LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

PARAMETRIZĒTU ALGORITMU PAĀTRINĀJUMI KVANTU DATORAM

BAKALAURA DARBS

Autors: **Aleksejs Jeļisejevs**
Studenta apliecības Nr.: aj17052
Darba vadītājs: doc., Dr.dat. Jevgēnijs Vihrovs

RĪGA, 2021

Anotācija

Darbā tiek aplūkots, kā uz kvantu datora iespējams paātrināt zināmus parametrizētus algoritmus problēmām "Closest string", "Cluster vertex deletion", "Cluster editing", "Vertex cover" un "Longest path" problēmu risinājumiem, ka arī to uzlabošana kvantu datoram, izmantojot kvantu algoritmus meklēšanas koka apstaigāšanai un dinamiskai programmēšanai.

Ir parādīts, kā izveidot kvantu algoritmus "Cluster vertex deletion" problēmas risinājumam ar laika sarežģītību $O(1.7321^k \sqrt{k}\, n^3)$ un "Vertex cover" problēmai ar laiku $O(1.175^k\, k^{O(1)} + n\sqrt{m})$, kas ir labāk, nekā labākajiem zināmajiem algoritmiem ar laiku $O(1.9102^k(n+m))$ un $O(1.2738^k + kn)$, attiecīgi. Ka arī ir parādīts, kā izveidot kvantu algoritmus "Closest string" problēmai ar laiku $O(kL + \mathrm{k}^2 d^{3/2}\sqrt{(\mathrm{d}+1)^d})$,"Cluster editing" problēmai ar laiku $O(1.7321^k \sqrt{k}\, n^3)$ un "Longest path" problēmai ar laiku $O(\left(1.7274\sqrt{e}\right)^k n^{O(1)})$, kas nav labāk, nekā ātrākiem pazīstamiem algoritmiem ar laiku $O(Lk + kd(|\Sigma|-1)^d \cdot 2^{3.25d}), O(1.62^k + m + n))$ un $O(2^k \cdot poly(n,k))$, attiecīgi.

Atslēgvārdi: Closest string, Cluster vertex deletion, Cluster editing, Vertex cover, Longest path, kvantu algoritmi, parametrizēti algoritmi.

Abstract

PARAMETRIZED ALGORITHMS SPEED UP FOR QUANTUM COMPUTERS

In this work few parametrized algorithms for solving “Closest string”, “Cluster vertex deletion”, “Cluster editing”, “Vertex cover” and “Longest path” problems are described, as well as their improvements for quantum computer, using quantum backtracking algorithm for solution three and quantum algorithm for dynamic programming.

There are shown, how to create quantum algorithms for “Vertex cover” problem solving with time complexity $O(1.175^k\ k^{O(1)} + n\sqrt{m})$ and for “Cluster vertex deletion” with time $O(1.7321^k\sqrt{k}\ n^3)$, improving best known classical algorithms with time $O(1.2738^k + \ kn)$ and $O(1.9102^k(n+m))$ respectively. Also, there are shown, how to create quantum algorithms for “Closest string” with time $O(kL + \ \mathrm{k}^2 d^{3/2}\sqrt{(\mathrm{d}+1)^d})$, for “Cluster editing” with time $O(1.7321^k\sqrt{k}\ n^3)$ and for “Longest path” with time $O(\left(1.7274\sqrt{e}\right)^k n^{O(1)})$, comparing to best known classical algorithms with time $O(Lk + kd(|\Sigma|-1)^d \cdot 2^{3.25d})$, $(1.62^k + m + n))$ and $O(2^k \cdot poly(n,k))$ respectively.

Keywords: Closest string, Cluster vertex deletion, Cluster editing, Vertex cover, Longest path, quantum algorithms, parametrized algorithms, backtracking.

SATURS

IEVADS

"Closest string", "Cluster vertex deletion", "Cluster editing", "Vertex cover" un "Longest path" ir pazīstamas NP-pilnas problēmas, līdz ar to viņiem nav zināms efektīvs risinājums. Viens no veidiem, ka uzkonstruēt efektīvu algoritmu, ir izmantot parametrizētus algoritmus, kuri izmanto papildus parametru k, kurš ierobežo doto problēmu. Par parametrizētu algoritmu sauksim algoritmu ar izpildes laiku $f(k) * n^c$, kur c ir konstante, neatkarīga no k un n. Tādu algoritmu labums ir, ka risinājuma laiks polinomiāli atkarīgs no n. Darbā tiek aprakstīti parametrizēti algoritmi "Closest string", "Cluster vertex deletion", "Cluster editing" un "Vertex cover" problēmām, kuri savā darba laikā konstruē meklēšanas kokus ar izmēru $f(k)$, ka arī parametrizēts algoritms "Longest path" problēmai, kurš izmanto dinamisku programmēšanu. Pagaidam vēl nav daudz rezultātu par kvantu parametrizētajiem algoritmiem, tāpēc tas ir liels lauks pētīšanai.

Liela kvantu datoru priekšrocība ir tas, ka vairākie algoritmi uz tiem strādā ar labāku laika sarežģitību, neka klasiskiem datoriem. Viens no pazīstamākajiem kvantu algoritmiem ir Grovera algoritms meklešanai datubāzē[6]. Algoritmus, kuri izmanto meklēšanās kokus, ir iespējams būtiski paātrināt, izmantojot kvantu algoritmu [2], kurš apstrādā meklēšanas koku laikā $\tilde{O}(\sqrt{T}n^{3/2})$. Šīs algoritms palīdzēs samazināt $f(k)$ eksponenciālo funkciju. Dinamisku programmēšanu arī ir iespējams kvantiski paātrināt [4]. Darbs ir fokusēts uz eksponentfunkcijas bāzes samazināšanu, lai minimizētu algoritmu laika sarežģītību. Atmiņas pieprasījums algoritmiem darbā ietvaros netiek apskatīts.

Katrai problēmai ir norādīta tā definīcija, pēc tam ir aprakstīti klasiskie algoritmi risinājumam. Beigās ir aprakstīts, kā pielāgot klasisko algoritmu kvantu uzlabojumam.

1. INSTRUMENTI

1.1. Parametrizēti algoritmi

Parametrizēts algoritms – ir algoritms ar darba laiku $f(k) \cdot n^c$, kur c ir konstante, neatkarīga no n un k. Svarīga īpašība tadiem algoritmiem ir, ka viņu laika sarežģītība ir polinomiāli atkarīga no n. Parametrizētus algoritmus meklē tad, kad vispārīgā gadījumā labākais zināmais atrisinājums nav polinomiāla laikā.

1.2. Koka apstaigāšanas kvantu algoritms

Rakstā [2] tika atrasts kvantu algoritms, ar kura palīdzību ir iespējams uzlabot meklēšanas koka apstaigāšanas laiku, jeb paātrināt to. Kvantu algoritms var noteikt, vai meklēšanas grafā eksistē $true$ lapa izmantojot $O(\sqrt{T_1 n}\log\frac{1}{\epsilon})$ pieprasījumus melnas kastes funkcijām, kur T_1 un n ir meklēšanas koka izmērs un dziļums, bet $1-\epsilon$ ir pareizas atbildes varbūtība. Talāk darbā $\log\frac{1}{\epsilon}$ daļa nebūs norādīta laikā sarežģītībā, jo lielais O to paslēpj ka konstanti. Lai izmantot kvantu algoritmu, mums jānodrošina, ka:

1) Ir pieejama sakne s meklēšanas kokam τ.
2) Ir pieejama melnas kastes funkcija, kura, saņemot virsotni v, atgriež tās bērnu skaitu $d(v)$.
3) Ir pieejama melnas kastes funkcija, kura, saņemot virsotni v un $i \in [d(v)]$, atgriež virsotnes v i-to bērnu.
4) Visas virsotnes kokā atbilst daļējam uzdevuma atrisinājumām.
5) Koka sakne atbilst tukšam risinājumam.
6) Virsotnes v_x bērni v_y atbilst iespējamam daļēja risinājuma x paplašinājumiem y, kurus meklēšanas algoritms var pārbaudīt, tajā secībā, kurā algoritms tas, pārbaudīs.

Turklāt mums ir jādefinē melnas kastes funkcija, kura katrai meklēšanas koka virsotnei:

1) Atgriež $true$, ja virsotne ir meklēšanas koka lapa kura atbilst problēmas veiksmīgam atrisinājumam.
2) Atgriež $false$, ja virstne ir meklēšanas koka lapa, kura neatbilst problēmas veiksmīgam atrisinājumam.
3) Atgriež $indeterminate$ citādi.

1.3. Kodolizācija

Kodolizācija (kernelization) – ir process, kurš orientēts uz sākotnējas problēmas samazināšanu, izmantojot dažādas redukcijas uzdevuma datu kopai. Kodolizācijas laikā risinās problēmas "vieglāka" daļa, lai pēc tām reducēt to uz sarežģītu problēmu – "kodolu". Gandrīz visiem darba aprakstītām problēmam mēs grībām paša sākumā, polinomiālā laikā samazināt virsotņu un šķautņu skaitu grafam un tikai tad risināt problēmu ar eksponenciālo laiku, izmantojot risinājuma meklēšanas koku.

1.4. Grovera meklēšana

[6] Ir dots masīvs $x_1, x_2, x_3, \dots x_n \in \{0,1\}$ un melnā kaste, kura, pavaicājot indeksu i, atgriež x_i vērtību. Grovera meklēšana ir kvantu algoritms, kas atrod indeksu i, kuram $x_i = 1$ vai pasaka, ka tāds elements neeksistē. Funkcijas laika sarežģītība ir $O(\sqrt{n})$.

1.5. Amplitūdas amplifikācija

[7] Pieņēmsim, ka mums ir varbūtisks algoritms, kas atgriež 0, kad atbilde ir 0 un, ar varbūtību p atgriež 1, kad atbilde ir 1. Var pierādīt, ka, lai dabūtu pareizo atbildi, pietiek atkārtot algoritmu vidēji $\frac{1}{p}$ reizes. Amplitūdas amplifikācija pārveido to par kvantu algoritmu, kuram nepieciešami tikai $O(\frac{1}{\sqrt{p}})$ atkārtojumi.

2. IEPRIEKŠĒJIE REZULTĀTI

Šajā nodaļā aprakstītas bakalaura darba autora iepriekšējie rezultāti, kas iegūti kursa darba ietvaros [5].

2.1. "Vertex cover" problēma

Mums dots grafs G ar n virsotnēm un m šķautnēm. Kopa $S \subseteq \mathrm{V(G)}$ tiek saukta par "vertex cover", ja katrai grafa G šķautnei vismaz viens galapunkts atrodas kopā S.
"K-vertex cover" problēmā mums ir dots grafs G un vesels skaitlis k, un uzdevums ir noteikt, vai eksistē "vertex cover" ar izmēru ne lielāku par k.

Kursa darbā aprakstīti trīs klasiskie algoritmi [1], kuri rīsina šo problēmu laikā $O(2^k \cdot k \cdot n), O\left(1.6181^k \cdot k^2 + n\sqrt{m}\right)$ un $O\left(1.4656^k \cdot k^2 + n\sqrt{m}\right)$. Visi algoritmi ir parametrizēti pēc k un izmanto risinājuma meklēšanas kokus. Ar punktā 1.2. noradīto kvantu algoritma palīdzību bija iegūti kvantu algoritmi ar laika sarežģītību $O(\sqrt{2}^k\ k^{3/2}\ n)$, $O(1.272^k\ k^{5/2} + n\sqrt{m})$ un $O(1.211^k\ k^{5/2} + n\sqrt{m})$. Pēdējo divu kvantu algoritmu laika sarežģītība ir labāka, nekā ātrākam pazīstamām algoritmam ar laiku $O(1.2738^k + kn)$[3].

2.2. "Feedback vertex set" problēma

Neorientētam grafam G kopu $X \subseteq \mathrm{V(G)}$ sauksim par "feedback vertex set", ja $G - X$ ir aciklisks grafs. "Feedback vertex set" uzdevums ir noteikt, vai grafam G eksistē "feedback vertex set" ar izmēru ne vairāk par k. Kursa darbā aprakstīts klasiskais algoritms[1], kurš rīsina šo problēmu laikā $O((3k)^k \cdot\ n^{O(1)})$ un izmanto meklēšanas koku ar izmēru $O((3k)^k)$. Ar punktā 1.2. noradīto kvantu algoritma palīdzību laika eksponentes daļa tika samāzināta līdz $O\left((\sqrt{3k})^k\right)$, un iegūts kvantu algoritms ar laika sarežģītību $O((\sqrt{3k})^k\ n^{O(1)} \log\frac{1}{\epsilon})$.

3. “CLOSEST STRING” PROBLĒMA

3.1. Problēmas apraksts

Mums ir dots alfabets Σ, k simbolu virknes $x_1, \dots, x_k$, katra virkne ar garumu L un piedēr Σ^L, ka arī vesels skaitlis d. Uzdevums ir noteikt, vai eksistē tāda virkne $y \in \Sigma^L$, ka $d_H(y, x_i) \leq d$ visiem $i \in \{1, \dots, k\}$. Šeit $d_H(x, y)$ ir Heminga attālums starp virknem x un y, jeb poziciju skaits, kur x atšķīras no y. Sauksim jebkuru tadu virkni y par centrālo virkni. Labākais pazīstamais klasiskais algoritms[12] atrisina “closest string” laikā $O(Lk + kd(|\Sigma| - 1)^d \cdot 2^{3.25d})$.

3.2. Kodolizācija

[1]Apzimēsim virknes x p-to simbolu ka $x[p]$. Līdz ar to $x = x[1]x[2] \dots x[L]$ virknei ar garumu L. Virknes x un y atšķiras p-ta pozīcijā, ja $x[p] \neq y[p]$. Mums ir dotas k virknes, katra ar garumu L, līdz ar to mēs varam domāt par tiem kā par $k \times L$ simbolu matricu. Tas nozīme, ka j-ta kolonna sastāv no simboliem $x_1[j], x_2[j], \dots, x_k[j]$. Sauksim kolonnu par sliktu, ja tā satur vismāz divus dažādus simbolus no alfabeta Σ, un sauksim kolonnu par labu pretējā gadījumā. Acīmredzams, ja j-ta kolonna ir laba, tad mums ir j-tais simbols atbildei un $y[j] = x_1[j] = x_2[j] = \dots = x_k[j]$. Līdz ar to mums ir pirmais kodolizācijas solis – noņēmam visas labas kolonnas. To ir iespējams izdarīt laikā $O(kL)$.

Ja uzdevumam ir atrisinājums, tad atbilstoša $k \times L$ simbolu matrica saturēs ne vairāk kā kd sliktas kolonnas [1: Lemma 3.13]. Otrais kodolizācijas solis – ja matrica satūr vairāk nekā kd sliktas kolonnas, tad atrisinājuma nav.

3.3. Klasiskais algoritms

Eksistē klasiskais algoritms [1], kurš atrisina “Closest string” problēmu laikā $O(kL + kd(d + 1)^d)$ [1: Theorem 3.14.]. Pieņēmsim, ka centrāla virkne y eksistē. Algoritma ideja ir sākt ar vienu no k virknem, piemēram ar x_1, kura būs mūsu “kandidātvirkne”. Apzīmēsim to ar z. Kamēr eksistē virkne x_i, $i \in \{1, \dots, k\}$, kurai izpildas $d_H(x_i, z) \geq d$+1, vismaz vienai no pozicijam p, kur z un x_i atšķīras, būs $y[p] = x_i[p]$. Tāpēc mēs varam rekursīvi pārbaudīt jebkuras $d + 1$ iespējas pietuvināt “kandidātvirkni” z virknei y. Tas ir, izvelēties pozīciju p, kur virkne z atšķīras no x_i un pieškīrt $z[p] := x_i[p]$. Ar katru soli mēs tuvojamies virknei y, un no sākuma $z = x_1$ un $d_H(y, x_1) \leq d$, līdz ar to mēs dabūjam ierobežotu variantu skaitu.

Klasiskam algoritmam ir sekojoši soļi:

No sākuma mēs pielietojam kodolizāciju. Ja uzdevums vel nav atrisināts, tad mēs dabūjam k virknes $x_1, x_2, \dots, x_k$, katra ar garumu $L \leq kd$. Šīs solis pieprāsa laiku $O(kL)$.

Uzkonstruēsim funkciju, kurai kā parametrus padodam "kandidātvirkni" z un veselo skaitli $l \leq d$. Tas ir rekursīva funkcija, kura veidos atrisinājuma meklēšanas koku. Vispirms funkcija pārbauda, vai z jau ir centrāla virkne laikā $O(k^2 d)$. Ja ir, tad funkcija atgriež pozitīvu rezultātu. Citādi, ja $l = 0$, tad funkcija atgriež negatīvu rezultātu, jo šīs rekursijas zars neatroda centrālo virkni y. Atlikušā gadījumā $l > 0$ un eksistē x_i, kuram $d_H(x_i, z) \geq d + 1$. Kopa D sastāv no jebkurām $d + 1$ pozīcijam, kurās x_i un z atšķīras. Tad funkcija rekursīvi sazarojas $d + 1$ gadījumos: katram $p \in D$ nodefinēsim $z_p = z$, izņēmot pozīciju p, kur $z_p[p] = x_i[p]$, un palaižam funkciju visiem parametriem $(z_p, l - 1)$. Ja kaut viens rekursijas zars atgriež pozitīvu rezultātu, tad arī paša funkcija atgriež pozitīvu rezultātu. Citādi, funkcija atgriež negatīvu rezultātu.

Pašā sākuma palaižam algoritmu ar parametriem $z = x_1$ un $l = d$. Rekursijas zari veido meklēšanas koku τ, ar maksimālo dziļumu d, ka arī katrai koka virsotnei ir ne vairāk kā $d + 1$ bērni. Līdz ar to, maksimālais meklēšanas koka τ izmērs T ir vienāds ar $(\mathrm{d} + 1)^d$. Kopēja algoritma laika sarežģītība ir $O(kL + \mathrm{k}^2 d(d + 1)^d)$. Avotā [1: Theorem 3.14.] ir norādīts, ka šo algoritma laiku ir iespējams optimizēt līdz $O(kL + kd(d + 1)^d)$, tomēr tas nav svarīgi, jo mūs interesē tikai eksponentes daļa.

3.4. Kvantu uzlabojums

Pielāgosim klasisko algoritmu punktā 1.2. aprakstītam kvantu algoritmam. Kvantu algoritms apstaigās tādu pašu meklēšanas koku τ. Katra virsotne tajā būs aprakstīta ar "kandidātvirkni" z un veselo skaitli l. Mums ir funkcija $F((z, l))$, risinājuma pārbaudei, kura strādā līdzīgi klasiskam algoritmam aprakstītai, jeb pārbauda, vai z ir centrāla virkne. Ja ir, tad $F((z, l))$ atgriež tukšu kopu, citādi atgriež kopu ar $d + 1$ jauniem "kandidātvirknem", katra atšķīras no z tikai vienā pozicijā. Funkcijas laika sarežģītībā ir $O(\mathrm{k}^2 d)$. Šādu funkciju mēs varam izmantot gan koka apstaigāšanai, gan risinājuma pārbaudei. Sanāca, kā:

- Sakne s kokam τ ir paris (x_1, d).
- Funkcija $F((z, l) = v)$ ir melnas kastes funkcija un $|F((z, l) = v)|$ ir koka virsotnes bērnu skaits $d(v)$.

- Funkcija $F((z,l)=v)$ ir melnas kastes funkcija, kura atgriež 0 vai $d+1$ elementus, līdz ar to, virsotnes v i-tais bērns ir $((F(v)[i], l-1)$ kur $[i]$ apzīmē i-to masīva elementu.

Ka arī mums ir nepieciešama melnas kastes funkcija P risinājuma pārbaudei. Šeit arī varam izmantot funkciju F. P saņemot virsotni $((z,l)=v)$ pārbaudei palaiž $F((z,l))$:

1) Ja $|F((z,l))| = 0$ un $l \geq k$, tad risinājums ir atrasts un P atgriež $true$.

2) Ja $|F((z,l))| = d+1$ un $l = 0$, tad šis zars nesatur risinājuma un P atgriež $false$.

3) Citādi P atgriež $indeterminate$.

No 3.3. dabūjam, ka koka τ maksimālais izmērs $T_1 = (\mathrm{d}+1)^d$ un dziļums $= d$. Apvienojot visu kopā, mēs varam izveidot kvantu uzlabojumu klasiskām algoritmam. Kopējā laika sarežģītība ir $O(kL + \mathrm{k}^2 d\sqrt{T_1 d})$ = $O(kL + \mathrm{k}^2 d^{3/2}\sqrt{(\mathrm{d}+1)^d})$. Mums sanāca samāzināt laika eksponentes daļu no $O((\mathrm{d}+1)^d)$ līdz $O((\sqrt{\mathrm{d}+1})^d)$.

4. “CLUSTER VERTEX DELETION” PROBLĒMA

4.1. Problēmas apraksts

Mums dots neorientēts grafs G ar n virsotnēm un m šķautnēm, ka arī pozitīvs vesels skaitlis k. Uzdevums ir noteikt, vai ir iespējams nodzēst no grafa ne vairāk kā k virsotnes, lai atlikušas virsotnes veidotu klasteru, jeb atdalītu kliķu kopu. Labākais pazīstamais klasiskais algoritms[8] atrisina “cluster vertex deletion” laikā $O(1.9102^k(n+m))$.

4.2. Kodolizācija

No paša sākuma visas virsotnes ar pakāpem 0 jau veido 1-kliķes, līdz ar to varam noņēmt tās no G. To ir iespējams izdarīt laikā $O(n)$. Ka arī mums vajag noņemt no grafa visas 2-kliķes, jeb virsotnes ar pakāpi 1, kuras ir savienotas viena ar otru. Lai to izdarīt, varam iterēt caur visam virsotnem ar pakāpi 1 un pārbaudīt, vai vienīga kaimiņa pakāpe arī ir 1. To arī ir iespējams izdarīt laikā $O(n)$.

4.3. Klasiskais algoritms

Mūs interesē parametrizēts algoritms, kura laika sarežģītība ir polinomiāli atkarīga no n un m. Līdz ar to, vairāki soļi klasiskām algoritmam var būt neoptimizēti, kamēr viņi neietekmē meklēšanas koka izmēru. No sākuma, pielietosim kodolizāciju, lai noņēmt visas 1-kliķes un 2-kliķes. Kā uzzināt, vai virsotne piedēr lielākai kliķei? Virsotne v ar pakāpi $d(v) \geq 2$ piedēr kliķei tad un tikai tad, ja visie tā kaimiņi ir pa pāriem savienoti, un katram virsotnes v kaimiņam v_x izpildās $d(v) = d(v_x)$. Vienai virsotnei vajag pārbaudīt ne vairāk kā $O(n^2)$ kaimiņu parīšus. Visam grafam nepieciešams ne vairāk kā n tādas pārbaudes. Kopā sanāca laiks $O(n^3)$, jeb $O(poly(n))$. No tā seko vel viena svarīga īpašība: katra grafa komponente vai nu ir kliķe, vai tajā var atrast virsotni, kurai ir 2 nesavienoti kaimiņi. Tādu virsotni arī varam atrāst ar naivu algoritmu laikā $O(n^3)$.

Tagad mēs varam izveidot algoritmu, kurš atrisina “Cluster vertex deletion” problēmu laikā $O(3^k \cdot n^3)$. No sākuma, pielietosim kodolizāciju, tad uzkonstruēsim funkciju, kurai kā parametru padodam kopu S ar tām grafa G virsotnēm, kuras jau ir noņemtas. Funkcija katrā solī meklē grafā virsotni v_x, kurai ir divi nesavienoti kaimiņi - virsotnes $v_y \; un \; v_z$, kuri nepieder kopai S. Viena kliķē nevar būt vienlaicīgi virsotnes v_x, $v_y \; un \; v_z$, vismaz vienu no tām vajag noņemt no grafa.

1. Ja tāda virsotnes eksistē un $|S| < k$, tad rekursīvi palaižam funkciju vispirms ar kopu $S \cup v_x$, tad ar kopu $S \cup v_y$, pēc tām ar kopu $S \cup v_z$. Ja kaut vienai no apakš funkcijām

būs pozitīvs rezultāts, tad arī pašai funkcijai ir pozitīvs rezultāts, jeb šīs rekursijas zars atrada problēmas risinājumu. Citādi rezultāts ir negatīvs.

2. Ja tāda virsotne eksistē un $|S| = k$, tad funkcijas rezultāts ir negatīvs.

3. Ja grafā nav tādas virsotne, jeb katra virsotne piedēr kaut kādai kliķei, tad funkcija atgriež pozitīvu rezultātu arī tad, ja līdz šīm bija noņemtas mazāk nekā k virsotnēs.

Palaižam to funkciju ar tukšo kopu, ja rezultāts ir pozitīvs, tad atrisinājums eksistē. Ja rezultāts ir negatīvs, tad atrisinājuma nav. Ja mēs grībam atrast pašas virsotnēs, kuras vajag noņemt, tad funkcijai, pozitīva rezultāta gadījumā, ir jāatgriež arī saraksts ar noņemtajām virsotnēm. Rekursijas zari veido meklēšanas koku τ, ar maksimālo dziļumu k. Maksimālais koka τ izmērs T ir vienāds ar 3^k. Katrai τ virsotnei ir nepieciešams laiks $O(n^3)$, lai atrastu virsotnes $v_z, v_x\ un\ v_y$, vai konstatēt, ka tādas neeksistē. Kopēja algoritma laika sarežģītība ir $O(n + \ 3^k \cdot n^3) = O(3^k \cdot n^3) = O(\ f(k) \cdot n^c)$.

4.4. Kvantu uzlabojums

Pielāgosim klasisko algoritmu kvantu algoritmam. Kvantu algoritms apstaigās tādu pašu meklēšanas koku τ. Katra virsotne tajā būs aprakstīta ar nodzēstu virsotņu kopu S. Mums ir funkcija $F(S)$ risinājuma pārbaudei, kura strādā līdzīgi klasiskam algoritmam aprakstītai, jeb atgriež trīs virsotnes v_x, v_y un v_z, kur v_y un v_z ir nesavienoti virsotnes v_x kaimiņi, un nepieder kopai S, vai atgriež tukšu kopu, ja tādas neeksistē. Funkcijas laika sarežģītībā ir $O(n^3)$. Šādu funkciju mēs varam izmantot gan koka apstaigāšanai, gan risinājuma pārbaudei. Sanāca, kā:

- Sakne s kokam τ ir tukša kopa, vai masīvs $\{\}$.
- Funkcija $F(S = v)$ ir melnas kastes funkcija un $|F(S = v)|$ ir koka virsotnes bērnu skaits $d(v)$.
- Funkcija $F(S = v)$ ir melnas kastes funkcija, kura atgriež 0 vai 3 elementus, līdz ar to, virsotnes $S = v$ i-tais bērns ir $v\ \cup F(S = v)[i]$, kur $[i]$ apzīmē i-to masīva elementu.

Ka arī mums ir nepieciešama melnas kastes funkcija P risinājuma pārbaudei. Šeit arī varam izmantot funkciju F. P saņemot virsotni $(S = v)$ pārbaudei palaiž $F(S = v)$:

1) Ja $|F(S = v)| =\ 0$ un $|S| \leq k$, tad risinājums ir atrasts un P atgriež $true$.

2) Ja $|F(S = v)| =\ 3$ un $|S| = k$, tad šis zars nesatur risinājuma un P atgriež $false$.

3) Citādi P atgriež $indeterminate$.

No 4.3. dabūjam, ka koka τ maksimālais izmērs ir $T_1 = 3^k$ un dziļums $= k$. Apvienojot visu kopā, mēs varam izveidot kvantu uzlabojumu klasiskām algoritmam. Kopējā sarežģītībā ir $O\left(\sqrt{T_1 k}\, n^3\right) = O(1.7321^k \sqrt{k}\, n^3)$. Mums sanāca samazināt laika eksponentes daļu no $O(3^k)$ līdz $O(1.7321^k)$.

5. “CLUSTER EDITING” PROBLĒMA

5.1. Problēmas apraksts

Mums dots neorientēts grafs G ar n virsotnēm un m šķautnēm, ka arī pozitīvs vesels skaitlis k. Uzdevums ir noteikt, vai ir iespējams sakoriģēt ne vairāk kā k grafa šķautnes, lai atlikušais grafs veidotu klasteru, jeb atdalītu kliķu kopu. Šeit katra koriģēšana ir vai nu pievienot grafam vienu šķautni, vai nodzēst vienu jau eksistējošo šķautni. Labākais pazīstamais klasiskais algoritms[9] atrisina “cluster editing” laikā $O(1.62^k + m + n))$.

5.2. Kodolizācija

“Cluster editing” prolēmai būs tada paša kodolizācija kā “Cluster vertex deletion” problēmai. Paša sākumā noņēmsim no grafa G visas 1-kliķes un 2-kliķes laikā $O(n)$. Kliķes noņēmšana no grafa neietekmēs rezultātu. Acīmredzāms, ka mums nav jēgas noņēmt šķautnes no jau izveidotas kliķes. Ka arī nav jēgas veidot jaunas šķautnes no citas grafa komponentes A uz kliķi K, jo ja $A \cup K$ veidos kliķi, tad arī paša A jau veido kliķi.

5.3. Klasiskais algoritms

Mūs interesē parametrizēts algoritms, kura laika sarežģītība ir polinomiāli atkarīga no n un m. Klasiskais algoritms būs līdzīgs “Cluster vertex deletion” problēmai aprakstītam. No sākuma pielietosim kodolizāciju, tad uzkonstruēsim funkciju, kurai kā parametru padodam kopu S ar tām grafa G virsotņu pāriem, starp kuriem jau bija šķautnes koriģēšana. Svarīgi, ka tadi parīši var atkārtoties, jeb viena šķautne var būt noņemta kāda laika pēc pievienošanas, vai otrādi. Funkcija katrā solī meklē ar kopu S koriģētā grafā virsotni v_x, kurai ir divi nesavienoti kaimiņi - virsotnes $v_y \; un \; v_z$. Viena kliķē nevar būt vienlaicīgi virsotnes $v_x, \; v_y \; un \; v_z$, līdz ar to mums ir trīs opcijas: noņēmt no grafa šķautni $(v_x, \; v_y)$, noņēmt no grafa šķautni $(v_x, \; v_z)$ vai pievienot šķautni $(v_y, \; v_z)$.

1. Ja tāda virsotnes eksistē un $|S| < k$, tad rekursīvi palaižam funkciju vispirms ar kopu $S \cup (v_x, v_y)$, tad ar kopu $S \cup (v_x, v_z)$, pēc tām ar kopu $S \cup (v_y, v_z)$. Ja kaut vienai no apakš funkcijām būs pozitīvs rezultāts, tad arī pašai funkcijai ir pozitīvs rezultāts, jeb šīs rekursijas zars atrada problēmas risinājumu. Citādi rezultāts ir negatīvs.

2. Ja tāda virsotne eksistē un $|S| = k$, tad funkcijas rezultāts ir negatīvs.

3. Ja grafā nav tādas virsotne, jeb katra virsotne pieder kaut kādai kliķei, tad funkcija atgriež pozitīvu rezultātu arī tad, ja līdz šīm bija noņemtas mazāk nekā k virsotnēs.

Ir garantēts, ka šis algoritms neieciklēsies, jo katrā rekursijas solī, kopas S izmērs palielinās par 1.

Palaižam to funkciju ar tukšo kopu, ja rezultāts ir pozitīvs, tad atrisinājums eksistē. Ja rezultāts ir negatīvs, tad atrisinājuma nav. Rekursijas zari veido meklēšanas koku τ, ar maksimālo dziļumu k. Maksimālais koka τ izmērs T ir vienāds ar 3^k. Katrai τ virsotnei ir nepieciešams laiks $O(n^3)$, lai atrastu virsotnes $v_z, v_x\ un\ v_y$, vai konstatēt, ka tādas neeksistē. Kopēja algoritma laika sarežģītība ir $O(n + 3^k n^3) = O(3^k \cdot n^3) = O(f(k) \cdot n^c)$.

5.4. Kvantu uzlabojums

Pielāgosim klasisko algoritmu kvantu algoritmam. Kvantu algoritms apstaigās tādu pašu meklēšanas koku τ. Katra virsotne tajā būs aprakstīta ar kopu S kura sastāv no tiem grafa G virsotņu pāriem, starp kuriem jau bija šķautnes koriģēšana. Mums ir funkcija $F(S)$, risinājuma pārbaudei, kura strādā līdzīgi klasiskam algoritmam aprakstītai, jeb atgriež trīs virsotņu parīšus $(v_x, v_y), (v_x, v_z)$ un (v_y, v_z), kur v_y un v_z ir nesavienoti virsotnes v_x kaimiņi, ja tādas neeksistē. Funkcijas laika sarežģītībā ir $O(n^3)$. Šādu funkciju mēs varam izmantot gan koka apstaigāšanai, gan risinājuma pārbaudei. Sanāca, kā:

- Sakne s kokam τ ir tukša kopa, vai masīvs $\{\}$.
- Funkcija $F(S = v)$ ir melnas kastes funkcija un $|F(S = v)|$ ir koka virsotnes bērnu skaits $d(v)$.
- Funkcija $F(S = v)$ ir melnas kastes funkcija, kura atgriež 0 vai 3 elementus, līdz ar to, virsotnes $S = v$ i-tais bērns ir $v\ \cup F(S = v)[i]$, kur $[i]$ apzīmē i-to masīva elementu.

Ka arī mums ir nepieciešama melnas kastes funkcija P risinājuma pārbaudei. Šeit arī varam izmantot funkciju F. P saņemot virsotni $(S = v)$ pārbaudei palaiž $F(S = v)$:

1) Ja $|F(S = v)| = 0$ un $|S| \leq k$, tad risinājums ir atrasts un P atgriež $true$.

2) Ja $|F(S = v)| = 3$ un $|S| = k$, tad šis zars nesatur risinājuma un P atgriež $false$.

3) Citādi P atgriež $indeterminate$.

No 5.3. dabūjam, ka koka τ maksimālais izmērs $T_1 = 3^k$ un dziļums $= k$. Apvienojot visu kopā, mēs varam izveidot kvantu uzlabojumu klasiskām algoritmam. Kopējā sarežģītībā ir $O\left(\sqrt{T_1 k} \cdot n^3\right) = O(1.7321^k \sqrt{k}\ n^3)$. Mums sanāca samazināt laika eksponentes daļu no $O(3^k)$ līdz $O(1.7321^k)$.

6. “VERTEX COVER” PROBLĒMA

6.1. Problēmas apraksts

Mums dots grafs G ar n virsotnēm un m šķautnēm. Kopa $S \subseteq \mathrm{V(G)}$ tiek saukta par “vertex cover”, ja katrai grafa G šķautnei vismaz viens galapunkts atrodas kopā S. “K-vertex cover” problēmā mums ir dots grafs G un vesels skaitlis k, un uzdevums ir noteikt, vai eksistē “vertex cover” ar izmēru, ne lielāku par k. Labākais pazīstamais klasiskais algoritms[3] atrisina “k-vertex cover” laikā $O(1.2738^k + kn)$.

6.2. Klasiskais algoritms ar meklēšanas koka izmēru $\mathbf{1.3803^k}$.

Kursa darba [5: 3.4.] aprakstīts algoritms[1], kurš atrīsina “k-vertex cover” problēmu un izmanto meklēšanas koku ar izmēru 1.4656^k. Sākumā notiek kodolizācijas process, algoritms laikā $O(n\sqrt{m})$ atrod kodolu – grafu G ar ne vairāk kā $2k$ virsotnēm [1: Theorem 2.21, Corollary 2.23.]. Rekursīva algoritma idēja: katra solī atrodam grafā virsotni v_m ar vislielāko pakāpi, tad vai nu noņēmam no grafa virsotni v_m, vai noņēmam visus virsotnes v_m kaimiņus $N(v_m)$. Gadījumā, kad maksimāla virsotnes pakāpe ir 1 vai 2, algoritms var neturpināt zarošanas procedūru, jo tādam grafam ir iespējams atrast “vertex cover” laikā $O(k^2)$. Citos gadījumos, pirmā rekursijas zarā skaitlis k samāzinās par 1, bet otrā vismāz par 3. Tas nozīme, ka meklēšanas koka zarošanas vektors ir (1, 3). Pēc zarošanas vektora varam aprēķināt maksimālo meklēšanas koka izmēru. Vektoram (1, 3) atbīlst meklēšanas koka izmērs 1.4656^k [1: 3.2.].

Eksistē klasiskais algorims [10], kurš atrīsina “k-vertex cover” problēmu grafam, ar virsotņu pakāpem ≤ 3 laikā $O(1.1616^k \cdot k^{O(1)})$. Izmantosim šo algoritmu rekursīvai procedūrai, lai apstrādāt gadījumu, kad v_m pakāpe ir vienāda ar 3, jo tas ir atrāk, neka turpināt rekursiju ar zarošanas vektoru (1, 3). Tad rekursiju turpinām tikai tad, kad v_m pakāpe ir ≥ 4, līdz ar to, uzlabotai procedūrai meklēšanas koka zarošanas vektors būs (1, 4) un meklēšanas koka τ izmērs T_1 sliktākā gadījumā būs $O(1.3803^k)$. Pats algorims strādā laikā $O(1.3803^k \cdot k^{O(1)})$.

6.3. Kvantu uzlabojums

Varam modificēt kvantu algoritmu, kurš aprakstīts kursa darbā [5: 3.6.]:

- Funkcija $F(S = v)$ ir melnas kastes funkcija. Ja $|N(v_m)| \leq 3$, tad virsotnei v bērnu nav. Bērnu nav arī tad, ja $|S| \geq k$. Citādi, virsotnei v ir 2 bērni.

  P saņemot virsotni $(S = v)$ pārbaudei palaiž $F(S = v)$:

1) Ja $|N(v_m)| = 0$ un $|S| \leq k$, tad risinājums ir atrasts un P atgriež $true$.

2) Ja $|N(v_m)| = 1$ un $|S| < k$, tad pārbaudām rezultātu polinomiālā laikā un atgriežam $true$, ja atrisinājums eksistē un $false$ pretēja gadījuma.

3) Ja $|N(v_m)| = 2$ un $|S| < k$, tad pārbaudām rezultātu laikā $O\left(1.1616^k \cdot k^{O(1)}\right)$ un atgriežam $true$, ja atrisinājums eksistē un $false$ pretēja gadījuma.

4) Ja $|N(v_m)| = 3$ un $|S| < k$, tad pārbaudām rezultātu polinomiālā laikā un atgriežam $true$, ja atrisinājums eksistē un $false$ pretēja gadījuma.

5) Ja $|N(v_m)| > 3$ un $|S| < k$, tad atgriežam $indeterminate$.

6) Citādi atgriežam $false$.

No 6.2. dabūjam, ka koka τ maksimālais izmērs $T_1 = 1.3803^k$ un dziļums $= k$. Apvienojot visu kopā, mēs varam izveidot kvantu uzlabojumu klasiskām algoritmam. Kopējā sarežģītībā ir $O\left(\sqrt{T_1 k} \cdot k^{O(1)} + n\sqrt{m}\right) = O(1.175^k \, k^{O(1)} + n\sqrt{m})$.

7. “LONGEST PATH” PROBLĒMA

7.1. Problēmas apraksts

Mums dots grafs G ar n virsotnēm un m šķautnēm, ka arī pozitīvs vesels skaitlis k. Uzdevums ir noteikt, vai grafā eksistē ceļš no k virsotnēm. “Longest path” problēma ir NP-sarežģīta. Labākais pazīstamais klasiskais algoritms[11] atrisina “Longest path” laikā $O(2^k \cdot poly(n,k))$.

7.2. Klasiskais algoritms

Klasiskais algoritms[1] izmantos krāsu kodēšanu. Sākumā, nokrāsosim katru grāfa virsotni ar kādu no k krāsām. Krāsu katrai virsotnei izvēlēsimies ar vienādu varbūtību. Tāgad, mēģināsim atrast grafā ceļu no k virsotniem, kuru krāsas ir pārī atšķirīgas.

[1: Lemma 5.4.] Pieņēmsim, ka U ir kopa ar izmēru n, un $X \subseteq U$, ir apakškopa ar izmēru k. Pieņēmsim, ka $\chi: U \to [k]$ ir kopas U elementu krāsojums, kur katras virsotnes krāsa ir izvēlēta nejauši. Tad varbūtība, ka visi kopas X elementi ir nokrāsoti ar pārī atšķirīgam krāsām ir vismaz e^{-k}.

[1: Lemma 5.5.] G ir orientēts vai neorientēts grafs un $\chi: V(G) \to [k]$ ir to virsotņu krāsojums ar k krāsām. Eksistē deterministisks algoritms, kurš pārbauda laikā $2^k n^{O(1)}$ vai G satūr ceļu no k virsotniem, kuru krāsas ir pārī atšķirīgas.

Pieņēmsim, ka $V_1, \dots, V_k$ ir tāds grafa G virsotņu sadalījums, kā visas virsotnes no V_i ir nokrāsotās ar i-to krāsu. Klasiskais algoritms izmantos dinamisku programmēšanu: katrai netukšai kopas $\{1, \dots, k\}$ apakškopai S un virsotnei $u \in \bigcup_{i \in S} V_i$, mēs definēsim būla vertību $PATH(S,u)$, kura ir vienāda ar $true$, ja grafā eksistē daudzkrāsainais ceļš ar garumu $|S|$, kura virsotnes ir nokrāsotās ar visam krāsam no S, un ceļa galapunkts ir virsotne u. Gadijumā, kad $|S| = 1$, $PATH(S,u)$ ir vienāds ar $true$ katrai $u \in V(G)$ tad un tikai tad, ja $S = \{\chi(u)\}$. Gadijumā, kad $|S| > 1$, izmantosim sekojošu rekurenci:

$$PATH(S,u) = \begin{cases} \vee \{PATH(S \setminus \{\chi(u)\}, v) : vu \in E(G)\}, & ja\ \chi(u) \in S \\ false, & citādi \end{cases}$$

Citiem vārdiem, ja grafā eksistē ceļš, kurš beidzās virsotnē u un izmanto visas krāsas no S, tad grafā jābūt arī ceļs, kurš beidzās ar vienu no virsotnes u kaimiņiem v, un izmanto visas krāsas no $S \setminus \{\chi(u)\}$. Visas $PATH$ vertības var būt aprēķinātas laikā $2^k n^{O(1)}$, izmantojot norādito rekurenci. Grafā eksistē krāsains ceļš no k virsotnēm tad un tikai tad, ja $PATH([k], v)$ ir vienāds ar $true$ kādai virsotnei $v \in V(G)$.

Mēs varam papildināt šo algoritmu, lai viņš saglabātu ne tikai ceļā galapunktu, bet arī sākuma punktu. $PATH(S,b,u)$ ir vienāds ar $true$, ja grafā eksistē daudzkrāsainais ceļš ar garumu $|S|$, kura virsotnes ir nokrāsotās ar visam krāsam no S, ceļa sākuma punkts ir virsotne b un ceļa galapunkts ir virsotne u. Gadijumā, kad $|S| = 1$, $PATH(S,u,u)$ ir vienāds ar $true$ katrai $u \in V(G)$ tad un tikai tad, ja $S = \{\chi(u)\}$. Gadijumā, kad $|S| > 1$, izmantosim sekojošu rekurenci:

$$PATH(S,b,u) = \begin{cases} \vee \{PATH(S \setminus \{\chi(u)\}, b, v) : vu \in E(G)\}, & ja\ \chi(u), \chi(b) \in S \\ false & , \quad citādi \end{cases}$$

Analoģiski sākotnējam algoritmam, visas $PATH$ vertības var būt aprēķinātas laikā $2^k n^{O(1)}$, izmantojot norādito rekurenci. Grafā eksistē krāsains ceļš no k virsotnēm tad un tikai tad, ja $PATH([k],b,v)$ ir vienāds ar $true$ kādiem virsotnēm $b, v \in V(G)$. Papildinātajam algoritmam ir vel viena svarīga īpašība:

$PATH(S,b,u) = \vee \{PATH(X,b,v) \wedge PATH(Y,v,u)\}$, kur $X \cup Y = S$ un $X \cap Y = \chi(v)$.

Izmantojot šo īpašību, varam aprēķināt $PATH$ vērtības ne tikai pievienojot vienu krāsu apakškopai, bet arī apvienojot lielākas krāsu apakškopas.

Varbūtība dabūt "labu" virsotņu krāsojumu ir vismāz e^{-k}, lai atrīsināt "Longest path" problēmu ar konstantu varbūtību, mums būs nepieciešami e^k algoritma atkartojumi. Kopēja klasiska algoritma laika sarežģītība ir $O((2e)^k n^{O(1)})$.

7.3. Kvantu uzlabojums

Dinamisku programmēšanu uz apakškopam ir iespējams paātrināt, izmantojot kvantu algoritmu, ko piedāvāja [4]. Pamata ideja ir, ja mēs gribām atrāst funkcijas vērtību kopai S, mums ir nepieciešams aprēķināt un saglabāt atmiņā funkcijas vertību visām S apakškopam ar izmēru $\frac{|S|}{4} + 1$, tad izmantot Grovera meklēšanu uz apakškopam. Pamatalgoritms ir aprakstīts pētījumā [4: 1.2.] un to izmanto, lai noteiktu, vai grafs satur Hamiltona ciklu. Mēs varam izmantot to "Longest path" problēmai, jo mums vajag atrāst ceļu caur visam krāsu kopam $V_1, \dots, V_k$. Piezīme: lai pārveidot ceļu ar k krāsam uz ciklu, mēs varam pievienot grafam vel vienu virotni, kura ir savienota ar visām citam, un nokrāsot to ar unikālu krāsu. Vienīga atšķirība ir, ka "Longest path" problēmai, kopu apvienošanas laikā vajag pārbaudīt $O(n)$ sākuma virsotnes un $O(n)$ gala virsotnes, līdz ar to laika sarežģitībai parādas papildus $O(n^{O(1)})$ sastāvdaļa. Šīs kvantu algoritms uzlabo klasiska algoritma laiku līdz

$O((1.755e)^k n^{O(1)})$. Pielietojot amplitūtas amplifikāciju, dabūjam laiku $O(\left(1.755\sqrt{e}\right)^k n^{O(1)})$.

Pētījumā [4: Theorem 8] ir aprakstīts optimizēts kvantu paātrinājums "Travelling salesman" problēmai. Lai to izmantot, no sākuma vajag aprēķināt $PATH$ vertības visam kopas S apakškopam ar izmēru ne lielāku par $(1-a)n/4$, kur $a \approx 0.055362$. Tad izmantojot pētijumā norādītu algoritmu, "Longest path" problēma ir atrisināma laikā $O(\left(1.7274\sqrt{e}\right)^k n^{O(1)})$. Mums sanāca samāzināt laika eksponentes daļu no $O((2e)^k)$ līdz $O(\left(1.7274\sqrt{e}\right)^k)$.

REZULTĀTI UN SECINĀJUMI

Bakalaura darba ietvāros autoram izdēvas iegūt sekojošus rezultātus:

1) Tika iegūts kvantu algoritmu, kas atrisina "Closest string" problēmu laikā $O(kL + \mathrm{k}^2 d^{3/2}\sqrt{(\mathrm{d}+1)^d})$. Gadijumā, kad $|\Sigma| - 1 > \sqrt{d+1}$, kvantu algoritma laika sarežģītība ir labāka, nekā ātrākam pazīstamām algoritmam ar laiku $O(Lk + kd(|\Sigma| - 1)^d \cdot 2^{3.25d})$[12].

2) Tika iegūts kvantu algoritmu, kas atrisina "Cluster vertex deletion" problēmu laikā $O(1.7321^k \sqrt{k}\, n^3)$. Kvantu algoritma laika sarežģītība ir labāka, nekā ātrākam pazīstamām algoritmam ar laiku $O(1.9102^k(n+m))$[8].

3) Tika iegūts kvantu algoritmu, kas atrisina "Cluster editing" problēmu laikā $O(1.7321^k \sqrt{k}\ n^3)$. Kvantu algoritma laika sarežģītība nav labāka, nekā ātrākam pazīstamām algoritmam ar laiku $O(1.62^k + m + n))$[9].

4) Tika iegūts abūjam kvantu algoritmu, kas atrisina "k-vertex cover" problēmu laikā $O(1.175^k\ k^{O(1)} + n\sqrt{m})$. Kvantu algoritma laika sarežģītība ir labāka, nekā ātrākam pazīstamām klasiskam algoritmam ar laiku $O(1.2738^k + kn)$[3] un ātrākam pazīstamām kvantu algoritmam ar laiku $O(1.211^k\ k^{5/2} + n\sqrt{m})$[5].

5) Tika iegūts kvantu algoritmu, kas atrisina "Longest path" problēmu laikā $O(\left(1.7274\sqrt{e}\right)^k n^{O(1)})$. Kvantu algoritma laika sarežģītība nav labāka, nekā ātrākam pazīstamām algoritmam ar laiku $O(2^k \cdot poly(n,k))$[11].

6) Daži algoritmi [9], kurus potenciāli ir iespējams kvantiski paātrināt, nav apskatīti bakalaura darba ietvāros, tomēr viņi ir interesanti tālākai izpētei.

Darba rezultāti liecina, ka kvantu algoritmiem ir liels potenciāls arī citu, sarežģītāku klasisku algoritmu uzlabošanai.

PATIECĪBAS

Autors pateicās darba vadītājam Jevgēnijam Vihrovam par interesanto pētniecisko tēmu kursa un bakalaura darbam, kā arī par sadarbību un palīdzību pētījuma laikā.

IZMANTOTĀ LITERATŪRA

[1] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk and Saket Saurabh *"Parameterized Algorithms"* 2016. Pieejams internetā: https://www.mimuw.edu.pl/~malcin/book/parameterized-algorithms.pdf?fbclid=IwAR2bv4ImvjV3hjHzxQS0gvsgFoRpjizfSh2lnGcccvxwxww42DArTcfQM8s

[2] Andris Ambainis, Martins Kokainis *"Quantum algorithm for tree size estimation, with applications to backtracking and 2-player games"* 2017. Pieejams internetā: https://arxiv.org/pdf/1704.06774.pdf

[3] Jianer Chen, Iyad A. Kanj, Ge Xia *"Improved upper bounds for vertex cover"* 2010. Pieejams internetā: https://www.sciencedirect.com/science/article/pii/S0304397510003609?via%3Dihub

[4] Andris Ambainis, Kaspars Balodis, Jānis Iraids, Martins Kokainis, Krišjānis Prūsis, and Jevgēnijs Vihrovs *"Quantum Speedups for Exponential-Time Dynamic Programming Algorithms"* 2018. Pieejams internetā: https://arxiv.org/pdf/1807.05209.pdf

[5] A. Jeļisejevs "Parametrizētu algoritmu paātrinājumi kvantu datoram", Kursa darbs, LU Datorikas fakultāte, 2021.

[6] L.K.Grover, "A fast quantum mechanical algorithm for database search", 1996. Pieejams internetā: https://arxiv.org/abs/quant-ph/9605043

[7] G.Brassard. P.Høyer, M.Mosca, A.Tapp "*Quantum Amplitude Amplification and Estimation*", 2000. Pieejams internetā: https://arxiv.org/abs/quant-ph/0005055

[8] A. Boral. M. Cygan, T. Kociumaka, M. Pilipczuk "*A Fast Branching Algorithm for Cluster Vertex Deletion*", 2016. Pieejams internetā: https://link.springer.com/article/10.1007/s00224-015-9631-7

[9] S. Böcker "*A golden ratio parameterized algorithm for Cluster Editing*", 2012. Pieejams internetā: https://www.sciencedirect.com/science/article/pii/S1570866712000597

[10] M. Xiao "*A Note on Vertex Cover in Graphs with Maximum Degree 3*", 2010.

DOI:10.1007/978-3-642-14031-0_18. Pieejams internetā: https://www.researchgate.net/publication/221427022_A_Note_on_Vertex_Cover_in_Graphs_with_Maximum_Degree_3

[11] R. Williams "*Finding a path of length k in* $O^*(2^k)$ *time*", 2008.
Pieejams internetā: https://arxiv.org/pdf/0807.3026.pdf

[12] L. Wang, B. Zhu "*Efficient Algorithms for the Closest String and Distinguishing String Selection Problems*", 2009.
Pieejams internetā: https://link.springer.com/chapter/10.1007/978-3-642-02270-8_27

Bakalaura darbs “Parametrizētu algoritmu paātrinājumi kvantu datoram” izstrādāts Latvijas Universitātes Datorikas fakultātē.

Ar savu parakstu apliecinu, ka darbs izstrādāts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti.

Autors: Aleksejs Jeļisejevs __.05.2021.

Rekomendēju darbu aizstāvēšanai

Darba vadītājs: doc., Dr.dat. Jevgēnijs Vihrovs __.05.2021.

Recenzents: prof., Dr.sc.comp. Juris Vīksna

Darbs iesniegts Datorikas fakultātē 31.05.2021.

Dekāna pilnvarotā persona: vecākā metodiķe Ārija Sproģe

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē

__.06.2021. prot. Nr. ___.

Komisijas sekretārs: doc., Dr.sc.comp. Jevgēnijs Vihrovs