**RĪGAS TEHNISKĀ UNIVERSITĀTE**

Datorzinātnes un informācijas tehnoloģijas fakultāte

Vadības informācijas tehnoloģijas katedrā

**Dmitrijs Siņickis**

Informācijas tehnoloģija

students, stud. apl. nr. 201RDB360

**Vairākslāņu grafu vizualizēšanas metožu eksperimentāls novērtējums ar “JavaScript” bibliotēkām**

**BAKALAURA DARBS**

Zinātniskais vadītājs:

Profesors

Jānis Grabis

RĪGA 2023

**RĪGAS TEHNISKĀ UNIVERSITĀTE**

**DATORZINĀTNES UN INFORMĀCIJAS TEHNOLOĢIJAS FAKULTĀTE**

Informācijas tehnoloģijas institūts

**bakalaura darbs izpildes lapa**

Noslēguma darba autors:

students(-e) Dmitrijs Siņickis\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts, datums)

Noslēguma darbs ieteikts aizstāvēšanai:

Zinātniskais vadītājs:

Profesors Jānis Grabis \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(paraksts, datums)

Anotācija

Bakalaura darba mērķis ir noteikt efektīvāko veidu vairākslāņu grafu implementēšanai ar “JavaScript” bibliotēkām. Darba uzdevumi ir izpētīt vairākslāņu grafu vizualizācijas pamatprincipus, izveidot vairākslāņu grafu prototipu tīmekļa lapā ar “JavaScript” bibliotēkām, pārbaudīt “JavaScript” bibliotēku veiktspēju ar lieliem datiem. eksperimentāli salīdzināt “JavaScript” bibliotēkas. Darba izstrādes procesā tika izveidoti vairākslāņu grafu prototipi ar trim bibliotēkām, pēc tam modificēti algoritmi, lai pārbaudītu “JavaScript” bibliotēkas veiktspēju un kvalitatīvi salīdzinātu. Rezultātā tika konstatētas “JavaScript” bibliotēku vājās un stiprās puses, algoritmu izmēri un novērtēta vairākslāņu grafu interaktivitāte . Darba apjoms 56.lpp., 10 tabulas, 34 attēli.

Abstract

The purpose of the bachelor's thesis is to determine the most efficient way to implement multilayer graphs with JavaScript libraries. The objectives are studying the basic principles of visualization of multilayer graphs, creating a prototype of layered graphs on a web page with JavaScript libraries. Performance testing of big data JavaScript libraries. Experimental comparison of JavaScript libraries. During the development of the research, multilayer graph prototypes were created with three libraries, then the algorithms were modified to test the performance of the JavaScript library and make a qualitative comparison. As a result, the strengths and weaknesses of loading the JavaScript library, the size of the algorithms and the interactivity of multilayer graphs in different libraries were identified. The volume of work consists of 56 pages, 10 tables, 34 images.

Satura rādītājs

[Ievads 5](#_Toc135856953)

[1. Grafu teorija 7](#_Toc135856954)

[2. Vairākslāņu grafu teorija 12](#_Toc135856955)

[3. Grafu vizualizācija 20](#_Toc135856956)

[3.1. Grafu vizualizēšanas algoritmu process 20](#_Toc135856957)

[3.2. d3.js bibliotēka 21](#_Toc135856958)

[3.2.1 Secinājumi par d3.js bibliotēku 27](#_Toc135856959)

[3.3. cytoscape.js bibliotēka 27](#_Toc135856960)

[3.3.1. Secinājumi par cytoscape.js bibliotēku 31](#_Toc135856961)

[3.4. vis.js bibliotēka 31](#_Toc135856962)

[3.4.1 Secinājumi par vis.js bibliotēku 33](#_Toc135856963)

[3.5 Bibliotēku vērtēšanas kopsavilkums 34](#_Toc135856964)

[4. Eksperimentāls salīdzinājums 35](#_Toc135856974)

[4.1. Eksperimentu objekts 36](#_Toc135856975)

[4.1.1. D3.js bibliotēka koda modificēšana 37](#_Toc135856976)

[4.1.2. Cytoscape.js bibliotēka koda modificēšana 39](#_Toc135856977)

[4.1.3. Vis.js bibliotēka koda modificēšana 40](#_Toc135856978)

[4.2. Eksperiments 42](#_Toc135856979)

[4.3. Eksperimentu rezultāti 48](#_Toc135856980)

[Secinājumi 52](#_Toc135856981)

[Izmantotie informācijas avoti](#_Toc135856982)

Ievads

Vairākslāņu grafs ir svarīgākais instruments dažādu sarežģītu sistēmu analīzei un aprakstīšanai visā pasaulē. Vairākslāņu grafus visbiežāk var redzēt medicīnas, socioloģijas, humanitārajās zinātnēs un citās jomās.

Ar vairākslāņu palīdzību var apvienot dažādas datu bāzes, kas var būt savstarpēji saistītas un vizualizēt to visu vienā vairākslāņu grafā. Šādu grafu veidu sāka izmantot tikai pirms dažiem desmitiem gadiem, šobrīd vairākslāņu grafi piesaistījuši ievērojamu pētnieku uzmanību daudzās jomās to augstās lietderības dēļ, modelējot savstarpēji atkarīgas sistēmas.

Šobrīd ir daudz dažādu variantu, kā var izveidot un vizualizēt vairākslāņu grafus ar programmēšanas valodu palīdzību. Darbā izmantota “JavaScript” valoda, kura ir aktuāla programmēšanas jomā šodien. Tāpat “JavaScript” ir populāra programmēšanas valoda sabiedrībā, tāpēc ir daudz dažādu bibliotēku, kuras ir saistītas ar vairākslāņu grafu tēmu. Pateicoties “JavaScript” valodai ir iespējama grafu vizualizāciju integrācija dažādās tīmekļa lapās un tīmekļa lietojumprogrammās. Bibliotēku “JavaScript” veiktspēja ir ļoti svarīgs aspekts, kas jāizvērtē pirms gatava produkta izlaišanas. No tā ir atkarīgs projekta darbības ātrums un projekta lielums.

Mūsdienās “JavaScript” ir daudz dažādu bibliotēku, kas var vienkāršot vairākslāņu grafu izveidi. Tomēr nav zināma to veiktspēja ar lielu datu apjomu.

**Bakalaura darba mērķis:** Noteikt efektīvāko veidu vairākslāņu grafu implementēšanai ar “JavaScript” bibliotēkām, eksperimentāli salīdzinot vismaz trīs bibliotēkas.

**Bakalaura darba uzdevumi:**

1) Aprakstīt grafu teoriju un vairākslāņu grafu jēdzienu.

2) Izpētīt grafu vizualizēšanas algoritmus

3) Salīdzināt “JavaScript” bibliotēkas funkcionālu (D3.js, Cytoscape.js un Vis.js)

4) Vairākslāņu grafu implementēšana.

5) Eksperimentāls salīdzinājums.

**Problēmas nostādne:**

Bakalaura darbs satur 4 nodaļas, secinājumus un izmantotos informācijas avotus.

Pirmajā daļā ir pamatinformācija par grafu teoriju. Aprakstīta grafu teoriju terminoloģija.

Otrajā daļā apraksta vairākslāņu grafu jēdzienus. Aplūkoti dažādi vairākslāņu grafu tipi un lietošanas piemēri.

Trešajā daļā tika izveidotas vienkāršas vairākslāņu grafu vizualizācijas ar dažādām “JavaScript” bibliotēkām un funkcionalitātes salīdzinājumi.

Ceturtajā daļā tika modificēti algoritmi no trešās daļas un izveidoti eksperimentālie veiktspēju salīdzinājumi.

1. Grafu teorija

Grafi rodas un tiek izmantoti visdažādāko situāciju modelēšanai. [5]

Grafu teorija – ir bāze vairākslāņu grafiem, lielākā daļa teorijas aizgūtā tieši no grafu teorijas. Tāpēc, lai pilnībā apzinātu vairākslāņu grafus, ir jāpēta grafu teoriju.

Parasts grafs sastāv no virsotnes un šķautnes. Virsotnes apzīmē ar burtu V un šķautnes apzīmē ar burtu E, sanāk, ka grafa formula ir G = (V, E), grafu apzīmēts ar burtu G. – sk. 1.1. att. [4]



1.1. attēls. Grafs, kur ar sarkanu krāsā apzīmētas virsotnes un ar zilā krāsā apzīmētas šķautnes. (Autora veidots)

Tagad ir vairāki grafu veidi, pirmais grafu veids ir neorientēts grafs un otrs grafu veids ir orientēts grafs, atšķirība ir tā, ja šķautnēm ir virziens, šķautnēs šādu veidu sauc par lokiem, grafs ar lokiem saucas par “orientētu grafu”, bet grafs ar šķautnēm saucās par “neorientētu grafu”. Ja grafs ir neorientēts, tad virsotņu secībai nav nozīmes. – sk. 1.2. att. [5]

Attēls, kurā ir elektronika

Apraksts ģenerēts automātiski

1.2. attēls. Grafu veidi, neorientēts grafs un orientēts grafs. (Autora veidots)

Trešais grafu veids ir “Jaukts grafs”, kur ir gan šķautnes, gan loki. – sk. 1.3. att. [5]

A picture containing circle, line, sketch, black and white

Description automatically generated

1.3. attēls. Jaukts grafs (Autora veidots)

Ceturtais grafu veids ir “Multigrafs”, kur viena virsotne savienota ar vairākām virsotnēm caur šķautnēm. – sk. 1.4. att. [5]

A picture containing circle, line, sketch, design

Description automatically generated

1.4. attēls. Multigrafs (Autora veidots)

Piektais ir “Hipergrafs” – grafs, kur šķautnes savieno trīs un vairāk virsotnes. – sk. 1.5. att. [5]

A picture containing circle, line, black and white

Description automatically generated

1.5. attēls. Hipergrafs (Autora veidots)

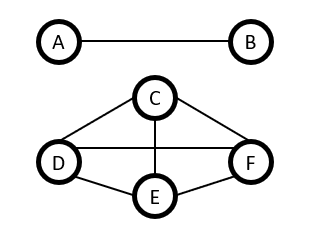
Pēdējais sestais grafs ir “Pseidografs”, kur šķautne sākās un beidzās tanī pašā virsotnē, šo īpašību sauc par cilpu. – sk. 1.6. att. [5]

A picture containing sketch, line, circle, design

Description automatically generated

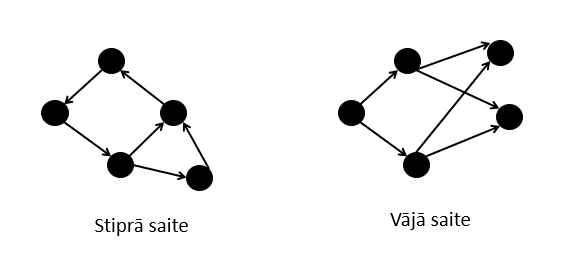
1.6. attēls. Pseidografs (Autora veidots)

Grafus var iedalīt arī saistītos un nesaistītos grafos. Nesaistīts grafs ir tad, kad virsotnes veido daļas, piemērs: virsotne A un virsotne B ir savienoti ar šķautni, virsotne C, D, E, F ari savienotas ar šķautnēm, bet AB ar CDEF nav savienoti ar kopējām šķautnēm. – sk. 1.7. att. [5]



1.7. attēls. Nesaistīts grafs piemērs. (Autora veidots)

Saistīts grafs dalās uz diviem veidiem, pirmais veids ir stipra saite, otrais veids ir vāja saite. Stipra saite, kur no katras virsotnes ir loki uz citu virsotni. Vāja saite ir pilnīgs pretstats, kur ne no katras virsotnes ir loki uz citu. – sk. 1.8. att. [5]



1.8. attēls. Saistīts grafs (Stiprā un vājā saites) (Autora veidots)

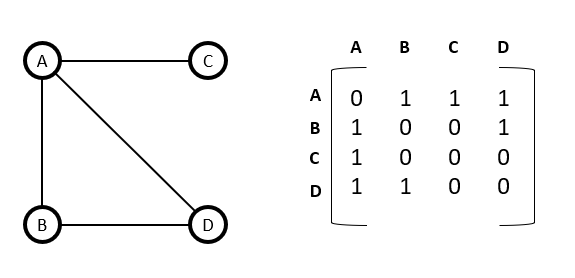
Taisīsim grafu, kas sastāv no Latvijas novadiem, kur virsotnes ir A (Latgale), B (Zemgale), C (Vidzeme), D (Kurzeme). Ceļu no vienas virsotnes uz citu sauc par maršrutu. Šķautnes maršruta skaitu sauc par maršruta garumu. Lai sasniegtu punktu D no punkta A. Izveidosim maršrutu A-B-D, sanāk, ka maršruta garums ir 2, ja cits maršruts A-C-B-D, mazliet garāks, tad sanāk, ka maršruta garums ir 3. – sk. 1.9. att. [5]

Attēls, kurā ir karte

Apraksts ģenerēts automātiski

1.9. attēls. Latvijas novada grafs (Autora veidots)

Matrica, ir vēl viena metode, kā var parādīt grafu. Grafu, kur ir virsotnes un šķautnes ( G = (V,E) ) sadalīsim uz rindām un kolonnām (p, q grafs). Izveidosim p × p matricu A = [aij], kur rindas un kolonnas atbilst V virsotnei. Izveidosim vienkāršu algoritmu: Ja virsotne vi atrodas blakus virsotnei vj, tad aij = 1. Citi varianti būs aij = 0, un izmantosim ciklu, lai algoritms aprēķinātu katru virsotni (i = 0,1,2,…,p.). Sanāk saistību matrica, kura ir pilnīgi simetriska. 1.10. attēlā var redzēt rezultātu. [5]



1.10. attēls. Grafs un matrica (algoritma rezultāts)

2. Vairākslāņu grafu teorija

Vairākslāņu grafs – ir datu vizualizācija, kur virsotnes un šķautnes ir sadalītas vairākos slāņos, katrs no tiem pārstāv noteiktu datu aspektu. Katrs grafu slānis var saturēt savas virsotnes un šķautnes.

Vairākslāņu grafu visbiežāk izmanto sarežģītu sistēmu modelēšanai un savstarpējā sakarā starp dažādiem datu tipiem. [6]

Vairākslāņu grafā katrs slānis var attēlot dažādus aspektus vai savienojumu tipus. Piemēram, pirmais aspekts – iedarbība ar kaut ko un otrais aspekts – laiks. Aspektus apzīmēsim kā *d*, Viens slānis ar d1 un otrais slānis d2, lai apzīmētu šo slāni lieto terminu – elementārais slānis. Lai formulētu vairākslāņu grafu, vispirms izveidosim secības formulu L = {La}dα =1 (alfa vai α nozīme slāņu kopa), lai izveidotu elementāro slāņu “komplektu”. Vairākslāņu grafu apzīmēsim kā M, tad M=(VM, EM, V, L), ir piemēram 4 virsotnes. Tad V = {1,2,3,4} un d1, d2. Elementāri slāņi ir L1 = {A,B} un L2 ={X,Y}. Sanāk 4 dažādi slāņi: (A,X), (A,Y), (B,X) un (B,Y). Virsotnes šādā veidā ir izvietoti slāņos: VM = {(1,A,X), (2,A,X), (3,A,X), (2,A,Y), (3,A,Y), (1,B,X), (3,B,X), (4,B,X), (1,B,Y)} ⊆ V × L1 × L2. Virsotnes savienotas gan slāņu iekšpusē, kuras saucās par iekšējām malām (nepārtraukta līnija), gan starp slāņiem, kuri saucas par starpslāņu malām (pārtraukta līnija).

Šķautne ((i, α), (j, β)) ∈ EM, kas nozīmē, ka šķautnes virziens iet no virsotnes i uz slāni α līdz virsotnei j uz slāni β. – sk. 2.1. att. [1]

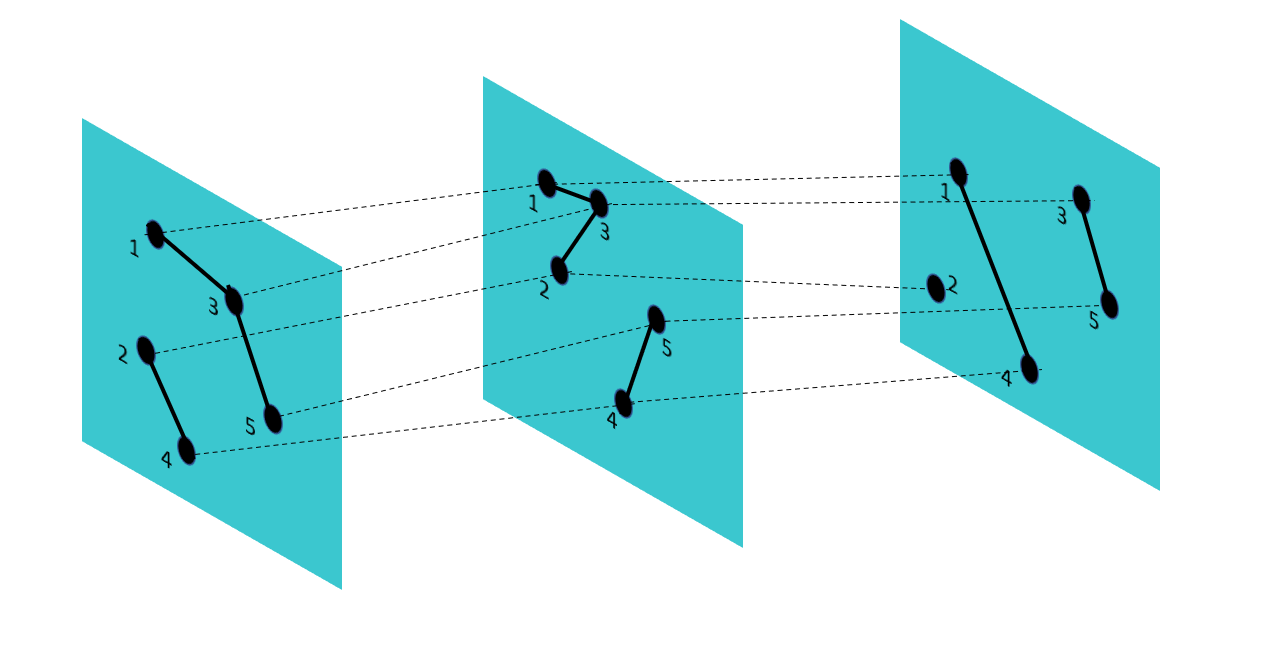
Attēls, kurā ir teksts, rakstāmgalds, galds, laimētava

Apraksts ģenerēts automātiski

2.1. attēls. Vairākslāņu grafs no piemēra. (aizgūts no (Kivelä & Arenas, 2014))

Vairākslāņu grafu priekšrocības ir spēja attēlot sarežģītākas un daudzveidīgākas kopsakarības nekā vienkāršais grafs. Ar vairākslāņu grafu ir iespējams elastīgāk modelēt un analizēt sistēmas, tostarp daudzus faktorus, kas ietekmē mijiedarbību starp virsotnes.

Tāpat kā grafu teorijā, vairākslāņu grafiem ir dažādi tipi. Pirmais tips ir **Multiplekss grafs**, tas ir slāņu kopa, kas ir {Ga; α ∈ {1, …, M}}, kur katrs slānis ir grafs Gα = (Xα, Eα), kur visos slāņos obligāti jābūt vienādam virsotņu skaitam X1 = X2 = … = XM = X. Visas virsotnes uz slāņa savienojas tajās pašās virsotnēs citos slāņos Eαβ = {(x,x); x ∈ X katram α, β ∈ {1, … , M}, α ≠ β. Multiplekss grafs tiek izmantots sarežģītu tīklu modelēšanai un analīzei, kur savienojumi starp virsotnēm var būt ar dažādām īpašībām vai mijiedarbības tipiem. Piemēram, sociālajos tīklos var attēlot draudzīgas attiecības, darba attiecības un ģimenes attiecības starp cilvēkiem. – sk. 2.2. att. [7]



2.2. attēls. Multiplekss grafs (Autora veidots)

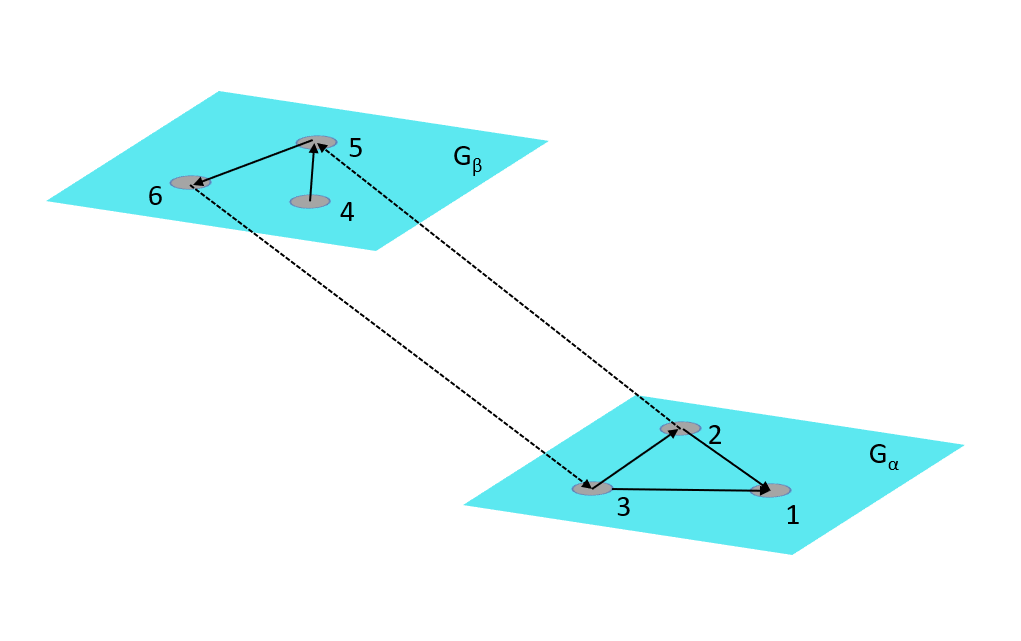
Otrais grafa tips ir **Temporāls grafs**, tas ir slāņu kopums, kas iet noteiktā secībā (G(t))Tt=1. Visi slāņi vairākslāņu grafā ir {G1,… , GT}, kur Gt = G(t). Temporāls grafs tas ir tīkls, kur savienojumi starp virsotnēm ir īslaicīgi. Visbiežāk ar tādu grafu modelēja transporta tīklus, epidemioloģiskos tīklus utt. – sk. 2.3. att. [7]

Attēls, kurā ir teksts, zīme, vizītkarte, aploksne

Apraksts ģenerēts automātiski

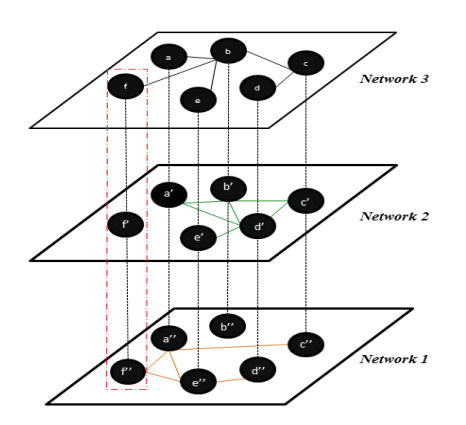
2.3. attēls. Temporāls grafs. G(t) t=1,2,3 (Autora veidots)

Trešais tips ir **Savstarpēji savienots grafs** (jeb angļu valoda “interconnected graph”), parasti starpslāņu malas iet no viena slāņa uz otru, un viss notiek vienā virzienā, bet savstarpēja savienota grafa var gan no slāņa Gα līdz slānim Gβ, gan no slāņa Gβ līdz slānim Gα. Izmanto datortīklos, lai vizualizētu vairākus lokālos tīklus, kas apvienoti vienotā tīklā. – sk. 2.4. att. [7]



2.4. attēls. Savstarpēji savienots grafs. (Autora veidots)

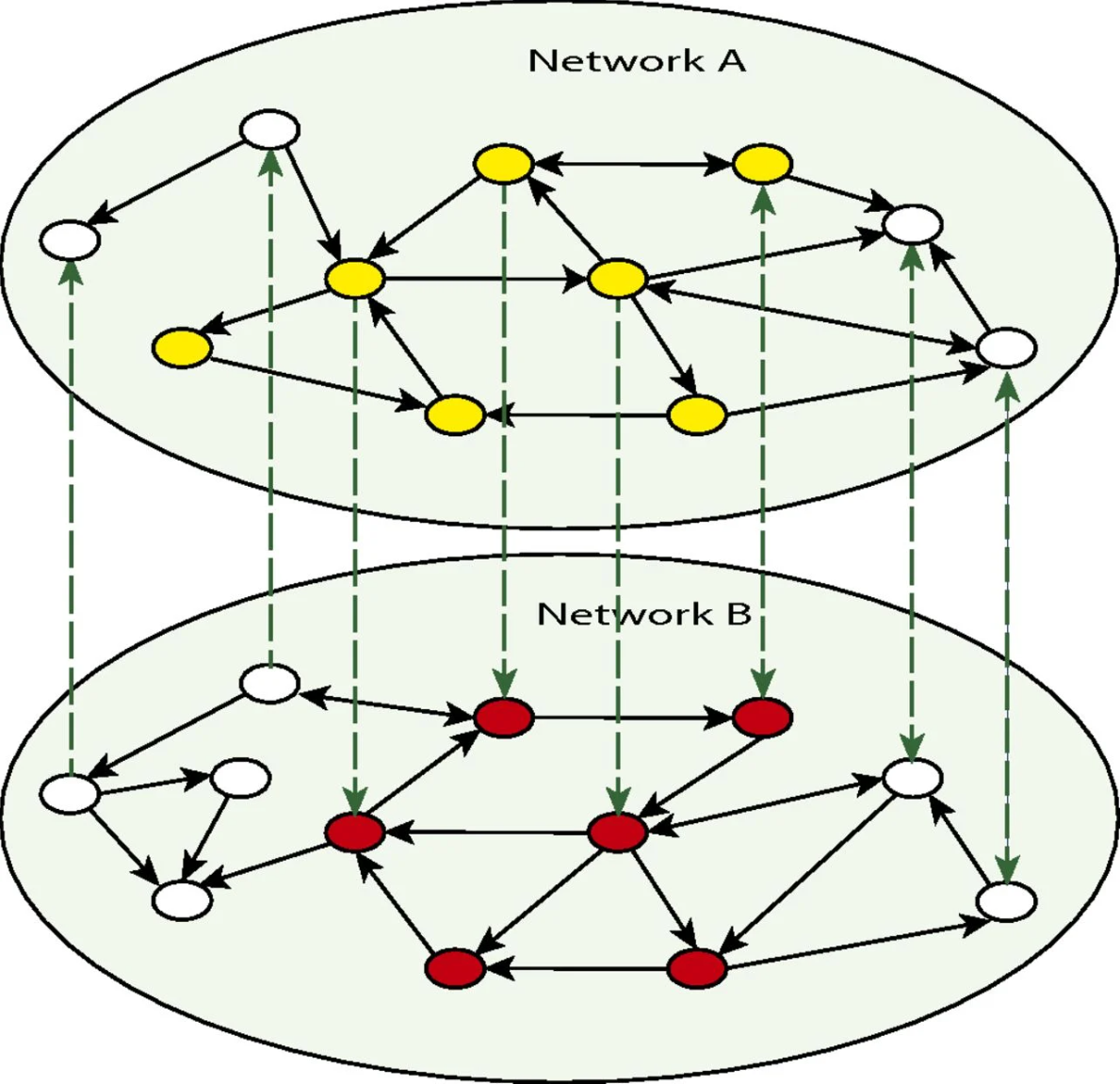
Ceturtais tips ir **Daudzdimensiju grafs**, vairākslāņu grafs, kur vairākos slāņos tiek izmantotas vienas un tās pašas virsotnes, bet katrā slānī var būt dažādi savienojumi starp virsotnēm. Grafa struktūra ir G=(V,E,L), kur V ir virsotņu kopa, E ir šķautnes kopa ar zīmi un L – zīme (jeb izmērs). Ar tādu grafu var vizualizēt sociālos tīklos datukopu ar parametriem: attiecību tips, vecums utt. – sk. 2.5. att. [8].



2.5. attēls. Daudzdimensiju grafs (aizgūts no (Bouanan & Ribault, 2015))

Piektais tips ir **Savstarpēji atkarīgs grafs**, dažādu slāņu un grafu kopums, kurš ir savstarpēji atkarīgs viens no otra. Virsotnes no viena slāņa ir atkarīgas no cita slāņa virsotnēm. Šādu struktūru sauc arī par mezostruktūru. [7]

Piemēram: ir Gα un Gβ, kur ir virsotnes ar parametriem kin un kout, sanāk Pα(kin, kout) un Pβ (kin, kout). Tad kin – virsotne, pie kuras nonāk informācija, jeb dati. Un kout – virsotne no kuras iznāk informācija, jeb dati. Var analizēt transporta sistēmu, kur ir kopsakarība starp ceļiem, transportu, luksoforu un citām sastāvdaļām. – sk. 2.6. att. [9]



2.6. attēls. Savstarpēji atkarīgs grafs (“Network A” ir Gα un “Network B” ir Gβ, virsotnes baltā krāsā (Gα ) - kin un virsotnes dzeltenā krāsā - kout (aizgūts no (Xueming & Stanley, 2015))

Sestais tips ir **Vairāklīmeņu grafs** – sastāv no trim elementiem M=(X,E,S), kur S = {S1, …, Sp}, tas nozīmē apakšgrafs Sj = (Xj, Ej). Vairāklīmeņu grafs identisks ar daudzdimensiju grafu tikai ar nosacījumu, ja virsotnes, kas pieder α slānim ir vienādas ar virsotnēm, kuras pieder β slānim (Xα=Xβ , 1 ≤ α, 1 ≤ β ). – sk. 2.7. att. [7]

Attēls, kurā ir galds, rakstāmgalds

Apraksts ģenerēts automātiski

2.7. attēls. Vairāklīmeņu grafs. (Autora veidots)

Septītais tips ir **Hipertīkla grafs –** pāris H′ = (X , H), kur virsotnes, kuras nav tukšas apzīmē kā X un hipertīkls kā H = {H1, …, Hp}. Ja dažādos slāņos ir vienādas virsotnes, tad nav nozīmes kādā secība ir slāni. Ļoti līdzīgs ar daudzdimensiju grafu – sk. 2.8. att. [7]

Attēls, kurā ir bulta

Apraksts ģenerēts automātiski

2.8. attēls. Hipertīkla grafs. (Hipertīkli (H­1, H2, H3) ar virsotnēm (X1, X2, X3, X4, X5) (aizgūts no (Boccaletti & Binconi, 2014))

Astotais tips ir **N-daļas grafs** (jeb angļu valoda “N-partite graph”) – tas ir grafs, kur virsotnes sadalās uz N bezgalīgu daudzumu. Šķautnes var būt tikai starp dažādu kopu virsotnēm. Katrs daudzums saucās kā “daļa” un grafs sastāv no tādiem N daļām. Jeb formula E ⊆ {(vi, vj) | vi ∈ Vi, vj ∈ Vj, i ≠ j}, kur Vn – virsotnes daudzums, E –šķautnes daudzums. – sk. 2.9. att. [6]

A picture containing line, colorfulness, circle, diagram

Description automatically generated

2.9. attēls. N-daļas grafs. (aizgūts no (Boccaletti & Binconi, 2014))

Devītais tips ir **Daudzfaktoru grafs** (jeb angļu valoda “Multivariate graphs”) – tas grafs, kur virsotnes vai šķautnes ir atribūti. Katrs grafu elements ir apzīmēts ar saviem parametriem vai mērīšanām. Šie atribūti var būt skaitliskie, kategoriskie var binārās vērtībās, kuri atspoguļo dažādus grafa aspektus vai īpašības. – sk. 2.10. att. [6]

A picture containing text, diagram, plan, line

Description automatically generated

2.10. attēls. Daudzfaktoru grafs, kur katrā virsotnē ir savi atribūti (aizgūts no (Boccaletti & Binconi, 2014))

Desmitais tips ir **Dinamiskais grafs** (jeb angļu valoda “Dynamic graphs”) – tas ir grafs, kur visa struktūra (virsotnes un šķautnes) laika gaitā mainās. Katrs slānis atspoguļo noteikta laika posmu. – sk. 2.11. att. [6]

A picture containing diagram, design

Description automatically generated with medium confidence

2.11. attēls. Dinamiskais grafs, kur notiek 4 laika posmi (aizgūts no (Boccaletti & Binconi, 2014))

Kopumā, vairākslāņu grafs ir spēcīgs instruments, lai modelēt sarežģītu sistēmu un analizēt dažādu tipu savstarpējo saišu. Vairākslāņu grafu izmantošana var palīdzēt izprast un prognozēt sistēmu uzvedību, atspoguļojot daudzveidīgus reālās pasaules aspektus. [2]

3. Grafu vizualizācija

Mūsdienās vairākslāņu grafu vizualizēšana ir iespējama ar “JavaScript” valodas programmēšanas palīdzību. “JavaScript” ir plaši lietota tīmekļa izstrādes programmēšanas valoda, tāpēc tā ir ideāla izvēle, lai realizēt interaktīvu un dinamisku vizualizāciju. Tāpat “JavaScript” valoda tiek atbalstīta tīmekļa pārlūkprogrammā, tāpēc to būs viegli integrēt jebkurā tīmekļa lietojumprogrammā.

“JavaScript” bibliotēkas grafu vizualizācijai sniedz dažādas metodes un algoritmus vairākslāņu grafu struktūru veidošanai. Tomēr, strādājot ar vairākslāņu grafu “JavaScript” vizualizāciju, ir svarīgi ņemt vērā veiktspēju un optimizāciju, strādājot ar lieliem un sarežģītiem datiem.

Tika izvēlētas šādas “JavaScript” bibliotēkas: “D3.js”, “Cytoscape.js” un “Vis.js”. “D3.js” ir vispopulārākā “JavaScript” bibliotēka. [30]

“Cytoscape.js” bija izvēlēta, tāpēc, ka bibliotēkas veidotāji kā pamatu izmantoja grafu teoriju. “Vis.js” mājaslapā rakstīts, ka tas ir viegli lietojams rīks (jeb angļu valoda – “easy to use”), tāpēc bija izvēlēta. [26]

3.1. Grafu vizualizēšanas algoritmu process

Lai implementēt vairākslāņu grafu prototipu tīmekļa lapā tika izveidots “HTML” dokuments, kur atradīsies vietnes struktūra, “CSS” dokuments vietnes vizuālās daļas pievienošanai un “JavaScript” dokuments, kur notiks vairākslāņu grafu realizācija. [13]

Demonstrācijas piemēram katrai bibliotēkai bija izveidots neorientēts temporāls grafs, kur ir 9 virsotnes, katrā slānī ir 3 virsotnes. Pirmā slānī tiek izveidotas 3 virsotnes, kuras ir savienotas viena ar otru.

“JavaScript” bibliotēkām nepieciešamās funkcijas:

* Virsotnes iestatīšana, tas nozīmē krāsu maiņu, koordinātu uzstādīšanu un virsotnes nosaukumu redzamību. Iespēja pievienot neierobežotu virsotnes daudzumu.
* Šķautnes iestatīšana, tas ir līnijas platuma maiņa, līnijas veids. Iespēja izmainīt šķautnes daudzumu.
* Vairākslāņu grafa atrašanās vietas maiņa, mērogošana, mijiedarbība ar vairākslāņu grafu elementiem.

3.2. d3.js bibliotēka

“D3.js” ir populārā “JavaScript” bibliotēka, kura ļauj manipulētu ar datiem un vizualizēt dažāda veida datus. Lai to izdarītu un vizualizētu datus, var izmantot modernās tehnoloģijas: “HTML”, “CSS” un “SVG”. Bibliotēka ir orientēta, lai strādātu modernās pārlūkprogrammās, kas ļauj izmantot visas tehnoloģijas, kuras izmanto tīmekļa izstrādē, tāpat ir viegli integrēt jebkurā vietnē, ja ir nepieciešams parādīt datu vizualizāciju. [10]

Pirmais posms, lai izveidotu vairākslāņu grafu vizualizāciju, tiek izveidots datu masīvs:

const data = [

  {

    layer: 1,

    nodes: [

      { id: 1, x: 100, y: 100 },

      { id: 2, x: 200, y: 100 },

      { id: 7, x: 150, y: 50 },

    ],

    links: [

      { source: 1, target: 2 },

      { source: 1, target: 7 },

      { source: 7, target: 2 },

    ],

  },…

,kur “let data” - tiek izveidots mainīgais ar nosaukumu “data”, “layer:1” – nozīme, ka slāni apzīmē ar numuru viens, “nodes” – iekšpusē norāda virsotnes, “id: n” – virsotnes nosaukums, x: n un y: n – virsotnes koordināti, “links”– iekšpusē norāda šķautnes, “source: n” – no kuras virsotnes sākās savienojums, “target: n” – kurā virsotnē beidzas savienojums.

Līdzīgi bija izveidoti vēl divi slāņi ar savām unikālajām virsotnēm un savienojumiem. Pēc tam pierakstam savienojuma parametrus, “D3.js” ir standarta rakstīšanas funkcija: [11]

let simulation = d3

  .forceSimulation()

  .force(

    "link",

    d3

      .forceLink()

      .id((d) => d.id)

      .distance(50)

  )

  .force("charge", d3.forceManyBody().strength(-50))

  .force("center", d3.forceCenter(400, 250));

,kur “let simulation” – radām mainīgo ar nosaukumu “simulation”, “d3.forceSimulation()” – izveido jaunu modelēšanu ar norādīto masīvu, funkcija, kas rada kaut kādu spēku, mūsu gadījumā tas ir savienojuma nostiepšanas spēku funkcija “force” (nosaukums, darbībā) - darbības nosaukums “link”, jo šī funkcija ietekmēs uz savienojumu lielumu starp virsotnēm. “forceLink()” – aprēķina savienojuma lielumu, “id()” – parāda, ka objektiem jābūt ID, lai tos savienotu. Un “distance(garums)” – savienojuma garums, parametrs, kas izmanto “forceLink()”, lai aprēķinātu savienojuma attālumu. “.force("charge", d3.forceManyBody().strength(-50))” – funkcija ar nosaukumu “charge”, “forceManyBody()” – izmantots, lai virsotnes nepievilktos viena otrai un varētu tās atšķirt, tādējādi uzdodot parametru “strength(-50)”, veidojam atgrūšanas spēku viena no otras (ja būs pozitīvs skaitlis, tad viņi sāks pievilkties viens otram). “force("center", d3.forceCenter(400, 250))” – funkcija ar nosaukumu ““center”, .forceCenter(x,y)” – nosaka grafa koordinātas, mūsu gadījumā tas atrodas centrā. [11]

Lai vizualizētu vairākslāņu grafu tīmekļa lapā, bija izveidots mainīgs “const svg = d3.select(“svg”)” ar nosaukumu “svg”, lai pēc tam izveidotu elementu grupas ar nosaukumu “g”, kur katrs “<g>” ir slānis:

const svg = d3.select("svg");

const layerSelection = svg

.selectAll(".layer")

.data(data)

.enter()

.append("g")

.attr("class", "layer");

,kur “d3.selectAll(".layer")” – atlasam visus elementus ar klasi “.layer” [12], pēc tam tiek izmantota metode “.data(data)”, lai norādītu, kādi dati jāizmanto [17]; “enter()” – pievieno datus masīvā, kas tiks izmantots [18]; “append(“g”)” – izveido elementu grupu ar nosaukumu “g”; “attr("class", "layer")” – katram “g” elementam pievieno klasi “layer” [20].

Nepieciešama funkcija, kura izveido līnijas un var regulēt krāsu un platumu, kas savieno virsotnes:

const linksSelection = layerSelection

.selectAll("line")

.data((d) => d.links)

.enter()

.append("line")

.attr("stroke", "black")

.attr("stroke-width", 0.5);

,kur “const linksSelection = layerSelection” – jauns mainīgais ar nosaukumu “linksSelection”, kas bāzēts uz “layerSelection” mainīgu; “.selectAll(“line”)” – izvēlamies visus elementus ar nosaukumu “line”. “data((d) => d.links)” – piešķir datus no datu masīva. “attr(“stroke”, “black”)” – piešķir krāsu un “attr("stroke-width", 0.5)” – piešķir platumu

Pēc tam, ar to pašu metodi bija izveidotas virsotnes:

const nodesSelection = layerSelection

.selectAll("circle")

.data((d) => d.nodes)

.enter()

.append("circle")

.attr("r", 6)

.attr("fill", (d) => {

switch (d.id) {

case 1:

case 2:

case 7:

return "blue";

case 3:

case 4:

case 8:

return "green";

case 5:

case 6:

case 9:

return "red";

}

});

,kur ar metodi “append(“circle”)” piešķiram jaunu elementu, ar metodi “attr(“r”, 6)” – izveidojam apļa rādiusu. Un ar “attr(“fill”, (d) => { switch (d.id) { case 1: …return “red”;}})” - katrai virsotnei “id” piešķiram savu krāsu (virsotnes, kuras pieder vienam slānim – būs vienā krāsā). [22]

Lai noteiktu, kura tieši ir šī virsotne, bija izveidots elements ar nosaukumu “tooltip”, kas parāda šīs virsotnes “id” un slāni:

const tooltip = d3

  .select("body")

  .append("div")

  .attr("class", "tooltip")

  .style("opacity", 0);

,kur “.select(“body”)” izvēlamies konteineru ar nosaukumu “body”, pēc tam “.append(“div”)” - izvēlamies “div” konteineru, kur “.attr(“class”, “tooltip”)” piešķiram jaunu klasi ar nosaukumu “tooltip” un “.style(“opacity”, 0)” – liekam redzamību uz nulli.

Lai “tooltip” aktivizētos tikai tad, kad kursors tiek uzvilkts uz virsotnes, bija uzrakstīta šāda funkcija:

nodesSelection

  .on("mouseover", (event, d) => {

    const layer = data.find((layer) => layer.nodes.includes(d));

    d3.select("#tooltip")

      .style("visibility", "visible")

      .html(`Slānis: ${layer.layer} Virsotne: ${d.id}`);

  })

  .on("mousemove", (event) => {

    d3.select("#tooltip")

      .style("top", event.pageY - 10 + "px")

      .style("left", event.pageX + 10 + "px");

  })

  .on("mouseout", () => {

    d3.select("#tooltip").style("visibility", "hidden");

  });

,kur uz “nodesSelection” pievienojam metodi “.on("mouseover", (event, d) =>…” - kursora kustības apdarinātājs. “const layer = data.find((layer) => layer.nodes.includes(d))” - funkcija, lai atrastu informāciju par virsotni un slāni. “d3.select("#tooltip").style("visibility", "visible").html(`Slānis: ${layer.layer} Virsotne: ${d.id}`)” – parāda virsotnes un slāņa nosaukumu tīmekļa lapā. Izmanto “on("mousemove", (event) => …”, lai teksts ar informāciju seko kursoram. Lai izveidot attālumu no kursora izmantots “.style("top", event.pageY - 10 + "px").style("left", event.pageX + 10 + "px")”, lai pēc tām teksts pazūd, izmantots “.on(“mouseout”, () => {d3.select("#tooltip").style("visibility", "hidden")}”. [23]

Interaktīvas vairākslāņu grafu izveidei izmanto funkciju “drag”, kas ļauj pārvietot virsotnes. Punkts no kura virsotne kustēsies sākumā:

const drag = d3

.drag()

.on("start", (event, d) => {

if (!event.active) simulation.alphaTarget(0.3).restart();

d.fx = d.x;

d.fy = d.y;

})

Pārvietošanas metode:

.on("drag", (event, d) => {

    d.fx = event.x;

    d.fy = event.y;

  })

Aptur pārvietošanos un atiestata iepriekšējās fiksētās koordinātus:

.on("end", (event, d) => {

if (!event.active) simulation.alphaTarget(0);

d.fx = null;

d.fy = null;

});

Animācijas aktivizācijai tiek izmantota virkne “nodesSelection.call(drag)”.

Beigās ir jāuzraksta divas metodes un vienu funkciju. Pirmā metode: “simulation.nodes(data.flatMap((d) => d.nodes))”, kur visa informācija par virsotnēm un slāņiem tiek nodota simulācijai. [24]

Otrā metode: “simulation.force("link").links(data.flatMap((d) => d.links))”, kur ir parādīti visi savienojumi starp virsotnēm, lai to simulētu.

Pēdēja funkcija nosaka virsotņu un saišu koordinātas simulācijā: [11]

simulation.on("tick", () => {

  linksSelection

    .attr("x1", (d) => d.source.x)

    .attr("y1", (d) => d.source.y)

    .attr("x2", (d) => d.target.x)

    .attr("y2", (d) => d.target.y);

  nodesSelection.attr("cx", (d) => d.x).attr("cy", (d) => d.y);

});

Bija uzlabots “HTML” dokuments:

    <p>BAKALAURA DARBS - Dmitrijs Sinickis 201RDB360</p>

    <div id="svgContainer">

        <svg width="800" height="500"></svg>

        <div id="tooltip" style="position: absolute; visibility: hidden;"></div>

    </div>

    <script src="script.js"></script>

    <p>D3.js piemērs</p>

Pievienota informācija par autoru, kāda bibliotēka tika izmantota un rezultāts ir vairākslāņu grafu vizualizācija. – sk. 3.2.1. att.

A screenshot of a computer

Description automatically generated with medium confidence

3.2.1. attēls. Rezultāts ar D3.js bibliotēku.

3.2.1 Secinājumi par D3.js bibliotēku

“D3.js” ir ērts instruments, kur var regulēt dažādas detaļas, sākot no virsotnes un šķautnes izskata līdz regulējumam attāluma starp slāņiem un punktiem. “D3.js” ir iespēja izveidot no statiskā grafa līdz interaktīvam grafam. Dokumentācija ir detalizēta un saprotama. Negatīvās puses: grafa uzsākšanas darbībai ir nepieciešams uzrakstīt daudz parametru un mainīgo, kā rezultātā var viegli sajaukt. Kā arī nevar izveidot slāni izmantojot krāsu, kas atvieglotu darbu.

3.3. Cytoscape.js bibliotēka

“Cytoscape.js” ir pilna funkcionāla teorijas grafu bibliotēka, kura palīdz vizualizēt un modelēt dažādus datus, piemēram: bioloģiskos vai sociālajos tīklos. Lai vizualizētu vairākslāņu grafu, nepieciešams izveidot “HTML”, “CSS” un “JavaScript” dokumentus. “HTML” dokumentā bija pievienots “div” konteineris ar “id” nosaukumu “cy”, lai vizualizētu skripta rezultātus tīmekļa lapā. [25]

“JavaScript” dokumentā vispirms vajag pievienot mainīgu ar nosaukumu “cy”, kur būs savienojums starp konteineru un skriptu. Izmantojot DOM modeli:

let cy = cytoscape({

container: document.getElementById("cy")…

Nākamais solis, jāpievieno slāņu datus, mūsu gadījuma būs trīs slāņi, ar nosaukum “a”, ”b” un ”c”: [25]

elements: [

    {

      data: { id: "a" },

    },

    {

      data: { id: "b" },

    },

    {

      data: { id: "c" },

    },]…

Lai būtu viegli atšķirt slāni, ir jāpievieno krāsu, mūsu gadījumā tas būs debeszila krāsa (jeb HEX formāta: #aadcff): [25]

 style: [

    {

      selector: ".layers",

      style: {

        "background-color": "#aadcff ",

Label: “data(id)”,

      }….

,kur ”Label” ir data(id), lai parādītu visus “id” nosaukumus.

Pēc tam bija pievienoti mainīgie ar nosaukumu “nodes1”, “nodes2” un “nodes3”. Tā ir funkcija, kura pievieno dažādas virsotnes: [25]

let nodes1 = cy.add([

  {

    data: { id: "a1", parent: "a" },

    position: { x: 0, y: -50 },

    classes: "node1",

  },

{

    data: { id: "a2", parent: "a" },

    position: { x: 50, y: -100 },

    classes: "node1",

  }…

,kur “cy.add()” – pievieno elementus grafā, mūsu gadījumā pievieno virsotnes ar “id” nosaukumiem “a1”, “a2” un klasi “node1”, kurš pieder slānim ar “id” nosaukumu “a”, pirmās virsotnes sākumpunkts ir ass x = 0 un ass y = -50 un otrās virsotnes sākumpunkts ir ass x = 50 un ass y = -100. Līdzīgi bija izveidota trešā virsotne.

Mainīga ar nosaukumu “cy” bija pielikta nodaļā “style” jauns stils, lai izveidotu atšķirības starp virsotnēm no dažādiem slāņiem: [25]

 {

      selector: ".node1",

      style: {

        "background-color": "blue",

      },

    },

    {

      selector: ".node2",

      style: {

        "background-color": "green",

      },

    }…

Lai savienotu visas virsotnes, bija uzrakstīta funkcija ar mainīgo nosaukumu “edges”, kura izveido šķautnes starp virsotnēm:

let edges = cy.add([

  {

    data: { id: "edge0", source: "b1", target: "a1" },

  },

  {

    data: { id: "edge1", source: "b3", target: "c1" },

  },

  {

    data: { id: "edge2", source: "b1", target: "b2" },

  },

  {

    data: { id: "edge3", source: "b1", target: "b3" },

  },

  {

    data: { id: "edge4", source: "a1", target: "a2" },

  }…

,kur “id: “x”” - katrai šķautnei pievieno nosaukumu, “source: “x”” - no kuras virsotnes sākas savienojums, “target: “x”” – kurā virsotnē beidzās savienojums. [25]

Analoģiski situācijai ar virsotnēm, bija pievienots stils, lai varētu redzēt šķautnes:

selector: "edge",

      style: {

        width: 1,

        "line-color": "black",

        "curve-style": "straight",

        "label": "",

      },

,kur “width: 1” - ir platums 1 pikselis, ““line-color”: “black”” – ir šķautnes melna krāsa, ““curve-style”: “straight”” – ir līnijas tips, mūsu gadījumā tā ir taisna līnija. Un ““label”: “”” – ir tukšs, lai būtu redzami tikai virsotņu un slāņu nosaukumi, bet ne šķautnes. [25]

Lai vairākslāņu grafs paradās korekti un ir kā interaktīvs grafs, bija izveidota funkcija ar nosaukumu “animation”, kur izmantota metode “cy.animate()”, lai ar parametru “pan” novietot noteiktā vietā un, lai redzētu vairākslāņu grafu pilnīgi, bija izmantots parametrs ar nosaukumu “zoom”: [25]

var animation = cy.animate({

  pan: {x: 400, y: 250},

  zoom:1,

})

“HTML” dokumentā pievienota informācija par autoru un izmantojama bibliotēka:

<body>

    <p>BAKALAURA DARBS - Dmitrijs Sinickis 201RDB360</p>

    <div id="cy"></div>

    <script src="script.js"></script>

    <p>Cytoscape.js piemērs</p>

</body>

Prototipa integrācijas rezultāts ar cytoscape.js bibliotēku. – sk. 3.4.1. att.

A screenshot of a computer

Description automatically generated

3.3.1. attēls. Rezultāts ar Cytoscape.js bibliotēku.

3.3.1. Secinājumi par Cytoscape.js bibliotēku

“Cytoscape.js” tā ir bibliotēka, kura ļauj izveidot jebkādu vairākslāņu grafu, kur ir redzami slāņi, viegli paradīt slāņu un virsotnes nosaukumu. Var regulēt virsotnes krāsu, šķautnes platumu un krāsu, ari ir iespēja izveidot slāni ar savu krāsu, lai vizuāli būtu saprotams, kur ir kāds slānis. Ar “Cytoscape.js” bibliotēku var izveidot interaktīvu grafu, bet tanī nebūs sarežģītas animācijas.

3.4. Vis.js bibliotēka

“Vis.js” ir dinamiska vizualizācijas bibliotēka, kura pamatojas uz tīmekļa lapu. Bibliotēka izveidota, lai vizualizētu dažādu veidu un apjoma datus. Ar šo bibliotēku var izveidot dažādus vizualizācijas tipus un arī var veidot vairākslāņu grafu. Lai vizualizētu šo vairākslāņu grafu, ir nepieciešams izveidot “HTML”, “CSS” un “JavaScript” formāta dokumentus. Vispirms tika izveidots “div” konteineris ar “id” nosaukumu “graph” “HTML” dokumentā, lai skripta rezultāti parādās tīmekļa lapā. [26]

Pirms datu vizualizēšanas vairākslāņu grafa saglabāšanai tika izveidots mainīgais ar nosaukumu “nodes”, kur bija pievienoti dati ar funkciju “vis.DataSet”.

let nodes = new vis.DataSet([

  { id: 1, label: "Node 1", group: "group1" },

  { id: 2, label: "Node 2", group: "group1" },

  { id: 3, label: "Node 3", group: "group1" }…

,kur “id: x” – ir datu apzīmējums, “label: “x”” – ir datu nosaukums, “group: “x”” – ir grupu nosaukums, kurām pieder šī virsotne. [27]

Ar tādu pašu funkciju “vis.DataSet” bija izveidotas šķautnes starp virsotnes mainīgo ar nosaukumu “edges”.

let edges = new vis.DataSet([

  { id: 1, from: 1, to: 2 },

  { id: 2, from: 1, to: 3 },

  { id: 3, from: 2, to: 3 },

  { id: 4, from: 1, to: 4 }…

,kur “id: x” – ir šķautnes apzīmējums, “from: x” – ir no kurās virsotnēs sākās savienojums, “to: x” – ir kurā virsotnē beidzas savienojums. [27]

Pēc tam bija pievienota metode ar nosaukumu “layers”, kura sadala virsotnes uz krāsām, lai būt vieglāk tos atšķirt. Krāsu pievienošana noteiktai grupai:

let layers = {

  groups: {

    group1: { color: "blue"},

    group2: { color: "green"}…

Lai uzlabotu teksta redzamību virsotnēs, bija izmantota teksta krāsas rediģēšanas opcija. [28]

nodes: {

    font: {

      color: "white"

    }}

Pēdējais posms, lai iegūtu rezultātu, bija izveidots mainīgais ar nosaukumu “data”, kas ir datu objekts ar informāciju par virsotni un šķautni. [27]

let data = {

  nodes: nodes,

  edges: edges

};

Viss rezultāts realizēts konteinerā, kurš bija izveidots “HTML” dokumentā:

let container = document.getElementById("graph");

Un visa simulācija notiek mainīgajā “network”, kur nodota informācija par konteineru, virsotnēm, šķautnēm un slāņiem. [27]

let network = new vis.Network(container, data, layers);

Prototipa integrācijas rezultāts ar vis.js bibliotēku. – sk. 3.4.1. att.

A screenshot of a computer

Description automatically generated with medium confidence

3.4.1. attēls. Rezultāts ar Vis.js bibliotēku.

3.4.1 Secinājumi par Vis.js bibliotēku

“Vis.js” bibliotēka var viegli izveidot vairākslāņu grafu, nav vajadzības rakstīt daudz elementu, lai simulētu grafu. Ir iespējams regulēt virsotnes krāsas un nosaukumi, visi dati nosaukumi jau ir integrēti grafā. Interaktīvs grafs automātiski izveidos. Bet nevar detalizēti regulēt pozīciju vai virsotnes formu, ari nav iespējams vizualizēt slāni ar savu krāsu.

Nepieciešamo funkciju novērtēšana rezultāts ir dots 3.1. tabulā.

3.1. tabula

Bibliotēku vērtēšanas kopsavilkums

|  |  |  |  |
| --- | --- | --- | --- |
| Funkcijas | D3.js | Cytoscape.js | Vis.js |
| Virstoņu apstrāde | + | + | + |
| Šķautņu apstrāde | + | + | + |
| Slāņu apstrāde | - | + | - |

4. Eksperimentāls salīdzinājums

Pirms eksperimenta veikšanas nepieciešams izveidot vienādus nosacījumus un kritērijus.

**Kopīgie nosacījumi:**

Katrai bibliotēkai ir vajadzīgs vizualizēt 1200 virsotnes, 3 slāņos un 600 šķautnes. Nav nozīmes kādā secībā tiks savienotas virsotnes un kādos slāņos tās atradīsies. Kritēriji, pēc kuriem tiks novērtēts rezultāts, ir norādīti 4.1. tabulā.

4.1. tabula

Kritēriji veidi

|  |  |  |
| --- | --- | --- |
| Kritērija numurs | Kritēriji | Apraksts |
| 1 | Algoritma ātrums | Cik ātri algoritms ielādējas lapā, lai to pārbaudītu, tiek izmantota “Google Chrome” pārlūkprogramma, kur ir integrēta funkcija ar nosaukumu “Performance”, kas nozīmē tīmekļa lapas veiktspēju. Rezultātā var redzēt koda veiktspēju un cik ātri lapa tiek ielādēta. Parametru saraksts:   * Iekraušana (jeb “Loading”) – parametrs, kas parāda patērētā “HTML”, “CSS”, “JavaScript” laika resursu lejupielādi. * Skriptēšana (jeb “Scripting”) – parameters, kas parāda, cik daudz laika patērēts, lai izpildītu “JavaScript” kodu. * Renderēšana (jeb “Rendering”) – parametrs, kas parāda, cik daudz laika patērēts, lai lietotu “CSS” stilus un novietotu elementus lapā. * Vizualizācija (jeb “Painting”) – parametrs, kas parāda patērētu laiku uz pikseļu aizpildīšanu ekrānā. Izveidošana un atjaunošanā tīmekļa lapas vizuālo izvadi. * Sistēma (jeb “System”) – parametrs, kas parāda patērēto laiku, veicot pārlūkprogrammas, operētājsistēmas un datora komponentu sistēmas darbības. Darbības ir notikumu apstrāde, montāža un citas operācijas. * Tukšgaitā (jeb “Idle”) – parametrs, kas parāda patērēto laiku, pārlūkprogrammu datu ievades gaidīšana un tīkla atbildes. * Kopsumma (jeb “Total”) – ir visu parametru laika summa, kas nepieciešama, lai ielādētu un parādīt lapu. [29] |
| 2 | Algoritma garums | “JavaScript” formāta faila lielums un rindu skaits |
| 3 | Grafu interaktivitāte | Mijiedarbība ar vairākslāņu grafu vizualizācijai. Cik daudz animācijas ir un vizuāli rāda virsotnes nosaukumu. |

4.1. Eksperimentu objekts

Lai veiktu eksperimentu korekti, trīs dažādi algoritmi ar savu bibliotēku bija modificēti. Vispārējais koda uzlabošanas princips ir ciklu pievienošana, kas automātiski ģenerē vajadzīgo virsotnes un šķautnes daudzumu. Tiek izmantots ari “Math” metode kodā, lai uzģenerēt nejaušus virsotnes nosaukumus un šķautnes savienojumu.

4.1.1. D3.js bibliotēka koda modificēšana

Lai veiktu eksperimentu ar d3.js kodā tika pievienoti divi cikli. Pirmais cikls veido noteiktu slāņu skaitu un pašu slāni, kā objektu , kur būs dati par slāņa numuru, virsotnes un šķautnes:

      let layerCount = 3;

      for (let layerIndex = 1; layerIndex <= layerCount; layerIndex++) {

        let layer = {

          layer: layerIndex,

          nodes: [],

          links: [],

        };

,kur “const layerCount = 3” – ir mainīga ar nosaukumu “layerCount”, kur norāda slāņu skaitu, mūsu gadījumā tas ir 3 slāņi. Cikls “for” tiek izmantots, lai izveidotu katru slāni, tiks izpildīts, līdz “layerIndex” viendāds ar “layerCount” skaitu. Ciklā tiek izveidots objekts ar nosaukumu “layer”, kas būs slānis. Objekta parametri: “layer: layerIndex” – informācija par slāņa numuru, “nodes: []” – tas ir tukšs masīvs, kur saglabās datus par virsotnēm (pagaidām nekas nav), “links: []” – tukšs masīvs, kur būs informācija par šķautnēm.

Otrais cikls izveido noteiktu virsotņu skaitu un nejaušā secībā sadala slāņos. Katrai virsotnei pievieno nosaukumu un atrašanās vietas koordinātes:

let nodeCount = Math.floor(1200 / layerCount);

let startNodeId = (layerIndex - 1) \* nodeCount + 1;

        for (let nodeIndex = 1; nodeIndex <= nodeCount; nodeIndex++) {

          let nodeId = startNodeId + nodeIndex - 1;

          let node = {

            id: nodeId,

            x: Math.floor(Math.random() \* 400) + 50,

            y: Math.floor(Math.random() \* 300) + 50,

          };

if (layerIndex > 1) {

            let randomNodeIndex = Math.floor(

              Math.random() \* data[layerIndex - 2].nodes.length

            );

            let randomNode = data[layerIndex - 2].nodes[randomNodeIndex];

            layer.links.push({

              source: nodeId,

              target: randomNode.id,

            });

          }}

,kur pirmā mainīgā ar nosaukumu “nodeCount” – nosaka virsotņu skaitu (mūsu gadījumā tas ir 1200 virsotnes) un sadala slāņos, “Math.floor” tiek izmantots, lai katrā slānī būtu proporcionāls virsotnes daudzums. Otrā mainīgā ar nosaukumu “startNodeId” – ir izmantota, lai pievienotu katrai virsotnei savu unikālu “id” nosaukumu. Cikls “for” veido virsotnes katrā slānī tiks izpildīts līdz momentam, kad “nodeIndex” būs vienāds ar “nodeCount”. Ciklā ir mainīga ar nosaukumu “nodeId” – notiek identifikatora aprēķināšana caur formulu “startNodeId + nodeIndex -1”, lai būt pareizs “id” numurs. Pēc tam ciklā veidots objekts ar nosaukumu “node” (tas ir virsotne) ar parametriem: “id” – unikāls nosaukums, “x: Math.floor…” – koordināti x asi, un “y:Math.floor…” – koordināti y asi. Tiek izmantots “Math.floor”, lai būtu laidena pārvietošana animācija. “Math.random” – veido nejauši koordinātus, reizināšanu un saskaitīšanu, lai pievienotu attālumu starp virsotnēm. Pēc tam tiek izmantots operātors “if” ar nosacījumu “layer >1”, lai nebūtu savienojuma ar pirmo slāni, jo tas ir pirmais un visas šķautnes nāk no pirmā slāņa, nevis otrādi. Pēc tam mainīgā “randomNodeIndex” ģenerē nejaušu indeksa numuru masīva garuma robežas (virsotnes no iepriekšējā slāņa), mainīgā “randomNode” tiek izmantots, lai atrastu virsotni ar nejaušu indeksu no “randomNodeIndex”. Bija izveidots objekts ar parametriem “source” un “target”, kur “source” nozīmē sākumpunkts un “target” ir virsotne, kur savienojums beidzas.

Lai būtu redzams no jebkura rakursa, ir pievienota funkcija ar kursora pārvietošanu:

const zoom = d3.zoom().on("zoom", (event) => {

        layerSelection.attr("transform", event.transform);

      });

,kur mainīgā ar nosaukumu “zoom” – ir izveidots objekts ar funkciju “d3.zoom” (tas ir funkcija no “D3.js” bibliotēkas). “on(“zoom”, (event)…” – tas ir metodes izsauksme, kurš pievieno notikumu apdarinātāju. Funkcijā ir “layerSelection.attr…” – ir atribūts, kurš satur informāciju par mērogošanu un pārvietošanu.

4.1.2. Cytoscape.js bibliotēka koda modificēšana

Eksperimenta veikšanai tika modificēts kods ar “Cytoscape.js” bibliotēku. Bija pievienoti 2 cikli un modificēta viena funkcija.

Pirmais cikls veido 1200 :

let nodes1 = [];

let nodes2 = [];

let nodes3 = [];

for (let i = 1; i <= 400; i++) {

  let nodeId = i.toString();

  let nodeA = { data: { id: "a" + nodeId, parent: "a" }, classes: "node1" };

  let nodeB = { data: { id: "b" + nodeId, parent: "b" }, classes: "node2" };

  let nodeC = { data: { id: "c" + nodeId, parent: "c" }, classes: "node3" };

  nodes1.push(nodeA);

  nodes2.push(nodeB);

  nodes3.push(nodeC);

}

cy.add(nodes1);

cy.add(nodes2);

cy.add(nodes3);

,kur mainīgie “nodes1”, “nodes2”, “nodes3” – ir tukšie masīvi. Cikls “for” pievienos 400 virsotnes katrā slānī. Mainīgais “nodeId” izveido “i” kā tekstu. Mainīgie “nodeA”, “nodeB”, “nodeC” pievienoti objekti ar virsotnes parametriem (nosaukums, “id” numurs un slānis). Tālāk visi mainīgie ar parametriem pārvietojas uz tukšiem masīviem ar metodi “push”. Pēc tam “cy.add” pievieno virsotnēm grafā.

Otrais cikls izveido 600 šķautnes, kas nejauši savieno virsotnes:

let nodes = cy.nodes();

let nodeCount = nodes.length;

for (let i = 0; i < 600; i++) {

        let sourceNode = nodes[Math.floor(Math.random() \* nodeCount)];

        let targetNode = nodes[Math.floor(Math.random() \* nodeCount)];

        if (sourceNode.id() !== targetNode.id()) {

          cy.add([

            {

              data: {

                id: sourceNode.id() + "-" + targetNode.id(),

                source: sourceNode.id(),

                target: targetNode.id(),

              },

            },

          ]);

        }

      }

,kur mainīgais “nodes” – iegūt visas grafa virsotnes, mainīgais “nodeCount” – iegūt virsotņu masīva garumu. Cikls “for” izmantots, lai izveidotu 600 šķautnes nejaušā secībā starp diviem virsotnēm. Mainīgais “sourceNode” – iegūst nejaušu virsotni no masīva “nodes” un tas ir sākumpunkts. Mainīgais “targetNode” – iegūst nejaušu virsotni no masīva “nodes” un virsotnes, kur savienojums beidzas. Operators “if” pārbauda, lai izvēlētās virsotnes nav vienādas. Ja virsotnes nav vienādas, tad notiks algoritms blokā. Metodē “cy.add”, kur atrodas objekts ar datiem par šķautni: “source” – virsotnes sākumpunkts, “target” – virsotnes, kur savienojums beidzās.

Tika modificēta funkcija, lai virsotnes izveido nejaušās koordinātes:

 let layout = cy.layout({

        name: "random",

        spacingFactor: 2,

      });

,kur mainīgais “layout” ir “cy.layout” metode, lai izveidotu objektu ar parametriem “name” – tas nozīmē virsotnes pozīciju, mūsu gadījumā bija izvelēta “random”, lai virsotnēm būtu nejaušas koordinātes un “spacingFactor” – tas ir attālums starpēm.

4.1.3. Vis.js bibliotēka koda modificēšana

“Vis.js” bibliotēka koda bija modificēta, lai veiktu eksperimentu, tika pievienoti 3 cikli.

Pirmais cikls ģenerē 1200 virsotnes ar savu nosaukumu:

let nodes = new vis.DataSet();

for (var i = 1; i <= 1200; i++) {

        nodes.add({

          id: i,

          label: "Node " + i,

          group: "group" + Math.floor(Math.random() \* 3) + 1,

        });

,kur mainīgais ”nodes” izveido jaunu objeku ar konstruktoru “vis.DataSet()”, tas ir objekts, kur saglabājās virsotnes. Cikls “for” izveidos 1200 virsotnes, “nodes.add” pievienos jaunas virsotnes objektā “nodes” ar parametriem “id” – virsotnes numurs, “label” – virsotnes nosaukums un “group” ar nejaušu secība pievienos vienam no trim slāņiem.

Otrais cikls veido 600 šķautnes nejaušā secībā:

let edges = new vis.DataSet();

for (var i = 1; i <= 600; i++) {

let from = Math.floor(Math.random() \* 1200) +1;

let to = Math.floor(Math.random() \* 1200) +1;

if (from != to) {

edges.add({

id: i,

from: from,

to: to,

        });

}

}

,kur mainīgais ”edges” izveido jaunu objektu ar konstruktoru “vis.DataSet()”, tas ir objekts, kur saglabās šķautnes. Cikls “for” izveidos 600 šķautnes. Mainīgais “from” ģenerē nejaušu sākumpunktu, mainīgais “to” ģenerē nejaušu galapunktu. Operators “if” pārbauda, ja sākumpunktu un galapunktu nav vienādi. Ja tādi virsotnes nav vienādās, tad pievienos objektā “egdes” ar parametriem: “id” – virsotnes numuru, “from” – virsotnes sākumpunkts un “to” virsotnes galapunkts.

Trešais cikls sadala virsotnes uz slāņos:

for (var i = 0; i < groups.length; i++) {

        let group = groups[i];

        let color = layerColors[Math.floor(Math.random() \* 3)];

        layers.groups[group] = { color: color };

      }

,kur cikls “for” izveido un pievieno dati katrai grupai, mainīgais “group” iegūst masīvu elementu ar indeksu “i”, mainīgais “color” pievieno nejaušu krāsu katrai virsotnes grupai. Un krāsu parametrs saglabās “layers.groups”

4.2. Eksperiments

Lietotā specifikācijas iekārta: “Windows 11” versijā 22H2, procesors: Intel Core i9-9900KF, operatīva atmiņa: 48 GB ar ātrumu 3200 Mhz, video kartes atmiņa: 12 GB. Viss kods tiek izpildīts programmā “Visual Studio Code” versijā 1.78.2., tīmekļa lapa ģenerēta pārlūkprogrammā “Google Chrome” versijā 113.0.5672.127.

4.2.1 D3.js eksperiments

Eksperiments tika veikts ar “D3.js” bibliotēku. Bija vizualizētas 1200 virsotnes un 600 šķautnes, savienojumi tika izveidoti nejaušā secībā. Virsotnes tika sadalītas trīs slāņos nejaušā secībā. Var redzēt vienkāršu vairākslāņu grafu, kur ir neorientētas šķautnes, tāpēc pilnīgi var nosaukt, ka tas ir vienkāršs neorientēts vairākslāņu grafs. Algoritma ātrumu rezultāti pēc “Google Chrome” funkcijas “performance” izmantošanas rāda, ka visvairāk laika aizņēma “JavaScript” koda izpildīšana – 3595 milisekundes. Vismazāk laika aizņēma parametrs “Iekraušana” – 5 ms, tas ir ļoti ātri, kas nozīme, ka failu izmēri neietekmēja lejupielādi. Visi algoritma ātrumu rezultāti ir norādīti 4.2.1. tabulā.

4.2.1. tabula

D3.js algoritma ātrums

|  |  |
| --- | --- |
| Parametrs | Aizņem laiku (milisekunde) |
| Iekraušana | 5 ms |
| Skriptēšana | 3595 ms |
| Renderēšana | 1339 ms |
| Vizualizācija | 248 ms |
| Sistēma | 51 ms |
| Tukšgaitā | 21 ms |
| Kopsumma | 5259 ms |

“D3.js” algoritma faila izmēru un rindas kodā skaitu parādīti 4.2.2. tabulā.

4.2.2. tabula

D3.js algoritma garums

|  |  |
| --- | --- |
| Algoritma faila izmērs (KB) | Rindas kodā |
| 3.94 | 83 |

**Grafu interaktivitāte:** Var redzēt katras virsotnes nosaukumu un redzēt, pie kura slāņa pieder. Virsotnes var pārvietot jebkurā virzienā. – sk. 4.2.1. att.

Attēls, kurā ir ekrānuzņēmums, krāsainība

Apraksts ģenerēts automātiski

4.2.1. attēls. D3.js vairākslāņu grafu animācija.

Vispārējais vizualizācijas un veiktspējas rezultāts. – sk. 4.2.2. att.

A screenshot of a computer

Description automatically generated

4.2.2. attēls. D3.js eksperimenta rezultāts.

4.2.2 Cytoscape.js eksperiments

Eksperiments tika veikts ar “Cytoscape.js” bibliotēku. Bija vizualizētas 1200 virsotnes un 600 šķautnes, savienojumi tika izveidoti nejaušā secībā. Virsotnes tika sadalītas trijos slāņos, pirmajā slānī ir 400 virsotnes, otrajā slānī 400 virsotnes un trešajā slānī 400 virsotnes. Var redzēt vienkāršu vairākslāņu grafu, kur ir neorientētas šķautnes, tāpēc pilnīgi var nosaukt, ka tas ir vienkāršs neorientēts vairākslāņu grafs. Algoritma ātrumu rezultāti pēc “Google Chrome” funkcijas “performance” izmantošana rāda, ka daudz laiku aizņem parametrs “Tukšgaitā” – 4740 ms. Visātrāk notiek “Iekraušana” – 4 ms. Visi “Cytoscape.js” algoritma ātrumu rezultāti ir 4.2.3. tabulā.

4.2.3. tabula

Cytoscape.js algoritma ātrums

|  |  |
| --- | --- |
| Parametrs | Aizņem laiku (milisekunde) |
| Iekraušana | 4 ms |
| Skriptēšana | 1653 ms |
| Renderēšana | 6 ms |
| Vizualizācija | 12 ms |
| Sistēma | 42 ms |
| Tukšgaitā | 4740 ms |
| Kopsumma | 6457 ms |

“Cytoscape.js” algoritma faila izmēru un rindas kodā skaitu parādīti 4.2.4. tabulā

4.2.4. tabula

Cytoscape.js algoritma garums

|  |  |
| --- | --- |
| Algoritma faila izmērs (KB) | Rindas kodā |
| 2.16 | 32 |

**Grafu interaktivitāte:** Var redzēt katras virsotnes nosaukumu un redzēt katru virsotni un slāni. Virsotnes un slāni var pārvietot uz jebkuru vietu. – sk. 4.2.3. att.

Attēls, kurā ir Elektriski zila, Majorelle zilā krāsa, krāsainība, kobaltzils

Apraksts ģenerēts automātiski

4.2.3. attēls. Cytoscape.js vairākslāņu grafu animācija.

“Cytoscape.js” bibliotēkas eksperimenta kopējais rezultāts – sk. 4.2.4. att.

A screenshot of a computer

Description automatically generated

4.2.4. attēls. Cytoscape.js eksperimenta rezultāts.

4.2.3 Vis.js eksperiments

Eksperiments tika veikts ar “Vis.js” bibliotēku. Bija vizualizēti 1200 virsotnes un 600 šķautnes, savienojumi un virsotnes izveidoti nejauša secība savienojumi tika izveidoti nejaušā secībā. Virsotnes tika sadalītas trīs slāņos nejaušā secībā. Var redzēt vienkāršu vairākslāņu grafu, kur ir neorientētas šķautnes, tāpēc pilnīgi var nosaukt, ka tas ir vienkāršs neorientēts vairākslāņu grafs. Algoritma ātrumu rezultāti pēc “Google Chrome” funkcijas “performance” izmantošanas rāda, ka visvairāk laika aizņēma “Skriptēšana” – 14595 ms. Parametrs ar vismazāko rezultātu ir “Vizualizācija” – 0 ms. “Vis.js” algoritma ātrumu rezultāti ir 4.2.5. tabulā.

4.2.5. tabula

Vis.js algoritma ātrums

|  |  |
| --- | --- |
| Parametrs | Aizņem laiku (milisekunde) |
| Iekraušana | 3 ms |
| Skriptēšana | 14595 ms |
| Renderēšana | 3 ms |
| Vizualizācija | 0 ms |
| Sistēma | 15 ms |
| Tukšgaitā | 40 ms |
| Kopsumma | 14657 ms |

Vis.js algoritma faila izmēru un rindas kodā skaitu parādīti 4.2.6. tabulā.

4.2.6. tabula

Vis.js algoritma garums

|  |  |
| --- | --- |
| Algoritma faila izmērs (KB) | Rindas kodā |
| 1.10 | 18 |

**Grafu interaktivitāte:** Var redzēt katras virsotnes nosaukumu un redzēt, pie kura slāņa pieder. Virsotnes var pārvietot jebkurā virzienā. – sk. 4.2.5. att.

Attēls, kurā ir bērnu māksla, aplis, rinda

Apraksts ģenerēts automātiski

4.2.5. attēls. Vis.js vairākslāņu grafu animācija.

“Vis.js” bibliotēkas eksperimenta kopējais rezultāts – sk. 4.2.6. att.

A screen shot of a computer

Description automatically generated with medium confidence

4.2.6. attēls. Vis.js eksperimenta rezultāts.

4.3. Eksperimentu rezultāti

Pēc eksperimenta veikšanas, var secināt, ka algoritms ar “D3.js” bibliotēku ir ātrāks nekā “Cytoscape.js” bibliotēka un “Vis.js” bibliotēka. Ar 3 slāņiem, 1200 virsotnēm un 600 šķautnēm – “D3.js” algoritms augšupielādēts tīmekļa lapā ar laiku 5259 ms, “Cytoscape.js” par 1198 ms izrādījās lēnāks, bet tā nav būtiska atpalicība no “D3.js”. “Vis.js” algoritms ar 1200 virsotnēm un 600 šķautnēm tika galā tikai par 14657 ms, tas ir vairāk nekā divreiz ilgāk, ja salīdzināt ar “D3.js” algoritmu. Tāpēc "Vis.js” lejupielādes ātrums ir vissliktākais. – sk. 4.3.1. att.

4.3.1. attēls. Algoritma ātruma rezultāts, cik milisekundēs ielādēti algoritmi tīmekļa lapā

Ja detalizēti salīdzina trīs bibliotēkas pēc katra parametra, rezultāti sanāk šādi (4.3.2. attēls):

* Parametrs “Iekraušana” visām trim bibliotēkām ir aptuveni vienāds laiks, bet “D3.js” uz 2 ms ielādējās ilgāk nekā “Cytoscape.js” – 3 ms.
* Parametrs “Skriptēšana” ir vāja vieta “Vis.js” bibliotēkā, ja salīdzina ar trim bibliotēkām, tad "Vis.js” skripts tiek realizēts visilgāk – 14595 ms. Visātrākā “Skriptēšana” izpildīta “Cytoscape.js” bibliotēkā – 1653 ms.
* “Renderēšana” pie “Cytoscape.js” un “Vis.js” gandrīz ir vienādā ātrumā – 6 ms un 3 ms, realizācija notiek ātrāk nekā “D3.js” – 1339 ms.
* “Vizualizācija” – minimālais rezultāts “Vis.js” bibliotēkai, tas ir 0 ms, jo "Vis.js” bibliotēkā nav instrumenta, kas regulē virsotnes, šķautnes un slāņu izskatu. Tāpēc korektāk ir salīdzināt “Cytoscape.js” un “D3.js”, kur “D3.js” bibliotēka ģenerē vizualizāciju par 236 ms ilgāk.
* “Sistēmā” rezultāts ir ātrāk “Vis.js” – 15 ms. “D3.js” bibliotēkas apstrādei nepieciešams ilgāks laiks – rezultāts 51 ms un cytoscape.js par 9 sekundēm ātrāk nekā "D3.js” bibliotēka.
* “Kopsummas” rezultāti minēti augstāk.

4.3.2. attēls. Detalizēts algoritmu ātrumu rezultāts

Algoritma garuma rezultāti ir tādi, ka “D3.js” algoritmā ir 83 koda rindas un "Javascript” faila izmērs ir 3.94 KB, tas ir vairāk nekā “Cytoscape.js” algoritms un “Vis.js” algoritms. “Vis.js” algoritmā ir vismazāk koda rindas – 18 un faila izmērs ir 1.10 KB. “Cytoscape.js” algoritmā ir 32 rindas kodā un 2.16 KB faila izmērs. Tas nozīme, ka “Vis.js” var aizņemt mazāk vietas kaut kāda projektā nekā ar “D3.js” vai “Cytoscape.js”. Par algoritmu garumu kritēriju var secināt, ka “D3.js” ir vislielākais, bet izmērs ir tik nenozīmīgs, ka nevar globāli ietekmēt uz reālo projektu. – sk. 4.3.3. att.

4.3.3. attēls. Algoritma garums rezultāts, faila izmērs (kilobaitos) un cik ir rindas kodā.

Par grafu interaktivitāti, katra bibliotēka ļauj mijiedarboties ar virsotnēm, detalizēti apskatīt vairākslāņu grafu, redzēt virsotņu nosaukumu “D3.js” un “Cytoscape.js” var redzēt slāņu nosaukumu, bet tikai “Cytoscape.js” bibliotēka ļauj mijiedarboties ar veselu slāni, jo “Cytoscape.js” bibliotēka viskorektāk attēlo vairākslāņu grafu atšķirībā no “D3.js” un “Vis.js” bibliotēkas. – sk. 4.3.4 att. un 4.3.1. tabulā.

Attēls, kurā ir taisnstūris, raksts, ekrānuzņēmums, tirkīzs

Apraksts ģenerēts automātiski

4.3.4. attēls. Spēja darboties ar visu slāni Cytoscape.js bibliotēkā.

4.3.1. tabula

Grafu interaktivitāte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Bibliotēka | Mijiedarbība ar virsotnēm | Mijiedarbība ar slāņiem | Detalizēts apskats | Virsotnes nosaukums | Slāņa nosaukums |
| D3.js | + | - | + | + | + |
| Cytoscape.js | + | + | + | + | + |
| Vis.js | + | - | + | + | - |

Ja būs izvēle, kādu no trim bibliotēkām labāk integrēt tīmekļa lapā, prioritātē būs “D3.js” bibliotēka, jo tā ātrāk strādā un nav problēmu ar lielu datu apjomu, bet vizuāli “D3.js” bibliotēka neizskatās kā kanoniska vairākslāņu grafa un vizuāli sarežģīti sadalīt veselus slāņus atšķirībā no “Cytoscape.js” bibliotēkas.

“Cytoscape.js” bibliotēka nepiekāpjas darba ātrumā un pareizāk vizualizēts vairākslāņu grafs, ka ir iespējams redzēt ar savu krāsu slāni un ka virsotnes no savas grupas atrodas viena slānī. Vieglāk regulēt, jo nav tik daudz mainīgo, lai simulētu vairākslāņu grafu nekā “D3.js” bibliotēkā. Bet ja ir neliela datu kopa un ir mērķis ātri un viegli integrēt mājaslapā, tad ieteikums izmantot “Vis.js” bibliotēku, jo tas ir viegli lietojams rīks. Ar tādu pašu funkcionāli kā “D3.js” bibliotēka.

Secinājumi

* Vairākslāņu grafi ir piemērots instruments sarežģītu sistēmu modelēšanai un dažādu tipu savstarpējo saišu analīzei. Tie attēlo dažādus reālās pasaules aspektus un palīdz izprast un paredzēt sistēmu uzvedību.
* Visas trīs “JavaScript” bibliotēkas vizualizē vairākslāņu grafu. Katra bibliotēka to attēlo pēc sava prāta, bet kopējais princips ir pareizs.
* Pirms eksperimenta izveidojās trīs vairākslāņu grafu prototipi ar dažādām bibliotēkām (“D3.js”, “Cytoscape.js” un “Vis.js”), lai saprastu bibliotēku pamatprincipus.
* Tikai “Cytoscape.js” bibliotēka atbilst visām trim nepieciešamajām funkcijām. “D3.js” un “Vis.js” nav iespējams mijiedarboties ar slāņiem.
* Modificēti visi trīs algoritmi, lai realizētu eksperimentu.
* Pēc eksperimenta ar trim bibliotēkām ar mērķi izveidot vairākslāņu grafu ar 1200 virsotnēm, 3 slāņiem un 600 šķautnēm, tiek atrasta veiktspējīga “JavaScript” bibliotēka – “D3.js”.
* Visilgāko lejupielādes rezultātu uzrādīja “Vis.js” bibliotēka – rezultāts sastādīja 14657 milisekundes, tas nozīmē, ka “Vis.js” bibliotēka nav labākais variants integrācijai tīmekļa lapā, ja nepieciešams strādāt ar lielo datu kopu.
* “Vis.js” uzrādīja labākus rezultātus algoritma ātruma ziņā, izņemot “Skriptēšanu”.
* “Cytoscape.js” bibliotēka ir parādījusi vairākslāņu grafu pareizāku un vizuāli saprotamāku variantu.
* Ar “Cytoscape.js” bibliotēkas palīdzību var izveidot interaktīvāko vairākslāņu grafu.
* Failu izmēri izrādījās nelieli, tāpēc nekādā gadījumā neietekmē procesu.
* “Vis.js” bija nepieciešams mazāk koda rindiņu, lai izveidotu vairākslāņu grafu. “D3.js” ir pilnīgs pretstats.

Izmantotie informācijas avoti

1. Kivelä, M., Arenas, A. u.c., Multilayer networks, *Journal of Complex Networks*. 2014, 2, 203.-271. lpp. ISSN 2051-1310
2. Hammound, Z. & Kramer, F., Big Data Analytics 5, *Multilayer networks: aspects, implementations, and application in biomedicine*. 2020, 2, 18. lpp. ISSN: 2058-6345
3. Liu, L., Kang, Z. u.c., Information Sciences, *Multilayer graph contrastive clustering network*. 2022. 256-267. lpp. ISSN: 0020-0255
4. Алексеев, В.Е. & Захарова, Д.В. Теория Графов. Нижний Новгород: Нижегородский госуниверситет, 2012. 57 lpp.
5. Gould, R., Graph Theory. New York: Courier Corporation, 2012. 335. lpp. ISBN 0486498069, 9780486498065.
6. McGee, F., Ghoniem., M. u.c., Computer Graphics forum, *The State of the Art in Multilayer Network Visualization*. 2019, 25. lpp. ISSN:1467-8659
7. Boccaletti, S., Binconi, G. u.c., Physics Reports, *The structure and dynamics of multilayer networks*. 2014. 1.-122. lpp. ISSN: 0370-1573
8. Bouanan, Y., Ribault, J. u.c., ScienceDirect, *Modeling and Simulation of Human Reaction in a Multidimensional Social Network.* 2015, 592.-597. lpp. ISSN: 2405-8963
9. Xueming, L., Stanley, E. u.c., Europhysics letters, *Breakdown of interdependent directed networks*, 2015, 1138.-1143. lpp. ISSN: 1286-4854
10. Bostock M., *Data-Driven Documents.* 2021 [skatīts 2023. g. 15. aprīlī] Pieejams: <https://d3js.org/>
11. [Kelleher C., *d3-force.* 2021 [skatīts 2023. g. 15. aprīlī] Pieejams: https://github.com/d3/d3-force](file:///C:\Users\dmitr\Downloads\Kelleher%20C.,%20d3-force.%202021%20%5bskatīts%202023.%20g.%2015.%20aprīlī%5d%20https:\github.com\d3\d3-force)
12. Bostock M., *d3-selection.* 2021 [skatīts 2023.g. 15.aprīlī] Pieejams: <https://github.com/d3/d3-selection>
13. *Ievads HTML valodā*. 2008 [skatīts 2023.g. 16.aprīlī] Pieejams: <https://www.uzdevumi.lv/p/informatika/11-klase/html-teksta-noformesana-7002/re-7ad62d77-cbbc-4b7b-b653-c87ea8cca862>
14. *HTML Basic*., 2023 [skatīts 2023.g. 16.aprīlī] Pieejams: [https://www.w3schools.com/h tml/html\_basic.asp#:~:text=All%20HTML%20documents%20must%20start,%3E%20and%20](https://www.w3schools.com/h%09tml/html_basic.asp#:~:text=All%20HTML%20documents%20must%20start,%3E%20and%20).
15. *How to add css.* 2023 [skatīts 2023.g. 16.aprīlī] Pieejams: <https://www.w3schools.com/css/css_howto.asp>
16. *JavaScript basics*. 2023 [skatīts 2023.g. 16.aprīlī] Pieejams: <https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics>
17. *http-server*. 2022 [skatīts 2023.g. 16.aprīlī] Pieejams: <https://www.npmjs.com/package/http-server>
18. Spruce E., *D3.js Tutorial – Data Visualization for Beginners.* 2021 [skatīts 2023.g. 16.aprīlī] Pieejams: <https://www.freecodecamp.org/news/d3js-tutorial-data-visualization-for-beginners/>
19. [*D3.js selection.enter() Function.* 2020 [skatīts 2023.g. 17.aprīlī] https://www.geeksforgeeks.org/d3-js-selection-enter-function/](file:///C:\Users\dmitr\Downloads\D3.js%20selection.enter()%20Function.%202020%20%5bskatīts%202023.g.%2017.aprīlī%5d%20https:\www.geeksforgeeks.org\d3-js-selection-enter-function\)
20. Shahin R., *Grouping Elements.* 2020 [skatīts 2023.g. 17.aprīlī] Pieejams: <https://datacrayon.com/visualisation-with-d3/grouping-elements/>
21. [*D3.js selection.attr() Function.* 2020 [skatīts 2023.g. 17.aprīlī] Pieejams: https://www.geeksforgeeks.org/d3-js-selection-attr-function/](file:///C:\Users\dmitr\Downloads\D3.js%20selection.attr()%20Function.%202020%20%5bskatīts%202023.g.%2017.aprīlī%5d%20Pieejams:%20%20https:\www.geeksforgeeks.org\d3-js-selection-attr-function\)
22. *JavaScript Switch Statement.* 2023 [ skatīts 2023.g. 18.aprīlī] Pieejams: <https://www.w3schools.com/js/js_switch.asp>
23. *Elements: mouseover event.* 2023 [skatīts 2023.g. 18.aprīlī] [https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseover\_event#browser\_compatibility](https://developer.mozilla.org/en-US/docs/Web/API/Element/mouseover_event%23browser_compatibility)
24. Riviere P., *D3-drag.* 2021 [skatīts 2023.g. 18.aprīlī] Pieejams: [https://github.com/d3/d3-drag/blob/v3.0.0/README.md#drag\_on](https://github.com/d3/d3-drag/blob/v3.0.0/README.md%23drag_on%20)
25. *Cytoscape.js*. 2015 [skatīts 2023.g. 18.aprīlī] Pieejams: <https://js.cytoscape.org/>
26. *Vis.js.* 2017 [skatīts 2023. 19.aprīlī] Pieejams : <https://visjs.org/>
27. *Vis.js – Network.* 2017 [skatīts 2023. 19.aprīlī] <https://almende.github.io/vis/docs/network/>
28. *Vis.js – Network groups.* 2017 [skatīts 2023. 19.aprīlī] Pieejams: [https://almende.github.io/vis/docs/network/groups.html#](https://almende.github.io/vis/docs/network/groups.html%23)
29. Basques K., *Analyze runtime performance*. 2017 [skatīts 2023. 15.maijā] Pieejams: <https://developer.chrome.com/docs/devtools/performance/>
30. Top 8 JavaScript Libraries for Data Visualization in 2023. 2023 [skatīts 2023. 20.maijā] Pieejams: <https://www.syncfusion.com/blogs/post/top-8-javascript-data-visualization-libraries-in-2023.aspx>