VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ BAKALAURO STUDIJŲ PROGRAMA

Variacinio kvantinio tiesinių lygčių sistemų algoritmo pritaikymo atraminių vektorių klasifikatoriui galimybių tyrimas

Feasibility study of variational quantum linear solver application for support vector machines

Bakalauro baigiamasis darbas

Atliko:	Pijus Petkevičius
Darbo vadovas:	Irus Grinis
Darbo recenzentas:	Linas Petkevičius

Vilnius – 2024

Santrauka

Šiame darbe nagrinėjamas kvantinių skaičiavimų pritaikymas mašininiam mokymuisi, ypač daug dėmesio skiriant variaciniam kvantiniam tiesinių lygčių algoritmui (VQLS) ir atraminių vektorių klasifikatoriui (SVM). Darbe išsamiai aprašoma algoritmo nuostolių funkcija, vektorių, matricų užkodavimas į kvantines grandines ir VQLS algoritmo įgyvendinimas. Pateikiama klasikinio SVM ir VQLS-SVM algoritmų klasifikavimo tikslumų analizė. Pateikiami eksperimentiniai rezultatai naudojant kvantinių skaičiavimų karkasą IBM Qiskit ir įvertinamas skirtingų optimizavimo funkcijų veikimas. Darbo pabaigoje pristatomi pasiūlymai, kaip ateityje galima patobulinti VQLS algoritmą, siekiant padidinti jo efektyvumą ir tikslumą, sprendžiant sudėtingesnius uždavinius su didesnėmis duomenų aibėmis.

Raktiniai žodžiai: IBM Qiskit, Variacinis kvantinis tiesinių lygčių sistemų algoritmas (VQLS), Atraminių vektorių klasifikatorius (SVM), Optimizavimo funkcijos, Kvantinis kompiuteris

Summary

The purpose of this paper is to explore the integration of quantum computing with machine learning, specifically focusing on the Variational Quantum Linear Solver (VQLS) and its application to Support Vector Machines (SVM). It defines algorithm cost function, encoding vectors and matrices into quantum circuits and the implementation of the VQLS algorithm. The paper provides comparative analysis of classical SVM and VQLS-SVM algorithms. It includes experimental results using quantum computing framework IBM Qiskit and evaluates the performance of different optimisation functions. The paper concludes with suggestions for future improvements in the VQLS algorithm, aiming to enhance its efficiency and accuracy in handling more complex data sets.

Keywords: IBM Qiskit, Variational quantum linear solver, Support vector machines, Optimization functions, Quantum computer

Turinys

Įvadas

Žiniasklaidoje didelio dėmesio sulaukęs kvantinių algoritmų ir kompiuterių sektorius atrodo nauja neištirta sritis, tačiau jos ištakas galime rasti praeito amžiaus 8 dešimtmetyje. 1980 metais Paul Benioff pasiūlė Tiuringo mašinos kvantinį modelį [Ben80]. 1985 metais David Deutsch išplėtojo kvantinių kompiuterių idėją teigdamas, kad naudojant kvantinius loginius elementus (*angl. quantum logic gates*), kvantinis kompiuteris galėtų veikti panašiu principu kaip klasikiniai kompiuteriai [Deu85]. XX a. paskutiniame dešimtmetyje, buvo sukurti kvantiniai algoritmai problemoms, kurių sprendimas naudojant kvantinius kompiuterius, yra eksponentiškai greitesnis, nei bandant išspręsti problemas klasikinių kompiuterių pagalba [DJ92; Gro96; Sho99; Sim97].

Tiesinių sistemų sprendimas yra viena iš pagrindinių mašininio mokymo, dalinių išvestinių skaičiavimo dalis [WZP18]. 2009 metais Seth Lloyd, Avinatan Hassidim, Aram Harrow sukūrė kvantinį tiesinių lygčių sistemų sprendimo algoritmą, naudojantis kvantiniu kompiuteriu [HHL09]. Tačiau šiuo metu, rinkoje esantys kvantiniai kompiuteriai yra jautrūs triukšmui, iki 1000 kubitų dydžio (*angl. Noisy intermediate scale-quantum (NISQ)*). Norint realizuoti HHL algoritmą, susiduriama su daug iššūkių, kadangi esami kubitai yra jautrūs išoriniam triukšmui, HHL algoritmo rezultatai tampa mažiau patikimi didėjant kvantinės grandinės ilgiui ir operacijų skaičiui. Dėl šių priežasčių, 2019 metais Carlos Bravo-Prieto sukūrė variacinį kvantinį tiesinių lygčių sistemų sprendimo algoritmą (*angl. variational quantum linear solver (VQLS)*) [BLC+23]. Nors šis algoritmas reikalauja didesnio grandinių skaičiaus nei HHL, jos yra žymiai mažesnės ir gaunami ypač geri rezultatai naudojant triukšmui jautrius kvantinius kompiuterius.

2023 metais Jianming Yi parašytame straipsnyje „Variational Quantum Linear Solver enhanced Quantum Support Vector Machine“ pristato algoritmą, kuris sujungia mašininio mokymo modelį (*angl. machine learning*) ir variacinį kvantinį tiesinių lygčių sistemų sprendimo algoritmą [YSM+23]. Straipsnyje naudotas mašininio mokymo modelis - atraminių vektorių klasifikatorius (*angl. support vector machine (SVM)*) ir klasifikatoriaus matricinis pavidalas (*angl. least squares support vector machine (LS-SVM)*). Šis variacinio kvantinio tiesinių lygčių sistemų sprendimo pritaikymas atraminių vektorių klasifikatoriui (*angl. variational quantum linear solver for support vector machines (VQLS-SVM)*) algoritmas naudoja kvantinių kompiuterių skaičiavimų našumą, kad pagerintų mašininio mokymosi modelių kokybę ir greitį.

Darbo tikslas. Atlikti VQLS-SVM algoritmo tyrimą, siekiant išplėsti jo taikymą iki 10 kubitų.

Darbo uždaviniai.

- Išanalizuoti VQLS-SVM algoritmą [YSM+23] straipsnyje, pritaikyti kodo optimizavimus, skirtus sumažinti algoritmo skaičiavimo trukmę.
- Atlikti tyrimus su skirtingomis duomenų aibėmis, palyginti klasikinio ir kvantinio atraminių vektorių klasifikatorių tikslumus.
- Atlikti įvairių gradientinių ir negradientinių optimizatorių bandymus, rasti efektyviausią optimizatorių problemos sprendimui.
- Išplėsti [YSM+23] straipsnyje pristatytą algoritmą iki 10 kubitų, siekiant padidinti apmo-

kymo duomenų aibės dydį ir duomenų požymių skaičių. Toks išplėtimas leistų algoritmui efektyviai apdoroti sudėtingesnius duomenis ir padidinti klasifikavimo tikslumą.

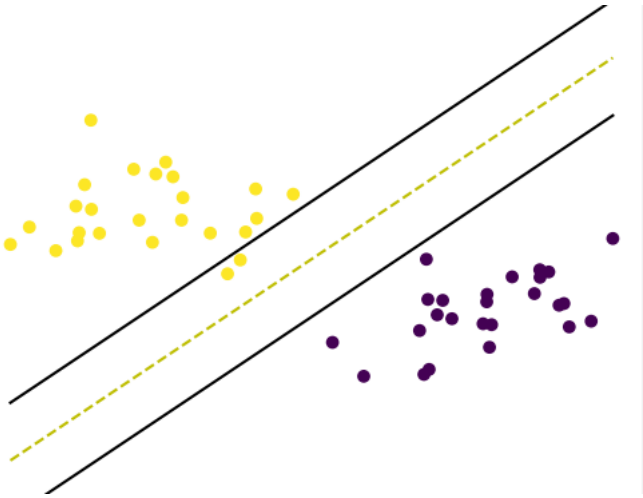
- Pritaikyti Luko Hantzko straipsnyje [HBG23] pristatytą efektyvų tenzorinės Paulio bazės dekompozicijos algoritmą, kai kubitų skaičius pasiekia 10.
- Ištirti ar Andrea Mari pristatytame straipsnyje nuoseklus variacinis kvantinis tiesinių lygčių sistemų algoritmas (*angl. coherent variational quantum linear solver*) turi tam tikrų privalumų sprendžiant atraminių vektorių klasifikatoriaus problemą lyginant su VQLS algoritmu.[BLC$^+$23; Mar19].

1. Literatūros analizė

1.1. Atraminių vektorių klasifikatorius

1.1.1. Klasikinis atraminių vektorių klasifikatorius

Atraminių vektorių klasifikatorius yra prižiūrimojo mašininio mokymo modelis, sukurtas 1995 metais Vladimir Vapnik ir Corinna Cortes, galintis klasifikuoti duomenis tarp dviejų klasių [CV95]. Šio algoritmo pagrindinė savybė yra gebėjimas rasti skiriamąją hiperplokštumą, kuri geriausiai atskiria duomenų aibėje esančių dviejų skirtingų klasių duomenis (1 paveikslėlis).



1 pav. Atraminių vektorių klasifikatoriaus pavyzdys

Atraminių vektorių klasifikatoriaus matematinę išraišką galime matyti žemiau pateiktoje formulėje (1), kur w yra atraminių vektorių klasifikatoriaus svoriai, (x_k, y_k) yra k-tieji duomenys ir atitinkamai priskirta duomenų klasė, b - laisvasis narys ir $\phi(...)$ yra netiesinė transformacijos funkcija, kuri atvaizduoja duomenis aukštesnėje dimensijoje.

$$\begin{cases} w^T\phi(x_k) + b \geq 1, & \text{jei } y_k = +1 \\ w^T\phi(x_k) + b \leq -1, & \text{jei } y_k = -1 \end{cases} \tag{1}$$

Atlikus (1) nelygybių sistemos pertvarkymus, hiperplokštuma, atskirianti duomenų aibės duomenis su skirtingomis klasėmis, įgauna formą:

$$\begin{cases} y_k\left[w^T\phi(x_k) + b\right] \geq 1 - \xi_k, & k = 1, \ldots, N \\ \xi_k \geq 0, & k = 1, \ldots, N \end{cases} \tag{2}$$

Naudojantis (2) nelygybių sistema, gaunama atraminių vektorių klasifikatoriaus minimizacijos funkcija:

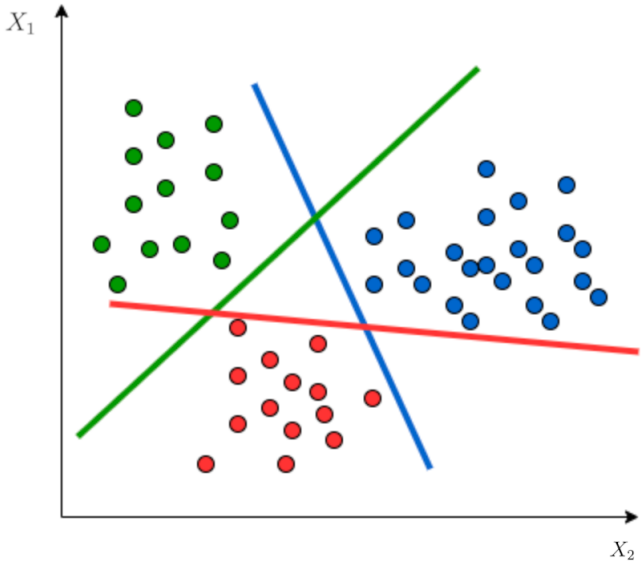
$$\min_{\vec{w},\xi_k} \mathcal{J}(\vec{w}, \xi_k) = \frac{1}{2}\vec{w}^T\vec{w} + c\sum_{k=1}^{N}\xi_k \tag{3}$$

Atraminių vektorių klasifikatorių galima naudoti ir n duomenų klasių klasifikavimui. Vienas iš būdų: vienas prieš visus (*angl. One-vs-All*). Šio metodo žingsniai:

1. norimos atskirti duomenų klasės duomenims priskiriame 1 reikšmę, visoms kitoms - -1;

2. randame hiperplokštumą, atskiriančią duomenų klasę nuo kitų klasių;
3. atliekame 1-2 žingsnius visoms n duomenų klasėms;

Šiuo būdu, reikia rasti tik n hiperplokštumų ir svorių. Paveikslėlyje 2 galime matyti sėkmingai atskirtus duomenis į 3 skirtingas duomenų klases, naudojantis trimis tiesėmis. Klasifikuojant duomenis, jie patikrinami su visais n svoriais ir priskiriama klasė, kurioje gaunama didžiausia tikimybė.



2 pav. Atraminių vektorių klasifikatoriaus su $n = 3$ klasėmis pavyzdys

1.1.2. Atraminių vektorių klasifikatorius pritaikius mažiausių kvadratų metodą

1999 metais Johan Suykens pristatė mažiausių kvadratų metodo pritaikymą atraminių vektorių klasifikatoriui (*angl. least squares support vector machine (LS-SVM)*) [SV99]. Atraminių vektorių klasifikatoriaus formulės buvo performuluotos į matricinį pavidalą:

$$\begin{bmatrix} 0 & \vec{1}_N^T \\ \vec{1}_N & \Omega + \gamma^{-1} I_N \end{bmatrix} \cdot \begin{bmatrix} d \\ \vec{\theta} \end{bmatrix} = \begin{bmatrix} 0 \\ \vec{y} \end{bmatrix} \tag{4}$$

Kur $\vec{1}_N = \underbrace{[1; \ldots; 1]}_{N}^T$, $\Omega = X^T X$ yra, šiuo atveju, tiesinė branduolio matrica, γ yra reguliavimo parametras, d - poslinkio reikšmė, $\vec{\theta}$ - Lagranžo daugikliai , $\vec{y}$ - duomenų aibės klasių reikšmės. Išsprendus šią lygčių sistemą ir gavus d ir $\vec{\theta}$ galima tiksliai klasifikuoti testinius duomenis [SV99]. Ši atraminių vektorių klasifikatoriaus forma, itin naudinga norint rasti atraminius vektorius tam tikrai duomenų aibei, naudojantis kvantiniais kompiuteriais.

1.2. Variacinis kvantinis tiesinių lygčių sistemų algoritmas

Variacinis kvantinis tiesinių lygčių sistemų sprendimo algoritmas yra platesnės variacinių kvantinių algoritmų (*angl. variational quantum algorithm*) klasės dalis. Šie hibridiniai algoritmai turi pagrindines dalis:

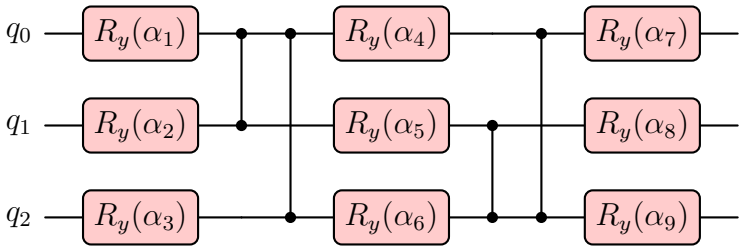
- vektoriaus $|x\rangle$ skaičiavimas pasitelkiant *ansatz* grandines;
- nuostolių funkcijos apibrėžimas specifinei problemai;
- paprasti ir specialieji Hadamardo testai;
- vektorių ir matricų užkodavimas į kvantines grandines;

1.2.1. Ansatz

Viena iš pagrindinių VQLS algoritmo dalių - galimo lygties sistemos sprendinio užkodavimas naudojantis kvantinėmis grandinėmis. Tam atlikti, naudojama *ansatz* grandinė. *Ansatz* - sprendinio prielaida, šiuo atveju, vektoriaus $|x\rangle$ spėjimas. *Ansatz* grandinė žymima $V(\alpha)$, kur $\alpha = \{\alpha_1,\alpha_2,\ldots,\alpha_n\}, n \in \mathbb{N}, \alpha_i \in \mathbb{R}$. *Ansatz* užkoduoja sprendinio prielaidą naudojantis formule (5).

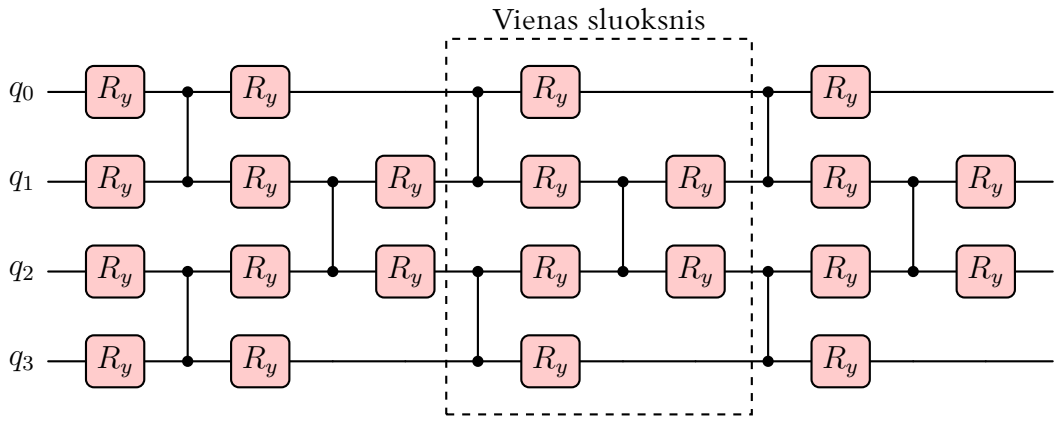
$$V(\alpha)|0\rangle = \frac{|x\rangle}{\|x\|}, \quad \text{jei } \|x\| \neq 1, \text{ kitu atveju - } V(\alpha)|0\rangle = |x\rangle \tag{5}$$

Kiekvienai specifinei užduočiai, *ansatz* grandinė turi būti atitinkamai parinkta, kad būtų pakankamai lengva realizuoti naudojantis tikru kvantiniu kompiuteriu. [BLC+23] straipsnyje pristatyta efektyvi *ansatz* grandinė, trijų kubitų sistemai, kuri lengvai sukonstruojama, naudojantis tikru kvantiniu kompiuteriu (*angl. hardware-efficient*). Grandinę galime matyti žemiau pateiktame paveikslėlyje 3 , kur naudojami 9 α parametrai.

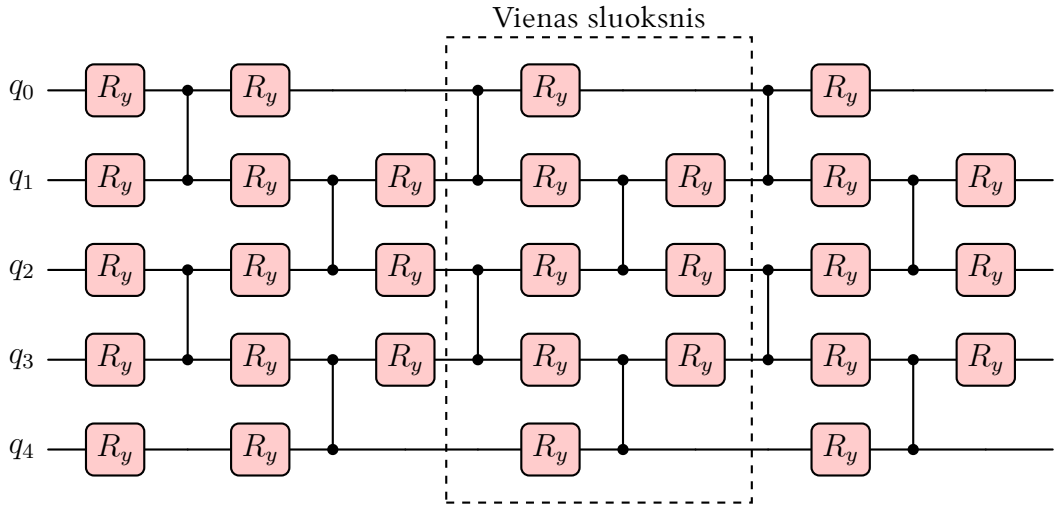


3 pav. 3 kubitų *ansatz* grandinė

Kadangi šiame darbe siekiama atlikti eksperimentus su didesniu kubitų skaičiumi nei 3, buvo pasitelkta nekintančios struktūros, sluoksniuota, lengvai sukonstruojama naudojantis tikru kvantiniu kompiuteriu *ansatz* grandinė *(angl. Fixed-structure layered Hardware-Eficient Ansatz)*. [BLC+23] straipsnyje pristatyta 10 kubitų atvejis, bet galima generalizuoti bet kokiam kubitų skaičiui. Žemiau pateiktuose paveikslėliuose 4 ir 5 galima matyti jog *ansatz* grandinės sudaryto iš visų kubitų pasukimo naudojantis R_y vartais ir vieno sluoksnio, sudaryto iš CZ ir R_y vartų, kartojimo n kartų. Lyginiu ir nelyginiu kubitų atvejais, *ansatz* grandinių realizacijos šiek tiek skiriasi, tačiau tai nesutrukdo rasti problemos sprendinį. Darbe naudojama 3 sluoksnių *ansatz* grandinės, siekiant sumažinti resursų naudojimą, bet išlaikant gerą tikslumą.



4 pav. 4 kubitų *ansatz* grandinė



5 pav. 5 kubitų *ansatz* grandinė

1.2.2. Nuostolių funkcija

Apibrėžkime tam tikrus, šioje užduotyje reikalingus ir svarbius žymėjimus. Klasikinė lygčių sistemos išraiška matoma žemiau pateiktoje lygtyje (6), kur A yra matrica, x ir b yra vektoriai.

$$A\vec{x} = b \tag{6}$$

Performulavę šią lygtį, naudojantis kvantinės mechanikos žymėjimais, gauname:

$$|\psi\rangle = A|x\rangle \approx |b\rangle \tag{7}$$

$$|b\rangle = U|0\rangle \tag{8}$$

$$A = \sum_{l=0}^{N} c_l A_l, \quad c \in \mathbb{C} \tag{9}$$

$$A_l \in \{I,X,Y,Z\}^{\otimes N} \tag{10}$$

$$AV(\alpha)|0\rangle \approx U|0\rangle \qquad \text{(įstatome (5) ir (8) į (7))} \tag{11}$$

Pagrindinė algoritmo užduotis: turime rasti tokius $\alpha = \{\alpha_1,\alpha_2,\ldots,\alpha_n\}$ kad formulės (11) kairės ir dešinės formulės pusių skirtumas būtų artimas 0. Apibrėžkime nuostolių funkciją VQLS problemai. Norime, kad nuostolių funkcijos reikšmė būtų kuo mažesnė, kai $A|x\rangle$ ir $|b\rangle$ yra artimi vienas kitam ir kuo didesni, kai vektoriai ortogonalūs. Tam pasitelkiame [BLC+23] straipsnyje pristatytą „projekcijos“ Hamiltonianą, matomą žemiau pateiktoje formulėje (12). Hamiltonianas - operatorius, kuris reprezentuoja visą sistemos energiją: kinetinę ir potencinę energijas. Šis operatorius itin svarbus kvantinėje mechanikoje, nes jo dėka galima išsiaiškinti, kaip tam tikra kvantinė sistema elgsis priklausomai nuo laiko.

$$H_P \;=\; \mathbb{I} \;-\; |b\rangle\langle b| \tag{12}$$

Galime apibrėžti nuostolių funkciją (*angl. cost function*) naudojantis hamiltonianu H_P ir $|\psi\rangle$:

$$C_P \;=\; \langle\psi|H_P|\psi\rangle \;=\; \langle\psi|(\mathbb{I} \;-\; |b\rangle\langle b|)|\psi\rangle \;=\; \langle\psi|\psi\rangle \;-\; \langle\psi|b\rangle\langle b|\psi\rangle \tag{13}$$

Norint padidinti nuostolių funkcijos tikslumą, reikia atlikti $|\psi\rangle$ normalizaciją:

$$|\psi_{norm}\rangle = \frac{|\psi\rangle}{\|\psi\|} = \frac{|\psi\rangle}{\sqrt{\langle\psi|\psi\rangle}} \quad \left(\langle\psi_{norm}| = \frac{\langle\psi|}{\sqrt{\langle\psi|\psi\rangle}}\right) \tag{14}$$

$$\hat{C}_P \;=\; \langle\psi_{norm}|\psi_{norm}\rangle \;-\; \langle\psi_{norm}|b\rangle\langle b|\psi_{norm}\rangle \;=$$

$$= \frac{\langle\psi|\psi\rangle}{\langle\psi|\psi\rangle} \;-\; \frac{\langle\psi|b\rangle\langle b|\psi\rangle}{\langle\psi|\psi\rangle} \;=\; 1 \;-\; \frac{\langle\psi|b\rangle\langle b|\psi\rangle}{\langle\psi|\psi\rangle} \;=\; 1 \;-\; \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle} \tag{15}$$

Formulėje (15) turime didesnio tikslumo nuostolių funkcijos išraišką $1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle}$. Pabandykime išreikšti $\langle\psi|\psi\rangle$ ir $|\langle b|\psi\rangle|^2$ pasinaudoję tiesine unitarinių matricų kombinacijos (*angl. linear*

combination of unitaries (LCU)) formule (9):

$$\langle\psi|\psi\rangle = \langle 0|V(\alpha)^\dagger A^\dagger A V(\alpha)|0\rangle = \langle 0|V(\alpha)^\dagger \Big(\sum_m c_m\, A_m\Big)^\dagger \Big(\sum_n c_n\, A_n\Big) V(\alpha)|0\rangle =$$
$$= \sum_m \sum_n \overline{c_m} c_n \langle 0|V(\alpha)^\dagger A_m^\dagger A_n V(\alpha)|0\rangle \tag{16}$$

$$|\langle b|\psi\rangle|^2 = |\langle b|AV(\alpha)|0\rangle|^2 = |\langle 0|U^\dagger AV(\alpha)|0\rangle|^2 = \langle 0|U^\dagger AV(\alpha)|0\rangle\langle 0|V(\alpha)^\dagger A^\dagger U|0\rangle =$$
$$= \sum_m \sum_n \overline{c_m} c_n \langle 0|U^\dagger A_n V(\alpha)|0\rangle\langle 0|V(\alpha)^\dagger A_m^\dagger U|0\rangle \tag{17}$$

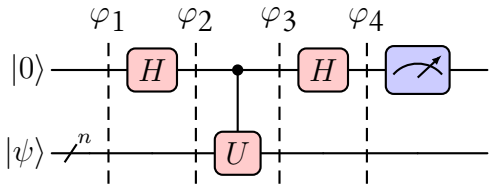
Atlikus perstatymus gauname, kad norint apskaičiuoti $\langle\psi|\psi\rangle$, reikia gauti $\langle 0|V(\alpha)^\dagger A_m^\dagger A_n V(\alpha)|0\rangle$ visų kombinacijų reikšmes, o $|\langle b|\psi\rangle|^2$ reikšmė gaunama apskaičiavus $\langle 0|V(\alpha)^\dagger A_m^\dagger U|0\rangle$ visiems m.

1.2.3. Hadamardo testas

Variaciniuose kvantiniuose algoritmuose, skaičiuojant nuostolių funkcijos reikšmę, itin dažnai reikia gauti tikėtiną reikšmę (*angl. expectation value*) pritaikius tam tikrą operatorių U. Kvantinėje mechanikoje tikėtinos reikšmės gavimas apibrėžiamas formule $\langle U\rangle_\psi = \langle\psi|U|\psi\rangle$, kur $|\psi\rangle$ yra normalizuotas būsenos vektorius (*angl. state vector*). Norint apskaičiuoti tikėtiną reikšmę kvantinio kompiuterio pagalba, pasinaudosime Hadamardo testu. Hadamardo testas - tai vienas iš paprasčiausių kvantinių algoritmų [DL23], kuris susideda iš 3 dalių:

1. viršutinio kubito superpozicijos paruošimas, naudojantis Hadamardo operatoriumi;
2. kontroliuojamo U operatoriaus pritaikymas;
3. Hadamardo operatoriaus pritaikymas viršutiniam kubitui ir jo matavimas.

Hadamardo testo grandinę galima matyti 6 paveikslėlyje.



6 pav. Hadamardo testo grandinė

Suskaičiuokime kubitų būsenas, po kiekvieno operatoriaus pritaikymo:

$$\varphi_1 = |0\rangle \otimes |\psi\rangle \tag{18}$$

$$\varphi_2 = \frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\psi\rangle}{\sqrt{2}} \tag{19}$$

$$\varphi_3 = \frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes U|\psi\rangle}{\sqrt{2}} \tag{20}$$

$$\varphi_4 = \frac{\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes |\psi\rangle + \frac{|0\rangle-|1\rangle}{\sqrt{2}} \otimes U|\psi\rangle}{\sqrt{2}} = \frac{|0\rangle \otimes |\psi\rangle + |1\rangle \otimes |\psi\rangle + |0\rangle \otimes U|\psi\rangle - |1\rangle \otimes U|\psi\rangle}{2} =$$
$$= \frac{|0\rangle \otimes (|\psi\rangle + U|\psi\rangle) + |1\rangle \otimes (|\psi\rangle - U|\psi\rangle)}{2} = \frac{1}{2}(|0\rangle \otimes (I+U)|\psi\rangle + |1\rangle \otimes (I-U)|\psi\rangle) \tag{21}$$

Norėdami suskaičiuoti kokia yra tikimybė gauti 0 ir 1 reikšmes, pamatavus viršutinį kubitą, pasinaudosime projekcijos operatoriumi, matomą žemiau pateiktoje formulėje (22), įstatydami į turimą formulę tikimybei apskaičiuoti, žemiau pateiktoje formulėje (23).

$$P_i = |i\rangle\langle i| \tag{22}$$

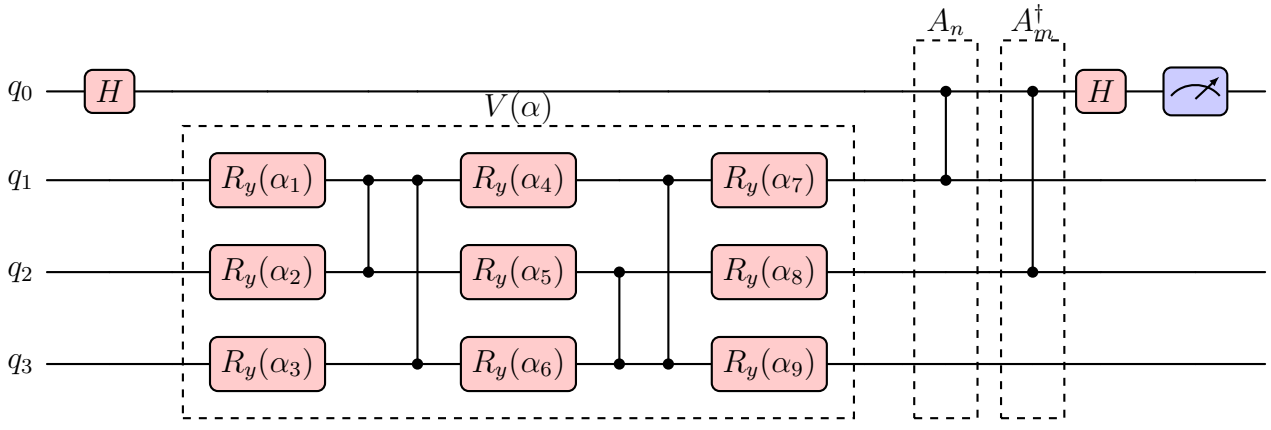
$$P(i) = \langle\varphi|P_i|\varphi\rangle = \langle\varphi|i\rangle\langle i|\varphi\rangle \tag{23}$$

$$P(0) = \frac{1}{4}\langle\psi|(I+U^\dagger)(I+U)|\psi\rangle = \frac{1}{4}\langle\psi|(2I+U+U^\dagger)|\psi\rangle =$$
$$= \frac{1}{4}(2 + \langle\psi|U|\psi\rangle + \langle\psi|U^\dagger|\psi\rangle) = \frac{1}{2}(1 + \mathrm{Re}\langle\psi|U|\psi\rangle) \tag{24}$$

$$P(1) = \frac{1}{4}(2 - \langle\psi|U|\psi\rangle - \langle\psi|U^\dagger|\psi\rangle) = \frac{1}{2}(1 - \mathrm{Re}\langle\psi|U|\psi\rangle) \tag{25}$$

$$\mathrm{Re}\langle\psi|U|\psi\rangle = P(0) - P(1) \tag{26}$$

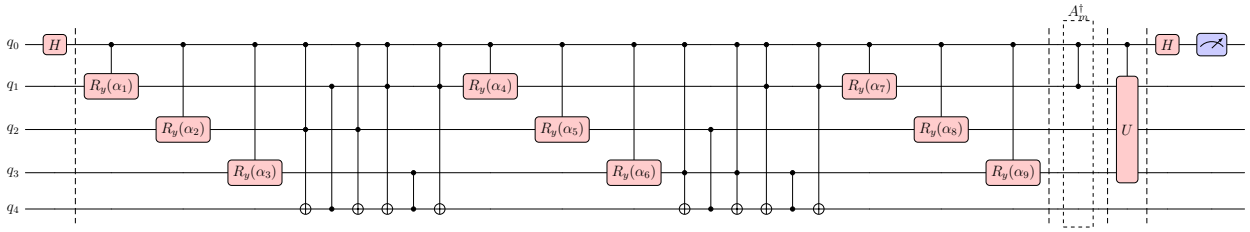
Formulėje (26) gauta lygybė, skirta apskaičiuoti realiai $\langle\psi|U|\psi\rangle$ reikšmei. Jei $|\psi\rangle = V(\alpha)|0\rangle$ ir $U = A_m^\dagger A_n$, pradinę Hadamardo testo grandinę galima pertvarkyti į $\langle\psi|U|\psi\rangle = \langle 0|V(\alpha)A_m^\dagger A_n V(\alpha)|0\rangle$ kvantinę grandinę, matomą žemiau pateiktame paveikslėlyje 7, kur $A_n = Z \otimes I \otimes I$, $A_m = I \otimes Z \otimes I$.



7 pav. $\langle 0|V(\alpha)A_m^\dagger A_n V(\alpha)|0\rangle$ grandinė

1.2.4. Specialus Hadamardo testas

Turint formulę (17) reikia rasti $\langle 0|V(\alpha)^\dagger A_m^\dagger U|0\rangle$. Šiai reikšmei rasti pasinaudosime Jianming Yi „Variational Quantum Linear Solver enhanced Quantum Support Vector Machine“ (VQLS-QSVM) straipsnyje pristatytu specialiu Hadamardo testu [YSM+23]. Pirmoje žemiau matomo paveikslėlio 8 dalyje vaizduojama kontroliuojama *ansatz* grandinė, antroje - matrica $A_m^\dagger = Z \otimes I \otimes I$, trečioje - kontroliuojama matrica U skirta užkoduoti vektorių $|b\rangle$ į kvantinę grandinę.

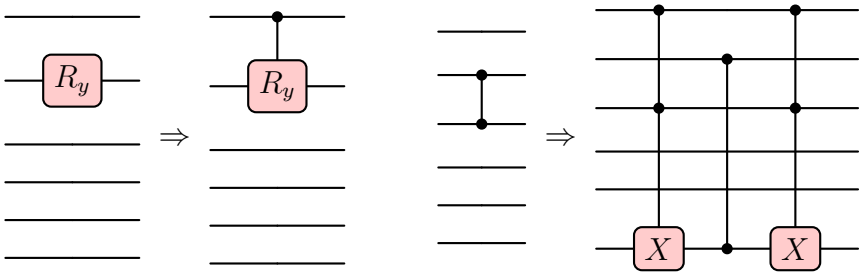


8 pav. $\langle 0|V(\alpha)^\dagger A_m^\dagger U|0\rangle$ grandinė

Galima lengvai pastebėti, kad paveikslėlyje 8 naudojama kontroliuojama *ansatz* grandinė yra sukonstruota naudojantis *ansatz* grandine matoma paveikslėlyje 3 ir keleta elementarių taisyklių.

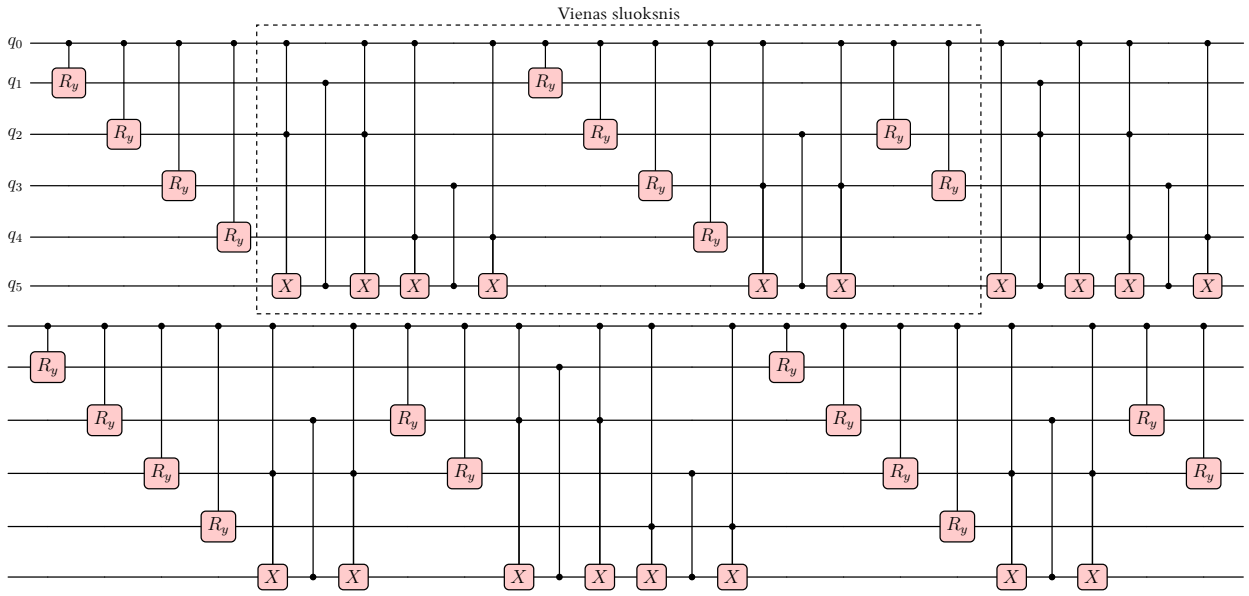
Pirma, visi posūkių vartai pakeičiami į kontroliuojamus posūkių vartus, kur kontroliuojamas kubitas yra 1 ancilla (pirma eilė paveikslėlyje 8). Toliau visi kontroliuojami Z vartai yra pakeičiami į simetrišką dvigubai kontroliuojamų X ir kontroliuojamų Z vartų kombinacijas, naudojant 2 ancilla (pirma ir penkta eilė paveikslėlyje 8).

Taisykles galima matyti ir žemiau pateiktame paveikslėlyje 9 .



9 pav. Taisyklės, siekiant paversti paprastą *ansatz* grandinę į kontroliuojamą *ansatz* grandinę

Pritaikius paveikslėlyje 9 esančias taisykles *ansatz* grandinei, kai kubitų skaičius didesnis už 3, gauname žemiau esančiame paveikslėlyje 10 grandinę. Šios taisyklės sėkmingai pritaikomos ir didesnėms *ansatz* grandinėms.



10 pav. 4 kubitų specialaus Hadamardo testo kontroliuojama ansatz grandinė

1.2.5. Vektoriaus užkodavimas į kvantinę grandinę

Amplitudės užkodavimas (*angl. amplitude encoding*) yra vienas iš dažniausiai naudojamų būdų, skirtų užkoduoti vektoriaus informaciją į kvantines būsenas (*angl. quantum states*) [KA23]. Šiuo informacijos užkodavimo būdu, klasikinis vektorius normalizuojamas, kad visų vektoriaus narių kvadratų suma būtų lygi 1, matematinė išraiška matoma žemiau pateiktoje formulėje (28).

$$|\psi\rangle = c_0|x_0\rangle + c_1|x_1\rangle + \ldots + c_{n-1}|x_{n-1}\rangle, \qquad c_i \in \mathbb{C} \tag{27}$$

$$\begin{gathered} |x_0\rangle = [1;0;\ldots;0]^T \\ |x_1\rangle = [0;1;\ldots;0]^T \\ \vdots \\ |x_{n-1}\rangle = [0;0;\ldots;1]^T \end{gathered}$$

$$\sum_{i=0}^{n-1} |c_i|^2 = 1 \tag{28}$$

Atlikus vektoriaus normalizaciją, kiekvienas vektoriaus elementas yra priskiriamas atitinkamai kubitų būsenai. Pertvarkius klasikinę *ket* vektoriaus formulę (27) gauname žemiau matomą lygtį (29), kur x_i yra vektoriaus i-as elementas, i yra kubito būsena, n - kubitų skaičius.

$$|\psi_x\rangle = \sum_{i=1}^{2^n} x_i|i\rangle \tag{29}$$

Pavyzdys 1. Turime vektorių $|x\rangle = [1; 0; -5{,}5; 0]^T$. Normalizuotas vektorius matomas (30) formulėje. Vėliau jis užkoduojamas naudojant visas 4 dviejų kubitų būsenas (31) formulėje.

$$|x_{norm}\rangle = \frac{|x\rangle}{\|x\|} = \frac{1}{\sqrt{31.25}}[1; 0; -5{,}5; 0]^T \tag{30}$$

$$|\psi_{x_{norm}}\rangle = \frac{1}{\sqrt{31{,}25}}(1|00\rangle + 0|01\rangle - 5{,}5|10\rangle + 0|11\rangle) = \frac{1}{\sqrt{31{,}25}}(|00\rangle - 5{,}5|10\rangle) \tag{31}$$

Jei norimas užkoduoti vektorius $x \in \mathbb{R}^k$ ir $\forall m \in \mathbb{N} : 2^m \neq k$, prie vektoriaus x pridedamos papildomos, nereikšmingos konstantos (dažniausiai naudojama 0), kad vektoriaus dimensija būtų dvejeto laipsnis.

Pavyzdys 2. Turime vektorių $|x\rangle = [1; 1; 1; 1; 1]^T$, jo dimensija - 5, reikalingų kubitų skaičius, norint užkoduoti vektorių x: $\lceil \log_2(5) \rceil = 3$. Reikalingų nereikšmingų konstantų skaičius, kuriuo turime papildyti vektorių x: $2^3 - 5 = 3$. Galutinis vektorius, su papildytomis konstantomis: $|x\rangle = [1; 1; 1; 1; 1; 0; 0; 0]^T$.

IBM Qiskit turi *prepare_state(x)* funkciją, kuri užkoduoja normalizuotą vektorių į kvantinę grandinę. Vektoriaus dimensija turi būti dvejeto laipsnis, priešingu atveju grąžinama klaida. Sukurta funkcija, skirta išvengti šios problemos. Jos pseudokodą galime matyti apačioje esančiame algoritme 1:

1 algoritmas. Vektoriaus užkodavimo į kvantines būsenas pseudokodas

Input: Norimas užkoduoti vektorius x.
Output: Kvantinė grandinė, kurioje užkoduotas vektorius x amplitudės užkodavimo būdu.
function EncodeVector(x)
    qubits = $\lceil \log_2(\text{length}(x)) \rceil$
    if length(x) $\neq 2^{qubits}$ **then**
        $x = x \oplus \bigoplus_{i=0}^{qubits - \text{length}(x)} \langle 0 \rangle$
    end if
    $x = \frac{x}{\|x\|}$
    circuit = QuantumCircuit(*qubits*)
    circuit.prepare_state(x)
    return *circuit*
end function

1.2.6. Matricos užkodavimas į kvantinę grandinę

Matricų užkodavimas į kvantines grandines yra vienas iš svarbiausių žingsnių kuriant kvantinius algoritmus [CV22]. Kiekvienas kvantinis operatorius (*angl. quantum gate*) yra unitarinė matrica, kuri atitinka formulėje (32) matomą savybę.

$$UU^\dagger = U^\dagger U = I \tag{32}$$

Tačiau, jei matrica yra ne unitarinė, ji negali būti įprastai užkoduojama. Šiai problemai išspręsti, pasinaudosime tiesine unitarinių matricų kombinacijos (LCU) formule (33). Šioje lygtyje matome, kad turima neunitarinė matrica išskaidoma į unitarines, Paulio bazės matricas, su koeficientais c. Matematinę išraišką galime matyti apačioje esančioje formulėje (34).

$$A = \sum_{l=0}^{N} c_l A_l, \quad c \in \mathbb{C} \tag{33}$$

$$A_l \in \{I,X,Y,Z\}^{\otimes N} \tag{34}$$

Formulėje (34) pavaizduotą Paulio bazę galima sugeneruoti su algoritmu 2. Jame tenzorinės sandaugos dėka gaunamos visos, N kubitų dydžio Paulio bazės. Pavyzdžiui, *pauliBasis(numQubits=2)* grąžintų:

$$\{I,X,Y,Z\}^{\otimes 2} = \{I,X,Y,Z\} \otimes \{I,X,Y,Z\} =$$
$$= \{II,IX,IY,IZ,XI,XX,XY,XZ,YI,YX,YY,YZ,ZI,ZX,ZY,ZZ\} \tag{35}$$

2 algoritmas. *PauliBasis* funkcijos pseudokodas

```
1:  Input: Kubitų skaičius numQubits.
2:  Output: Paulio bazės dydžio numQubits visi elementai.
3:  function PauliBasis(numQubits)
4:      pauliOneQubit = {I, X, Y, Z}
5:      if numQubits = 1 then
6:          return pauliOneQubit
7:      end if
8:      pauli = pauliOneQubit
9:      for i = 0 to numQubits − 1 do
10:         pauli = pauliOneQubit ⊗ pauli
11:     end for
12:     return pauli
13: end function
```

Formulėje (33) esančius koeficientus ir Paulio bazės matricas galime gauti naudojantis žemiau pateiktu išretintos Paulio bazės (*eng. Sparse Pauli*) funkcijos algoritmu 3. Sugeneravus N kubitų Paulio bazę, skaičiuojama kiekvieno Paulio bazės elemento koeficiento reikšmė naudojant (36) funkciją, kur N yra kubitų skaičius, A_i yra Paulio bazės matrica, A - norima dekomponuoti matrica, tr(...) - matricos pėdsako funkcija (formulė (37)). Jei c_i nėra artimas 0, koeficientas ir

Paulio bazės elementas išsaugojamas į masyvą.

$$c_i = \frac{\mathrm{tr}(A_i \cdot A)}{2^N} \tag{36}$$

$$\mathrm{tr}(A) = \sum_{i=1}^{2^n} a_{ii} \tag{37}$$

3 algoritmas. *SparsePauliOP* funkcijos pseudokodas

```
Input: Matrica inputMatrix, kurią norima išskaidyti į unitarines su koeficientais;
       reikalingų matricai užkoduoti kubitų skaičius numQubits.
Output: Paulio bazės elementai su atitinkamais koeficientais.
function SparsePauliOP(inputMatrix, numQubits)
    paulis = ∅
    coefficients = ∅
    basis = PauliBasis(numQubits)
    for i = 0 to length(basis) - 1 do
        pauliBasisMatrix = [ ]
        for j = 0 to length(basis[i]) do
            if pauliBasisMatrix = [ ] then
                pauliBasisMatrix = basis[i][j]
            else
                pauliBasisMatrix = pauliBasisMatrix ⊗ basis [i][j]
            end if
        end for
        coefficient = tr(pauliBasisMatrix · inputMatrix) / 2^numQubits
        if coefficient ≉ 0 then
            paulis = paulis ∪ basis[i]
            coefficients = coefficients ∪ coefficient
        end if
    end for
    return (paulis, coefficients)
end function
```

Tiesinių unitarinių matricų gavimui yra sukurti ir kiti algoritmai (H2ZIXY, Pauli-Composer, tensorized pauli decomposition), tačiau IBM Qiskit algoritmas yra lengviausiai suprantamas ir realizuojamas. Funkcijų 2 ir 3 pseudokodo realizacijas galima rasti IBM Qiskit atvirajame kode [Qis23].

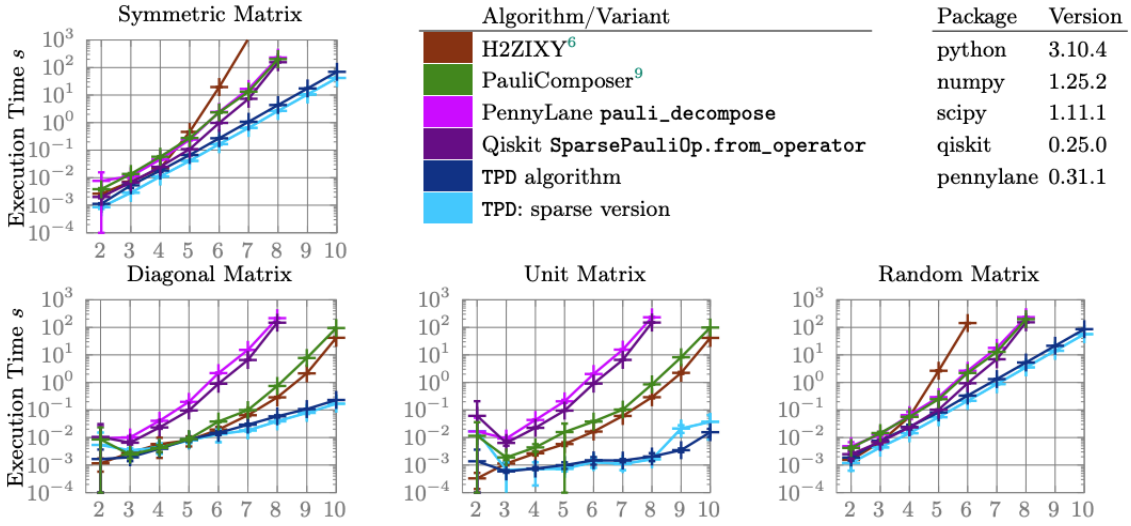
1.2.7. Tenzorinė Paulio dekompozicija (*angl. Tensorized pauli decomposition (TPD)*)

Ankščiau minėti algoritmai unitarinių, Paulio bazės matricų radimui naudoja matricų daugybą, kas yra gana daug skaičiavimo resursų reikalaujantis metodas. Norint gauti Paulio bazės matricas 8 kubitų atveju, IBM Qiskit pristatytas algoritmas užtrunka 100 sekundžių [HBG23].

Tenzorinė Paulio dekompozicija tai Lukas Hantzko sukurtas algoritmas, kuris vengia brangaus matricų daugybos metodo ir naudoja matricos skaidymą į mažesnes, rekursyviai pritaikant algoritmą ir taip pagreitinant Paulio matricų, sudarančių duotą matricą, radimą. Žemiau pateiktoje formulėje 38 galima matyti, kad matrica A išskaidoma į keturias $2^{n-1} \times 2^{n-1}$ matricas ir tada atitinkamai randamas sprendinys kiekvienai. Jei $n \geq 2$, kiekvieną $2^{n-1} \times 2^{n-1}$ matricą galima toliau skaidyti, kol pasiekiama 2×2 dydžio matrica.

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \sum_{t \in Q} \Omega_{*t} \otimes \sigma^t \tag{38}$$

Rezultatus, kurie buvo gauti [HBG23] straipsnyje tiriant tiesinės unitarinės matricos dekompozicijos algoritmus su skirtingomis matricomis, galime matyti paveikslėlyje 11. Tenzorinės Paulio matricų dekompozicijos algoritmas sėkmingai susidoroja su 8×8 matricų skaičiavimu 10 kartų greičiau nei kiti Paulio matricų dekompozicjos algoritmai.



11 pav. Paulio matricų radimo algoritmų efektyvumo priklausomybė nuo kubitų skaičiaus [HBG23]

Šiame darbe, eksperimentinės dalies metu, bus naudojamas tenzorinės Paulio matricų dekompozicijos algoritmas.

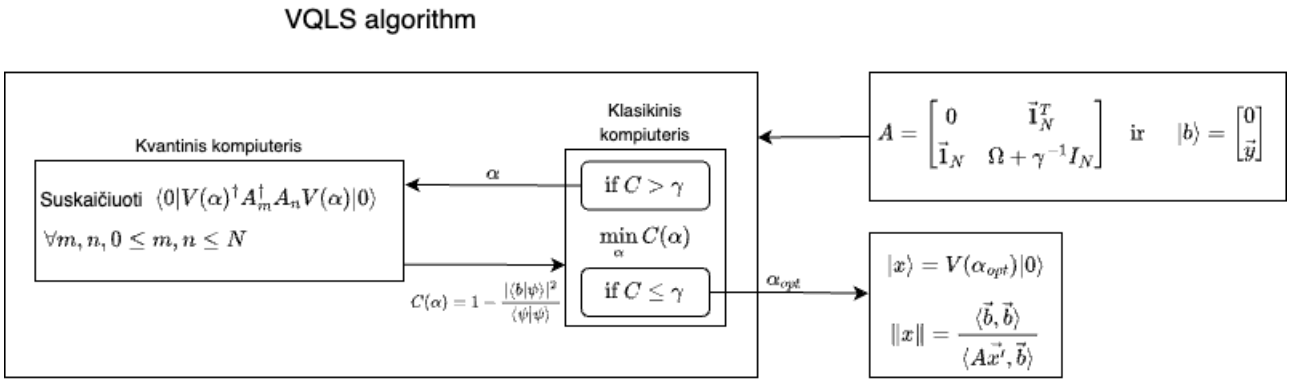
1.2.8. VQLS-SVM konstravimas

Jian-ming Yi straipsnyje „Variational Quantum Linear Solver enhanced Quantum Support Vector Machine“ pristatytas LS-SVM ir VQLS sujungiantis algoritmas, kurio dėka atraminių vektorių klasifikatoriaus mokymas atliekamas naudojant kvantinį kompiuterį [YSM+23]. Žemiau pateiktame algoritmo pseudokode 4 galime matyti šiuos žingsnius.

4 algoritmas. VQLS-SVM algoritmo pseudokodas

Input: Duomenų aibė X;
    kiekvienam rinkinio įrašui priskiriamos klasių reikšmės y.
Output: LS-SVM tiesinės lygčių sistemos sprendinys $\vec{x}$.
function VQLS-SVM(X, y)
    y_{norm} = Normalizuojamas duomenų aibės klasių vektorius y.
    // Parenkami 7 atsitiktiniai duomenų rinkinio įrašai, jų duomenys normalizuojami, duomenys išsaugomi į *trainData* kintamąjį.
    A = Sukonstruojama LS-SVM matrica, naudojantis (4) formule ir *trainData* reikšmėmis.
    K,c = Matricos A išskaidymas į unitarinių, su koeficientais, sumą (33 formulė).
    // Parinkę parametrus α, minimizuojame $1 - \frac{|\langle b|\psi\rangle|^2}{\langle\psi|\psi\rangle}$ (15 formulė).
    α_{opt} = *minimize(K,c,* y_{norm} *)* (6 funkcija)
    // Gavus optimalius parametrus α_{opt}, suskaičiuojame $\vec{x'} = \frac{\vec{x}}{\|x\|}$ (57 formulė).
    $\vec{x'} = V(\alpha_{opt})|0\rangle$
    // Apskaičiuojame vektoriaus normą, naudojantis A, $\vec{x'}$ ir y.
    $\|x\| = \frac{\langle\vec{y},\vec{y}\rangle}{\langle A\vec{x'},\vec{y}\rangle}$
    // Turėdami vektoriaus normą $\|x\|$ gauname vektorių $\vec{x}$ (61 formulė).
    $\vec{x} = \|x\| \cdot \vec{x'}$
    return $\vec{x}$
end function

Šis algoritmas paruošia duomenis naudojantis klasikiniu kompiuteriu, gauta LS-SVM matrica A ir vektorius b užkoduojami į kvantines grandines. Kvantinis kompiuteris paleidžia visas sukonstruotas grandines ir gauna nuostolių funkcijos reikšmę su parametrais α. Klasikinis kompiuteris, naudodamasis *COBYLA* optimizatoriumi, pakoreguoja α reikšmes, grandinės konstruojamos iš naujo ir skaičiuojama nuostolių funkcija, kol iteracijų skaičius pasiekia nustatytą ribą arba nuostolių funkcijos reikšmė yra nedidesnė už tam tikrą parametrą γ. Gavus α_{opt}, tikroji vektoriaus reikšmė $\vec{x}$ randama naudojantis klasikiniu kompiuteriu apdorojant kintamuosius. Visus VQLS-SVM algoritmo atliekamus žingsnius galime matyti apačioje pateiktame paveikslėlyje 12.



12 pav. VQLS-SVM algoritmo diagrama

[YSM+23] straipsnyje pristatytas VQLS algoritmo nuostolių funkcijos pseudokodas pasižymi vienu neigiamu bruožu: grandinių konstravimo skaičiumi. Naudojant testinius duomenis, kurių Paulio bazės matricų skaičius siekė 46, reikalingų vienai epochai sukonstruoti grandinių skaičius: $3 \cdot 46 \cdot 46 = 6348$. Kai epochų skaičius - 200, iš viso sukonstruotų grandinių skaičius: $200 \cdot 6348 = 1269600$. Buvo atliktos pseudokodo modifikacijos, skirtos sumažinti konstruojamų grandinių skaičių, iškeliant konstravimą į atskirą funkciją *prepareCircuits(...)*. Žemiau pateiktame algoritme 5 galime matyti šios funkcijos pseudokodą. Sukonstruojamos visos A_n, A_m paprasto ir specialaus Hadamardo testų grandinių kombinacijos. Grandinės konstruojamos naudojantis IBM Qiskit parametrizuotų grandinių funkcionalumu ir yra transpiliuojamos. Kiekvienos iteracijos metu parametrizuojamas parametras α yra panaudojamas grandinėse ir apskaičiuojama iteracijos nuostolių funkcijos reikšmė.

5 algoritmas. Visų paprastųjų ir specialiųjų Hadamardo grandinių sukonstravimas

Input: Paulio bazės elementai A, gauti naudojantis funkcijos (3) pseudokodu; vektorius b.
Output: Visos paprasto ir specialaus Hadamardo testo kvantinės grandinės, reikalingos VQLS algoritmui.
function PrepareCircuits(A, b)
    hadamardTestCircuits = []
    specialHadamardTestCircuits = []
    for A_m in $\{A_1, A_2, \ldots, A_N\}$ **do**
        for A_n in $\{A_1, A_2, \ldots, A_N\}$ **do**
            qc = Sukonstruojame Hadamardo testo grandinę
            naudojantis A_n ir A_m (7 grandinė).
            hadamardTestCircuits = *hadamardTestCircuits* $\cup$ *qc*
            qc = Sukonstruojame specialaus Hadamardo testo grandinę
            naudojantis A_m, A_n ir b (8 grandinė).
            specialHadamardTestCircuits = *specialHadamardTestCircuits* $\cup$ qc
        end for
    end for
    return (*hadamardTestCircuits*, *specialHadamardTestCircuits*)
end function

Kitas algoritmo žingsnis - VQLS nuostolių funkcija. Žemiau pateiktame algoritme 6 galime matyti pertvarkytą *minimize(...)* funkcijos pseudokodą. Pirmiausia, paruošiamos visos paprasto ir specialaus Hadamardo testo grandinės. Vėliau, kiekvienos epochos metu, α reikšmės pritaikomos visoms sukonstruotoms grandinėms. Šis nedidelis pertvarkymas padeda sumažinti grandinių konstravimo skaičių nuo 1269600 iki $2 \cdot 46 \cdot 46 = 4232$ grandinių.

Taip pat, iš (16) formulės, galime pastebėti, kad:

$$\overline{c_m} c_n \langle 0|V(\alpha)^\dagger A_m^\dagger A_n V(\alpha)|0\rangle = \overline{\overline{c_n} c_m \langle 0|V(\alpha)^\dagger A_n^\dagger A_m V(\alpha)|0\rangle} \tag{39}$$

Šis pastebėjimas leidžia dar labiau sumažinti grandinių konstravimo skaičių. Paulio bazės matricų skaičiui esant 46, Hadamardo testui sukonstruotas grandinių skaičius bus lygus sumai nuo 1 iki 46 arba $\frac{46 \cdot 47}{2} = 1081$. Specialiajam Hadamardo testui reikia sukonstruoti tik 46 grandines, vėliau $|\langle b|\phi\rangle|^2$ reikšmė suskaičiuojama pritaikant 46 grandinių kombinacijas.

Galutinis grandinių skaičius, reikalingas sukonstruoti 46 Paulio matricų atveju - $46 \cdot 47 \cdot \frac{1}{2} + 46 = 1127$, nedidelis patobulinimas lyginant su [YSM+23] straipsnyje pristatytu algoritmu.

6 algoritmas. VQLS minimizacijos funkcijos pseudokodas

Input: Paulio bazės elementai A;
  kiekvieno elemento A koeficientai c;
  vektorius b.
Output: Optimalūs kintamiesiems A ir c α parametrai α_{opt}.
function Minimize(A, c, b)
  $\epsilon = 0.01$, $C = 1$, *maxIteration* $= 200$, α, *numIteration* $= 0$, shots $= 10000$
  hadamardTestCircuits, *specialHadamardTestCircuits* $=$ PrepareCircuits(*A, b*)
  while $C > \epsilon$ **or** *numIteration* $<$ *maxIteration* **do**
    sum1$= 0$, *sum2* $= 0$
    // Pritaikome α hadamardTestCircuits ir specialHadamardTestCircuit grandinėms.
    for A_m in $\{A_1, A_2, \ldots, A_N\}$ **do**
      for A_n in $\{A_1, A_2, \ldots, A_N\}$ **do**
        hadamardTestCircuits$[m \cdot N + n]$.execute(*shots*)
        // Pamatuojame pagalbinį kubitą q_a.
        // Suskaičiuojame $p_{q_a}(|0\rangle) - p_{q_a}(|1\rangle)$ norint gauti $\langle 0|V(\alpha)^\dagger A_m^\dagger A_n V(\alpha)|0\rangle$.
        sum1$+= \overline{c_m} \cdot c_n(p_{q_a}(|0\rangle) - p_{q_a}(|1\rangle))$
        specialHadamardTestCircuits$[m]$.execute(*shots*)
        // Pamatuojame pagalbinį kubitą q_a.
        // Suskaičiuojame $p_{q_a}(|0\rangle) - p_{q_a}(|1\rangle)$ norint gauti $\langle 0|U^\dagger A_n V(\alpha)|0\rangle$.
        specialHadamardTestCircuits$[n]$.execute(*shots*)
        // Pamatuojame pagalbinį kubitą q_a.
        // Suskaičiuojame $p_{q_a}(|0\rangle) - p_{q_a}(|1\rangle)$ norint gauti $\langle 0|V(\alpha)^\dagger A_m U|0\rangle$.
        sum2$+= \overline{c_m} \cdot c_n \langle 0|U^\dagger A_n V(\alpha)|0\rangle\langle 0|V(\alpha)^\dagger A_m U|0\rangle$
      end for
    end for
    // *sum1* reikšmė yra lygi $|\langle b|\psi\rangle|^2$.
    // *sum2* reikšmė yra lygi $\langle\psi|\psi\rangle$.
    $C = 1 - \frac{sum1}{sum2}$
    numIteration++
    $\alpha =$ Atnaujiname parametrus naudojant **COBYLA** optimizatorių.
  end while
  return α_{opt}
end function

1.2.9. Koherentinis variacinis kvantinis tiesinių lygčių sistemos algoritmas

Koherentinis variacinis kvantinis tiesinių lygčių sistemos algoritmas *(angl. Coherent variational quantum linear solver algorithm)*, pristatytas Pennylane puslapyje [Mar19], pateikia alternatyvų būdą užkoduoti matricą A, fiziškai pritaikant kaip koherentinę operaciją.

Pagrindinis CVQLS algoritmo skirtumas lyginant su VQLS algoritmu - matricos A kompo-

nentų A_i koeficientai turi būti neneigiami ir jų suma turi būti lygi vienetui. Matematinę išraišką galima matyti žemiau pateiktoje formulėje (40).

$$c_i \geq 0 \; \forall i, \quad \sum_{i=0}^{L-1} c_i = 1 \tag{40}$$

Šis VQLS algoritmo patobulinimas yra netinkamas VQLS-SVM problemai, nes formulėje (4), matrica gali turėti Paulio bazės komponentus su bet kokiais koeficientais c_i. CVQLS algoritmas sprendžia tik tam tikrą poaibį matricų, kurioms formulėje (40) esančios sąlygos galioja.

1.2.10. Tiesinės lygčių sistemos sprendinio apdorojimas

Suradus tinkamus svorius α_{opt} kvantinei grandinei, pritaikome $V(\alpha_{opt})$ matricą pasinaudoję paveikslėlyje 3 esančia grandine, norint gauti normalizuotą svorių vektorių $\vec{x'}$. Matematinė išraišką galime matyti formulėje (5). Vektoriaus $\vec{x'}$ svorių ženklai pritaikius $V(\alpha_{opt})$ nėra tinkami. Norint pagerinti sprendinį, reikia rasti tokią ženklų kombinaciją, kad skirtumas tarp $A \cdot x''$ ir b būtų kuo mažesnis, kur x'' - vektorius x' pritaikius geriausią ženklų kombinaciją:

$$C = \{(y_1, y_2, \ldots, y_n)^T \mid y_i \in \{-1,1\}, \forall i \in 1,2,\ldots,n\}, \quad n = \dim(\vec{x'}) \tag{41}$$

$$\vec{x'}_{sign} = \min_{\forall y \in C} \|A \cdot (y \odot x') - b\| \tag{42}$$

kur $|\vec{x'}|$ - vektoriaus dimensija, $y \odot x'$ Hadamardo daugyba.

Tačiau, kai kubitų skaičius lygus 10, vektoriaus $\vec{x}$ dimensija būtų $2^{10} = 1024$. Bandant visas ženklų kombinacijas, reiktų patikrinti $2^{2^{10}} = 2^{1024}$, kas yra daugiau kombinacijų negu protonų skaičius visatoje [EN88]. Norint išvengti visų kombinacijų perrinkimo, autorius pristato šiai problemai išspręsti dvejetainės paieškos algoritmu paremtą ženklų paieškos pseudokodą, matomą žemiau esančiame algoritme 7. Pagrindinė algoritmo veikimo idėja - paeiliui einant per ženklus nuo 1-ojo iki vektoriaus dimensijos, patikrinti ar vektoriaus norma sumažėja, pakeičiant i-tąjį ženklą neigiamu. Jei norma sumažėja, ženklą paliekame ir kartojame su $i+1$-tuoju ženklu.

Įrodykime, kad pataikius vieną ženklą, vektorių skirtumo norma sumažėja. Pirma, pertvarkykime formulę (43) į patogesnę formą. Tarkime c teisingų ženklų vektorius, o y - spėjamų ženklų vektorius, $\dim(c) = \dim(y), \forall i \in 1,2,\ldots,\dim(c) : c_i, y_i \in \{-1,1\}$:

$$A \cdot (y \odot x') - b \tag{43}$$

$$b = A \cdot (c \odot x') \tag{44}$$

$$c \odot x' = x' \odot c = \text{diag}(x') \cdot c \tag{45}$$

$$A \cdot \text{diag}(x') \cdot y - A \cdot \text{diag}(x') \cdot c \qquad \text{Pritaikome (44) ir (45)} \tag{46}$$

$$A \cdot \text{diag}(x') \cdot (y - c) \tag{47}$$

$$\|Ax\| \leq \|A\| \cdot \|x\|, \quad A \in \mathbb{R}^{n \times n}, x \in \mathbb{R}^n \tag{48}$$

Pasinaudoję matricų ir vektoriaus normų nelygybės formule (48), pritaikome formulės (47) išraiš-

kai:

$$\|A \cdot \text{diag}(x') \cdot (y - c)\| \leq \|A \cdot \text{diag}(x')\| \cdot \|y - c\| \tag{49}$$

$$\|A \cdot \text{diag}(x') \cdot (y - c)\| \leq \|A \cdot \text{diag}(x')\| \cdot \sqrt{(y_1 - c_1)^2 + (y_2 - c_2)^2 + \ldots + (y_n - c_n)^2} \tag{50}$$

Tarkime turime atvejį, kai atspėjame pirmo vektoriaus nario ženklą, t.y.: $y_1 = c_1$:

$$n_a = \sqrt{\underbrace{(c_1 - c_1)^2}_{=0} + (y_2 - c_2)^2 + \ldots + (y_n - c_n)^2} \tag{51}$$

Ir turime priešingą situaciją, kai pirmo vektoriaus nario ženklo neatspėjame:

$$y_1 = -c_1 \tag{52}$$

$$n_b = \sqrt{\underbrace{(-c_1 - c_1)^2}_{=4c_1^2} + (y_2 - c_2)^2 + \ldots + (y_n - c_n)^2} \tag{53}$$

Parodykime, kad n_b yra daugiau už n_a, t.y., vektoriaus $x - c$ norma didesnė, kai vektoriaus elemento ženklas klaidingas:

$$n_b > n_a \tag{54}$$

$$\sqrt{4c_1^2 + (y_2 - c_2)^2 + \ldots + (y_n - c_n)^2} > \sqrt{(y_2 - c_2)^2 + \ldots + (y_n - c_n)^2} \tag{55}$$

$$4c_1^2 > 0 \tag{56}$$

Kadangi formulės (49) dešinėje pusėje esanti $\|A \cdot \text{diag}(x')\|$ matricos norma yra neneigiama bet kokiai matricai, atspėjant vektoriaus $y - c$ ženklus pažingsniui, kairės nelygybės (49) pusės reikšmė irgi mažės. Automatiškai kairė nelygybės (49) pusė bus visada mažesnė, kas reiškia, kad atspėjus vieną vektoriaus nario ženklą formulėje (47) esančio vektoriaus norma mažės.

7 algoritmas. Geriausios ženklų kombinacijos radimo algoritmas

Input: Normalizuotas, su netinkamais ženklais vektorius x, matrica A, vektorius b.
Output: Vektorius x su geriausia ženklų kombinacija.
function BestMatchingSignsVector(x)
    minDifference = $\|A \cdot x - b\|$
    for $i = 0$ to len(x) **do**
        x[i] = -1 · x[i]
        difference = $\|A \cdot x - b\|$
        if minDifference < difference **then**
            x[i] = -1 * x[i]
        else
            minDifference = difference
        end if
    end for
    return x
end function

Šis pseudokodo algoritmas padeda sumažinti perrinkimų skaičių nuo $2^{2^{10}}$ iki 2^{10}.

Radus tinkamą ženklų konfigūraciją, reikia rasti vektoriaus $\vec{x}$ normą. Autorius pristato formulę, kurios dėka randama vektoriaus norma, nenaudojant tiesinės regresijos sprendimo, pristatymo [YSM+23] straipsnyje. Atliekami tokie formulių pertvarkymai:

$$\vec{x'} = \frac{\vec{x}}{\|\vec{x}\|} \;\rightarrow\; \vec{x} = \vec{x'} \cdot \|\vec{x}\| \tag{57}$$

$$A\vec{x} = \vec{b} \tag{58}$$

$$\|x\| \cdot A\vec{x'} = \vec{b} \qquad \text{(įstatome (57) lygtį į (58))} \tag{59}$$

$$\|x\| \cdot \langle A\vec{x'}, \vec{b} \rangle = \langle \vec{b}, \vec{b} \rangle \tag{60}$$

$$\|x\| = \frac{\langle \vec{b}, \vec{b} \rangle}{\langle A\vec{x'}, \vec{b} \rangle} \tag{61}$$

Vektorius $\vec{x}$ gaunamas naudojantis formulių (57) ir (61) pagalba suskaičiuota vektoriaus norma. Gautas vektorius yra tinkamas naudoti testavimo duomenų rinkinio klasifikavimui.

2. Eksperimentinė dalis

Šiame skyriuje yra pateiktos visos technologinės ir programinės priemonės, kurios buvo naudojamos atliekant eksperimentus: SVM ir VQLS-SVM algoritmų palyginimo, VQLS-SVM algoritmo mokymosi efektyvumo tyrimai su skirtingais optimizatoriais ir algoritmo lygiagretinimo tyrimų rezultatai.

2.1. Įranga

Google Colab – debesų kompiuterijos platforma, suteikianti nemokamą prieigą prie skaičiavimo išteklių ir galimybę paleisti pasirinktą Jupyter užrašinę (*angl. Jupyter notebook)*. Šioje platformoje prieinami CPU, GPU ir TPU resursai, tad tinka mašininio mokymosi, giliojo mokymosi (*angl. deep learning*). Jei naudojant nemokamą versiją, prireikia daugiau resursų, Google siūlo įsigyti Pro versiją, kur gaunami galingesni GPU ir TPU resursai ir sesijos palaikymas, kol atliekami skaičiavimai. Vienas minusas, lyginant su eksperimentų skaičiavimu naudojant asmeninį kompiuterį, kad pasibaigus sesijai, visi papildomi failai, dingsta. Taip pat, jei projektas yra išskaidytas į daug failų susiduriama su viso projekto įkėlimu į Google Colab aplinką problemomis.

Taip pat darbo metu atlikti tyrimus buvo naudojamas asmeninis nešiojamas kompiuteris „Macbook M2 Pro“, 32 GB operatyvioji atmintis (*angl. Random access memory (RAM))*, 12 branduolių.

2.2. Įrankiai

Duomenų apdorojimui ir VQLS-SVM algoritmo tyrimui buvo naudota:

1. Python programavimo kalba (3.11.4 versija);
2. IBM Qiskit karkasas, skirtas kvantinėms grandinėms konstruoti (1.0.2 versija);
3. nuostolių funkcijos reikšmei optimizuoti panaudota Scipy biblioteka;
4. Numpy biblioteka skirta matematiniams skaičiavimams;
5. grafikų braižymui pasitelkta Matplotlib biblioteka.

2.2.1. IBM Qiskit

Jau anksčiau minėtas IBM Qiskit yra atviro kodo kvantinių skaičiavimų karkasas, kuris leidžia kurti, simuliuoti, paleisti kvantinius algoritmus ir grandines, naudojant simuliatorius ir tikrus kvantinius kompiuterius. Ši biblioteka parašyta „IBM“ kompanijos 2017 metais naudojantis Python programavimo kalba, kad kvantiniai skaičiavimai ir algoritmai būtų prieinami plačiai auditorijai ir lengvai suprantami. IBM Qiskit suteikia galimybę dirbti su realiu kvantiniu kompiuteriu, leidžiant mokslininkams, studentams ir kvantinių skaičiavimų entuziastams išbandyti savo algoritmus realioje aplinkoje. Karkasas palaiko ir kvantinių kompiuterių simuliavimą naudojantis klasikiniais kompiuteriais. Simuliatoriai suteikia galimybę atlikti įvairius eksperimentus be prieigos prie fizinio kvantinio kompiuterio. Šio karkaso vienas iš pagrindinių privalumų – galimybė pasiekti labai didelį skaičių mokomosios medžiagos parašytos „IBM“ kompanijos ir programuo-

tojų, prisidėjusių prie atvirojo kodo ir teorinės medžiagos apie kvantinius skaičiavimus kūrimo. Kadangi IBM Qiskit yra vienas iš populiariausių karkasų kvantinių skaičiavimų srityje, naujai sukurti algoritmai yra greitai pridedami į jos biblioteką, užtikrinant, kad vartotojai turėtų prieigą prie pažangiausių kvantinių skaičiavimo metodų. Ši biblioteka yra taikoma kvantinio mašininio mokymosi (*angl. quantum machine learning*), chemijos tyrimams ir algoritmų kūrimui.

Šiame darbe IBM Qiskit karkasas naudojamas paruošti parametrizuotoms grandinėms, jas translipiuoti, skaičiuoti VQLS-SVM algoritmo nuostolių funkcijos reikšmę ir paleisti su įvairiais simuliatoriais.

2.2.2. Scipy

SciPy yra pagrindinė mokslinių skaičiavimų atviro kodo biblioteka, skirta įvairiems sudėtingesnių uždavinių sprendimams: tiesinė algebra, integravimas, funkcijų interpoliacija, optimizavimas, signalų ir vaizdų apdorojimas. Ši biblioteka itin gerai vertinama dėl savo universalumo ir integracijos su kitomis bibliotekomis: NumPy, Pandas, Matplotlib.

Šiame darbe naudojamas pagrindinis SciPy funkcionalumas - galimybė efektyviai optimizuoti įvairias funkcijas, šiuo atveju, nuostolių funkciją. Naudojamos įvairios išvestinių nereikalaujančių optimizatorių (*angl. derivative-free optimization*) ir gradientinės funkcijos.

2.2.3. NumPy

NumPy yra pagrindinė Python biblioteka, skirta matematiniams skaičiavimams. Ši biblioteka parašyta Travis Oliphant 2005 metais, sujungus Numeric ir Numarray bibliotekas, siekiant sukurti vieną galingą masyvų apdorojimo paketą. Dėl efektyvaus didelių duomenų rinkinių tvarkymo ir sparčių operacijų šios bibliotekos naudojimas plačiai paplito atliekant mokslinius skaičiavimus inžinerijoje ir duomenų moksle. Kadangi NumPy bibliotekos programinis kodas yra viešai prieinamas, tai suteikia galimybę vartotojams naudotis gausia funkcijų aibe. Tai padeda suprasti šių algoritmų ir metodų įgyvendinimo būdus. Taip pat skatinamas bendruomenės narių indėlis, leidžiant prisidėti prie algoritmų gerinimo ir klaidų taisymo.

Šiame darbe NumPy biblioteka naudojama duomenų rinkinių paruošimui. Jo didelės manipuliavimo masyvais galimybės leidžia efektyviai apdoroti ir paruošti duomenų aibes apmokymui.

2.2.4. Matplotlib

Matplotlib yra dažniausiai naudojama 2D,3D grafikų kūrimui Python programavimo kalba. Ją 2003 m. sukūrė John D. Hunter, siekdamas Python alternatyvą Matlab programavimo aplinkai. Matplotlib palaiko daugybę diagramų tipų: linijines diagramas, stulpelines diagramas ir kt. Matplotlib galima paprastai integruoti su NumPy ir Pandas programavimo karkasais, todėl tai yra pagrindinis duomenų mokslininkų, analitikų ir tyrėjų įrankis. Dėl gebėjimo kurti sudėtingus grafikus vos keliomis kodo eilutėmis, išsamios dokumentacijos ir atviro kodo, kurį galima pataisyti, radus klaidų, šis karkasas yra plačiai naudojamas.

2.3. Duomenų rinkinys

SVM ir VQLS-SVM algoritmų tyrimui buvo pasirinkti irisų [Fis88] ir krūties auglių duomenų aibės [WS95]. Irisų duomenų aibė sudaryta iš 150 irisų duomenų. Kiekvienas irisas duomenų rinkinyje turi 4 požymius:

- taurėlapio ilgis (cm);
- taurėlapio plotis (cm);
- žiedlapio ilgis (cm);
- žiedlapio plotis (cm);

Duomenų rinkinys turi tris klases: Iris Setosa, Iris Versicolour ir Iris Virginica.

Krūties auglių duomenų rinkinys sudarytas iš 569 auglių. Kiekvienas auglys duomenų rinkinyje turi 30 požymių. Duomenų rinkinys turi dvi klases: piktybinis, t.y. vėžinis (*angl. malignant*), ir gerybinis (*angl. benign*).

2.4. Duomenų paruošimas

VQLS-SVM ir SVM algoritmų mokymui iš irisų duomenų aibės pašalinta Iris Virginica duomenys ir atsitiktinai parinkti 7 įrašai 3 kubitų atveju, 15 įrašų 4 kubitų atveju ir 31 - 5 kubitų. Iris Setosa duomenų klasė pakeista į -1, Iris Versicolour į 1. Toks duomenų paruošimas buvo atliktas [YSM+23] straipsnyje.

Taip pat buvo ištirtas 1 prieš visus (*angl. one vs all*) metodas. Irisų duomenų aibės įrašų kiekis kiekvienam eksperimentui buvo atitinkamai 7, 15, 31 įrašai. Atsitiktinai pasirenkama viena iš irisų duomenų aibės klasė, ji pakeista į 1, kiti duomenys buvo pakeisti į -1.

Krūties auglių duomenų rinkinio atsitiktinai paimti 7 įrašai 3 kubitų atveju, 15 įrašų 4 kubitų atveju ir 31 - 5 kubitų. Benign duomenų klasė pervadinta į -1, malignant į 1.

2.5. Duomenų apdorojimas

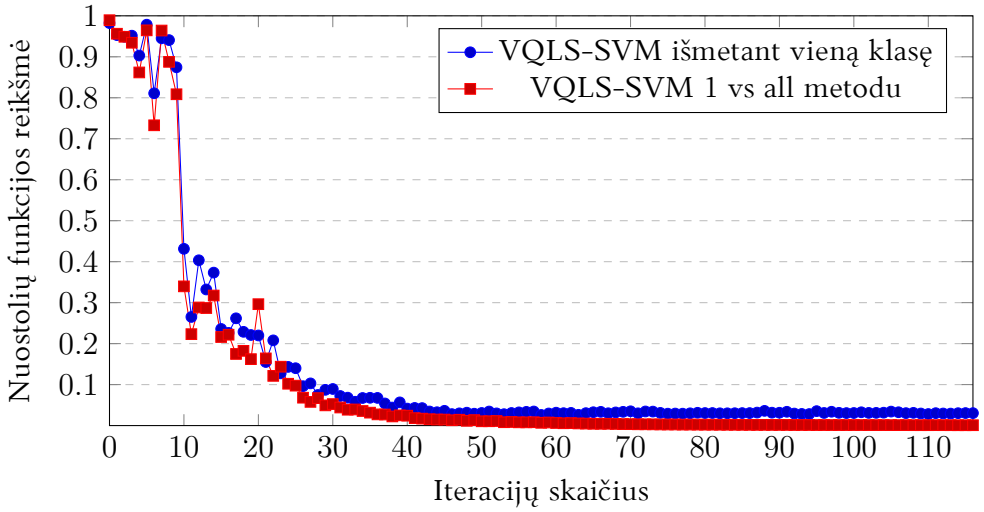
Prieš perduodant irisų ir krūties auglių duomenis VQLS-SVM ir SVM algoritmams, duomenys apdorojami. Sukonstruojama lygybėje (4) esanti matrica, naudojantis duomenimis. Duomenų klasės atitinkamai užkoduojamos į $[0\ \vec{y}]^T$. Paruošta LSSVM matrica išskaidoma į unitarines, su koeficientais naudojantis funkcijų 2 ir 3 realizacijomis Python programavimo kalba.

2.6. VQLS-SVM ir SVM algoritmų mokymai ir gauti rezultatai

2.6.1. Algoritmo tyrimai su kubitų skaičiumi lygiu 3

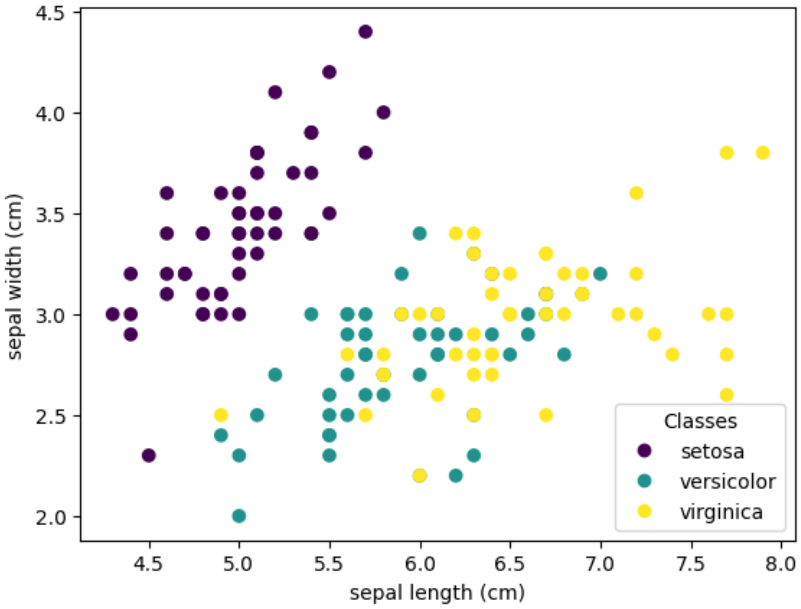
VQLS-SVM algoritmo minimizacijos funkcijos 6 pseudokodas realizuotas naudojantis Python programavimo kalba ir IBM Qiskit karkasu. Geriausių svorių paieška truko 200 epochų, su reguliarizacijos parametru $\gamma = 0{,}01$. Minimizacijos funkcijos optimizavimo panaudota funkcija: *COBYLA*. Kvantinės grandinės buvo leidžiamos naudojantis *aer_simulator* simuliatorių. Kiekviena kvantinė grandinė buvo paleista $shots = 10000$ kartų. Apmokymas su atsitiktinai paimtais duomenimis buvo kartotas 10 kartų ir paimtas nuostolių funkcijos reikšmių aritmetinis vidurkis.

VQLS-SVM nuostolių funkcijos grafikus su irisų duomenų rinkiniu galime matyti paveikslėlyje 13. Apmokant VQLS-SVM algoritmą su irisų duomenimis, programa pasiekė minimalią reikšmę $C(\alpha_{opt}) = 0{,}03$ po 115 iteracijų. Algoritmo mokymo truko apytiksliai 1 valandą.

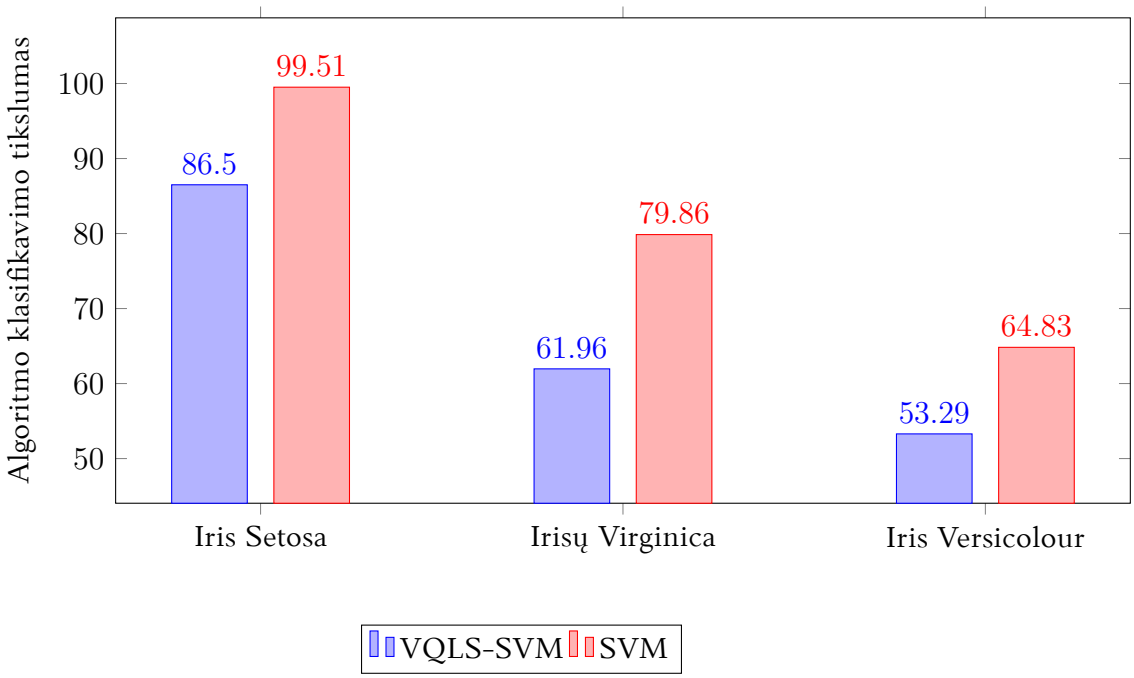


13 pav. Nuostolių funkcijos grafikai apmokant VQLS-SVM algoritmą su irisų duomenų aibe

Taip pat buvo atliktas vienas prieš visus metodo tyrimas pasirenkant vieną iš irisų duomenų aibės klasę, o kitas pakeičiant į neigiamą klasę. Atlikti eksperimentai su kiekviena kombinacija. Eksperimentas truko apie 3 valandas. Klasifikavimo tikslumus naudojant VQLS-SVM ir SVM algoritmus su skirtingomis pasirinktomis irisų duomenų klasėmis galima matyti žemiau esančiame paveikslėlyje 15. Galime pastebėti, kad klasifikavimo tikslumas yra sumažėjęs, kai kaip teigiama klasė yra pasirinkta Iris Virginica arba Iris Versicolour (61,96 ir 53,29 procentų klasifikavimo tikslumas). Taip yra todėl, nes šios klasės, lyginant su Iris Setosa (86,5 procentų klasifikavimo tikslumas) , yra gana panašios, jų duomenys glaudžiai pasiskirstę (apačioje esantis paveikslėlis 14).

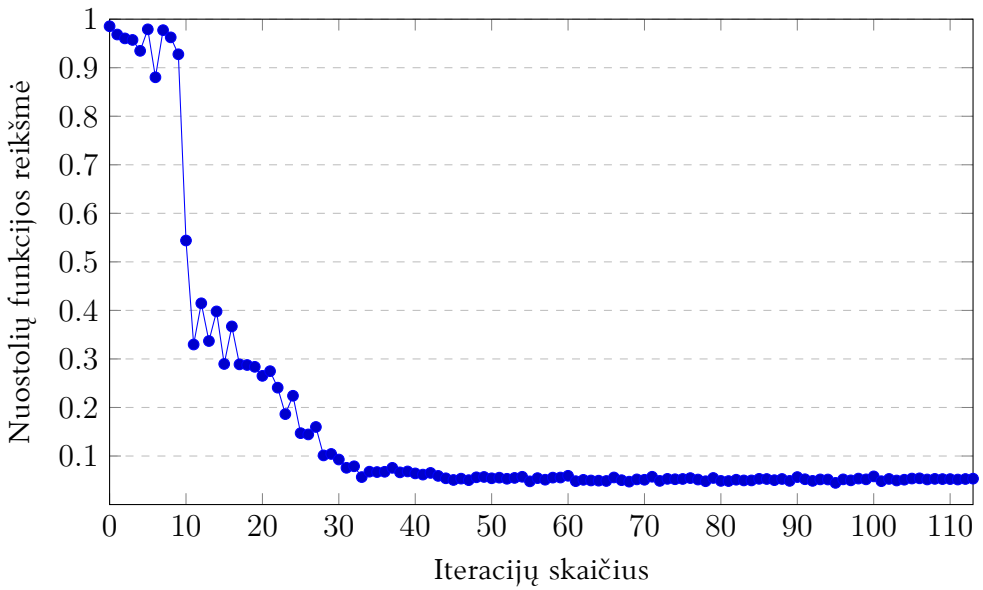


14 pav. Irisų duomenų aibės klasių pasiskirstymas



15 pav. VQLS-SVM ir SVM tikslumas irisų vienas prieš visus (*angl. one vs all*) metodu

Apmokant VQLS-SVM algoritmą su krūties auglių duomenimis, nuostolių funkcija pasiekė minimalią reikšmę $C(\alpha_{opt}) = 0{,}05$, 113 iteracijos metu. Algoritmo apmokymo truko apie 1 valandą. Nuostolių funkcijos grafiką galima matyti paveikslėlyje 16.



16 pav. Nuostolių funkcijos grafikas apmokant VQLS-SVM algoritmą su krūties auglių duomenų aibe

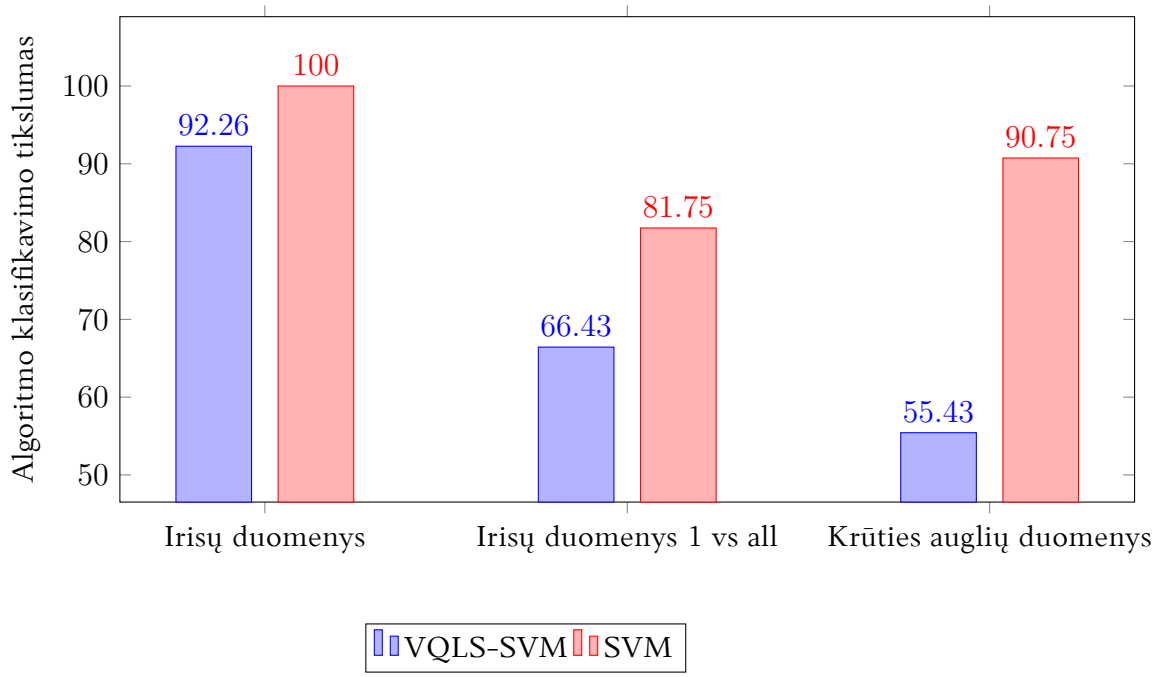
Naudojantis tomis pačiomis irisų ir krūties auglių apmokymo duomenų aibėmis, buvo apmokyti klasikiniai atraminių vektorių klasifikatoriai. Klasikinio SVM ir kvantinio VQLS-SVM algoritmų klasifikavimo tikslumus kiekvienam duomenų rinkiniui matome paveikslėlyje 17. Matyti, kad VQLS-SVM klasifikavimo tikslumas irisų aibei, naudojantis tik 7 duomenų rinkinio įrašais - 92,26%. Klasikinis SVM pasiekė 100% tikslumą. Su krūties auglių duomenų aibe VQLS-

SVM algoritmas pasiekė 55,43% tikslumą, o klasikinis SVM - 90,75%. Šis klasifikavimo tikslumo sumažėjimas klasikinio ir kvantinio algoritmo atveju gali kilti dėl keleto priežasčių:

1. Krūties auglių duomenų aibės įrašai turėjo 30 požymių, irisų - 4.
2. Esant didesniam duomenų požymių skaičiui, reikia didesnio apmokymo rinkinio, negu požymių skaičius.
3. VQLS-SVM algoritmas esant nuostolių funkcijos reikšmei nelygiai 0 negauna tikslios vektoriaus x reikšmės. Netiksli sprendinio reikšmė gali turėti įtakos klasifikavimo tikslumui.

Su irisų duomenų aibe, tikslumų tarp SVM ir VQLS-SVM algoritmų skirtumas: 7,74%, su krūties auglių duomenų aibe - 35,34%. Padidėjęs atotrūkis tarp klasikinio ir kvantinio algoritmų klasifikavimo tikslumų gali būti dėl didesnio požymių skaičiaus negu apmokymų aibės dydis. Turint kvantinio algoritmo realizaciją, kai kubitų skaičius lygus 3, požymių skaičius didesnis už $2^3 - 1 = 7$, susiduriama su problema - sudėtingesnio duomenų rinkinio, turinčio daugiau nei 7 požymius, klasifikavimas gali būti nepakankamai tikslus.

Taip pat atlikus eksperimentus su vienas prieš visus metodu, kitaip, nei [YSM+23] straipsnyje, buvo pastebėta, kad VQLS-SVM algoritmo klasifikavimo tikslumas sumažėjo (66,43% klasifikavimo tikslumas). Tačiau taip nutinka, nes Iris Virginica ir Iris Versicolour duomenų aibės klasių elementai yra glaudžiai pasiskirstę ir sunku atskirti juos tarpusavyje.



17 pav. VQLS-SVM ir SVM tikslumas irisų ir krūties auglių testavimo duomenų rinkiniams

2.6.2. VQLS-SVM algoritmo tyrimas kubitų skaičiui lygiam 4 ir 5

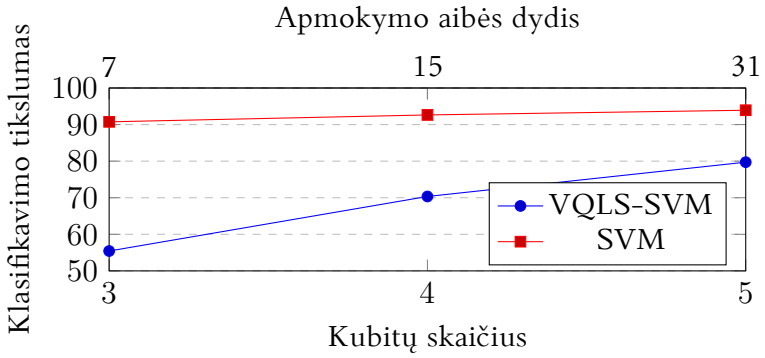
Norint ištirti, ar padidinus kubitų skaičių VQLS-SVM algoritmo tikslumas padidės, eksperimento metu buvo nustatyti bendri parametrai: reguliarizacijos parametru $\gamma = 0{,}01$. Minimizacijos funkcijos optimizavimo panaudota funkcija: *COBYLA*. Kvantinės grandinės buvo

leidžiamos naudojantis *aer_simulator* simuliatorių. Kiekviena kvantinė grandinė buvo paleista $shots = 10000$ kartų.

Atliekant eksperimentus su 4 kubitais, buvo pasirinktas epochų skaičius lygus 200. Apmokymas su atsitiktinai paimtais duomenimis buvo kartotas 10 kartų ir paimtas klasifikavimo tikslumų aritmetinis vidurkis. Algoritmo mokymo trukmė su irisų ir krūties auglių duomenų aibėmis - 8 valandos.

Apmokant VQLS-SVM algoritmą, kai kubitų skaičius lygus 5, pasirinktas epochų skaičius, lygus 50. Apmokymas buvo kartotas 5 kartus ir paimtas klasifikavimo tikslumų aritmetinis vidurkis. Tokie parametrai buvo pasirinkti, nes su epochų skaičiumi lygiu 200, vienas apmokymas trunka apie 10 valandų. Pastebėjus, kad paveikslėliuose 13 ir 16 nuostolių funkcija stabilizuojasi 40-50 epochų metu, galima epochų skaičių pasirinkti lygų 50. Bendra apmokymo trukmė: apytiksliai 37 valandos.

Gautus VQLS-SVM ir SVM algoritmų klasifikavimo tikslumus galima matyti žemiau paveikslėliuose 18 ir 19. Krūties auglių duomenų aibės atveju, matomas staigus tikslumo padidėjimas (nuo 55,43 iki 70,34). Tokį tikslumo padidėjimą lėmė didesnis požymių skaičius, nes 3 kubitų atveju, VQLS-SVM algoritmas nesutalpino visų krūties auglių duomenų aibės parametrų. Padidinus kubitų skaičių iki 5, matomas nedidelis, 9 procentų klasifikavimo tikslumo padidėjimas.

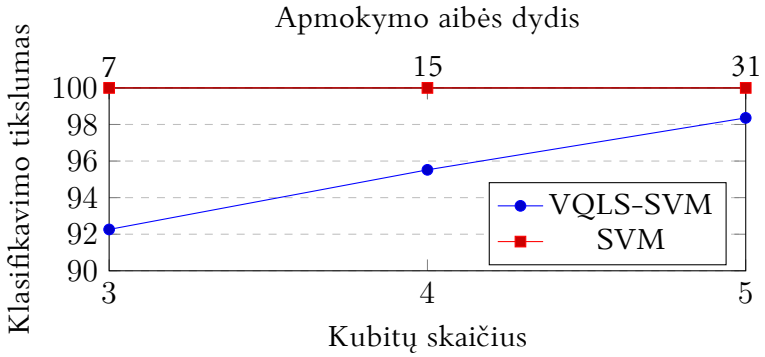


18 pav. Krūties auglių duomenų aibės klasifikavimo tikslumo priklausomybė nuo kubitų skaičiaus naudojant VQLS-SVM ir SVM algoritmus

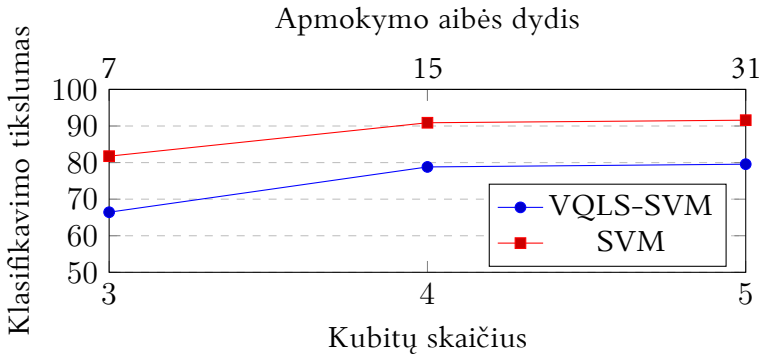
Irisų duomenų aibės atveju, matomas nedidelis tikslumo padidėjimas, duomenų aibė yra nedidelė ir turi 4 požymius, tai VQLS-SVM algoritmo kubitų skaičiaus didinimas padidina tik apmokymo aibės dydį. Padidinus apmokymo aibės dydį iki 31, irisų duomenų aibės klasifikavimo tikslumas, naudojantis VQLS-SVM algoritmą pasiekė 98,36 procentų tikslumą. Toliau didinti apmokymo aibę nėra verta, nes pasiekus aukštą klasifikavimo tikslumą, tolesnis algoritmo apmokymas su didesne apmokymo aibe gali sukelti VQLS-SVM algoritmo persimokymą (*angl. overfitting*), taip pakenkiant modelio veikimui ir jo klasifikavimo tikslumui su realiais duomenimis.

Apmokant VQLS-SVM ir SVM algoritmus vienas prieš visus metodu paruošta irisų duomenų aibe 4 ir 5 kubitų atveju, galima pastebėti tolygų klasifikavimo tikslumo padidėjimą padidinus duomenų aibę iki 15 elementų (78,81% VQLS-SVM ir 90.89% SVM algoritmų klasifikavimo tiks-

lumas) ir nežymus, 1 procento padidėjimas padidinus iki 31 apmokymo aibės elementų dydžio. Toliau didinant apmokymo aibės dydį, testavimo aibės klasifikavimo tikslumas gali nedidėti.



(a) Išmetus Iris Virginica duomenų klasės elementus



(b) Naudojantis vienas prieš visus (*angl. one vs all*) metodu

19 pav. Irisų duomenų aibės klasifikavimo tikslumo priklausomybė nuo kubitų skaičiaus naudojant VQLS-SVM ir SVM algoritmus

2.7. Skirtingų VQLS-SVM mokymui skirtų optimizatorių tyrimas ir rezultatai

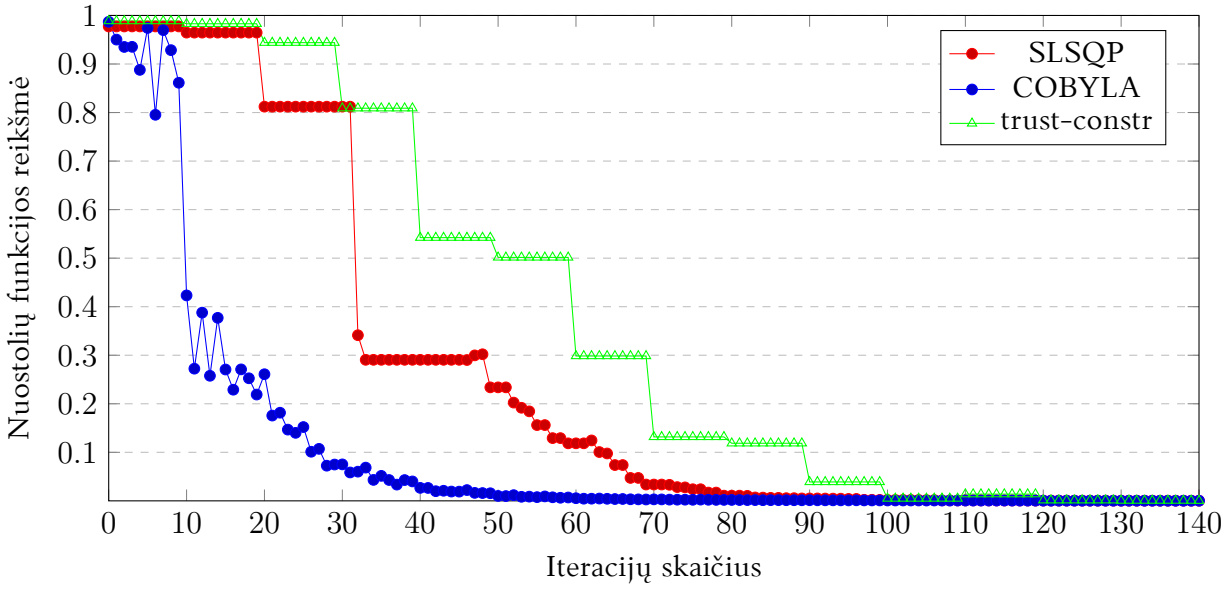
Šiame tyrime naudoti visi tokie patys parametrai, išskyrus optimizavimo funkciją. Kvantinės grandinės buvo leidžiamos naudojantis *state_vector* simuliatoriumi. Eksperimentai atliekami tik su irisų duomenų rinkiniu, kadangi paveikslėliuose 13 ir 16 matome, kad irisų duomenų ir krūties auglių nuostolių funkcijų grafikai panašūs, tyrimo rezultatai atitinkamai bus panašūs.

2.7.1. VQLS-SVM tyrimas naudojantis negradientiniais optimizatoriais

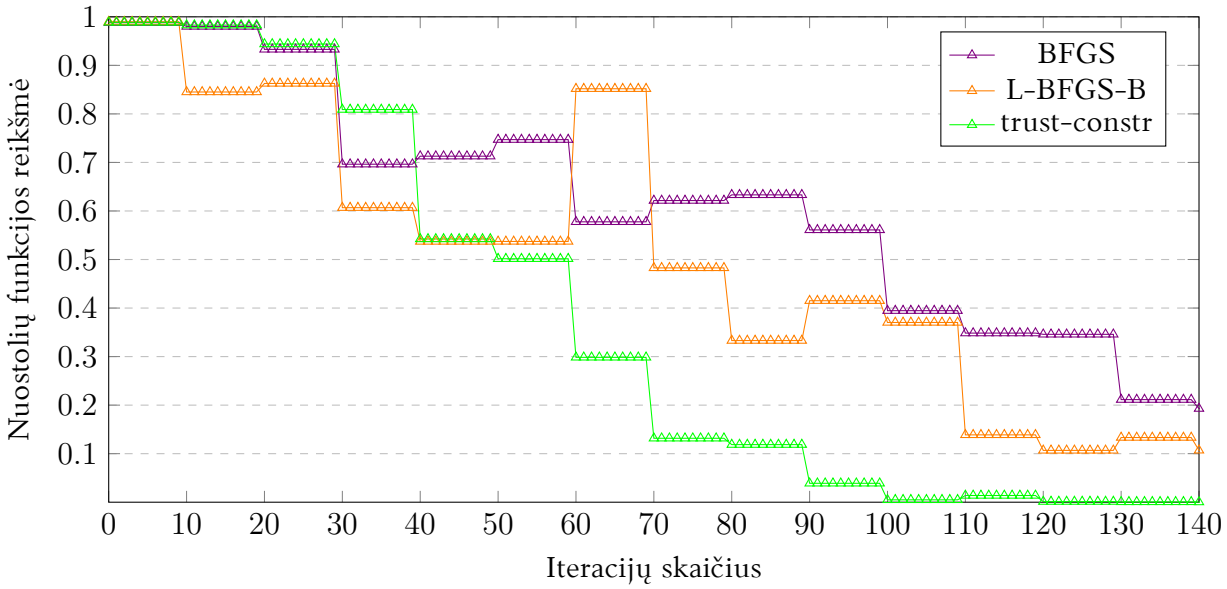
VQLS-SVM uždavinys sprendžiamas naudojantis optimizavimo funkcijomis: *COBYLA*, *SLSQP*, *BFGS*, *L-BFGS-B*, *trust-constr*. Norint gauti tikslesnius nuostolių funkcijų grafikus, eksperimentas buvo atliekamas 10 kartų su kiekviena optimizacijos funkcija. Atlikus šiuos eksperimentus, galutinė nuostolių funkcija buvo gauta apskaičiavus aritmetinį vidurkį su kiekvienos optimizavimo funkcijos nuostolių funkcijos reikšmėmis.

Atlikto eksperimento nuostolių funkcijos matomos žemiau pateiktame paveikslėlyje 20. Paveikslėlis išskaidytas į 2 mažesnius paveiklėlius, (a) paveikslėlyje matome 3 geriausias optimiza-

vimo funkcijas, o (b) - 3 blogiausias. *trust-constr* optimizavimo funkcijos grafikas naudojamas abiejuose paveikslėliuose, siekiant palyginti įvairius optimizatorius ir jų grafikus. Bendra eksperimento trukmė - 3 valandos 4 minutės. Naudojantis *COBYLA* optimizavimo funkcija, nuostolių funkcijos reikšmė $C(\alpha) = 0{,}1$ buvo pasiekta 26 iteracijos metu, kitiems optimizatoriams pasiekti 0,1 reikšmę reikėjo daugiau iteracijų (naudojant *L-BFGS-B* optimizavimo funkciją, $C(\alpha) = 0{,}1$ buvo pasiekta po 140 iteracijų). Taip pat galime pastebėti, kad *SLSQP*, *BFGS* ir kitų panašių algoritmų nuostolių funkcijos reikšmės nekinta tam tikrą iteracijų skaičių (*SLSQP* optimizavimo funkcijos reikšmė nekinta 20-30 iteracijų metu).



(a) *SLSQP*, *COBYLA* ir *trust-constr* optimizavimo funkcijos



(b) *BFGS*, *L-BFGS-B* ir *trust-constr* optimizavimo funkcijos

20 pav. Nuostolių funkcijos naudojantis skirtingas negradientines optimizavimo funkcijas

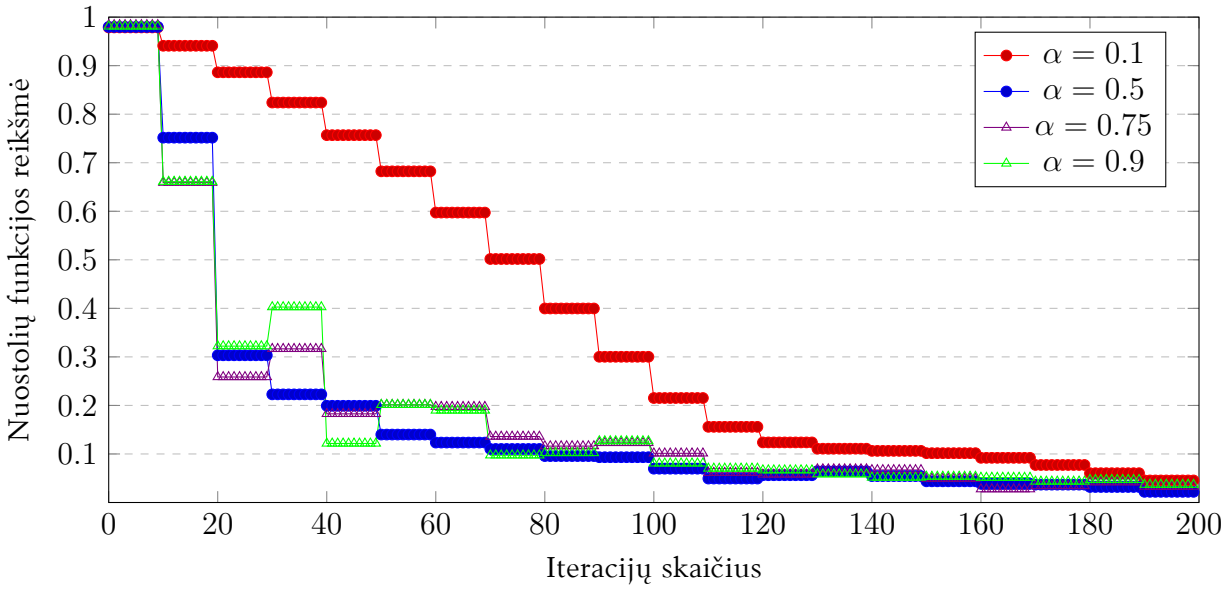
2.7.2. VQLS-SVM tyrimas naudojantis gradientiniais optimizatoriais

Buvo atliekami eksperimentai su VQLS-SVM ir 2 gradientinėmis optimizavimo funkcijomis: gradientinio nusileidimo (*angl. gradient descend)* ir *ADAM* (*Adaptive Moment Estimation*). Siekiant rasti geriausią mokymosi greičio reikšmę, algoritmai buvo ištestuoti su keleta skirtingų mokymosi greičio reikšmių: *ADAM* optimizatoriui atlikti eksperimentai su reikšmėmis 0,1; 0,5; 0,75; 0,9, o gradientinio nusileidimo su 0,01; 0,05; 0,1; 0,5; 0,6; 0,7; 0,8; 0,9 reikšmėmis. Kiekvienam mokymosi greičiui, eksperimentai buvo atliekami po 5 kartus ir paimamas nuostolių funkcijų reikšmių aritmetinis vidurkis. Bendra eksperimento trukmė- apytiksliai 13 valandų. Kadangi gradientinio nusileidimo funkcija su mokymosi greičiais 0,01; 0,05; 0,1 po 200 iteracijų nepasiekė nuostolių funkcijos reikšmes mažesnės nei 0,9, nuspręsta neatvaizduoti grafike. Nuostolių funkcijos reikšmės, gautos po 200 iteracijų, matomos žemiau pateiktoje lentelėje (1).

1 lentelė. *VQLS-SVM algoritmo nuostolių funkcijos reikšmės po 200 iteracijų su skirtingomis gradientinio nusileidimo optimizatoriaus mokymosi greičio reikšmėmis*

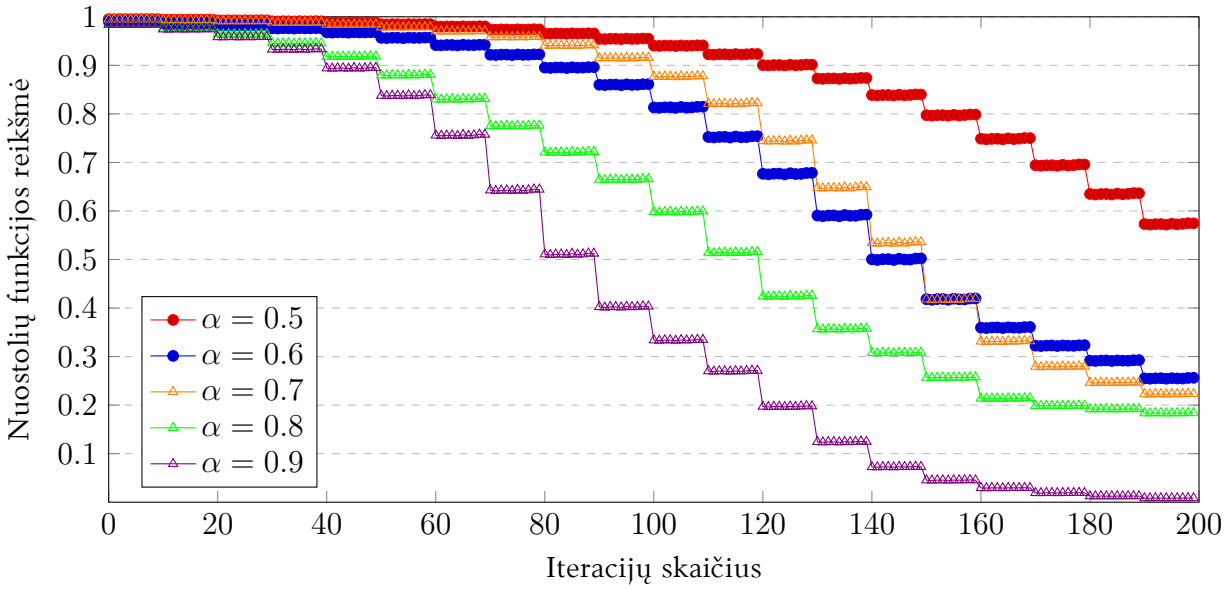
Mokymosi greičio reikšmė	**Nuostolių funkcijos reikšmė po 200 iteracijų**
0,01	0,98
0,05	0,95
0,1	0,93

Gradientinio nusileidimo ir *ADAM* optimizavimo funkcijų nuostolių funkcijų grafikus galima matyti žemiau esančiuose paveikslėliuose 21 ir 22 . *ADAM* optimizavimo funkcija pasiekė geriausius rezultatus, kai mokymosi greičio reikšmė lygi 0,5. Su didesnėmis α reikšmėmis, nuostolių funkcijos reikšmė padidėja (30, 50 iteracijos). didesnė mokymosi greičio reikšmė, šiuo atveju reiškia, kad optimali reikšmė yra peršokama, ir mokymosi procesas tampa nestabilus. Gradientinių ir negradientinių optimizatorių palyginime bus naudojama *ADAM* funkcija su $\alpha = 0.5$.



21 pav. ADAM optimizatoriaus nuostolių funkcijos su skirtingomis mokymosi greičio reikšmėmis

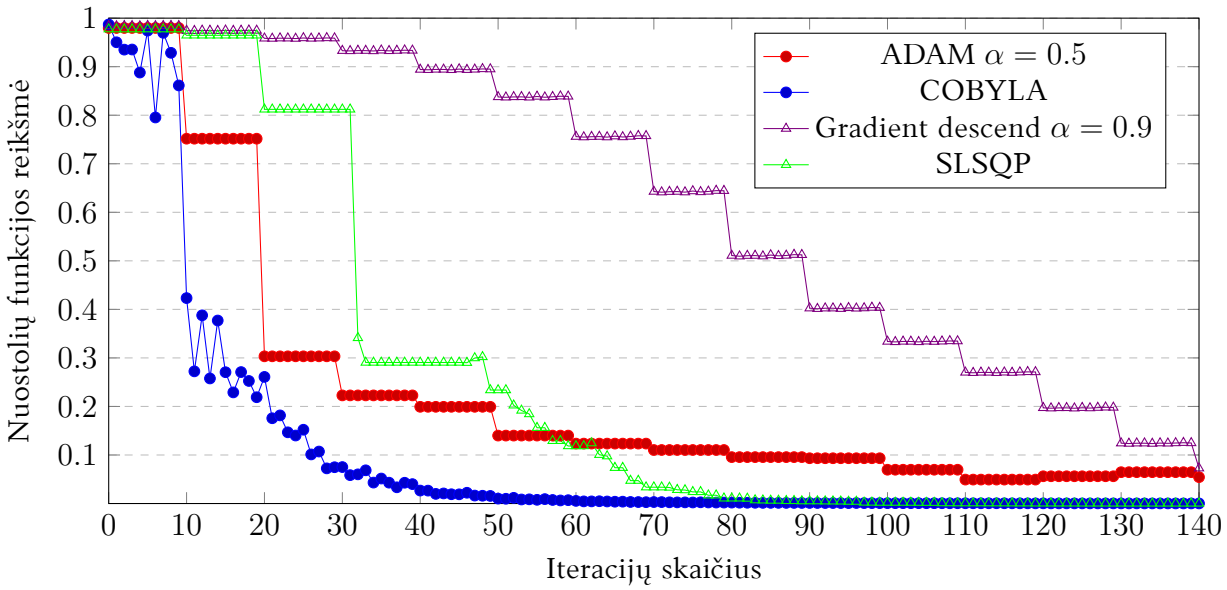
Gradientinio nusileidimo optimizatoriaus atveju galima matyti, kad didinant mokymosi greičio reikšmę, nuostolių funkcijos reikšmė sparčiau mažėja, kas reiškia greitesnį VQLS-SVM problemos sprendinio radimą. Toliau didinant mokymosi greitį, galima susidurti su problemomis, kad mokymosi procesas ne tik taps nestabilus, bet nuostolių funkcija gali diverguoti. Todėl VQLS-SVM problemai gradientinio nusileidimo optimizatoriaus atveju, toliau bus naudojama mokymosi reikšmė lygi 0,9.



22 pav. Gradientinio nusileidimo nuostolių funkcijos su skirtingomis mokymosi greičio reikšmėmis

2.7.3. VQLS-SVM tyrimo rezultatų palyginimas su gradientiniais ir negradientiniais optimizatoriais

Radus geriausias mokymosi greičio reikšmes *ADAM* ir gradientinio nusileidimo optimizatorių funkcijų atvejais, buvo palyginta su 2 geriausiais negradientiniais optimizatoriais - *COBYLA* ir *SLSQP*. Žemiau pateiktame paveikslėlyje 23 galima matyti 4 geriausias optimizatorių funkcijas: *COBYLA*, *SLSQP*, *ADAM* ir gradientinio nusileidimo. Galima pastebėti, kad *COBYLA* 26 išlieka pirmaujanti optimizavimo funkcija tarp gradientinių ir negradientinių funkcijų. *ADAM* optimizavimo funkcija pralenkia *SLSQP* optimizavimo funkciją, jei VQLS-SVM algoritmas yra apmokomas tik 50 epochų. Šiuo atveju, *ADAM* galėtų būti naudojamas kaip alternatyva *COBYLA* algoritmui.



23 pav. ADAM, Gradient descent ir COBYLA optimizavimo funkcijos

2.8. Išlygiagretinto algoritmo geriausių parametrų tyrimas

Pritaikius algoritmo lygiagretinimą kvantinių grandinių konstravime, transpiliavime ir VQLS-SVM vienos epochos nuostolių funkcijos reikšmės skaičiavime, norėta rasti, su kokiu gijų skaičiumi ir darbo dydžiu (*angl. job size*) algoritmas greičiausiai randa sprendinį. Darbo dydis šiame kontekste yra kiek VQLS-SVM nuostolių funkcijos reikšmei rasti reikalingų grandinių gauna viena gija.

Eksperimentai buvo atlikti su gijų ir darbo dydžiu nuo 1 iki 10 su MacBook M2 Pro ir iki 8 gijų naudojant Google Colab Pro aplinką, kadangi Google Colab turi tik 8 branduolius. Kiekvienai gijų ir darbo reikšmių kombinacijai, eksperimentas buvo atliktas 10 kartų ir paimtas rezultatų aritmetinis vidurkis. Naudota irisų duomenų aibė. Kiekvienai gijų ir darbo dydžio reikšmių kombinacijai, buvo naudotos tos pačios 10 irisų apmokymo aibės. Visas eksperimento truko apytiksliai 13 valandų. Žemiau pateiktoje lentelėje (2) galima matyti visus eksperimento rezultatus. Geriausi rezultatai su MacBook M2 Pro buvo pasiekti su 10 gijų ir 4 darbo dydžiu ir su 2 gijomis ir 2 darbo dydžiu Google Colab aplinkoje. Tačiau nedidelis, 9,31 sekundžių pagreitėjimas (1,7 sekundžių Google Cobal Pro atveju) lyginant su 1 gijos ir darbo dydžiu lygiam 1 atveju, parodo kad tik maža algoritmo dalis sėkmingai lygiagretinama. Tai reiškia, kad nepaisant gijų ir darbo dydžio padidinimo, tik nedidelė algoritmo dalis gali būti vykdoma lygiagrečiai, o likusi dalis yra atliekama nuosekliai, kas riboja bendrą našumo padidėjimą. Taip pat galime pastebėti, kad geriausias MacBook M2 Pro 10 gijų rezultatas (75,81 sekundės) nedaug skiriasi nuo geriausio 4 gijų rezultato (77,21 sekundės), galima sėkmingai naudoti programą ir su mažesniu gijų skaičiumi. Tai parodo, kad ne visada didinant gijų skaičių, VQLS-SVM algoritmo trukmė tiesiškai mažės.

2 lentelė. *VQLS-SVM algoritmo trukmė su skirtingomis gijų ir darbo dydžio reikšmėmis*

Gijų skaičius	**Darbo dydis**	**Trukmė (s) su MacBook M2 Pro**	**Trukmė (s) su Google Colab**
1	1	85,12	167,33
2	1	86,38	177,36
2	**2**	84,86	**165,63**
2	4	80,73	167,56
2	8	83,39	172,23
2	10	83,25	171,48
4	1	99,33	188,74
4	2	87,44	179,91
4	4	80,91	175,93
4	8	83,49	183,49
4	10	83,13	180,81
8	1	122,29	211,73
8	2	86,69	186,72
8	**4**	**77,21**	189,20
8	8	78,74	183,68
8	10	77,97	176,05
10	1	134,86	-
10	2	77,52	-
10	**4**	**75,81**	-
10	8	78,75	-
10	10	91,96	-

Rezultatai ir išvados

Šiame darbe buvo pristatyta atraminių vektorių klasifikatoriaus teorija, uždavinio performulavimas į mažiausių kvadratų formą (LS-SVM), variacinio kvantinio tiesinių lygčių sistemų algoritmo teorija, aptarti vieni iš svarbiausių žingsnių sprendžiant uždavinius naudojant kvantinius kompiuterius: vektorių, matricų užkodavimas į kvantines grandines. Sukonstruotas VQLS-SVM algoritmas ir pritaikyti optimizavimai, pritaikyti algoritmo lygiagretinimai, siekiant sumažinti algoritmo skaičiavimo trukmę. Atlikus eksperimentus su realizuotu VQLS-SVM algoritmu, gauti tokie rezultatai:

1. Gauti rezultatai ištyrus klasikinį SVM ir kvantinį VQLS-SVM algoritmus naudojantis irisų ir krūties auglių duomenų rinkiniais, siekiant palyginti klasikinio ir kvantinio algoritmų tikslumus;
2. ištirtas dimensijų mažinimo poveikis VQLS-SVM algoritmo klasifikavimo tikslumui;
3. ištirtas VQLS-SVM algoritmo tikslumo kitimas su skirtingomis kubitų reikšmėmis.
4. gauti rezultatai atlikus VQLS-SVM algoritmo tyrimą su skirtingomis gradientinėmis ir negradientinėmis optimizavimo funkcijomis;
5. ištirtas algoritmo efektyvumas su skirtingais gijų ir darbo dydžio reikšmėmis.

Iš šių rezultatų gautos tokios išvados:

1. VQLS-SVM algoritmas tiksliai klasifikuoja testavimo duomenų rinkinį, kai apmokymo duomenų aibė turi tik 7 įrašus ir yra padalinta [YSM+23] straipsnyje pristatytu būdu. Kvantinio atraminių vektorių klasifikatoriaus algoritmo tikslumas yra šiek tiek mažesnis lyginant su klasikiniu algoritmu. Gautas vektorius x nėra tikslus sprendinys, kas turi įtakos klasifikavimo tikslumui. Naudojant vienas prieš visus (*angl. one vs all*) metodą irisų duomenų aibei, klasifikavimo tikslumas sumažėjo, nes Iris Virginica ir Iris Versicolour yra gana panašios klasės, apmokant VQLS-SVM algoritmą, kur atrinkta viena iš šių klasių, klasifikavimo tikslumas yra mažesnis nei pasirinkus klasifikuoti tarp Iris Setosa ir kitų irisų klasių.
2. Kaip žinoma iš klasikinės mašininio mokymo teorijos, atlikus VQLS-SVM algoritmo tyrimą, buvo pastebėta, kad požymių skaičiui esant didesniam negu apmokymo duomenų aibė, klasifikavimo tikslumas yra mažesnis lyginant su duomenų rinkiniu, kurio požymių skaičius mažesnis už apmokymo aibę. Klasifikavimo tikslumui padidinti ir šiai problemai išspręsti galima pasirinkti didesnį apmokymo duomenų rinkinį.
3. Atlikus VQLS-SVM algoritmo patobulinimus ir išplėtus grandines iki 10 kubitų, buvo atlikti eksperimentai su kubitų reikšmėmis lygioms 4 ir 5 (apmokymo duomenų aibės dydžiai atitinkamai 15 ir 31), siekiant išsiaiškinti ar VQLS-SVM algoritmas tiksliau klasifikuoja su didesnėmis duomenų aibėmis. Krūties auglių duomenų aibės atveju, klasifikavimo tikslumas pakilo nuo 55,43 iki 70,34 procentų, padidinus iki 4 kubitų ir iki 79,72, 5 kubitų atveju. Irisų duomenų aibės atveju, padidinus VQLS-SVM algoritmo kubitų skaičių iki 5 ir apmokant su 31 elementų dydžio duomenų aibe, klasifikavimo

tikslumas nežymiai padidėjo nuo 92,26 iki 98,36 procentų. Tolimesnis kubitų didinimo žingsnis irisų duomenų aibei, gali sukelti VQLS-SVM modelio persimokymą.

4. *COBYLA* optimizatorius, iš tirtų išvestinių nereikalaujančių optimizatorių, greičiausiai pasiekė nuostolių funkcijos minimalią reikšmę. Optimizatoriaus dėka, jau po 26 iteracijų nuostolių funkcijos reikšmė yra mažesnė nei 0,1. Tačiau *SQLSP* optimizavimo funkcija taip pat gali būti efektyvi alternatyva, ypač atvejais, kai reikalingas didesnis skaičiavimų stabilumas, nes naudojant šį algoritmą, nuostolių funkcijos reikšmė didėjant iteracijų skaičiui, nedidėjo. Tiriant gradientinius optimizatorius, ADAM optimizavimo funkcija geriausiai pasirodė su mokymosi greičio reikšme lygiai 0,5, o gradientinio nusileidimo funkcija su $\alpha = 0.9$. Tolimesni mokymosi greičio reikšmių didinimai gali sukelti mokymosi proceso nestabilumą ir netgi nuostolių funkcijos divergavimą. Lyginant geriausius gradientinius ir negradientinius optimizatorius VQLS-SVM problemai, *COBYLA* pirmauja, antrąją vietą dalinosi *ADAM* ir *SLSQP* algoritmai.
5. Buvo atlikti VQLS-SVM algoritmo patobulinimai, išplečiant nuostolių funkcijos reikšmių ir grandinių konstravimo, transpiliavimo programinį kodą su galimybe naudoti keletą gijų. Eksperimentų rezultatai parodė, kad didžiausias VQLS-SVM algoritmo pagreitėjimas buvo su 10 gijomis ir darbo dydžiu lygiu 4 ir 8 gijomis ir darbo dydžiu lygiu 4 naudojantis MacBook M2 Pro, o Google Colab Pro aplinkoje su 2 gijomis ir darbo dydžiu lygiu 2. Tačiau algoritmo pagreitėjimas yra nežymus, tik 11 procentų pagreitėjimas lyginant su algoritmo veikimu naudojant vieną giją. Šie rezultatai rodo, kad naudojant keletą gijų sprendžiant VQLS-SVM problemą, tai nėra sprendimas stipriai paspartinantis algoritmo veikimą. Galimi tolimesni optimizavimai, kitose algoritmo srityse.

Atsižvelgiant į šiame darbe atliktus eksperimentus su VQLS-SVM algoritmu, jų rezultatais, ateityje būtų galima atlikti šiuos pakeitimus ir eksperimentus:

1. Atlikti tyrimus su VQLS-SVM algoritmu iki 10 kubitų, su įvairesnėmis, didesnėmis duomenų aibėmis: vyno, skaitmenų [AF91] [08]. Tokie eksperimentai suteiktų galimybę patikrinti algoritmo veikimą su didesnio masto duomenų apdorojimo atveju.
2. Didinant kubitų skaičių atliekant eksperimentus su VQLS-SVM algoritmu, didžiąją algoritmo dalį užėmė grandinių paruošimas, parametrizuotų grandinių transpiliavimas. Pennylane karkasas yra alternatyva IBM Qiskit, daug žadanti dėl efektyvaus grandinių tvarkymo ir jų skaičiavimo [BIS+22]. Galima atlikti IBM Qiskit ir Pennylane karkasų tyrimą, siekiant rasti kiekvieno iš šių karkasų pranašumus ir trūkumus sprendžiant VQLS-SVM problemą.
3. Jaehoon Hahm parašytame straipsnyje apie variacinių kvantinių algoritmų patobulinimus atliekant matavimo supaprastinimus (*angl. Improvement in variational quantum algorithms by measurement simplification*) pastebėta, kad pritaikius matavimo supaprastinimus, VQLS algoritmas pagreitėjo 184 kartus [HKP23]. Dabartinė VQLS-SVM algoritmo realizacija 5 kubitų atveju trunka 10 valandų. Atlikus straipsnyje pristatytus optimizavimus, teoriškai algoritmas 5 kubitų atveju galėtų trukti iki 3 minučių 15 sekundžių. Atlikus

šiuos patobulinimus, tolimesnis tyrimas su grandinių dydžiu didesniu nei 5 kubitai, būtų įmanomas.

4. Aruto Hosaka straipsnyje pristatyti būdai apdoroti apmokymo aibės matricą iš anksto (*angl. preconditioning*) siekiant sumažinti *ansatz* vartų sudėtingumą ir sluoksnių skaičių [HYK+24]. Straipsnyje sėkmingai ištirta su kubitų skaičiumi lygiu 8. Tyrimas rodo daug žadančių rezultatų, nes yra itin svarbu sumažinti grandinių sudėtingumą naudojantis NISQ kvantiniais kompiuteriais, padidinant atsparumą išoriniam triukšmui. Taip pat, pritaikius straipsnyje minėtus patobulinimus, tikėtinas VQLS-SVM algoritmo grandinių skaičiavimo pagreitėjimas.
5. Abeynaya Gnanasekaran straipsnyje pristatytas efektyvus VQLS algoritmas išretintoms matricoms (*angle. Efficient Variational Quantum Linear Solver for Structured Sparse Matrices*) [GS24]. Naudojant tiesine unitarinių matricų kombinacijos su specialia baze, kuri išnaudoja išretintos matricos savybes, galima sumažinti išteklių skaičių ir pagerinti VQLS-SVM algoritmo skaičiavimo efektyvumą ir trukmę.

Šaltiniai

[08] *Semeion Handwritten Digit* [UCI Machine Learning Repository]. 2008. DOI: https://doi.org/10.24432/C5SC8V.

[AF91] S. Aeberhard, M. Forina. *Wine* [UCI Machine Learning Repository]. 1991. DOI: https://doi.org/10.24432/C5PC7J.

[Ben80] P. Benioff. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics*. 1980, tomas 22, numeris 5, p. 563–591. Prieiga per internetą: https://doi.org/10.1007/BF01011339.

[BIS+22] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin ir kiti. *PennyLane: Automatic differentiation of hybrid quantum-classical computations*. 2022.

[BLC+23] C. Bravo-Prieto, R. LaRose, M. Cerezo, Y. Subasi, L. Cincio, P. J. Coles. Variational Quantum Linear Solver. *Quantum*. 2023, tomas 7, p. 1188. issn 2521-327X. Prieiga per internetą: https://doi.org/10.22331/q-2023-11-22-1188.

[CV22] D. Camps, R. Van Beeumen. FABLE: Fast Approximate Quantum Circuits for Block-Encodings. Iš: *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, 2022. Prieiga per internetą: https://doi.org/10.1109/qce53715.2022.00029.

[CV95] C. Cortes, V. Vapnik. Support Vector Networks. *Machine Learning*. 1995, tomas 20, p. 273–297.

[Deu85] D. Deutsch. Quantum theory, the Church-Turing principle and the universal quantum computer. *Proceedings of the Royal Society of London Series A*. 1985, tomas 400, numeris 1818, p. 97–117. Prieiga per internetą: https://doi.org/10.1098/rspa.1985.0070.

[DJ92] D. Deutsch, R. Jozsa. *Rapid Solution of Problems by Quantum Computation*. GBR: University of Bristol, 1992. Techninė ataskaita.

[DL23] Z. Ding, L. Lin. Even Shorter Quantum Circuit for Phase Estimation on Early Fault-Tolerant Quantum Computers with Applications to Ground-State Energy Estimation. *PRX Quantum*. 2023, tomas 4, numeris 2. issn 2691-3399. Prieiga per internetą: https://doi.org/10.1103/prxquantum.4.020331.

[EN88] A. S. Eddington, J. R. Newman. The world of mathematics, 4 vols. Iš: Second. Chichester: Tempus, 1988, p. 1074–1093.

[Fis88] R. A. Fisher. *Iris* [https://doi.org/10.24432/C56C76]. 1988.

[Gro96] L. K. Grover. A Fast Quantum Mechanical Algorithm for Database Search. Iš: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, p. 212–219. STOC '96. isbn 0897917855. Prieiga per internetą: https://doi.org/10.1145/237814.237866.

[GS24] A. Gnanasekaran, A. Surana. *Efficient Variational Quantum Linear Solver for Structured Sparse Matrices*. 2024.

[HBG23] L. Hantzko, L. Binkowski, S. Gupta. *Tensorized Pauli decomposition algorithm*. 2023.

[HHL09] A. W. Harrow, A. Hassidim, S. Lloyd. Quantum Algorithm for Linear Systems of Equations. *Physical Review Letters*. 2009, tomas 103, numeris 15. issn 1079-7114. Prieiga per internetą: https://doi.org/10.1103/physrevlett.103.150502.

[HYK+24] A. Hosaka, K. Yanagisawa, S. Koshikawa, I. Kudo, X. Alifu, T. Yoshida. *Preconditioning for a Variational Quantum Linear Solver*. 2024.

[HKP23] J. Hahm, H. Kim, Y. J. Park. *Improvement in Variational Quantum Algorithms by Measurement Simplification*. 2023.

[YSM+23] J. Yi, K. Suresh, A. Moghiseh, N. Wehn. *Variational Quantum Linear Solver enhanced Quantum Support Vector Machine*. 2023.

[KA23] M. Kashif, S. Al-Kuwari. The unified effect of data encoding, ansatz expressibility and entanglement on the trainability of HQNNs. *International Journal of Parallel, Emergent and Distributed Systems*. 2023, tomas 38, numeris 5, p. 362–400. issn 1744-5779. Prieiga per internetą: https://doi.org/10.1080/17445760.2023.2231163.

[Mar19] A. Mari. *Coherent Variational Quantum Linear Solver* [https://pennylane.ai/qml/demos/tutorial_coherent_vqls/]. Xanadu, 2019. Date Accessed: 2024-01-01.

[Qis23] Qiskit contributors. *Qiskit: An Open-source Framework for Quantum Computing*. 2023. Prieiga per internetą: https://doi.org/10.5281/zenodo.2573505.

[Sho99] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review*. 1999, tomas 41, numeris 2, p. 303–332. Prieiga per internetą: https://doi.org/10.1137/S0036144598347011.

[Sim97] D. R. Simon. On the Power of Quantum Computation. *SIAM Journal on Computing*. 1997, tomas 26, numeris 5, p. 1474–1483. Prieiga per internetą: https://doi.org/10.1137/S0097539796298637.

[SV99] J. Suykens, J. Vandewalle. Least Squares Support Vector Machine Classifiers. *Neural Processing Letters*. 1999, tomas 9, p. 293–300. Prieiga per internetą: https://doi.org/10.1023/A:1018628609742.

[WS95] S. N. Wolberg William Mangasarian Olvi, Street.W. *Breast Cancer Wisconsin (Diagnostic)* [https://doi.org/10.24432/C5DW2B]. 1995.

[WZP18] L. Wossnig, Z. Zhao, A. Prakash. Quantum Linear System Algorithm for Dense Matrices. *Physical Review Letters*. 2018, tomas 120, numeris 5. issn 1079-7114. Prieiga per internetą: https://doi.org/10.1103/physrevlett.120.050502.