generuojamas yra optimizacijų sąrašas. Kiekviena optimizacija irgi yra unikaliai identifikuota. Pavyzdžiui, po ANN analizės proceso bus gautas rezultatas $[2, 1, 3]$. Tai reiškia, kad turi būti atliktos tris optimizacijos apibrėžtos tokiais skaičiais.



21 pav. ANN modelio schema

6.4. Optimizacijų taikymas

Gauta optimizacijų tvarka yra perduodama Qiskit PassManager objektui. Šis objektas leidžia nurodyti optimizacijas, kurios turi būti atliktos transliavimo metu, arba prieš jį. Skirtumas yra tas, kad transliavimo metu generuojama adaptacija schemos topologijai, ir yra galimos papildomos optimizacijos, kurios yra neįmanomos, jei topologija nėra žinoma. Aprašytas optimizatorius keičia schemą prieš adaptavimą topologijai. Qiskit PassManager apibrėžia nemažą skaičių optimizacijų, bet yra leidžiama sukurti ir savo optimizacijas. Kiekviena optimizacija reikalauja atskiro etapo (Pass), po kurio gaunama laikina schema. Optimizuota schema yra konvertuojama atgal į QASM failą, kuris yra saugomas kompiuterio diske. Optimizacijos procesas yra baigtas.

7. Optimizatoriaus bandymai

Optimizatoriaus preprocesoriaus bei ANN modelio patikrinimui buvo atlikta keletas bandymų. Pirmiausia, buvo generuojamos atsitiktinės schemos, kurios vėliau buvo optimizuojamos. Taip buvo siekiama patikrinti, ar pografių generavimas ir rikiavimas veikia, nes turint atsitiktinę schemą yra mažiau šansų rasti šablonus be papildomo rikiavimo. Vykdant šį bandymą, buvo generuojamos schemos, kuriose gali būti iki N^3 vieno kubito vartų, kur N yra parinktas kubitų skaičius. Kiekvienam kubitų skaičiui atlikta 100 atsitiktinių schemų optimizacijų ir skaičiuojamas vidutinis panaikintų vartų skaičius. Buvo gauti rezultatai, parodyti lentelėje 6:

Schema	Pradinis operacijų skaičius	Optimizuota schema
random 3 qubit	20.2	18.7
random 4 qubit	46.1	42.9
random 5 qubit	89.1	82.9
random 6 qubit	152.6	141.9
random 7 qubit	239.8	224.0
random 8 qubit	355.9	332.1
random 9 qubit	503.6	471.5
random 10 qubit	689.6	645.8
random 11 qubit	914.0	857.5
random 12 qubit	1183.1	1110.4
random 13 qubit	1503.6	1410.3
random 14 qubit	1871.1	1760.2
random 15 qubit	2302.64	2162.76
random 16 qubit	2788.13	2621.68

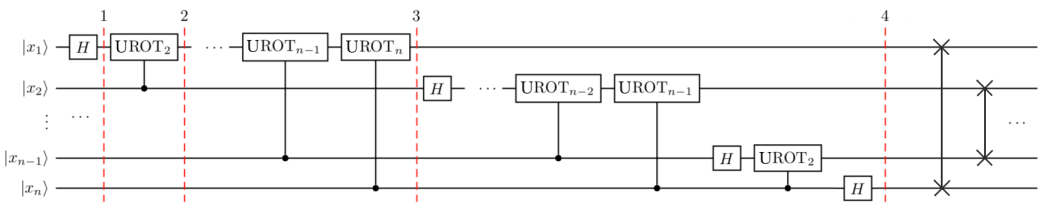
6 lentelė. Atsitiktinių schemų optimizavimo bandymų rezultatai

Kiti bandymai buvo atlikti su realiosiomis kvantinėmis schemomis. Visos schemos buvo konvertuotos į IBMQ bazinių vartų schemas. Žemiau yra pateiktos optimizuojamų algoritmų schemų struktūros bei gauti rezultatai.

7.1. Kvantinės Furjė transformacijos schema

Kvantinė Furjė transformacija, tai klasikinės diskrečiosios Furjė transformacijos adaptacija klasikiniams kompiuteriams. Šis algoritmas yra plačiai naudojamas kituose kvantiniuose algoritmuose, pavyzdžiui, Šoro algoritme.

Kvantinės Furjė transformacijos schema atrodo taip:

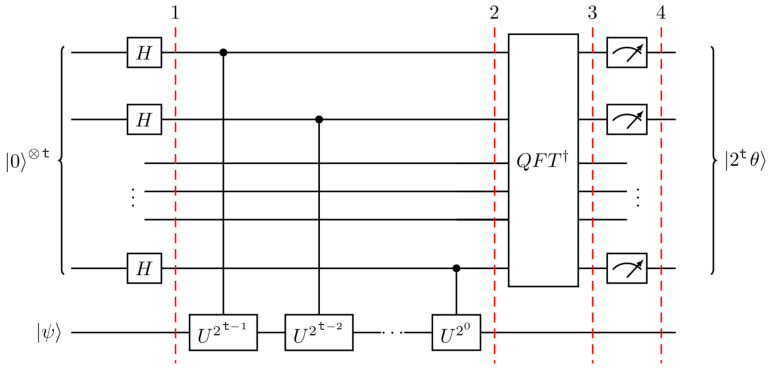


22 pav. N kubitų KFT schema

KFT sudėtingumas yra $O(NlogN)$

7.2. Kvantinio fazės aproksimavimo schema

Kvantinė fazės aproksimavimo schema, kaip ir KFT, įprastai yra didesnių algoritmų, tokių kaip Šoro ir HHL algoritmai, dalis. Šitos aprokcimacijos užduotis – rasti sistemos tikrinį vektorių σ, nurodantį sistemos pokytį.

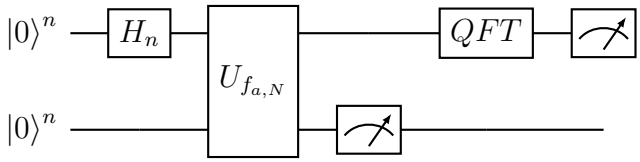


23 pav. N kubitų KFA schema

KFA algoritmo sudėtingumas yra lygus $O(1/\epsilon)$, kur ϵ – aproksimacijos paklaida.

7.3. Šoro algoritmo schema

Bendru atveju, Šoro algoritmo schema atrodo taip:



24 pav. Šoro algoritmo schema

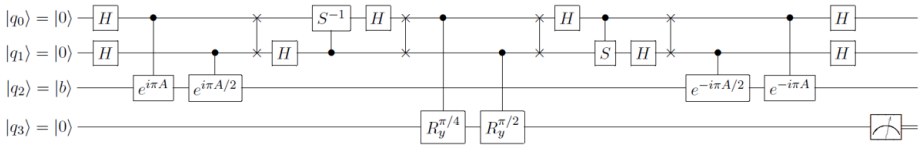
Šoro algoritmas reikalauja bent $3m$ kubitų, kur m – bitų skaičius faktorizuojamame skaičiuje, t.y. faktorizuodami skaičių 15 turime panaudoti bent 12 kubitų. Maža to, periodo paieškos funkcija $f_{a,N}$ generuoja būseną $\sum_{x=0}^{2^{m-1}} |x|a^x \mathrm{mod} N\rangle$, dėl ko vartų skaičius smarkiai išauga.

7.4. HHL algoritmas

HHL algoritmas yra naudojamas tiesinių lygčių sistemų sprendimo aproksimavimui. Pavyzdžiui, turėdami

$$A = \begin{bmatrix} 1 & -\frac{1}{3} \\ -\frac{1}{3} & 1 \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{25}$$

turime rasti tokį x, su kuriuo teisinga lygybė $Ax = b$.



25 pav. 4 kubitų HHL algoritmo schema

HHL algoritmo sudėtingumas yra $poly(\log N, k)$. Tai yra eksponentinis pagreitinimas, lyginant su greičiausiu klasikiniu algoritmu, kurio sudėtingumas yra $O(N\sqrt{k})$[HHL09].

7.5. Gautų rezultatų aptarimas

Kvantinių algoritmų optimizavimo rezultatai yra tokie:

Schema	Pradinis operacijų skaičius	Transliuota schema	Optimizuota schema
shor factor 9	18	14655	13747
shor factor 15	18	12360	11591
shor factor 21	22	23900	22423
shor factor 35	26	44481	41747
QFT 4 qubit	12	40	38
QFT 8 qubit	40	160	140
QFT 12 qubit	84	360	306
QFT 16 qubit	144	640	536
QPE 4 qubit	22	57	57
QPE 8 qubit	174	753	631
QPE 12 qubit	2142	10529	8469
QPE 16 qubit	32926	164385	131571
HHL 2x2	140	28278	28276
HHL 3x3	161	819289	819287

7 lentelė. Optimizatoriaus bandymų rezultatai

Atlikus bandymus buvo pastebėta, kad atsitiktinių schemų atveju optimizacija panaikindavo iki 7% vartų, o realių algoritmų atveju – iki 20% vartų. Tokį skirtumą galima paaiškinti tuo, kad realios schemos dažniausiai turi pasikartojančius elementus, pavyzdžiui smulkesnių funkcijų schemas, kurias galima aptikti ir panaikinti. Blogiausius rezultatus parodė HHL algoritmo optimizacija. Tai yra dėl to, kad transliuota schema beveik visiškai susideda iš vartų, kurių neįmanoma sukeisti vietomis, dėl ko atsiranda pografiai iš dviejų vartų ir *I* operacijos, ir jokia optimizacija nėra aptinkama.

Kitas pastebėjimas yra tas, kad panaikintų vartų dalis žymiai didėja optimizuojant didesnes schemas. Šitas pastebėjimas labiausiai tinka 16 kubitų kvantinės fazės aproksimavimo schemai. Taigi algoritmai, susidedantys iš tokių schemų gali būti optimizuojami labai efektyviai.

Darbo rezultatai ir išvados

Šiame darbe buvo sukurtas kvantinių schemų optimizatorius ir kvantinių schemų analizavimo mechanizmas, orientuotas į plečiamumą. Šis optimizatorius – tai galima bazė tolimesniems darbams, nes įgyvendintas funkcionalumas, kuris remiasi vartų komutatyvumo taisyklėmis, padeda atlikti paprasčiausius schemos pertvarkymus bei paruošti ANN modeliui tinkamus duomenis apie gautus pografius, kas toliau leidžia ieškoti šablonų ir vykdyti atitinkamus tiems šablonams veiksmus.

Atlikti bandymai parodė, kad praplėstas šablonų paieškos mechanizmas padės greičiau analizuoti kvantines schemas, ieškant galimų optimizacijų. Svarbu paminėti, kad bandymuose naudojamas optimizatorius ieškojo pografiuose tik komutatyvių vartų panaikinimo, kitos galimos optimizacijos buvo ignoruojamos. Sukurtas optimizatorius nėra visagalis – vis tiek galima surasti optimizacijų, kurias pastebi žmogus.

Buvo išanalizuoti baziniai NISQ kvantinių kompiuterių bruožai. Matome, kad vienintelis būdas efektyviai kovoti su klaidomis – sumažinti jų tikimybę mažinant naudojamų vartų skaičių. Darbo metu buvo išnagrinėta Qiskit platformos struktūra. Įrankių rinkinys, sukurtas IBM, yra gera platforma tolimesniems tyrimams, o klaidų simuliacijos mechanizmai pagreitina darbą, nes visus bandymus galima atlikti simuliatoriuje žinant, kad galutiniai rezultatai realiajame kompiuteryje bus panašūs.

Optimizacijos verifikavimui buvo panaudotas metodas, kuris remiasi logine išvada, kad schema, susidedanti iš apgręžiamų vartų, irgi būtinai yra apgręžiama. Ši išvada leidžia turėti kokybišką verifikavimo metodą nenaudojant automatinių teoremų įrodymo mechanizmų arba kvantinių schemų būsenų aproksimacijos.

Tolimesnis darbas. Šitas optimizatorius, kaip buvo minėta, neišnaudoja visų galimų šablonų. Taip yra dėl kelių priežasčių. Pirmiausia, pografių generavimo metodas nėra pakankamai apibendrintas, dėl ko kai kurios komutatyvumo taisyklės yra ignoruojamos. Papildomai, optimizatorius neatsižvelgia į kubitų topologijas, dėl ko kai kurie pertvarkymai taip pat yra ignoruojami. Kitas galimas optimizatoriaus pakeitimas yra variatyvaus dydžio pografių generavimas ir analizė. Šiuo metu naudojamas ANN modelis analizuoja tik pografius iš trijų vartų, bet kai kurie pertvarkymai yra galimi tik matant pilną pografį, kuris nebūtų apkarpytas. Atlikus minėtus optimizatoriaus pakeitimus, būtų ne tik padidinta optimizacijų kokybė, bet ir sumažintas poreikis papildomose, žmogaus vykdomose optimizacijose.

Šaltiniai

[AAG19] Abdullah Ash-Saki, Mahabubul Alam **and** Swaroop Ghosh. Study of decoherence in quantum computers: A circuit-design perspective. *CoRR*, abs/1904.04323, 2019. arXiv: 1904.04323. URL: http://arxiv.org/abs/1904.04323.

[ASD14] Nabila Abdessaied, Mathias Soeken **and** Rolf Drechsler. Quantum circuit optimization by hadamard gate reduction. **in** Shigeru Yamashita **and** Shin-ichi Minato, **editors***Reversible Computation*, **pages** 149–162, Cham. Springer International Publishing, 2014. ISBN: 978-3-319-08494-7.

[Ben02] C. H. Bennett. Notes on landauer's principle, reversible computation, and maxwell's demon. *Studies in History and Philosophy of Modern Physics*, 34:501–510, 2002.

[BRW20] Lukas Burgholzer, Rudy Raymond **and** Robert Wille. Verifying results of the ibm qiskit quantum circuit compilation flow, 2020. arXiv: 2009.02376 [quant-ph].

[DPR16] Maria De Marsico, Alfredo Petrosino **and** Stefano Ricciardi. Iris recognition through machine learning techniques: a survey. *Pattern Recognition Letters*, 82:106–115, 2016. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2016.02.001. URL: https://www.sciencedirect.com/science/article/pii/S0167865516000477. An insight on eye biometrics.

[GYW+21] Ming Gong, Xiao Yuan, Shiyu Wang, Yulin Wu **andothers**. Experimental exploration of five-qubit quantum error correcting code with superconducting qubits. *National Science Review*, 2021-01. ISSN: 2053-714X. DOI: 10.1093/nsr/nwab011. URL: http://dx.doi.org/10.1093/nsr/nwab011.

[HHL09] Aram W. Harrow, Avinatan Hassidim **and** Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical Review Letters*, 103(15), 2009-10. ISSN: 1079-7114. DOI: 10.1103/physrevlett.103.150502. URL: http://dx.doi.org/10.1103/PhysRevLett.103.150502.

[HRS17] Thomas Häner, Martin Roetteler **and** Krysta M. Svore. Factoring using 2n+2 qubits with toffoli based modular multiplication, 2017. arXiv: 1611.07995 [quant-ph].

[YM08] Noson S Yanofsky **and** Mirco A Mannucci. *Quantum computing for computer scientists*. Cambridge University Press, 2008.

[LB20] Frank Leymann **and** Johanna Barzen. The bitter truth about gate-based quantum algorithms in the nisq era. *Quantum Science and Technology*, 5(4):044007, 2020-09. ISSN: 2058-9565. DOI: 10.1088/2058-9565/abae7d. URL: http://dx.doi.org/10.1088/2058-9565/abae7d.

[LT07] Lev B Levitin **and** Tommaso Toffoli. Thermodynamic cost of reversible computing. *Physical review letters*, 99(11):110502, 2007.

[MS12] Igor L. Markov **and** Mehdi Saeedi. Constant-optimized quantum circuits for modular multiplication and exponentiation. *CoRR*, abs/1202.6614, 2012. arXiv: 1202.6614. URL: http://arxiv.org/abs/1202.6614.

[NRS+18] Yunseong Nam, Neil J. Ross, Yuan Su, Andrew M. Childs **and** Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1), 2018-05. ISSN: 2056-6387. DOI: 10.1038/s41534-018-0072-4. URL: http://dx.doi.org/10.1038/s41534-018-0072-4.

[Pre18] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[Qis21] Qiskit. Qiskit. 2021. URL: https://qiskit.org/documentation/stubs/qiskit.transpiler.passes.LookaheadSwap.html (**urlseen** 2021-04-27).

[VDO+19] Davide Venturelli, Minh Do, Bryan O'Gorman, Jeremy Frank, Eleanor Rieffel, Kyle EC Booth, Thanh Nguyen, Parvathi Narayan **and** Sasha Nanda. Quantum circuit compilation: an emerging application for automated reasoning. **in** *Proc. Schedul. Plan. Appl. Workshop*, 2019.

[WZ82] W. K. Wootters **and** W. H. Zurek. A single quantum cannot be cloned. *Nature*, 299(5886):802–803, 1982-10. ISSN: 1476-4687. DOI: 10.1038/299802a0. URL: https://doi.org/10.1038/299802a0.

Priedai

Programinis kodas

Visas programinis kodas, testai bei testų rezultatai yra patalpinti GitHub repozitorijoje.

Kodas buvo ištestuotas *Windows* sistemoje ir *macOS Big Sur* sistemoje su Apple M1 lustu. Windows sistemoje įmanoma importuoti optimizatorių į savo kodą ir iškviesti jį tokiu būdu:

```
from Optimizer_main import Adaptive_Optimizer
ao = Adaptive_Optimizer()
ao.run_optimization(<kelias įschemos .qasm ąfail>)
```

Apple M1 architektūroje nesikompiliuoja *SciPy* paketas, nuo kurio priklauso Qiskit. Todėl Qiskit kodas yra vykdomas Rosetta 2 aplinkoje. Tačiau Tensorflow paketas neveikia naudojant Rosetta 2. Dėl šitos priežasties yra aprašytas shell script failas, per kurį reikia paleisti optimizatorių:

```
Optimizer_m1.sh <.qasm failo pavadinimas>
```

Abiem atvejais bus sugeneruotas optimizuotas qasm failas.

Neuroninio tinklo sudarymo duomenys

Geriausiam mašininio mokymosi modelio suradimui buvo remiamasi tokiais duomenimis:

Sluoksnių 1 ir 2 dydžiai	Apmokymo tikslumas	Testavimo tikslumas	Rastos optimizacijos
5; 5	91.4%	43.8%	2;3
6; 6	87.1%	63.1%	1;2;3
7; 7	87.1%	45.1%	1;2;3
8; 8	90%	52.6%	1;2;3
9; 9	97.1%	57.6%	1;2;3
10; 10	95%	70.2%	1;2;3
11; 11	95.7%	57.8%	1;2;3
12; 12	94.2%	54.3%	1;2;3

8 lentelė. Skirtingų ANN modelio konfigūracijų rezultatai

Testai buvo vykdomi optimizuojant Šoro skaičiaus 9 faktorizavimo schemą. Rastos optimizacijos koduojamos taip: 1 – komutatyvus panaikinimas, 2 – *CNOT* panaikinimas, 3 – fazės posūkių optimizavimas.