VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Magistro darbas

MASINIO EISMO SIMULIAVIMAS

Atliko: 6 kurso, 10 grupės studentas
Raimondas Rimkus

Darbo vadovas:
Dr. Tadas Meškauskas

Vilnius
2012

Turinys

Sutartinių terminų sąrašas

Grafas: tai viršūnių ir jas jungiančių briaunų rinkinys

Viršūnė/briauna: Žemėlapio kontekste grafo viršūnė atitinka gatvių sankryžą, briauna – kelius jungiančius sankryžas.

Briaunos svoris: Analizuojamame grafe briaunos turi du svorius: briaunos (kelio atkarpos) pravažiavimo laiką esant nuliniai atkarpos apkrovai ir kraštinės kokybinį lygmenį. Jei tiesioginio kelio tarp viršūnių nėra, laikoma kad briaunos tarp jų svoris lygus begalybei.

Kelias grafe: Tai seka tarpusavyje sujungtų viršūnių. Kelio ilgis – suma šių viršūnių svorių.

Trumpiausias kelias: $min\sum_{i=0}^{n-1} d(p_i, p_{i+1})$, kur $p_0 = \text{Pradinis taškas}, p_n = \text{Galinis taškas},$ $d_n = \text{Atstumo tarp gretimų taškų f-ja}$.

Algoritmo sudėtingumas: Tai tokia funkcija $f(n)$ apibrėžianti skaičiavimo laiko priklausomybę nuo duomenų kiekio *n*. Naudojamas žymėjimas $O(f(n))$. $O(n^2)$, reikštų, kad padvigubinus duomenų kiekį, algoritmo skaičiavimo laikas paketurgubėja.

Neracionalus kelio paieškos algoritmas: Tradicinio Dijkstra algoritmo sudėtingumas $O(n^2)$. Pritaikius rikiuoto sąrašo duomenų struktūrą skaičiavimo laikas sutrumpėja iki $O(n\log(n))$. Aukštesnio sudėtingumo trumpiausio kelio paieškos algoritmas bus laikomas neracionaliu, nes verčiau naudoti tradicinį (ir paprastesnį) algoritmą.

Anotacija

Įvairūs grafo srautų minimizavimo arba maksimizavimo uždaviniai yra taikomi modeliuoti procesams logistikoje, kompiuteriniuose tinkluose, akcijų biržose, automobilių eisme ir t.t. Tokio uždavinio sprendimas esant dideliems grafams yra labai sudėtingas, todėl modelyje būna mažai kintančių parametrų arba modeliuojama tik dalis proceso.

Tokio uždavinio pavyzdys (kuris ir bus nagrinėjamas) yra eismo modeliavimas keliuose. Kiekvienas vairuotojas bando pasiekti savo tikslą per minimalų galimą tuo metu esančiomis sąlygomis laiką. Tokiu būdu yra užimama „geriausia vieta“ kelyje, o kitiems vairuotojams užimtas kelias gali tapti neberacionalus ir bus pasirenkamas alternatyvus. Šioje vietoje atsispindi esminė tokio tipo uždavinio savybė: kiekvienas proceso dalyvis siekia maksimalios naudos sau, tokiu būdu tą naudą atimdamas iš kitų dalyvių. Šiu atveju „nauda“ yra vieta eisme.

Darbe yra aprašytas eismo imitavimo sistemos modelis ir realizuotas tokia architektūra, kuri tinkama lygiagretiems skaičiavimams. Pateikta statistiniai gauto modelio analizavimo pavyzdžiai. Taip pat pasiūlyti keli algoritmai specifiniams uždaviniams spręsti, kaip dalinis modelio perskaičiavimas.

Summary

Mass Traffic Simulation

In this work we will try to apply various path search algorithm optimizations for solving traffic simulation problem. These optimizations are made on the basis of processes and patterns specific to car traffic. Other areas like logistics, money traffic and stocks should have their own specific optimizations. Primary selection criteria will be calculation time. Errors are also introduced by these optimizations, but benefits of solving more complex problems in much shorter time will outweigh those errors.

Most of described algorithms are just modifications of traditional Dijkstra's algorithm. This allows to show step by step every optimization introduced, which we can split into 3 categories:

- Algorithms for finding shortest path in untouched graph
- Algorithms for finding shorted path which need modified graph
- Algorithms specific for traffic simulation

Algorithms up to graph transformations have been analyzed in previous works. These optimizations proved to be very perspective in point-to-point path search. Thou applying them to traffic modeling and introducing modeling specific optimizations is a relatively new research area. Also an algorithm imitating traffic laws has to be implemented.

One of suggested optimizations could be called “fast partial recalculation”. This would allow a faster recalculation of main path loads. Assuming that traffic is the same and graph is slightly modified. A modified idea could be adapted for a recalculation with modified traffic loads.

Vastly improved calculations times would allow larger graphs or more combinations to be analyzed. This allows to create more precise models or more evolution paths to be analyzed in the same time.

Įvadas

Šio darbo tikslas – pademonstruoti kaip galima optimizuoti imitavimo procesą automobilių eismo modeliavimo uždaviniui spręsti. Automobilių eismas yra vienas paprasčiausių kelio paieškos uždavinių tipų . Taip pat jam galima taikyti daugelį optimizacijų (pvz: sluoksniavimas ir transformavimas į matricą). Šios optimizacijos remiasi dėsniais, kuriais remiasi paprasti vairuotojai darydami sprendimus kokį kelią pasirinkti. Kitose srityse reiktų ieškoti joms specifinių optimizacijų, kurios remtųsi prielaidomis apie grafe vykstančių procesų savybes. Kai kurios prielaidos galiojančios automobilių eismui gali galioti ir kitoms sritims. Analogiško tipo uždavinio pavyzdys būtų akcijų biržos modelių radimas priklausomai nuo to kur bus atliekama sąlyginai didelė (galinti pakeisti nusistovėjimą) investicija.

Dažniausiai tokio modelio radimas susideda iš trumpiausio (arba ilgiausio) kelio radimo kiekvienam modelio dalyviui. Tai sudaro didžiąją skaičiavimų dalį. Taip pat šiai sričiai yra taikoma daugiausia optimizacijų. Grafų lygmenų ir matricų transformacijos yra plačiai išnagrinėtos Dominik Schultes darbe [Sch08]. Šie algoritmai naudojami Google Maps sistemoje. Taip pat šių algoritmų palyginimas su tradiciniais algoritmais buvo atliktas ankstesniame mano darbe [Rim10].

Darbe nagrinėjamo uždavinio formuluotė: Duotame grafe rasti briaunų apkrovas, kuomet duotas eismo dalyvių sąrašas bando pasiekti savo tikslą trumpiausiu keliu, įtakodami vienas kitą. Gautas eismo modelis šio uždavinio spendimo eigoje gali būti panaudotas labai įvairiai. Pagrindinis panaudojamumas būtų eismo sąlygų prognozavimas miestuose. Tai gali būti ilgalaikių prognozių gavimas (pasitelkiant statistikas) ir modelio gavimas kintant eismo sąlygoms: srautų augimas, papildomi keliai arba pralaidumo kitimas. Egzotiškesnis panaudojamumas būtų trumpalaikės prognozės. Tai būtų modelio gavimas kelioms valandoms į ateitį. Tokią informaciją galima publikuoti internetiniuose puslapiuose arba skelbti per TMS paslaugą į GPS įrenginius. Taip vairuotojams būtų padedama vengti ne tik esamų bet ir prognozuojamų kamščių.

Pagrindinė problema, kuris bus nagrinėjama šiame darbe yra eismo dalyvių įtakos algoritmas. Pasirinktas įtakos algoritmas apibrėš modelio atitikimą realiam. Tuo pačiu įtakos skaičiavimų sudėtingumas ir trukmė daugiausia priklausys nuo parinkto algoritmo ir jo realizacijos.

Kylant apkrovoms eismas lėtėja. Tai, jog eismo dalyvis maršrutą pasirenka pagal tuo metu esančias eismo sąlygas vengiant tokių apkrautų atkarpų, bus laikoma kertine automobilių eismo savybe. Darbo

eigoje gavus modelį, kuris tenkins šią savybę, bus laikoma, kad sudarytas modelis tinkamai mėgdžioja automobilių eismą.

Eismo modelio architektūra

Eismo sąlygų imitavimas, tai daugybės trumpiausių kelio paieškos uždavinių sprendimas. Taip pat kiekvienas vairuotojas turi įtakoti eismo sąlygas kelyje, kuriuo važiavo. Trumpiausio kelio uždavinio sprendimas bus nagrinėjamas vėliau, dabar paanalizuokime principus kaip apjungsime eismo dalyvius į visumą.

Realiame gyvenime visi vairuotojai gali sėsti į savo mašinas ir išvažiuoti į darbą 7 valanda ryto. Sprendimai apie maršruto pasirinkimą yra atliekami realiu laiku. Programiškai paraleliai vienu metu paleisti visas mašinas yra labai sudėtinga. Taip pat tokio modeliavimo būdas bus neracionalus. Daug paprasčiau yra vienu metu paleisti vieną transporto priemonę pagal tuo metu esančias eismo sąlygas. Tai būtų lyg eismo sąlygų įšaldymas išvažiavimo momentu. Tokia optimizacija įneša nemažas paklaidas kintant apkrovai grafe, tačiau galutiniam modeliui tai turės mažai įtakos, nes esant tolydžiai apkrovai, atkarpų pravažiavimo laikai nusistovės (modelis stabilizuosis).

Kelio atkarpos apkrovos tikrinimas pagal realų atvykimo į ją laiko momentą, įneštų ir dar vieną didelę problemą: reikia žinoti kokios bus eismo sąlygos ateityje. Pavyzdžiui išvažiuojant 7 valandą į 2 valandas trunkančią kelionę reiktų žinoti kokios bus eismo sąlygos per artimiausias dvi valandas. Tačiau tai galima tik sumodeliavus eismą dvi valandas į ateitį. Jei naudojant ateities eismo sąlygas neįtraukiant sekančių išvykstančių mašinų, o tik leidžiant iš analizuojamų kelio atkarpų išvažiuoti ankstesnėms mašinoms, per kelionės laikotarpį bendra grafo apkrova nukris. Todėl taip skaičiuojant rastas kelionės laikas bus visada trumpesnis už realųjį.

Taip pat, eismo sąlygų įšaldymas leidžia lygiagrečiai keliems kompiuteriams prisijungus prie centrinės duomenų bazės vienu metu ieškoti trumpiausių kelių. Skaičiuojant tokiu būdu lygiagrečiai imituojamos transporto priemonės neįtakoja viena kitos. Esant sąlyginai mažam paralelių skaičiavimų kiekiui imituojamų transporto priemonių atžvilgiu, įnešama paklaida bus minimali.

Sekanti optimizacija: atvykimo į kelio atkarpą nefiksavimas. Realiai pasirinktu keliu yra važiuojama nuosekliai. Tačiau esant dideliam pravažiavimų istorijos sąrašui ir ilgiems atstumams, konkrečiu laiko momentu esančio atkarpos apkrovimo radimas tampa daug skaičiavimo laiko reikalaujančiu uždaviniu. Dėl šios priežasties atvykimo laikas nėra fiksuojamas. Tai būtų lyg važiavimas visomis gatvėmis

pasirinktame kelyje išvykimo momentu. Ši optimizacija taip pat turės mažai įtakos galutiniam modeliui.

Šis apkrovos skaičiavimo būdas dalinai kompensuos paklaidas atsirandančias nuo praeitos optimizacijos (eismo sąlygų įšaldymo išvykimo momentu). Visų vienu metu paleistų transporto priemonių apkrovos sumuosis ir apkrova laike kris lėčiau, kaip ir turėtų būti. Išlieka viena problema: apskaičiuotas pradinis kelionės laikas nesutaps su galutiniu saugomu pravažiavimo laiku konkrečiose kelio atkarpose.

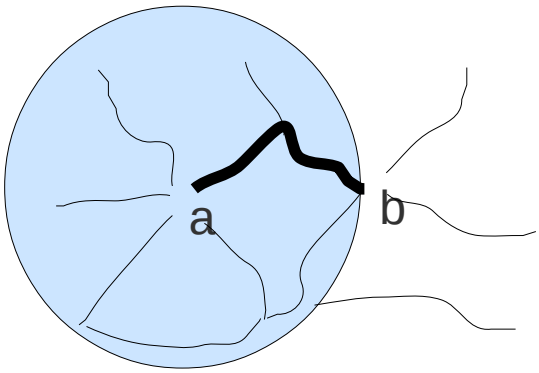
Kelio paieškos algoritmai

Imitavimo programoje naudojamas algoritmas susidarys iš kelių sub-algoritmų kurių kiekvienas bus detalizuotas. Taip pat panagrinėsim specifines jiems taikomas modifikacijas šiam uždaviniui spręsti. Pagrindinis algoritmo gerumo vertinimo kriterijus – skaičiavimo laiko minimizavimas. Algoritmų iliustracijos paimtos iš šaltinio [Rim10].

1. Dijkstra algoritmas

Tradicinis Dijkstra kelio paieškos algoritmas yra $O(n^2)$ sudėtingumo. Lyginant su likusiais darbe nagrinėjamais algoritmais, jis yra lėtas. Šį algoritmą iš visų likusių nagrinėjamų šiame darbe išskirianti savybė – jis vienintelis duodą tikslų trumpiausio kelio sprendinį. Taip pat jis bus išanalizuotas dėl to, kad visi likę nagrinėjami algoritmai (kaip ir dauguma egzistuojančių) yra tik Dijkstra algoritmo modifikacijos.

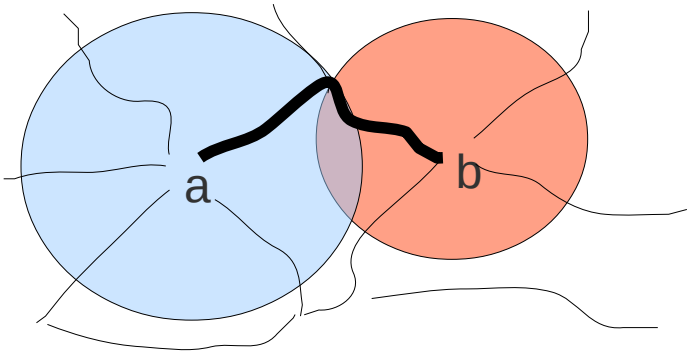
Šis algoritmas yra *one-to-many* paieškos tipo. Tai reiškia, kad nuo pradinio taško imamos gretimos viršūnės ir tokio didėjančio apskritimo principu einama iki kol bus atsiremta į viršūnę tenkinančią pabaigos sąlygas (tradiciniu atveju, tai yra tikslo viršūnė). Kiekvienos iteracijos metu yra surandamas vienas trumpiausias kelias į vieną artimiausią viršūnę. Skaičiavimo metu išanalizuojama *r* spindulio apskritimo zonoje esančios visos viršūnės, kur *r* yra atstumas tarp pradinio ir tikslo taškų.



1 pav. Vieno paieškos apskritimo iliustracija

2. Dvikryptis Dijkstra algoritmas

Šis algoritmas yra *meet-in-the-middle* tipo. Tai yra tradicinio Dijkstra algoritmo modifikacija. Vietoj kelio ieškojimo iš pradinio taško į galinį, kelio ieškoma dviem procesais iš pradinio ir galinio taško vienu metu. Algoritmų pabaigos sąlygas tenkinanti viršūnė- bendra abiejų procesų rasta viršūnė. Trumpiausias kelias: $A \rightarrow \text{bendras taškas} \rightarrow B$.



2 pav. Dviejų paieškos apskritimų iliustracija

Nors ir šio algoritmo sudėtingumas yra toks pats $O(n^2)$, tačiau jo vidutinis kelio paieškos laikas yra dvigubai trumpesnis. Tai galima pastebėti ir iš grafinės iliustracijos, nes dviejų apskritimų plotas bus dvigubai mažesnis, nei vieno esančio Dijkstra algoritme. Tai yra menkas pagreitėjimas, tačiau ši modifikacija yra būtina sekantiems algoritmams.

3. Kelių lygmenų Dijkstra algoritmas

Ši optimizacija tradiciniam kelio paieškos uždaviniui suteikia patį žymiausią pagreitėjimą. Pirmiau panagrinėkime kokiomis prielaidomis remiantis šis algoritmas yra sukonstruotas.

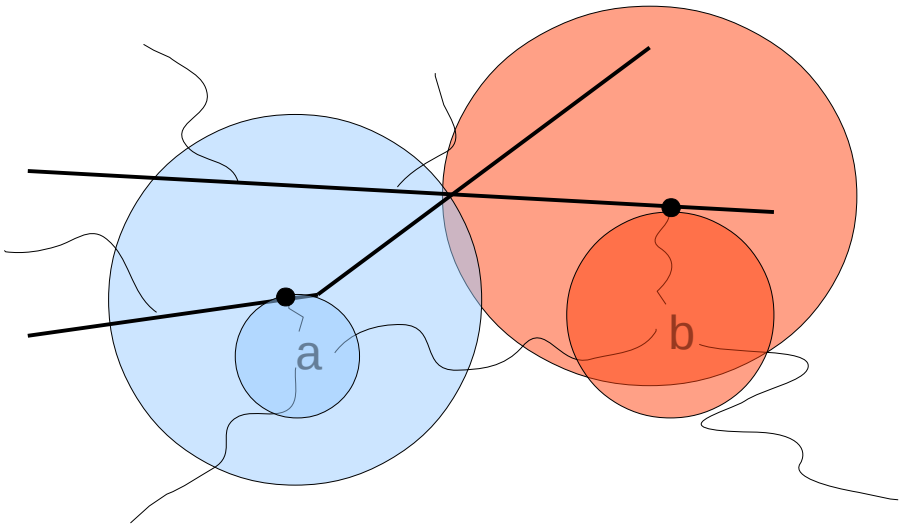
- Kelio lygmuo – tai skaičius apibrėžiantis kelio kokybinę grupę. Grafą rekomenduojama suskirstyti y 3-5 grupes, pvz: žvyrkeliai, užmiesčio, autostrados ir t.t.
- Mašina važiuojant iš pradinio taško pasirenka vis aukštesnio lygmens kelius, o artėjant prie tikslo išsukinėjama į žemesnio lygmens kelius. Maršrutas einantis per $\text{aukštesnio} \rightarrow \text{žemesnio} \rightarrow \text{aukštesnio}$ lygmens kelius yra negalimas.
- Mašina teikia pirmenybę aukštesnio lygmens keliui. Tai yra pradedant važiuoti ieškoma trumpiausių kelių iki aukštesnio lygmens kelio, o artėjant prie tikslo pasirenkamas kelias, kad žemesnio lygmens kelių būtų kuo mažiau.

Esant šioms prielaidoms galima sukonstruoti labai efektyvų kelio paieškos algoritmą. Taip pat jo rasti keliai dažnai atitinka žmogiškąjį privilegijavimą, nes dažnai pasirenkamas ne greičiausias, o geriausias (kokybiškai) maršrutas.

Šiam algoritmui realizuoti reikia atlikti tokią Dijkstra algoritmo modifikaciją:

- Kelio ieškoma iš pradinio ir galinio taško (analogiška dvikrypčiam Dijkstra).
- Kiekvienas procesas saugo skaičių apibrėžiantį kokio lygmens keliuose jis turi ieškoti kelio.
- Jei procesas randa aukštesnio lygmens viršūnę, tos viršūnės lygmuo išsaugomas ir tęsiama paieškos procedūra tame kokybiniame lygmenyje.
- Pabaigos viršūnės sąlyga lieka nepakitusi- bendra abiejų procesų viršūnė.

Trumpiausias kelias: $\text{A} \rightarrow \ldots \rightarrow \text{perėjimas į } \uparrow \rightarrow \ldots \rightarrow \text{bendras taškas} \rightarrow \ldots \rightarrow \text{perėjimas į } \downarrow \rightarrow \ldots \rightarrow \text{B}$



3 pav. Kelių lygmenų paieškos iliustracija

Toks kelio paieškos būdas įneša nemažas paklaidas, ypač trumpuose atstumuose. Paklaidų mažinimui galima įvesti dar kelias modifikacijas, kaip perėjimą į sekantį lygmenį tik radus fiksuotą kiekį viršūnių, arba visus fiksuoto spindulio zonoje esančius perėjimus, tačiau dėl jų komplikuotumo, jie nebus realizuojami imitavimo programoje.

4. Kelių lygmenų Dijkstra algoritmas su matrica

Paskutinioji modifikacija kelio paieškos tarp dviejų taškų algoritmui – aukščiausiojo lygmens trumpiausių kelių matricos sudarymas. Tai yra, iš anksto suskaičiuojama visi trumpiausi keliai tarp visų to lygmens viršūnių porų. Šie paruošti duomenys saugomi matricoje, o abiems procesams pasiekus šį lygmenį viena iteracija (per beveik konstantinį laiką) yra surandamas kelias tarp dviejų viršūnių. Skaičiavimo laiko sutrumpėjimas yra menkas, nes aukščiausiojo lygmens grafas yra labai retas ir jame paieška ir taip yra labai greita. Tačiau, ši optimizacija gali būti panaudota keliems specifiniams imitavimo uždaviniams spręsti.

5. Matricos sudarymas (Floyd–Warshall algoritmas)

Matricos sudarymui galima panaudoti Floyd–Warshall algoritmą. Juo galima rasti trumpiausius kelius tarp visų porų per $O(n^3)$ laiką. Duomenys yra saugomi sąraše tokiuose įrašuose: $(\text{from}, \text{to}, \text{via}, \text{atstumas})$. Tai yra, kelias iš vienos viršūnės į kitą, eina per “via” viršūnę. Tokiame sąraše rekursiškai galima atkurti pradinį kelią [Rei03]. Vėliau iš tokios lentelės būtų galima sudaryti

matricą, tačiau šis veiksmas nebus atliktas.

Duomenis saugoti pradiniame formate yra racionalu, nes turint pradinį ir galinį tašką, per $O(n)$ laiką galima surasti visus trumpiausius kelius per visus tarpinius taškus. Tokią savybę galima išnaudoti kuomet mutuojant pradiniam grafui, pirminis paskaičiuotas trumpiausias kelias tampa nebe trumpiausiu. Dažniausiai būna apkraunama viena kelio zona. Jos apvažiavimą galime imituoti trumpiausio kelio einančio per gretimą kelią pasirinkimu. Žemiau pateiktas duomenų pavyzdys:

1 Lentelė

Iš	Į	Per	Atstumas
1	2	null	2,5
3	4	null	1
2	3	null	1,3
2	4	3	2,6
1	4	2	5,1
…	...	...	...

Pradinis kelias atkuriamas tokia rekursija: $(1,4)=(1,2)+(2,4)=(1,2)+(2,3)+(2,4)$

Kiekvienoje poroje (a,b) gali būti saugomas atstumas tarp šių dviejų taškų. Tačiau tai yra tik suma tarpinių atstumų. Galimi du architektūriniai sprendimai kaip sinchronizuoti matricos svorius ir leisti jai mutuoti kartu su grafu: perskaičiuoti visą matricą ir naudoti svorius iš grafo arba saugoti svorius matricoje juos sinchronizuojant ir ieškoti kelių per alternatyvų kelią. Žemiau pateikta abiejų pasirinkimų argumentavimas:

Pilnas perskaičiavimas

- Paprastesnis paieškos algoritmas.
- Randamas tikslesnis kelionės laiko įvertis, nes visada naudojami laikai esantys grafe
- Randamas kelias nebus trumpiausias grafui kintant.
- Perskaičiavimas turi trukti labai greitai, nes jį reikia kartoti kuo dažniau. Gali reikti blokuoti kitus skaičiavimus arba kol skaičiuojama, turimi duomenys gali pasenti.
- Perskaičiavimo sudėtingumo įvertis: $O(n^3)$.

- Naudojant klasterine skaičiavimo architektūrą, galima paskirti vieną skaičiavimo vienetą šios matricos perskaičiavimams.

Atstumų perskaičiavimas

- Sudėtingesnis paieškos algoritmas.
- Kelionės laiko įvertis naudos perskaičiavimo metu gautus atstumus – netikslus apskaičiuotas kelionės laikas.
- Randamas trumpiausias alternatyvus kelias pradiniame kelyje esant tik 1 perkrautai atkarpai.
- Greitas perskaičiavimas ir jis gali vykti lygiagrečiai kelio paieškoms.
- Prognozuojamas perskaičiavimo sudėtingumo įvertis: $O(n)$.
- Alternatyvaus kelio radimo sudėtingumo įvertis: $O(n)$.
- Atstumą galima būtų saugoti tik pirminėse atkarpose, o suminį atstumą suskaičiuoti atkuriant pradinį kelią. Tuomet alternatyvių kelių paieškos procedūra tampa daug sudėtingesnė, nes kiekvienai tarpiniai viršūnei reikia perskaičiuoti jos kelio ilgį. Prognozuojamas tokios procedūros sudėtingumas: $O(n^2)$, kas atitinka pilno perrinkimo procedūrą.

Galimybė atstumus saugoti grafe ir vistiek ieškoti alternatyvių kelių yra iškart atmetama. Tuomet kiekvienos tikrinamos atkarpos atstumą reiktų perskaičiuoti ir gaunamas tokio algoritmo sudėtingumas bus $O(n^2)$, kas atitinka pilną perrinkimo procedūrą.

6. Floyd–Warshall matricos minimizavimas

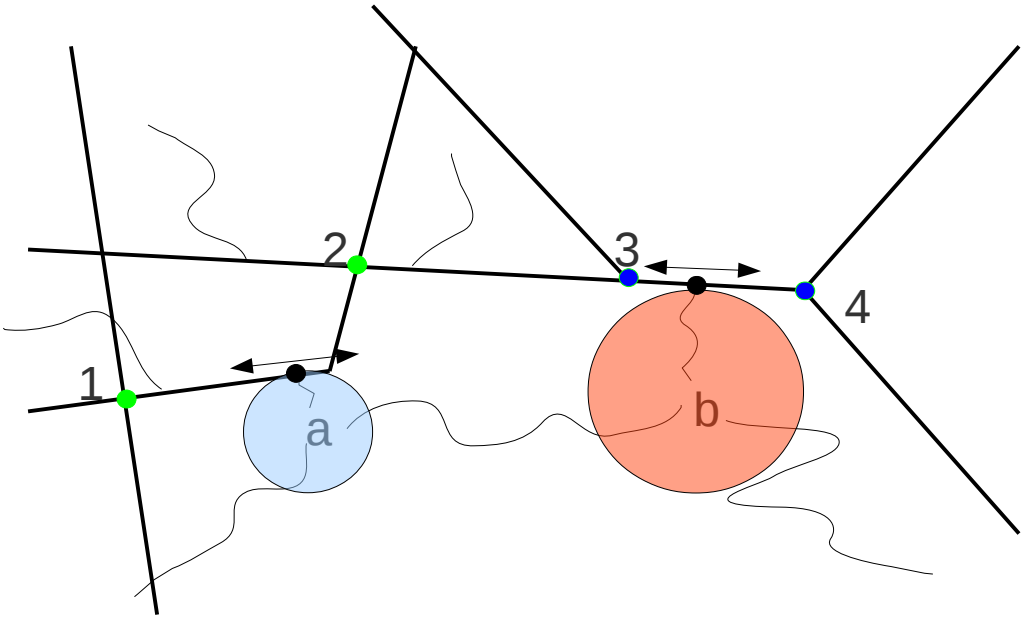
Žemėlapio sluoksniavimo atveju gaunami aukštesnieji sluoksniai yra labai reti. Tai yra dauguma viršūnių turi dvi briaunas. Iš anksto skaičiuoti visus trumpiausius kelius tokioje situacijoje yra neracionalu resursų atžvilgiu. Galima įvesti tokią modifikaciją: transformuoti tik „esmines“ tokio sluoksnio viršūnes. Šiuo atveju esminė būtų tokia viršūnė kuri turi ne 2 briaunas. Galinė viršūnė (turinti 1 briauną) laikoma esmine algoritmo paprastumo sumetimais.

Paanalizuokime detaliau kaip tuomet vyks paieška:

Tarkime turime matricą M kurioje yra surašyti visi trumpiausi keliai tarp esminių viršūnių. Norima rasti trumpiausią kelią tarp dviejų neesminių viršūnių. Pradiniam taškui surandamos dvi gretimos esminės viršūnės. Ši procedūra yra triviali, nes neesminė viršūnė turi dvi briaunas ir jomis einant visada

bus surastos dvi gretimos esminės viršūnės. Tokia pati procedūra atliekama ir tikslo viršūnei. Dabar turima po 2 esminius taškus pradiniam ir galiniam taškui. Iš 4 galimų kombinacijų yra išrenkamas trumpiausias, kas ir yra trumpiausias kelias. Jei pradinė arba tikslo viršūnė yra esminė, algoritme ta viršūnė naudojama kaip dvi artimiausios esminės.

Kelio radimas tokia procedūra yra bent 4 kartus ilgesnis, nes reikia tikrinti 4 kombinacijas, bei atkurti pilną kelią, nes naudojama praretinta matrica. Tačiau šie veiksmai yra labai greiti palyginus su laiko sąnaudomis surasti kelią iki šio lygmens.



Algoritmo analizė

1. Duomenų įvedimas (Mašinos)

Algoritmui reikalingi tokie duomenys apie mašinas:

- Mašinos išvykimo ir tikslo taškai
- Išvykimo laikas

2. Duomenų įvedimas (Grafas)

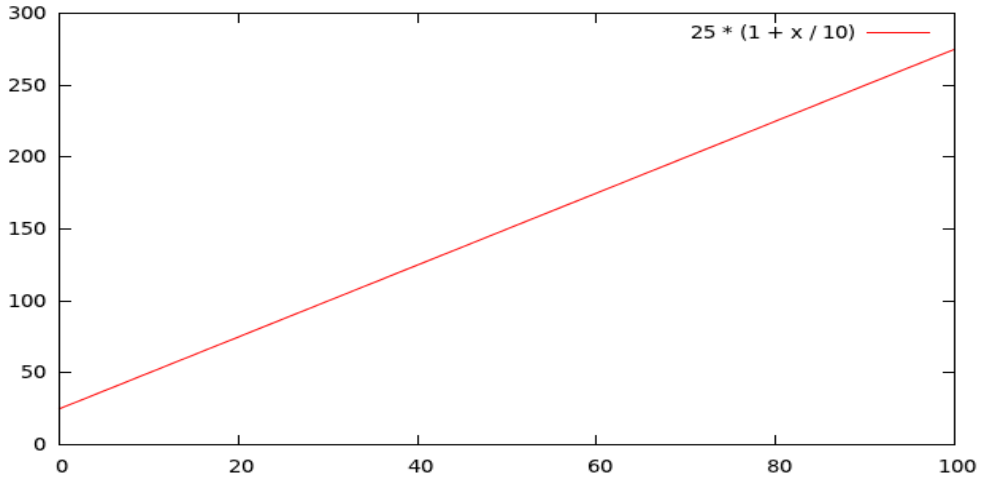
Algoritmui reikalingi šie duomenys apie grafą:

- Kiekvienos kelio atkarpos pravažiavimo laikas (esant nuliniai apkrovai)

- Kiekvieno kelio lygmens pravažiavimo laiko nuo apkrovos priklausomybės funkcija. Modelyje naudojama bendra funkcija visam grafui. Funkcija turėtų tenkinti tokias savybes:
  - $apkrova \in [0;+\infty)$ Kelio atkarpoje laiko momentu gali būti iki begalybes mašinų.
  - $f(apkrova=0)=pradinis\,kelio\,ilgis$ Esant nuliniai apkrovai – kelio pravažiavimo laikas turi sutapti su pradiniu.
  - $f(apkrova=\infty)=\infty$ Kelio atkarpoje esant begalybei mašinų važiavimo greitis lygus nuliui ir kelio atkarpos įveikimo laikas lygus begalybei.

  Šias savybes tenkinanti funkcija kuri buvo naudota bandymams yra:

  $f(length, load, level)=length\cdot(1+load\cdot 10^{-level})$. Kur *length* yra kelio pravažiavimo laikas nesant apkrovai, *load* – automobilių kiekis atkarpoje ir *level* – kelio kokybinis lygmuo (1 arba 2 pagal dabartinius duomenis). Funkcija parinkta bandymų būdu taip, kad modeliavimo metu pravažiavimo laikai neaugtų iki begalybės, bet augtų pakankamai, kad alternatyvių maršrutų pasirinkimas taptų racionaliu. Žemiau pateikta funkcijos grafikas pirmo lygmens keliui su pradiniu pravažiavimo laiku 25. Ši funkcija prastai atitinka realias eismo savybes. Pravažiavimo laikas turėtų augti lėtai esant žemai apkrovai ir augti sparčiau pasiekus kelio talpos ribą. Bandant tokias funkcijas modelis diverguodavo, nes rezultatų siuntimo metu smarkiai išaugdavo keletos atkarpų pravažiavimo laikai. Vėliau šios apkrovos perteklius nespėdavo nukristi.



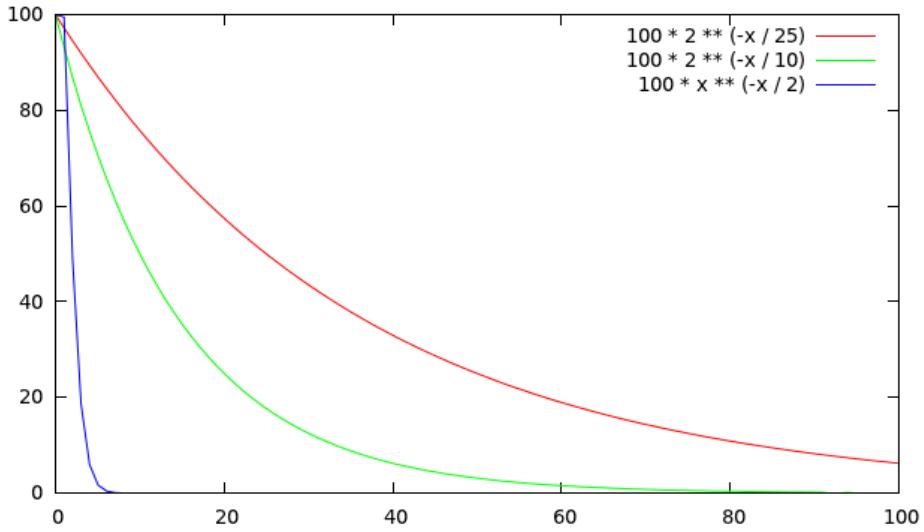
4 pav. Kelio atkarpos pravažiavimo laiko priklausomybės nuo mašinų kiekio grafikas

- Kiekvieno kelio lygmens apkrovos kritimo laike funkcija. Ši funkcija nurodo, kaip mažėja automobilių kiekis kelio atkarpoje, jei į ją neatvažiuoja daugiau automobilių. Ši funkcija yra

kažkokia (kol kas nežinoma) pravažiavimo laiko funkcijos transformacija. Funkcija turėtų tenkinti šias savybes:

- $apkrova \in [0; +\infty)$ Kelio atkarpoje laiko momentu gali būti iki begalybes mašinų.
- $g(apkrova = \infty) = \infty$ Esant begalybei mašinų kelio atkarpoje jų greitis = 0. Tai reiškia, kad bet kokio ilgio atkarpoje per bet kokį laiką mašinos išvažiuoti negali. Ši savybe parinktoje funkcijoje nėra tenkinama, nes kylant apkrovai dažnai patenka į tokią būseną.
- $g(\Delta laikas = \infty) = 0$ Esant baigtinei kelio apkrovai ir neatvykstant daugiau mašinų, kelio apkrova po kažkokio laiko *t* turi nukristi iki 0.
- $g'_{\Delta t} < 0$ Einant laikui apkrova tik mažėja
- $g(ilgis) < g(ilgis + \varepsilon)$ Ilgesnėje kelio atkarpoje apkrova krenta lėčau
- $g'_{\Delta t}(apkrova + \varepsilon) < g'_{\Delta t}(apkrova)$ Esant didesnei apkrovai ji krenta lėčiau, nes mašinų važiavimo greitis mažesnis. Ši savybė panaši antrajai. Ji taip pat nėra tenkinama.

Bandymų būdu parinkta funkcija: $g(load, \Delta time, length) = load \cdot 2^{\frac{-\Delta time}{length}}$. Kur *load* yra laiko momentu *t* buvusi kelio apkrova, *Δtime* – laiko skirtumas tarp *t* ir dabarties ir *length* – pradinis kelio atkarpos pravažiavimo laikas. Žemiau pateikta tokio grafiko pavyzdys *load* reikšmei esant 100 ir pradiniam kelio pravažiavimo laikui esant 25, 10 ir 2:



5 pav. Kelio atkarpos apkrovos nuo 100 automobilių kritimo grafikai

3. Pradiniai skaičiavimai

Prieš pradedant modeliavimą reikia sugeneruoti aukščiausiojo lygmens grafo trumpiausių kelių matricą. Tai yra atliekama tuščiame (be apkrovos) grafe. Apkrova imituojama keičiant pravažiavimo laikus, o kliūčių (apkrautų zonų) vengimas – ieškant alternatyvių kelių per tarpinį tašką.

Paruošiama vieta saugoti dabartiniai kiekvienos kelio atkarpos apkrovai.

Paruošiama vieta saugoti mašinos pasirinktam keliui ir užtruktam laikui.

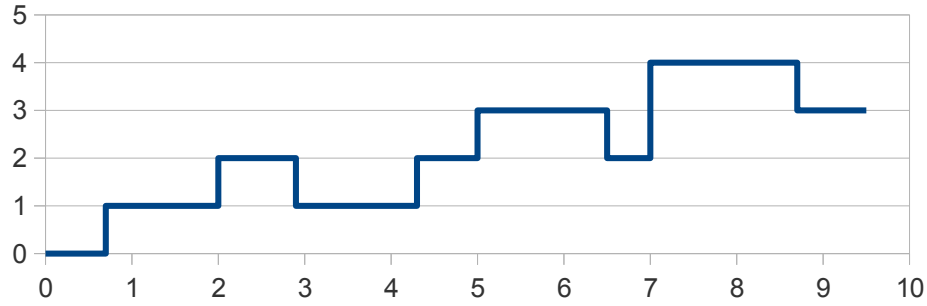
4. i-toji skaičiavimo iteracija

1. Pasirenkama neišvykusi mašina, kurios išvykimo laikas yra artimiausias dabartiniam sistemos laikui.
2. Sistemos laikas nustatomas į pasirinktos mašinos išvykimo laiką
3. Naudojantis aukščiau aprašytu trumpiausio kelio radimo algoritmu randamas tuo laiko momentu trumpiausias kelias.
4. Rasto trumpiausio kelio atkarpose apkrova padidinama vienetu.
5. Jei tai yra k-toji iteracija:
    1. į duomenų bazę nusiunčiamos apkrovos.
    2. paskaičiuojamas apkrovos kritimas per laiko intervalą nuo paskutinio atnaujinimo.
    3. skaičiavimo vienetuose atnaujinama grafo apkrovos ir pravažiavimo laikai.

Algoritmo optimizacijos

1. Tolydus automobilių kiekis kelio atkarpoje

Žemiau pateikta realios kelio atkarpos apkrovos grafiko pavyzdys:



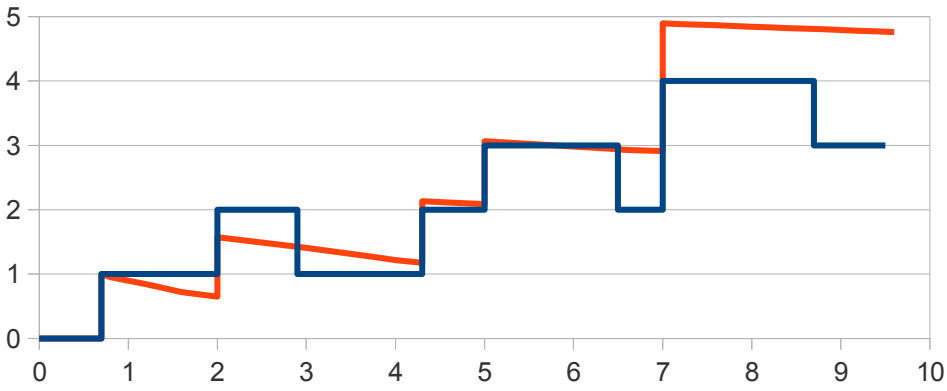
6 pav. Diskretaus automobilių kiekio kitimo pavyzdys

Tai yra tolydžiu laiko momentu egzistuoja diskretus automobilių kiekis. Tačiau tokio grafiko suformavimui reikia saugoti automobilių pravažiavimo istoriją ir kiekvienos iteracijos metų visą istoriją perskaičiuoti, tam kad gauti tikslų tuo metu esantį automobilių skaičių. Tai bus labai neracionalu.

Pasiūlymas:

- Neskaičiuojamas tikslus laiko momentas, kuomet mašina turės atvykti į kelio atkarpą, o apkrova suteikiama visam maršruto keliui išvykimo momentu.
- Kelio apkrova krinta tolydžiai, pagal “apkrovos kritimo” funkciją.

Tokiu būdu nereikia saugoti atvykimo ir išvykimo laikų kelio atkarpoms. Užtenka perskaičiuoti kiek nukris apkrova. Gaunama diskrečiau didėjanti ir tolygiai mažėjanti funkcija. Reiktų atkreipti dėmesį, į tai jog mažėjimo greitis priešingai proporcingas mašinų kiekiui. Žemiau pateiktas pavyzdys, kaip galėtų atrodyti optimizuota apkrovos funkcija:



7 pav. Mišraus automobilių kiekio kitimo pavyzdys

2. Greitojo perskaičiavimo režimas

Tiriant eismo intensyvumą dažniausiai orientuojamasi tik į pačius svarbiausiu kelius (pagal tiriamo ploto proporciją). Taip pat eismo spūstys dažniausiai susidaro tik pagrindiniuose keliuose. Todėl, išanalizavus rezultatus pakeitimai žemėlapyje dažniausiai atliekami tik pagrindiniuose keliuose. Dėl šių priežasčių bus analizuojama greitojo perskaičiavimo galimybė. Tarkime turime tokį panaudojimo scenarijų:

1. Sumodeliuojamos eismo sąlygos turimame grafe
2. Išanalizavus turimus rezultatus padaromas pakeitimas aukščiausiame grafo lygyje (įdedama, panaikinama kelio atkarpa ir/arba pakeičiama egzistuojančių pralaidumas)
3. Norima gauti preliminarų modelį kaip atrodys eismas esant toms pačioms apkrovoms

Šį rezultatą galima gauti tokia procedūra:

- Atliekant pirminį skaičiavimą išsaugoma automobilių atvykimo į aukščiausią kelio lygmenį laikai, pradiniai, bei galiniai taškai aukščiausiame lygmenyje.
- Padarius pakeitimą perskaičiuojama trumpiausių kelių matrica (arba jos dalis)
- Perskaičiuojamos eismo sąlygos tik aukščiausiame lygmenyje

Šios optimizacijos paklaidų šaltinis: prielaida, kad pakeitimas aukščiausiame kelio lygmenyje neįtakoja kelio pasirinkimo žemesniuose lygmenyse. Esant „kelių lygmenų Dijkstra algoritmo“ paklaidų mažinimo modifikacijoms, perskaičiuojant aukščiausią lygmenį būtų dalinai atsižvelgiama į priklausomybę tarp lygmenų, kas mažintų paklaidą.

Sistemos realizacija

1. Praktinės dalies tikslai

Šio mokslo tiriamojo darbo praktinės dalies tikslas yra realizuoti pavienius paieškos algoritmus ir juos apjungiančią modeliavimo sistemą. Algoritmai yra realizuoti taip, kad juos būtų galima naudoti kaip modulius kitose programose.

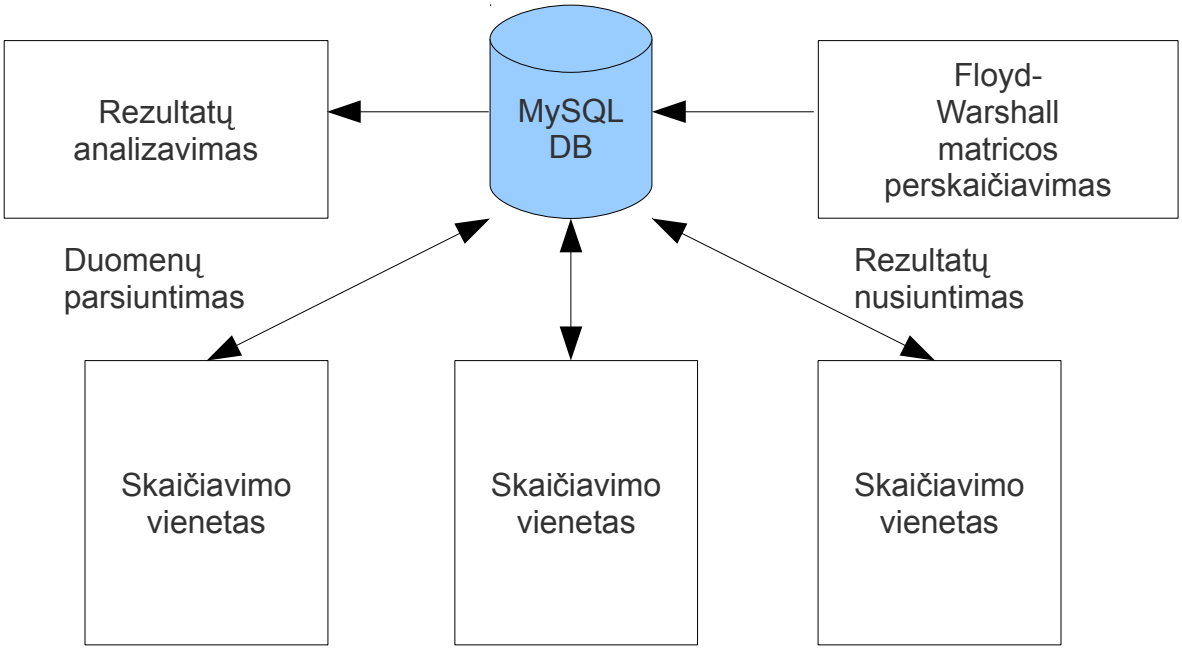
Didelių duomenų kiekių apdorojimui reikalinga lygiagrečiojo skaičiavimo galimybė. Kelio paieškos algoritmai yra visiškai tinkami lygiagretinimui. Problematiškas yra duomenų sinchronizavimas tarp kompiuterių. Ši problema paprasčiausiai išsprendžiama turint centrinę duomenų bazę. Tai nėra visiškai geras architektūrinis spendimas. Detaliau ši schema bei jos trūkumai bus nagrinėjama sekančiame skyrelyje.

Visi algoritmai buvo atskirai testuojami. Radus akivaizdžiai neefektyvius procesus, algoritmų architektūra buvo keičiama. Į realizuotų algoritmų sąrašą patenka ir transformavimas į matricą bei jos panaudojimo algoritmas. Šį algoritmas nėra plačiai analizuojamas, nes nebuvo išnaudojamas parinktai mažai grafo daliai.

Šio realizavimo tikslas nėra sukurti modulius tinkamus komerciniam naudojimui. Realizacija yra tik tiriamojo pobūdžio, kad parodyti algoritmų ir idėjų efektyvumą. Naudojant žemo lygio programavimo kalbą (pvz. C), procesoriaus ir atminties resursus būtų galima naudoti daug efektyviau.

2. Sistemos architektūra

Žemiau pateikta skaičiavimo klasterio iliustracija:



8 pav. Skaičiavimo klasterio iliustracija

Skaičiavimo vienetas:

- Vienas skaičiavimo procesas, kuriam priskirta procesorius/kompiuteris.
- Iš duomenų bazes yra parsisiunčiami visi duomenys ir laikomi lokalioje atmintyje.
- Rezultatai siunčiami į centralizuotą duomenų bazę: automobiliui parinktas maršrutas ir apkrovos padidėjimas grafe.

Matricos perskaičiavimas:

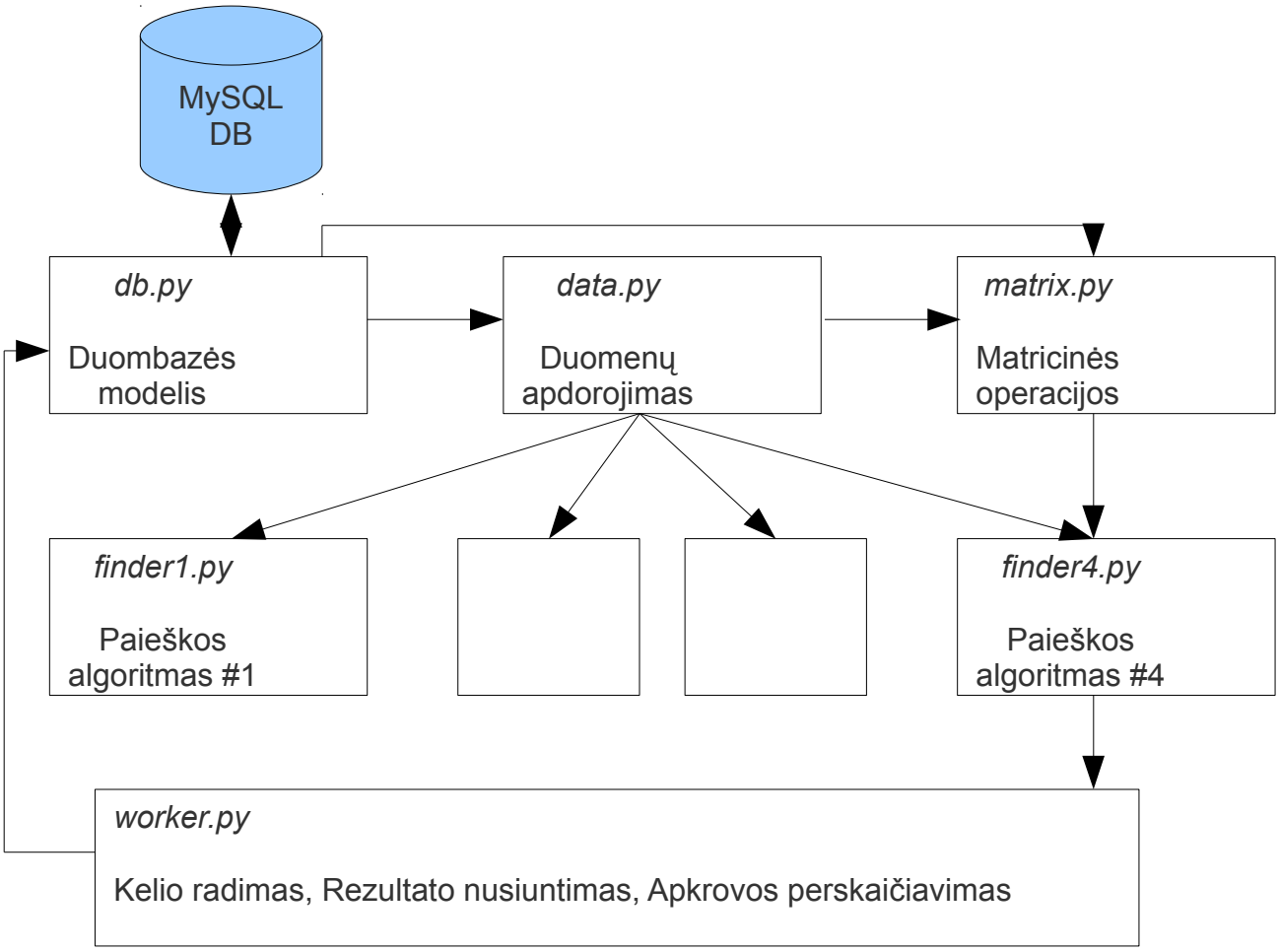
- Kas fiksuotą atliktų iteracijų kiekį reikia perskaičiuoti Floyd-Warshall matricą, tam kad ji liktų sinchronizuota su duomenų bazėje esančiais duomenimis.

Rezultatų analizavimas:

- Kadangi visi rezultatai yra renkami centralizuotai, jų analizavimas gali būti atliktas po imitavimo proceso.

3. Skaičiavimo vieneto architektūra

Žemiau pateikta skaičiavimo vieneto iliustracija:



9 pav. Skaičiavimo vieneto iliustracija

Finder klasės atitinka anksčiau aprašytus standartini, dvikrypti, kelių lygmenų ir matricinį Dijkstra algoritmus. Galutiniam modeliavimui bus naudojamas tik paskutinis (matricinis) algoritmas. Worker yra klasė naudojama modeliavimui. Ji ima darbus iš duomenų bazės, atlieka skaičiavimus naudojant matricinį algoritmą ir nusiunčia rezultatus atgal į duomenų bazę.

4. Sistemos modulių aprašymas

Sistemos klasių sąrašas su aprašais ir duomenų bazės schema pateikta prieduose. Taip prieduose glaustai aprašyta sistemos paruošimo instrukcija.

5. Problematika ir sprendimai

Duomenų paieška naudojant SQL užklausas yra labai lėta.

Dažniausiai atliekama užklausa yra „surask visas gretimas viršūnes duotajai“. Tokios užklausos vykdymo laikas pačiame MySQL serveryje, net naudojant indeksus, trunka apie 0,05 sekundės. Prie to pridėjus laikus ir resursus reikalingus SQL užklausos sugeneravimui, prisijungimui prie duombazės, rezultatų išsiuntimui ir jų išpakavimui, gaunamas apie 0,5 sekundės vykdymo laikas. Tuo pačiu smarkiai apkraunama duombazė ir SQLAlchemy biblioteka. Turint omenyje, kad bandymų metu tokių užklausų per vieną iteraciją buvo atliekama iki 140000 (naudojant dabartinius sintetinius duomenis ir efektyviausią matricinį algoritmą).
Sprendimas: lokali grafo kopija RAM atmintyje. Taip pat naudojama hash struktūra greitesnei paieškai.
Trūkumai: apribojamas grafo dydis kompiuterio atmintimi, reikalinga abipusė duomenų sinchronizacija tarp lokalios atminties ir duomenų bazės.

Aukščiausiame lygmenyje yra per daug viršūnių, kad jį būtų galima transformuoti į matricą

Antrame (aukščiausiame) lygmenyje yra apie 13000 viršūnių. Neskaičiuojant duomenų struktūros nuostolių, saugoti 13000^2 integer tipo elementų užims 650MB atminties. Taip pat grubus tokio dydžio matricos generavimo Floyd-Warshall algoritmu laikas dabartinėje sistemoje yra apie 160 parų.

Sprendimas: transformuoti tik dalį sluoksnio. Atrinktos 363 svarbios viršūnės. Gaunamas matricos generavimo apie 4 minutės. Tačiau didžioji šio laiko dalis yra sugaištama duomenų nuskaitymui ir sugeneruotos matricos surašymui į duomenų bazę. Šis laikas yra sąlyginai nedidelis, kas leidžia perskaičiuoti matricą modeliavimo eigoje ir nenaudoti sudėtingo matricos sluoksniavimo. Ši modifikacija detalizuota anksčiau pateiktame algoritmo aprašyme.

Trūkumai: sudėtingos matricos generavimo ir pradinio kelio atkūrimo procedūros.

Skaičiavimo rezultatų ir kelio apkrovų siuntimas trunka ilgiau nei pats skaičiavimas

Duomenų bazėje vykdyti daug smulkių užklausų yra neefektyvu. Esant galimybei užklausas reikia vykdyti kuo didesniais gabalais. Po skaičiavimo iteracijos į duomenų bazę siunčiami trijų skirtingų tipų duomenys:

- rasto kelio atstumas
- seka atkarpų, per kurias rastas kelias
- nauja apkrova kiekvienai naudotai kelio atkarpai

Sprendimas:

Lėčiausiai vykdoma operacija buvo kelio apkrovų perskaičiavimas. Visa logika buvo klientinėje dalyje. Perkėlus visa logika į vieną SQL užklausą, nebereikia ieškoti pavienių kelio atkarpų ir perskaičiuoti jų charakteristikas.

Sekanti ilgiausia operacija buvo rasto kelio atkarpų siuntimas, Kadangi šie duomenys nėra naudojami skaičiavimo eigoje, jų siuntimas perkeltas ant atskiros skaičiavimo gijos ir vyksta lygiagrečiai su skaičiavimu.

Trečioji optimizacija: rezultatų siuntimas gabalais po 250 darbų (keičiamas parametras). Dalis duomenų sukeliama į vieną užklausą iš visų darbų. Taip pat rečiau vykdomas atgalinis duomenų sinchronizavimas su lokalia atmintimi. Iš to atsirandančio paklaidos bus nagrinėjamos tolimesniame skyrelyje.

Įvykdžius šias optimizacijas duomenų sinchronizavimas sudarė mažąją skaičiavimo laiko dalį.

Skaičiavimų lygiagretinime atsirandančios klaidos

Kaip ir kuriant bet kokią kitą programą naudojančią kelias skaičiavimo gijas, reikia išspręsti branduolių sinchronizavimo niuansus. Paleidus sistemą ant 4 procesorių rastos šios klaidos:

- Resurso užrakinimas dviejų gijų (kol viena gija “rakina” resursą, kita spėja jį pasiimti)
- Skaičiavimo trunka ne vienodai laiko – rezultatai gražinami ne tokia tvarka, kaip buvo paskirstyti. Eismo modelio “laiko sąvoka” tampa neapibrėžta.
- Naudojant daugiau gijų, daugiau duomenų yra skaičiavimo būsenoje. Tai įneša papildomas paklaidas.

Dalis šių problemų buvo išspręsta, tačiau bandant sistemą lygiagretinti daugiau, turėtų atsirasti daugiau problemų. Taip pat naudojant kitokius kompiuterius, silpniausia skaičiavimo grandis gali tapti kitas skaičiavimo komponentas ir sistema pradės keistai veikti.

Sistemos modulių analizė

1. Bibliotekos naudojimo pavyzdys

Python interpretatorius gali veikti konsoliniame rėžime. Tai leidžia interaktyviai atlikti skaičiavimus arba naudoti iš anksto paruoštą skriptą. Žemiau pateiktas sesijos pavyzdys su komentarais:

```
1  import data                                  #importuojamas duomenų objektas
2  import finder4                               #importuojamas pasirinktas paieškos modulis
3  data.load(False)                             #nenaudojama lokali atmintis
4  path, dist = finder4.find(525687, 619211)    #atliekama paieška
5  print 'Total length: ' + str(dist)           #rasto kelio ilgis
6  print 'Total roads: ' + str(len(path))       #rasto kelio sankryžų kiekis
```

Bus atspausdintas toks rezultatas:

```
1  Total length: 6784.682261
2  Total roads: 391
```

2. Atskirų algoritmų tyrimų aplinka

Visi matavimai atliekami nešiojamu kompiuteriu su T5100 procesoriumi (2 banduoliai) ir 2GB darbinės atminties. Operacinė sistema Ubuntu 11.04 64-bit. MySQL duomenų bazė paleista lokaliame kompiuteryje kartu su imitavimo programa. Programai buvo skiriamas 1 branduolys, nes lygiagretinimo algoritmas nėra išbaigtas. Taip pat tai mažino apkrautos duomenų bazės (ir kitų procesų) įtaką skaičiavimo laikams ir atskiros skaičiavimo gijos neįtakojo viena kitos.

Linux operacinė sistema naudoja kietojo disko skaitymo ir rašymo buferius. Todėl pirmasis skaičiavimo ciklas trunka daug ilgiau už likusius. Dėl šios priežasties pirmieji skaičiavimo bandymai buvo ignoruojami.

Naudota Python 2.7.1 x64 programavimo kalba. Naudotas atsitiktiniu būdu sugeneruotas sintetinis grafas susidedantis iš 1973242 viršūnių. Remiantis [SS06] visos Europos kelių tinklo grafas susideda iš 18 mln. Atkarpų. Tai reiškia, kad sugeneruotas grafas, bent elementų kiekiu yra tinkamas tyrimams.

3. Lokalios atminties panaudojimo analizė

Duomenų modulio data.py inicializavimo funkcijai load() paduodamas parametras, kuris nurodo ar naudoti lokalią duomenų bazės kopiją. Duomenų grafą sudaro apie 1 milijonas atkarpų. Duomenų bazėje šie įrašai užima 36MB ir indeksai 80MB. Šių duomenų nuskaitymas, persiuntimas loopback interfeisu ir lokalus surašymas trunka 80-90 sekundžių. Tai yra labai ilgas laikas turint omenyje kiek vietos užima šie duomenys, bet laiko nebuvo skiriama išsiaiškinti to priežastims.

Lokaliame procese į paruoštą duomenų struktūrą užkrovus šiuos duomenis jie užima 614MB atminties. Todėl didesnių žemėlapių naudoti tyrimams nebuvo galima. Dalinai šios problemos išsispręstų algoritmą realizavus žemo lygio programavimo kalba kaip C.

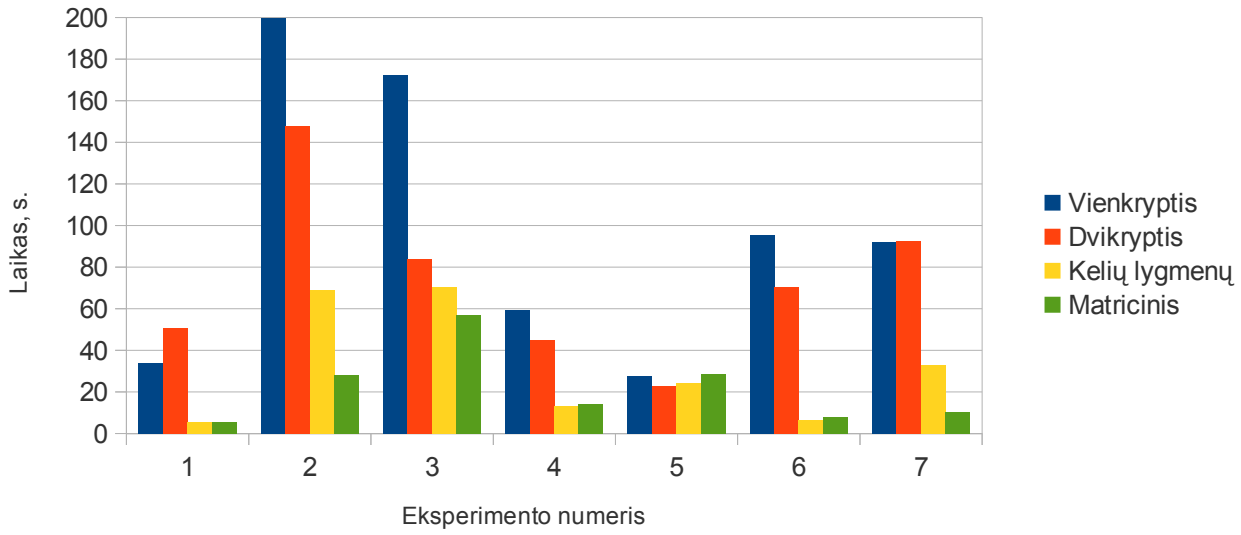
Nenaudojant lokalios atminties skaičiavimo laikai gaunami labai ilgi. Atliktas bandymas naudojant primityviausią Dijkstra algoritmą iš viršūnės 0 į viršūnę 917658. Gauti tokie skaičiavimo laikai:

- Su lokalia atmintimi: 0,5-0,8 sekundės
- Be lokalios atminties: apie 80 minučių

Taip pat visą procesoriaus apkrovą sudarė MySQL procesas. Duombazės struktūroje buvo naudojami atitinkami indeksai šiam procesui paspartinti, be kurių užklausų vykdymas trukdavo dar ilgiau. Taip pat dėl didelės apkrovos centrinėje duomenų bazėje tokia architektūra nebūtų tinkama lygiagretiems skaičiavimams.

4. Algoritmų skaičiavimų laikų palyginimas

Parinktų algoritmų efektyvumui patikrinti buvo atlikti 7 bandymai su atsitiktiniu būdu parinktais maršrutais. Žemiau pateikti šio bandymo rezultatai.

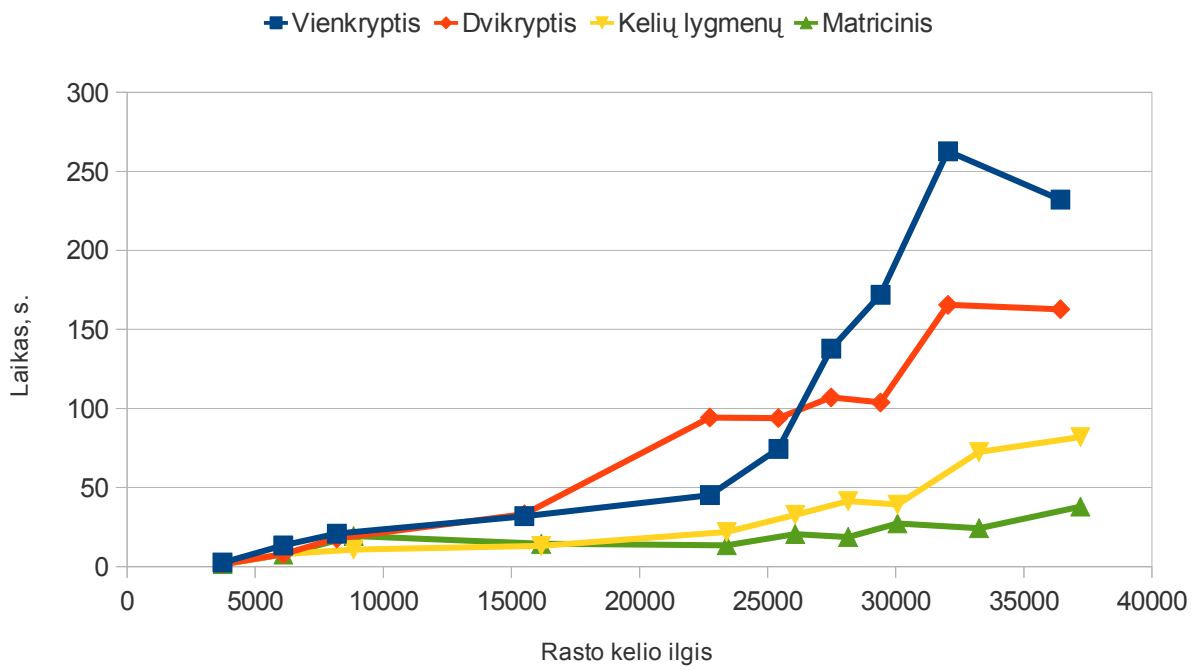


10 pav. Algoritmų skaičiavimo laikų palyginimo grafikas

Komentarai:

- Teoriniu požiūriu Dvikryptis algoritmas vidutiniškai turi būti dvigubai greitesnis už vienkrypti, nes jo analizuojamas plotas dvigubai mažesnis. Šis įvertis smarkiai išsikreipia dėl grafo netolygumų.
- Kelių lygmenų algoritmų skaičiavimo laikai yra mažiau priklausomi nuo maršruto ilgio. Tai rodo, kad jų sudėtingumas yra ne kvadratinis.
- Skaičiavimo laikų skirtumas didėtų naudojant didesnius grafus.
- Algoritmuose dar galima įvesti keletą optimizacijų, tačiau prognozuojamas pagreitėjimas yra minimalus.
- Skirtumai tarp algoritmų skaičiavimo laikų yra labai netolygūs. Tai yra dėl to, jog grafo tankis yra nevienodas, o algoritmų efektyvumas priklauso nuo grafo tankio skaičiuojamoje zonoje ir atstumo iki optimizuotos srities.

Tam kad geriau palyginti ir prognozuoti skaičiavimo laikus buvo atliktas dar vienas eksperimentas. Iš ankstesnio bandymo paimtas ilgiausias kelias (2-as bandymas), jis suskaldytas į 10 lygių dalių ir atlikta kelio paieška vis didesniu atstumu. Žemiau pateiktas šio eksperimento grafikas:

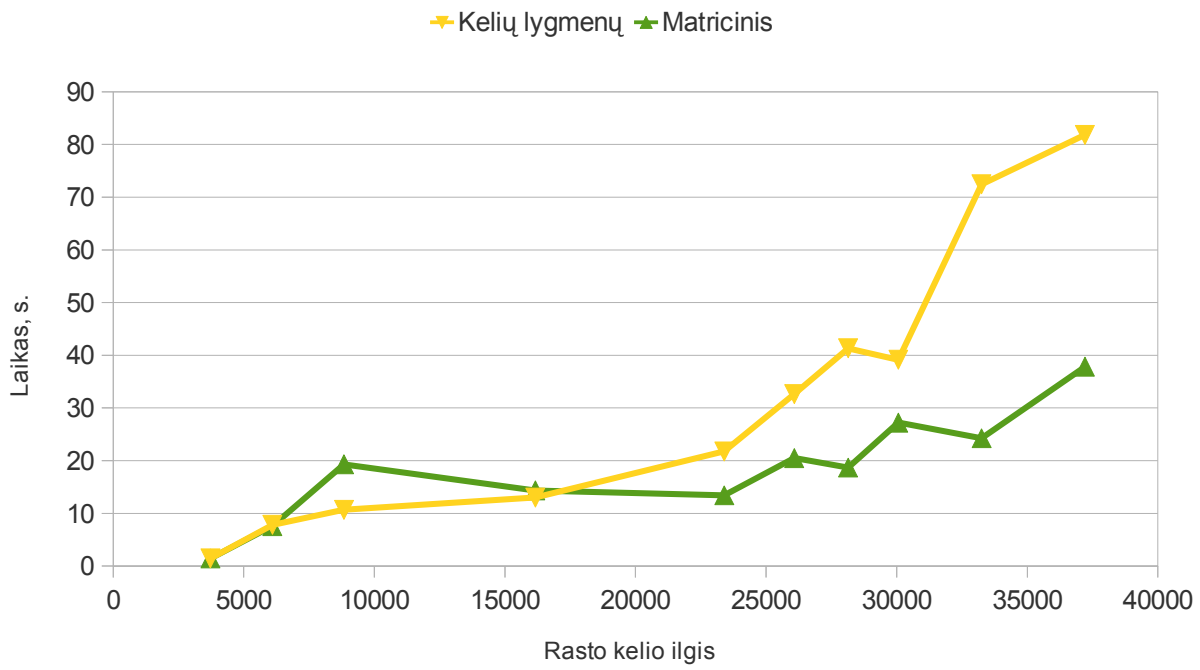


11 pav. Algoritmų skaičiavimo laikų palyginimo grafikas

Komentarai:

- Tolygaus skaičiavimo laiko išgauti nepavyko dėl tankio skirtumo apie kintančią tikslo viršūnę.
- Vienkrypčio algoritmo paskutinio bandymo laiko sumažėjimo neturėtų būti. Vieninteliai tokio skirtumo paaiškinimai gali būti klaida kode arba Python „garbage collector“ įtaka. Tai dar kartą parodo, kad šių bandymų paklaida yra didelė, tačiau bendra tendencija vis tiek pastebima.
- Vienkrypčio ir dvikrypčio algoritmų teorinis sudėtingumas – artimas kvadratiniam (atitinka teorinį).
- Optimizuota sritis pasiekiama tik nuo 3-io bandymo, tačiau pagreitėjimas nepastebimas, nes kelio dalis joje yra labai maža ir iškviečiamas beveik konstantinio sudėtingumo algoritmas.
- Optimizacija pastebima tik nuo 4-o bandymo.
- Didinant duomenų kiekį grafikų tendencija turėtų išlikti.

Žemiau pateikta tik dviejų paskutinių algoritmų skaičiavimo laikai, kad geriau matytųsi jų grafikai:



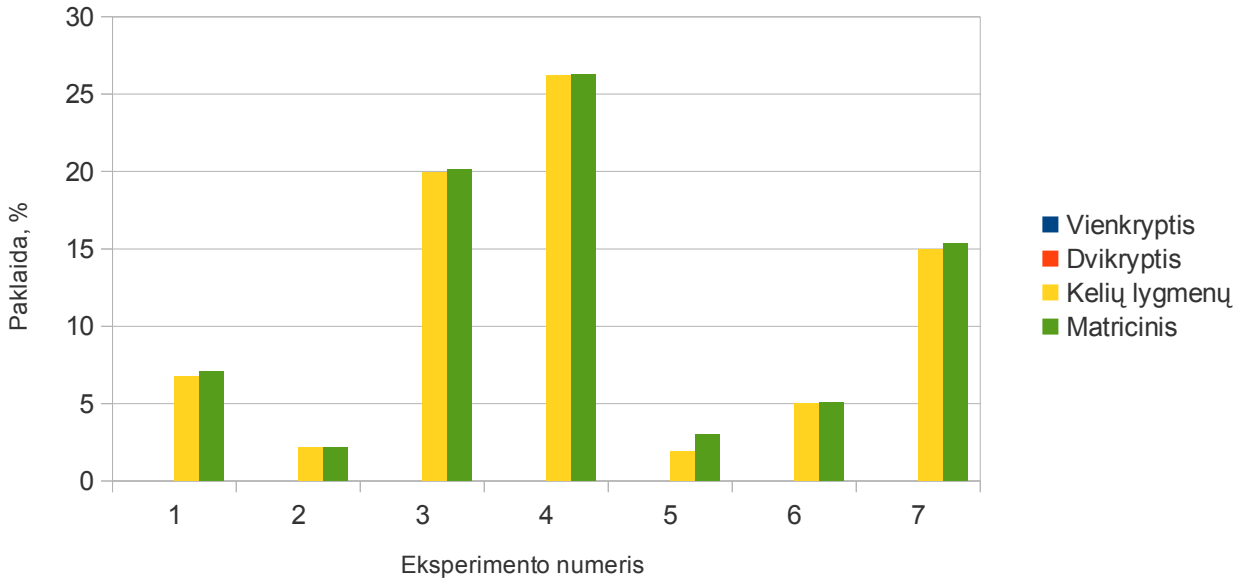
12 pav. Algoritmų skaičiavimo laikų palyginimo grafikas

Kaip ir reikėjo tikėtis matricinis algoritmas yra mažiausiai priklausomas nuo duomenų kiekio. Matematiškai išskaičiuoti šių algoritmų sudėtingumo lygį yra labai sunku, o šių duomenų nepakanka analitiniam sudėtingumo radimui. Tokiam tyrimui reiktų visiškai tolygaus grafo, tačiau toks grafas netinkamas sluoksniavimui, nes visos kelio atkarpos tampa lygiavertėmis. Dėl to negalimos šių algoritmų optimizacijos. Tikslingiausia būtų tokiems tyrimams naudoti realius duomenis, tačiau jų paruošimas užims per daug laiko.

Iš turimų duomenų galima spėti, kad matricinio algoritmo sudėtingumas bus bent tiesinio sudėtingumo, arba net logaritminio. Savaime aišku, kad šis rezultatas labai priklauso nuo turimų duomenų ir jų tinkamumo optimizacijoms.

5. Algoritmų skaičiavimų paklaidų palyginimas

Žemiau pateiktas parinkto kelio paklaidos grafikas. Paklaida buvo skaičiuojama imant santykinę paklaidą tarp vienkrypčio algoritmo rasto kelio ilgio ir kito algoritmo. Vidutinė matricinio algoritmo paklaida šių bandymų metu buvo 11%, kas turėtų būti žmogiškųjų klaidų ribose.



13 pav. Algoritmų skaičiavimo paklaidų palyginimo grafikas

Didinant sluoksnių kiekį paklaidos turėtų mažėti. Taip pat skaičiavimo tikslumą galima didinti įvedant keletą esamo lygmens prioritizavimų, tačiau algoritmas tampa labai komplikuotu ir aišku didėja skaičiavimo laikas.

Iš pirmo žvilgsnio atrodo, jog dvikryptis algoritmas turėtų visada rasti kelią be paklaidos. Taip pat šių bandymų metu visi sprendiniai sutapo su vienkrypčio algoritmo sprendiniais. Tačiau, taip nėra, nes tam tikrose situacijose galimos minimalios paklaidos paieškos apskritimų susilietimo srityje.

Kelių lygmenų ir matricinis algoritmai pagal jų architektūrą turėtų rasti identiškus sprendinius. Tačiau taip nėra, nes matricinis algoritmas griežtai laikosi kelio paieška tik esamame lygmenyje, o kelių lygmenų algoritmui leista matyti už esamo lygmens krašto. Tokiu būdu kartais „nukertamas kampas“ ties perėjimu tarp lygmenų.

6. Algoritmų užklausų kiekio palyginimas

Algoritmų skaičiavimo laikas priklauso nuo sistemos optimizavimosi. Duomenų bazės užklausų kiekis irgi yra tinkamas algoritmo efektyvumo vertinimo kriterijus. Jo trūkumas yra tai, kad užklausos nėra ekvivalenčios ir neįvertinamas sistemos optimizavimasis. Galutinis sistemos skaičiavimo laikas išlieka pagrindiniu vertinimo kriterijumi.

Žemiau pateikta vieno bandymo (iš viršūnės 832299 į viršūnę 731046) skirtingais algoritmais rezultatai. Buvo skaičiuojama užklausų kiekis į pagrindinę lentelę, kurioje saugomas pilnas kelių grafas.

Algoritmas	**Užklausų kiekis**
Vienkryptis	889931
Dvikryptis	699744
Kelių lygmenų	152285
Matricinis *	142129

** Matricinio algoritmo užklausos į matricos lentelę ir užklausos atkuriant pradinį kelią – neįskaičiuotos, nes naudoja kitokias užklausas arba lenteles su mažais duomenų kiekiais.*

Net ir naudojant optimizuotis algoritmus užklausų kiekis išlieka labai didelis. Naudojant realius duomenis, jie būtų dar didesni. Net ir parinkus sistemą su labai greita elementų paieška, kelio radimas nebus momentinis. Norint imituoti daugelio eismo dalyvių važiavimą, šis procesas tampa labai ilgu.

Dvikrypčiam algoritmui reikėjo 21% mažiau užklausų. Teoriškai vidutiniškai užklausų kiekis turėtų sumažėti 50%.

Kelių lygmenų algoritmui reikėjo 142000 užklausų pasiekti antrąjį lygmenį ir 10000 užklausų antrajame lygmenyje (išskaičiuojama iš skirtumo su matriciniu algoritmu). Tai parodo, kad pirmąjį lygmenį reiktų skaldyti į daugiau tarpinių lygmenų.

Sistemos analizė

1. Sistemos bandymų aplinka

Lygiagretinimo bandymas buvo naudojami du kompiuteriai. Anksčiau aprašytas (algoritmų bandymuose) kompiuteris atliko duomenų bazės serverio rolę, o skaičiavimai buvo atliekami šiuo kompiuteriu: Core i5-750 (Quad-Core), 8GB RAM, Ubuntu 10.10 x64. Kiekvienam iš keturių branduolių buvo leidžiama atskira programos gija. Kompiuteriai sujungti gigabitiniu tinklu.

Bandymams buvo naudojamas ne visas grafas (2 mln. briaunų), o tik jo dalis – 13663. Mažesnė zona pasirinkta dėl greitesnių skaičiavimų ir mažesnio reikalingo automobilių kiekio apkrovai sudaryti. Į šią zoną yra paleidžiama 100000 automobilių per sąlyginį sistemos laiką trunkantį 10000 vienetų. Pradinės ir tikslo viršūnės parinktos atsitiktinai. Automobiliai paleidžiami kas konstantinį 0,1 vnt. laiko intervalą.

2. Sistemos realizacijos problematika

Parinkta architektūra negali būti plečiama iki begalybės. Tai yra dėl centrinės duomenų bazės, kurios apkrova priklauso nuo skaičiavimo vienetų kiekio. Dabartinės bandymų aplinkos atveju (su aukščiau aprašytais kompiuteriais), klientinio kompiuterio visų branduolių apkrova buvo apie 70%, o DB serverio apie 50% (reiktų atkreipti dėmesį, kad kompiuterių pajėgumai labai skirtingi). Taip pat duomenų srautas tarp klientų ir duomenų bazės yra sąlyginai didelis: apie 384KB/s vienam skaičiavimo vienetui.

Tai reiškia, kad be tolimesnių SQL optimizacijų (arba specializuotos duomenų bazės sukūrimo), tokia sistema negali būti daugiau lygiagretinama. Parašius specialų serverį šio tipo duomenims saugoti ir padoroti, daugiau logikos būtų perkelta į serverio pusę mažinant tinklo srautą. Taip pat specializuota duomenų struktūra leistų greičiau apdoroti duomenis.

Taip pat, protingai padalinus grafą, gal būtų įmanoma suskirstyti skaičiavimo vienetus į zonas kuriose jie atlieka skaičiavimus. Tuomet mažėtų programos atminties poreikis.

3. Sistemos lygiagretinimo šalutiniai efektai ir galimybės

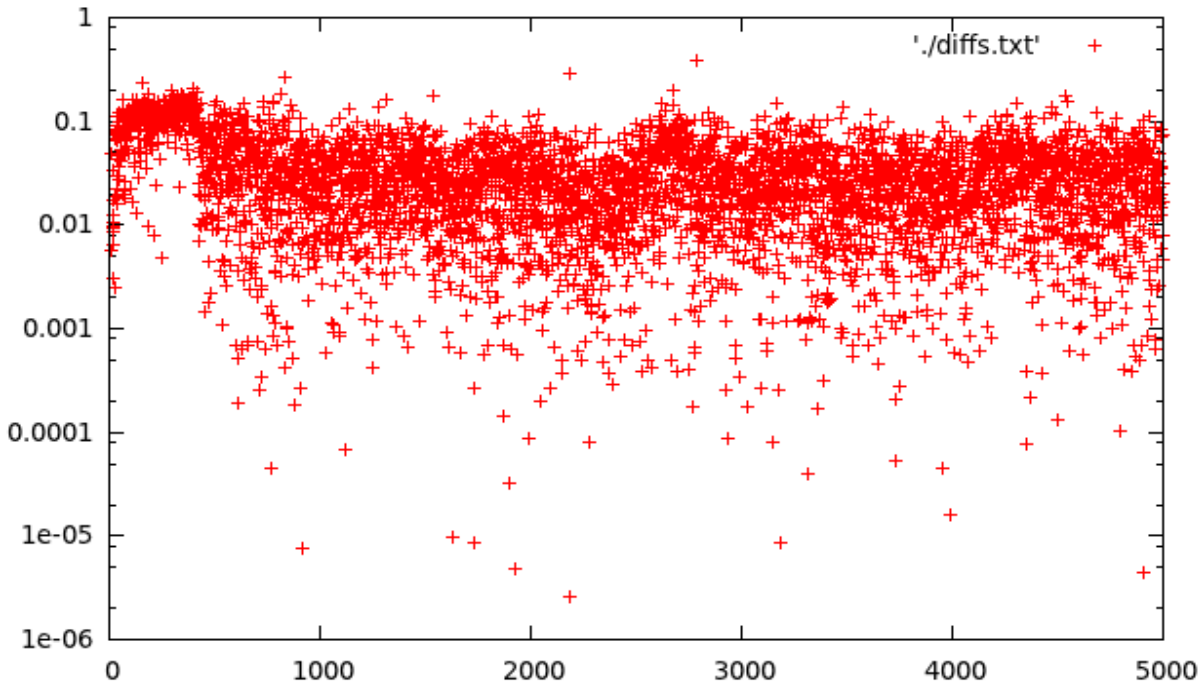
Skaičiavimo rezultatus siunčiant su uždelsimu ir skaičiavimo uždavinio skaidymas įneša papildomas paklaidas. Žemiau pateikta lentelė kuomet lyginamos paklaidos įvairiose situacijose. 1 procesoriaus su 10 darbų buferiu (labai mažu) ir laikoma kaip atskaitos taškas kitiems skaičiavimams. Paklaida yra vidutinis santykinis nuokrypis tarp rastų kelių ilgių.

Nr.	**buferis**	**procesorių**	**paklaida**
1	10	1	0
2	250	1	0.014981992581
3	1000	1	0.0180679103697
4	250	4	0.0280524677545

Panagrinėkime mechanizmą iš kur atsiranda paklaidos:

Uždelsiant rezultatų siuntimą, lokalioje atmintyje esantys duomenys pasensta. Modeliavimas vyksta neįtraukiant kątik rastų kelių. O rezultatų siuntimo momentu tam tikrų atkarpų apkrova gali neproporcingai pakilti, nes skaičiavimo eigoje sistema nemato tos kelio atkarpos apkrovos augimo. Tuo pačiu sekančioje iteracijoje ta perkrauta atkarpa bus per daug agresyviai vengiama. Dėl šių priežasčių gaunamas „bangavimo efektas“. Darbus vykdant gabalais po 250 ir naudojant 1 procesorių vidutinis nenusiųstų darbų kiekis yra 125.

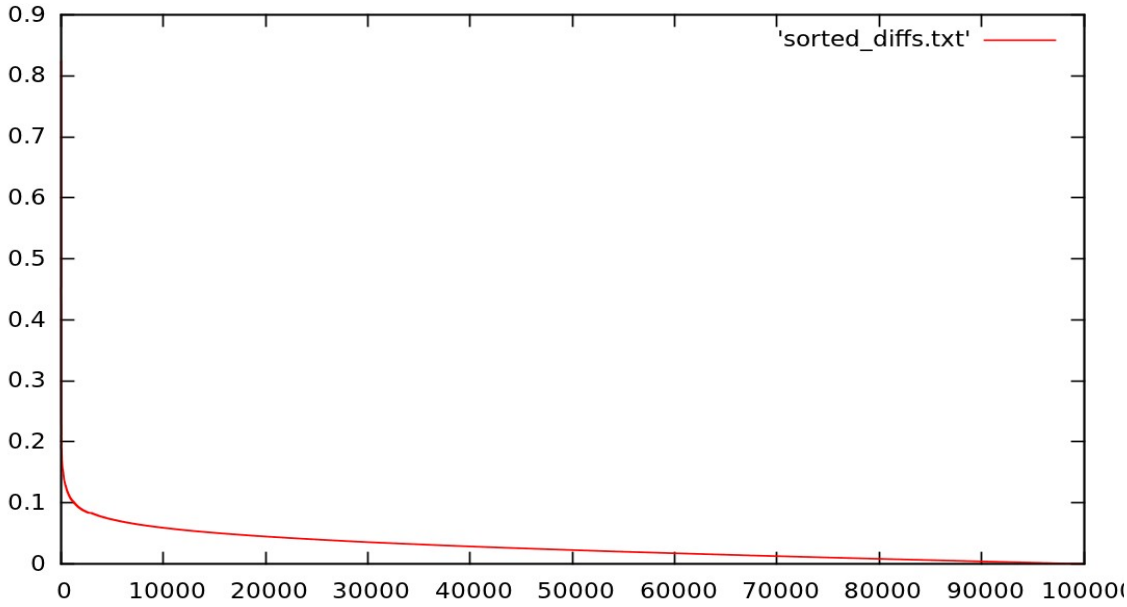
Lygiagretinant papildomai įnešama paklaida, dėl augančio nenusiųstų duomenų kiekio (250 buferis 4-iems procesoriams turės vidutinį 500 nenusiųstų darbų kiekį) ir dėl sistemos laiko nuokrypių. Vienas iš pavyzdžių būtų naudojant du procesorius su skirtingo sudėtingumo uždaviniais. Pirmasis procesorius greitai baigia skaičiavimo darbus ir nusiunčia rezultatus. Antrasis skaičiavimus baigia vėliau ir kuomet reikia perskaičiuoti apkrovas – jo skaičiavimai vyko ankstesniu laiku, nei naudojamas laikas duomenų bazėje. Rasti apkrovą skaičiavimo momentu ir pridėti papildomą tampa nebeįmanoma. Tuomet naudojama maksimali laiko vertė. Greičiausiai galimi ir kiti tokio laiko asinchronizacijos sprendimai.



14 pav. Sistemos lygiagretinimo paklaidų grafikas pagal iteracijas

Pateiktas 1-o ir 4-o bandymų skirtumo grafikas. Darant prielaidą, kad neišlygiagretintas skaičiavimas yra tikslus, gauname lygiagretinimo paklaidą. Pateikta pirmų 5000 automobilių duomenys, nes paklaidos lieka stabilios. Kaip pastebime pirmiem 500-am automobilių paklaida yra daug didesnė (šis skaičius sutampa su vidutiniu darbų kiekiu skaičiavimo buferyje naudojant 4 kompiuterius/procesorius).

Sistemą daugiau lygiagretinant pradinės paklaidos zona turėtų proporcingai didėti. Realaus modeliavimo atveju reiktų įvesti modelio apšilimo etapą, kuris ignoruotu šias pradines iteracijas arba apkrovą reiktų didinti palaipsniui.



15 pav. Sistemos lygiagretinimo surikiuotų paklaidų grafikas

Pateiktas rikiuotas kiekvieno modelyje dalyvaujančio automobilio rasto kelio paklaidų atsirandančių dėl lygiagretinimo grafikas. Pavienėse iteracijose paklaida siekia beveik 90%. Tačiau 95 percentilė yra lygi 0.072663 . Tai reiškia, kad 95% skaičiavimų atlikta su paklaida mažesne nei 7,3%.

4. Sistemos modelio generavimo pajėgumai

Su darbe naudojamu duomenų rinkiniu buvo atlikti trys laiko matavimo bandymai sistemos lygiagretinimo ir skaičiavimo pajėgumams nustatyti.

1. T5500 procesoriumi naudojant 1 skaičiavimo giją modelis sugeneruojamas per: 23 min. 44 s.
2. i5-750 procesoriumi naudojant 1 skaičiavimo giją modelis sugeneruojamas per: 21 min. 7 s.
3. i5-750 procesoriumi naudojant 4 skaičiavimo gijas modelis sugeneruojamas per: 8 min.

Tarp pirmo ir antro bandymų skirtumas yra mažas dėl tinklo įtakos. Pirmu bandymu duomenų bazę yra pasiekiama lokaliai. Laiko sutrumpėjimas su 4-iais kartais daugiau procesorių, yra 2,64 karto. Tiesinė priklausomybe neišlaikoma dėl bendro tinklo ir centralizuotos duomenų bazės.

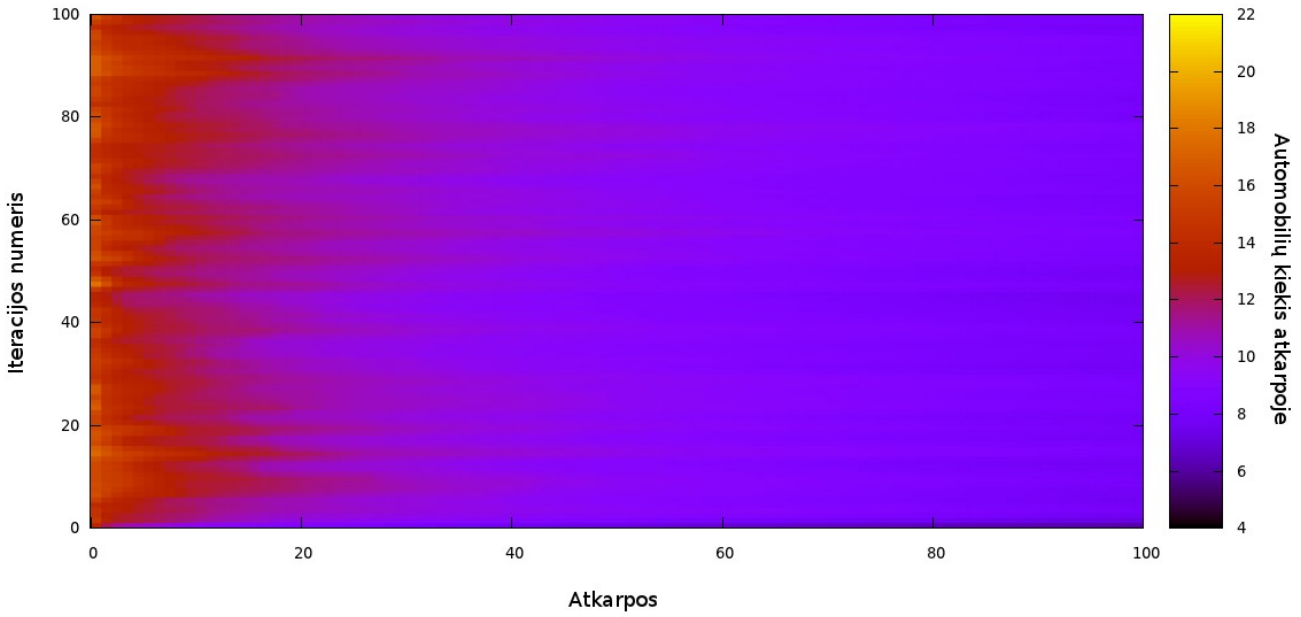
Modeliavimui naudojant visą grafą (2 mln. briaunų) vietoj jo dalies (13663), išlaikant tokį patį

automobilių tankį likusiame grafe ir išlaikant tokius pačius maršrutų ilgius reiktų apie 150 kartų daugiau resursų. Naudojant 3-io bandymo aplinką, tai truktų apie 20 valandų. Keičiant kiekvieną kitą parametrą, skaičiavimo laikas turėtų kisti ekvivalenčiai.

5. Sistema gauto modelio analizė

Grafo apkrovų kitimo stebėjimas buvo vykdomas tokiu būdu: kiekvieno rezultatų siuntimo momentu į tekstinį dokumentą surašoma visų kelio atkarpų apkrovų rikiuotas sąrašas. Vėliau iš šių dokumentų yra generuojami įvairūs pjūviai.

Pirmasis pjūvis: 100-o pirmų iteracijų (pirmi 25000 automobilių) 100-as labiausiai apkrautų atkarpų. X ašis atitinka 100 labiausiai apkrautų kelių surikiuotu sąrašu, Y ašis atitinka iteracijos numerį (santykinį sistemos laiką), o spalva- apkrovą toje kelio atkarpoje.



16 pav. Kelių apkrovos kitimas laike

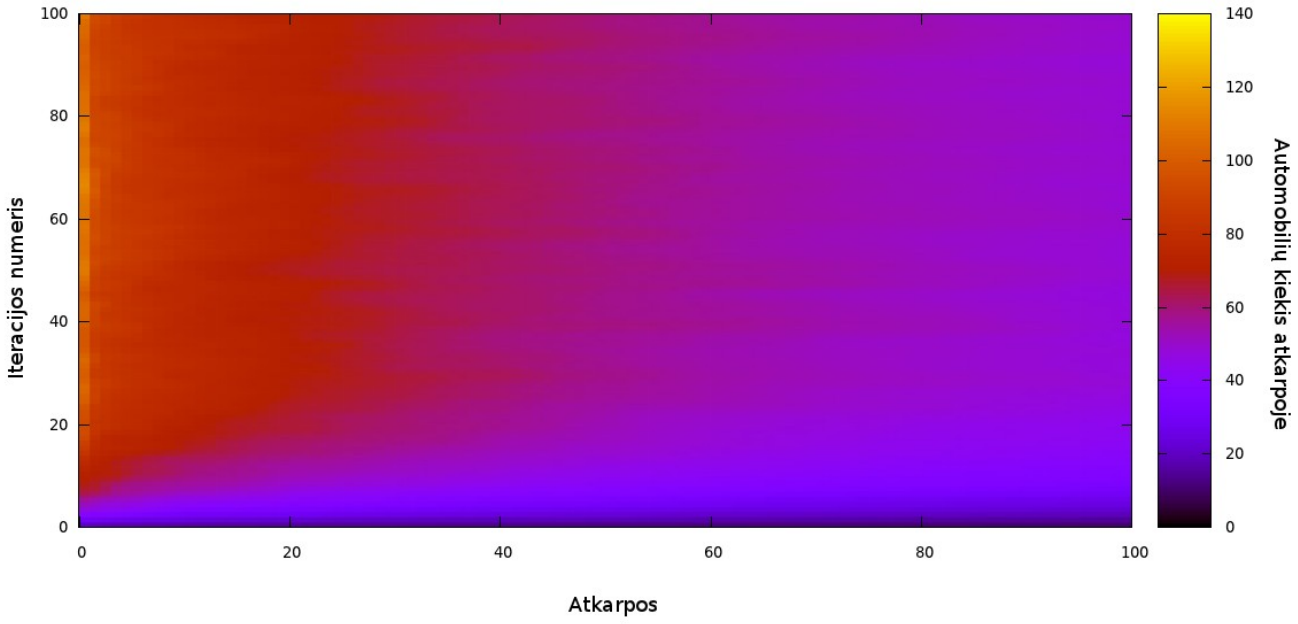
Komentarai:

- Aukštos apkrovos pastebimos tik labai mažoje dalyje atkarpų
- Apkrova neartėja link begalybės
- Gaunama pulsuojanti apkrova stipriai apkraunamose atkarpose
- Ribinė apkrova pasiekiama labai greitai

- Žema apkrova pasklinda po likusią grafo dalį. Retai naudojamos ir žemą apkrovą turinčio atkarpos klaidingai atsivaizduoja šiuose grafikuose, nes dėl reto jų naudojimo, retai perskaičiuojama to laiko momento apkrova.

Pagrindinė šio darbo prielaida, kurios tenkinimas turėtų išgauti realybę atitinkantį modelį yra apkrautų atkarpų vengimas privilegijuojant alternatyvius kelius. Jei ši savybė programoje būtų netinkamai realizuota, tai padidinus apkrovą ir išlaikant tuos pačius maršrutus, apkrovų pasiskirstymas turėtų išlikti panašus (ekvivalentus), arba augti sparčiau labiau apkrautose atkarpose. Tokį pasiskirstymą paaiškintų faktas, kad iš labiau apkrautų atkarpų automobilių išvykimas trunka dar ilgiau (eismas lėtesnis), kas reikštų dar didesnius kiekius automobilių šiose atkarpose.

Alternatyviai, jei šalutiniai keliai yra pasirenkami, apkrovai pakilus iki tam tikro lygmens- automobilių perteklius persikelia į gretimas atkarpas. O šių atkarpų perteklius pasirinktų dar kitas atkarpas ir t.t. Tai reiškia, kad maksimalios apkrovos turėtų kilti mažiau, nei proporcingai, o apkrovos perteklius atitekti kitoms kraštinėms. Žemiau pateiktas grafikas su 10 kartų trumpesniu eismo grafiku (10 kartų didesni srautai), kuris patvirtina šias prielaidas.



17 pav. Kelių apkrovos kitimas laike

Maksimalios apkrovos pakyla apie 5 kartus. Stipriai apkrautų atkarpų kiekis pakyla nuo maždaug 10 iki maždaug 30. Taip pat grafe galutinė apkrova pasiekiama per 20 iteracijų, vietoj 5, nes įsisavinamas

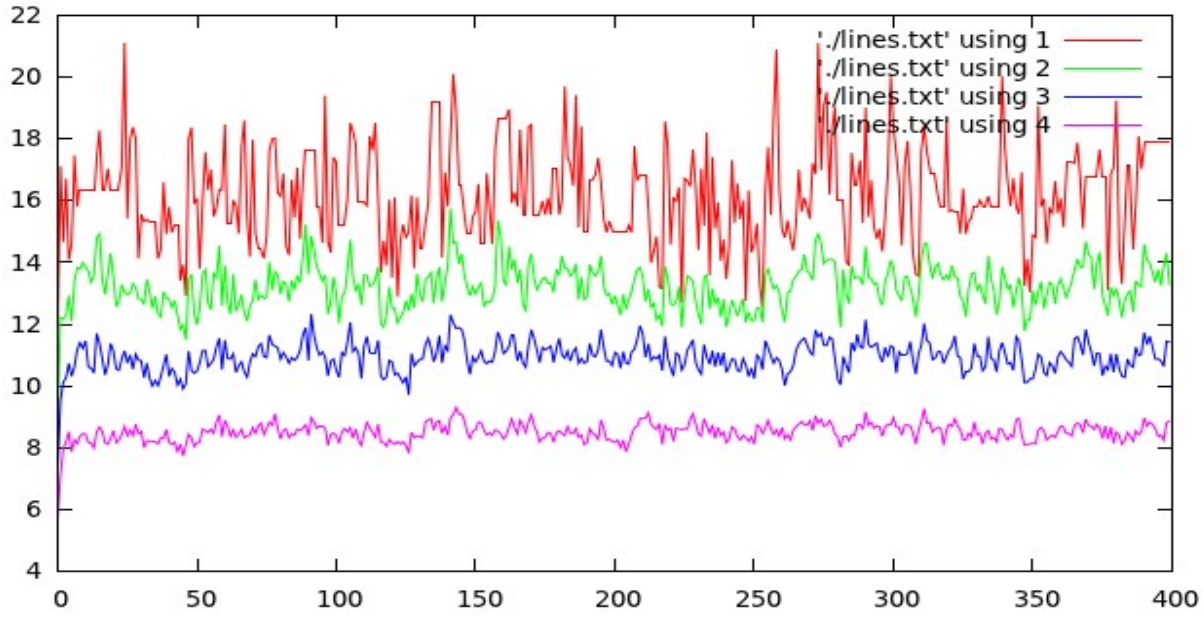
daug didesnis automobilių kiekis.

Abiejuose grafikuose išlieka svyruojančios pakrovos stipriai apkrautuose keliuose. Atliekant nuolatinius apkrovos perskaičiavimus retai naudojamose atkarpose, galbūt panašūs svyravimai būtų pastebimi ir šiose atkarpose.

Jei realiame eisme egzistuotų panašūs svyravimai, tokius duomenis galima naudoti ekstremumų skaičiavimams. Dažniausiai naudojamas tokios statistikos pavyzdys būtų „100-o metų potvynis“. Tai reiškia tokį potvynį, kurio tikimybė įvykti yra 1 kartas per 100-ą metų. Ekvivalenčiai eisme būtų galima skaičiuoti dienos, savaitės ar metų kamščius gautiems modeliams. Egzistuojančios statistikos yra tinkamos periodiškai kintamiems duomenims. Paros ir savaitės svyravimai apkrovose statistikoje gali būti kompensuojami.

Žemiau pateikti viso proceso (400 iteracijų, mažesnio srauto duomenys) atkarpų apkrovų grafikai (nuo aukščiausio):

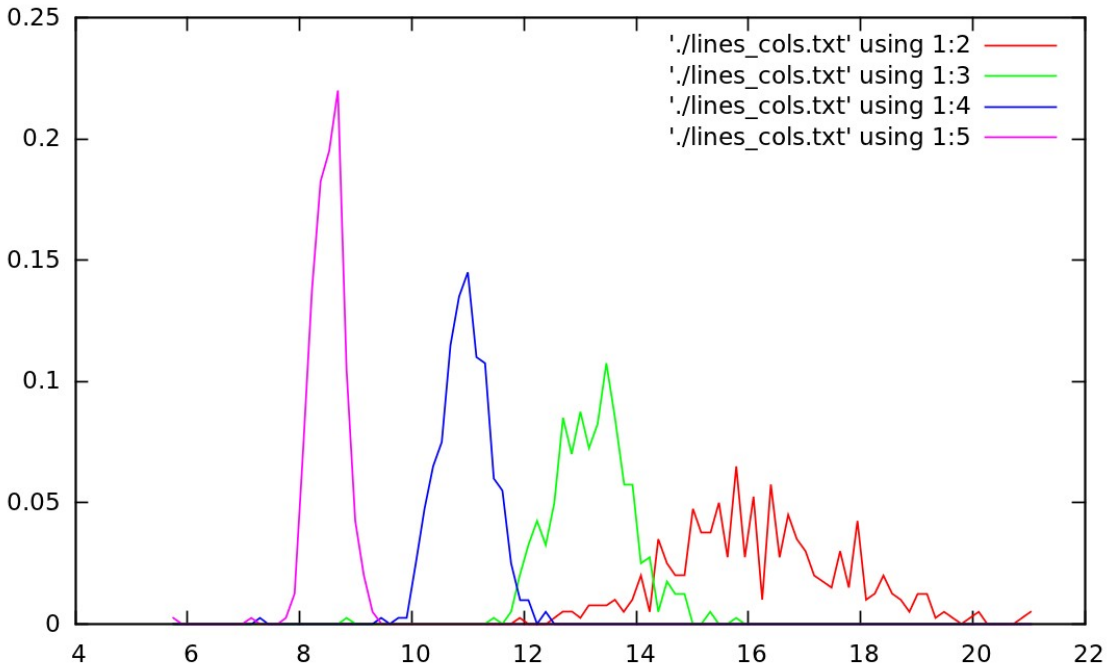
1. Maksimali apkrova kelio atkarpoje per iteraciją
2. 2-11 labiausiai apkrautų atkarpų vidurkis
3. 12-36 labiausiai apkrautų atkarpų vidurkis
4. 37-136 labiausiai apkrautų atkarpų vidurkis



18 pav. Kelių apkrovos kitimas laike

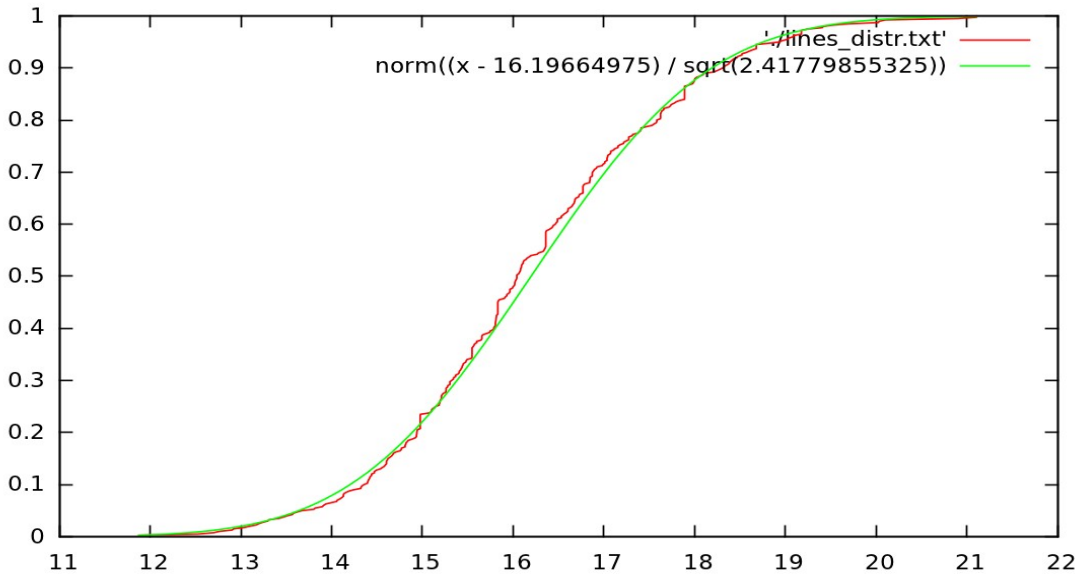
- 3 ir 4 grafikai yra labai stabilūs (daugiausia dėl didelio kiekio atkarpų vidurkio skaičiavime)
- Maksimali apkrova modeliavimo eigoje beveik neauga. Maksimumo linijinė aproksimacija: $f(x)=0.00091x+16.0141$. Tai reiškia, kad apkrova modeliavimo eigoje nesikaupia ir modelis stabilizuojasi labai greitai. Galutiniam modeliui gauti užtektų daug trumpesnio laiko intervalo, nei yra naudojamas bandymuose.
- Maksimumo f-jos vidurkis = 16.19664974, dispersija = 2.41779855325

Žemiau pateikti tų pačių grafikų histogramos.



19 pav. Kelių apkrovos pasiskirstymo histogramos

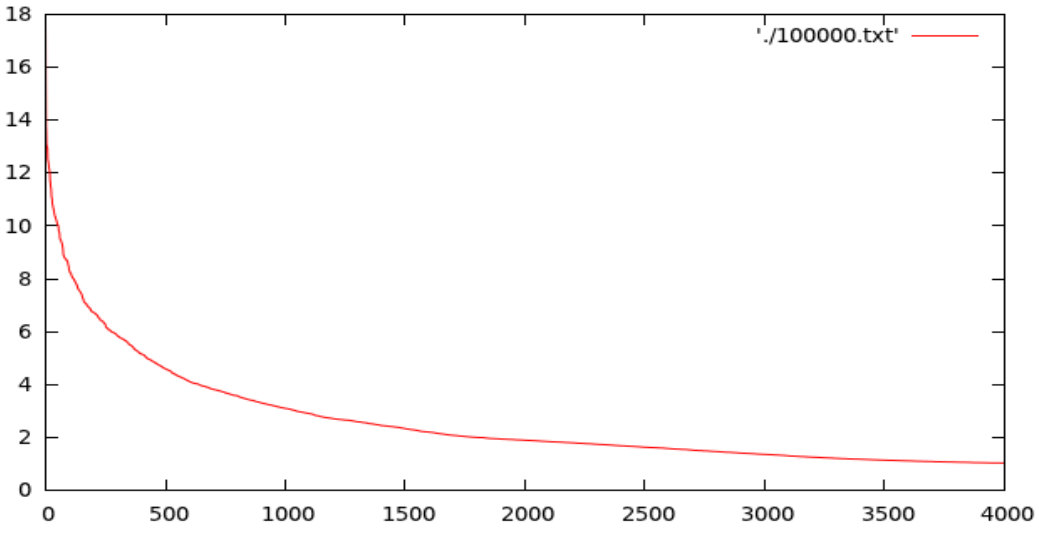
Maksimumo ir 2-6 atkarpų grafikai artimi normaliajam skirstiniui. Palyginkime:



20 pav. Kelių apkrovos maksimumo pasiskirstymo funkcijos palyginimas su normaliojo

Palyginus maksimumo pasiskirstymo funkciją su atitinkama normaliojo skirstinio pasiskirstymo funkcija, matome, kad maksimumai tikrai yra normaliojo skirstinio. Šią savybe galima naudoti ekstremumų statistikose.

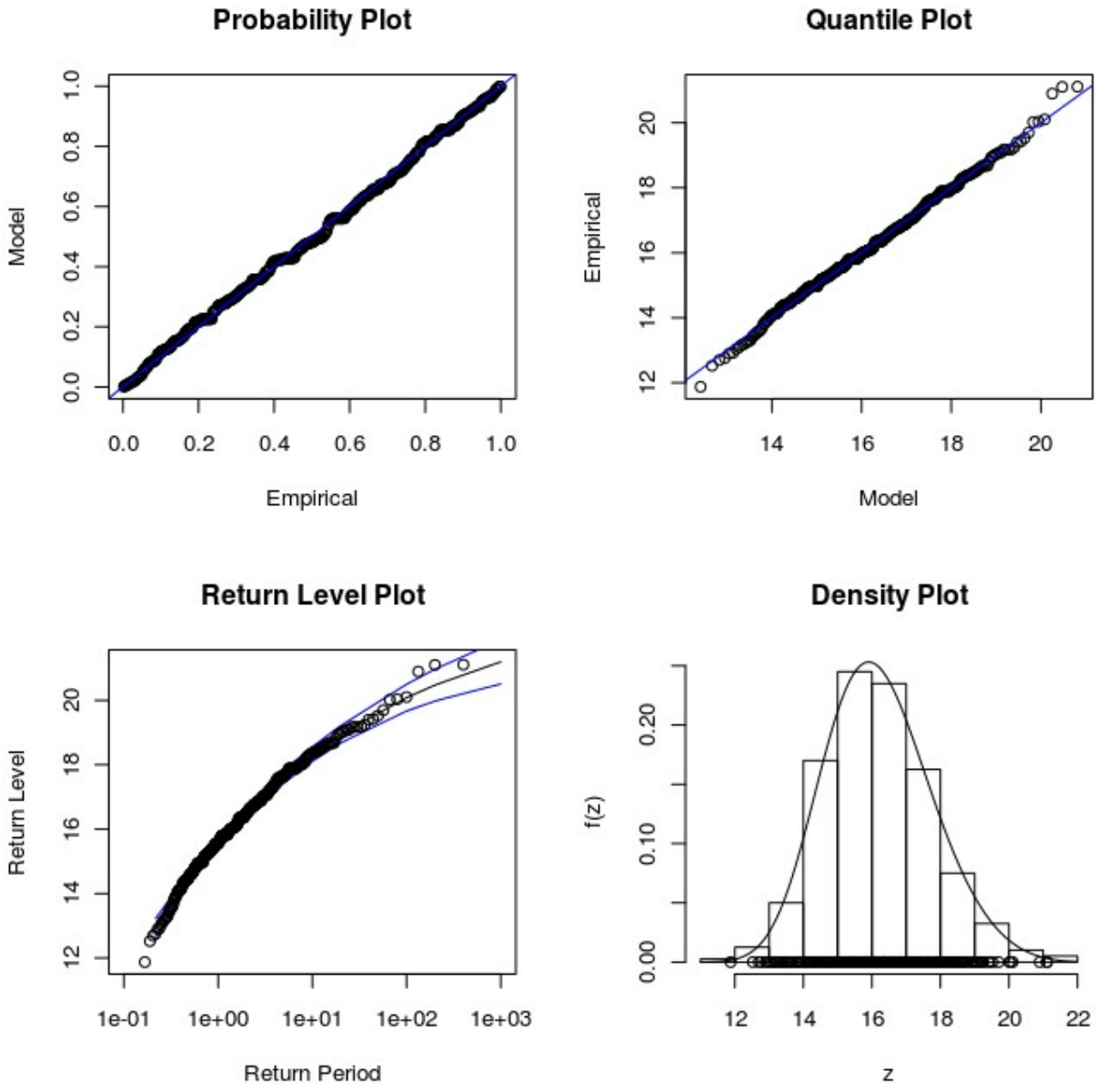
Baigus modeliavimą gautos tokios atkarpų apkrovos pateiktos grafike. Sąrašas yra surikiuotas ir vaizduojama pirmi 4000 atkarpų.



21 pav. Galutinių apkrovų pasiskirstymo grafikas

6. Ekstremumų statistika

Apkrovų maksimumų duomenis galima naudoti prognozavimui. Tam yra skirti paketai „evd“ ir „ismev“ programai R. Žemiau atvaizduojamos pagrindinės statistikos:



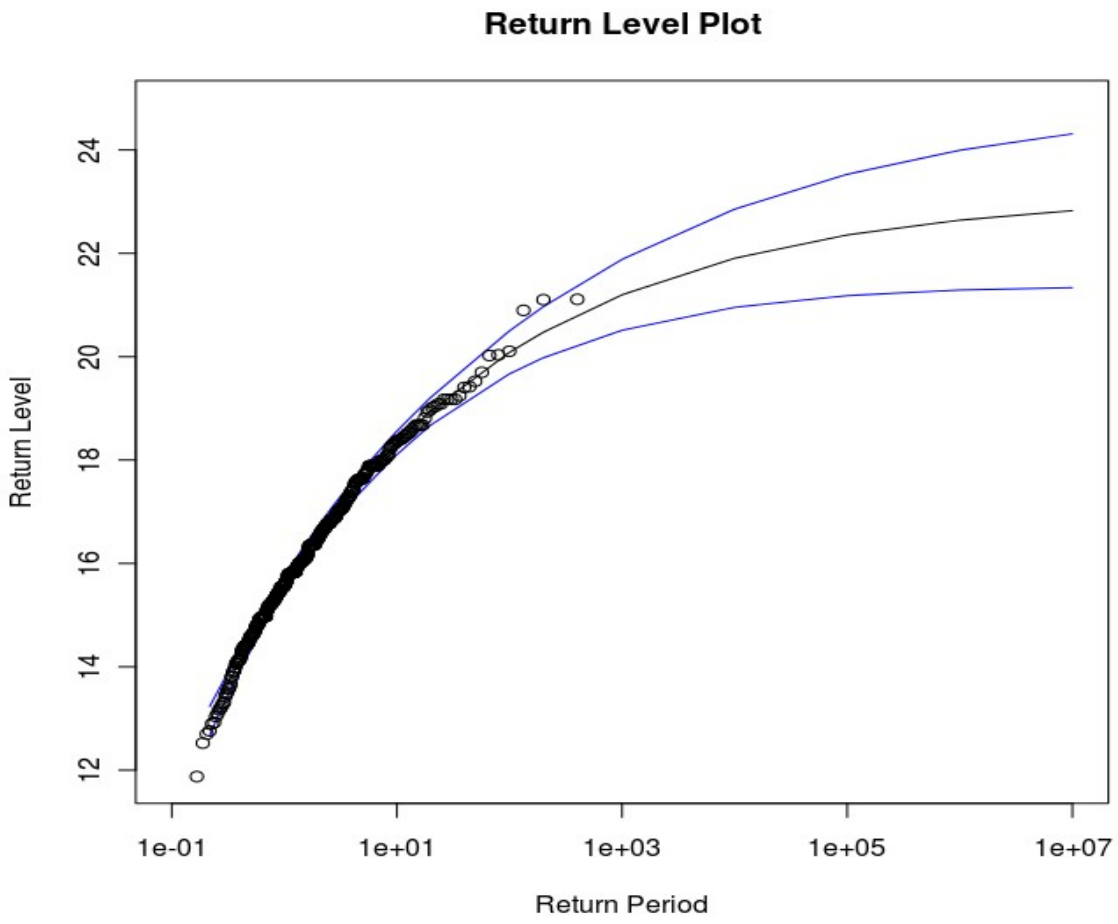
22 pav. Ekstremumų statistikos programoje R

Probability Plot ir Qantile Plot irgi patvirtina, kad maksimumai pasiskirstę pagal normalųjį skirstinį. Pastebimi maži nuokrypiai kraštuose, bet tai yra normalu, nes šiuose ruožuose mažai duomenų. Density

Plot atitinka anksčiau gautą. Return Level Plot nurodo apkrovos pasiskirstymą pagal tikimybes. Šis grafikas bus detaliau nagrinėjamas vėliau. Grafikai gali būti sugeneruoti šiuo kodu:

```
library(evd)
library(ismev)
data<-scan('line_single_max.txt')
fit<-gev.fit(data)
gev.diag(fit)
```

Žemiau pateiktas Return Level grafikas su praplėstais rėžiais (modifikuota gev.rl funkcija):



23 pav. Return Level statistika programoje R

Grafiko y ašyje yra pateikiama statistikos reikšmė (šiuo atveju automobilių kiekis atkarpoje).

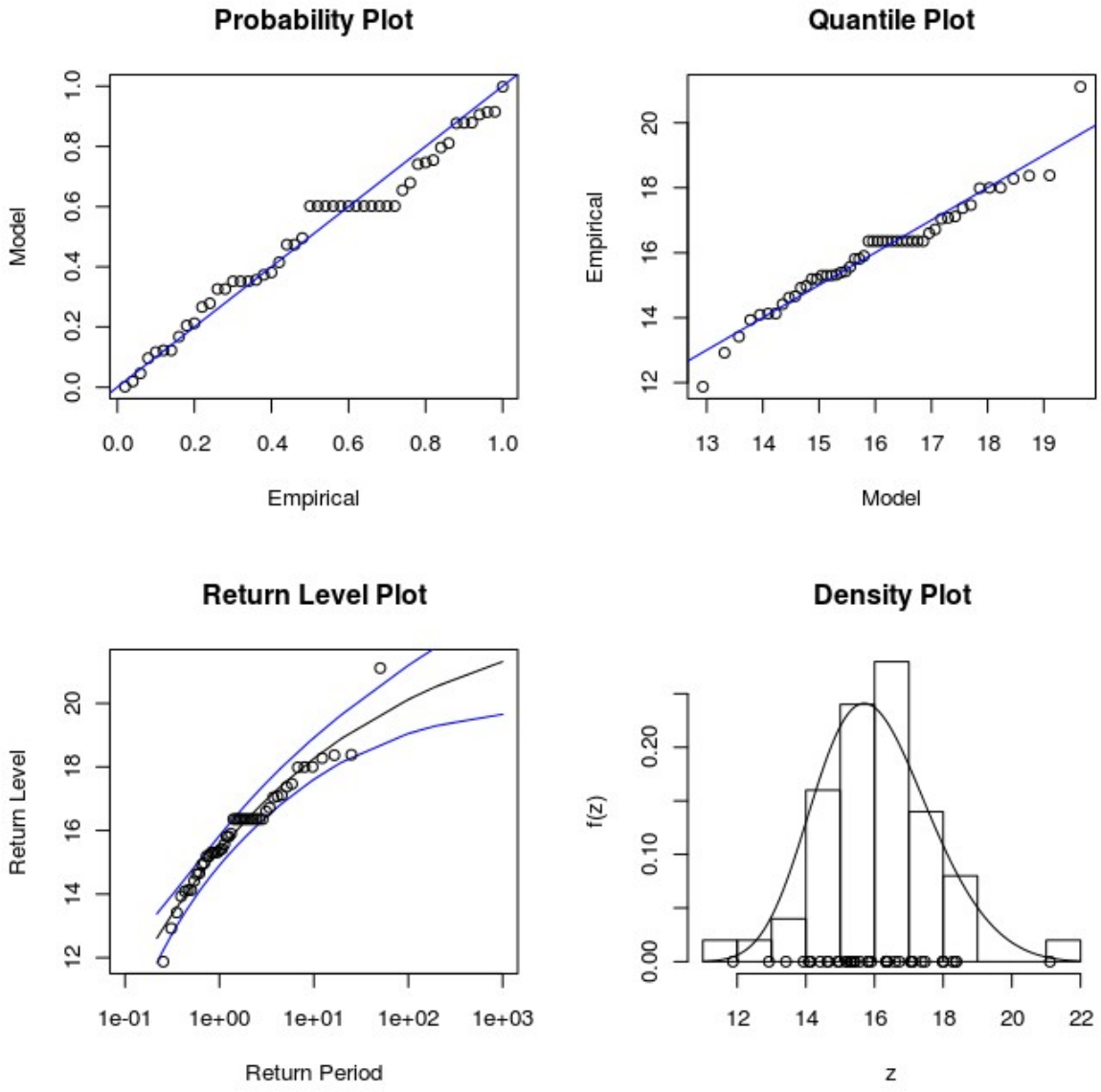
Grafiko x ašyje pateikta šios statistikos tikimybė: $\frac{1}{P(x>X)}$. Tai yra atvirkštinė tikimybė, kad atsitiktinis dydis neviršys reikšmės x. Tokį formatą taip pat galima apibrėžti kaip 1:N statistiką. Tai reikštų statistiką kas kiek imčių N atsitiktinis dydis viršija reikšmę x. Taip gaunamas „100-o metų audrų“ įvertis.

Šis grafikas patvirtina, kad didinant iteracijų kiekį maksimumai keisis mažai. Tikėtinas apkrovos maksimumas per 10^7 iteracijų (pagal atliktų 400 iteracijų duomenis) yra apie 23. Taip pat aproksimacijos grafikas greitai artėja prie konstantos. 95% pasikliautinasis šio įverčio intervalas yra apie [22;24]. Per 400 iteracijų apkrovos maksimumas buvo 21,1106 . Taip pat pagal grafiką galima nustatyti tikėtinus maksimumus mažinant iteracijų skaičių.

Laiko sumetimais galima būtų naudoti tokią schemą: modeliuoti trumpus intervalus, kol modelis pradeda stabilizuotis. Tuomet išskaičiuoti tikėtinus ilgalaikius maksimumus remiantis šia statistika. Žemiau patektas statistinis modelis su pirmu 50 iteracijų iš 400.

Kaip matoma, 50 iteracijų yra labai maža imtis. Tačiau išlaikoma normaliojo skirstinio požymių. Tikėtinas apkrovos maksimumas per 400 iteracijų pagal Return Period grafiką yra apie 20,5 su pasikliautinuoju intervalu apie [19,24]. Pasikliautinasis intervalas yra platus dėl mažos imties, bet aproksimuota maksimumo reikšmė yra artima gautai viso modeliavimo metu (21,1).

Apkrovos stipriai apkrautose atkarpose irgi turėtų būti pasiskirsčiusios normalųjį skirstinį. Tai reiškia, kad panašias statistikas ir principus galima taikyti apkrovų maksimumų prognozavimui pavienėse kelio atkarpose. Kaip modeliavimo pabaigos parametrą galima skaičiuoti maksimumų pasikliautinuosius intervalus. Kuomet šis pasiektų pakankamai mažą, modeliavimą galima stabdyti, o ilgalaikes modelio savybes išskaičiuoti.



24 pav. Ekstremumų statistikos programoje R

Naudojant kitokį eismo grafiką, gautos apkrovos nebūtų stabilios. Reiktų šalinti apkrovų periodiškumą (paros arba savaites periodai). Taip pat dėl netolygaus eismo grafiko išsiplėstų apkrovų skalė. Maksimumų skaičiavimui gali reikėti naudoti duomenų pjūvius. Visos priemonės atlikti šias transformacijas yra pateiktos bibliotekose „evd“ ir „ismev“. Metodai atlikti šias statistikas (ir anksčiau aprašytas) yra plačiau išnagrinėtos šaltinyje [Col08].

Rekomendacijos

Modifikacijų pagrindu pasirinktas Dijkstra algoritmas. Pritaikius visas siūlomas modifikacijas jo skaičiavimo laikas turėtų būti artimas nemodifikuotam A* algoritmui [Wiki1]. Daugumoje situacijų jis yra greičiausias egzistuojantis algoritmas. Problematika yra tai, kad jis remiamasi geografiniais atstumais tarp analizuojamos viršūnės ir tikslo viršūnių (metrinė erdvė). Toks matmuo egzistuoja, kuomet grafo svoriai yra fiziniai atstumai (ieškoma trumpiausio kelio). Tokiame grafe galima apibrėžti metriką (atstumą tarp dviejų taškų). Tačiau, ieškant greičiausio kelio, grafe saugoma atkarpų pravažiavimo laikai, kurie mažai priklauso nuo fizinio kelio atkarpos ilgio. Tokiame grafe neįmanoma apibrėžti metrikos. Galima nebent sukonstruoti jos įvertį, nuo kurios smarkiai priklausys algoritmo skaičiavimo laikas ir tikslumas. Taip pat šis algoritmas netinkamas kelių lygmenų grafams.

Pasirinkta programavimo kalba: Python. Kuriant sistemą skirtą ne vien algoritmų tyrimams, reiktų naudoti žemo lygio programavimo kalbą pvz C++. Python programos yra lengvai rašomos ir jai skirta daug bibliotekų, tačiau jos procesoriaus ir atminties „overhead“ yra labai didelis. Tai riboja duomenų kiekį kurį galima užkrauti ir lėtina skaičiavimus.

Realizuojant sistemą buvo pasirinkta SQL duomenų bazė informacijai saugoti. Tai yra dar vienas programavimą lengvinantis sprendimas. Tačiau taip saugomų duomenų paieška ir keitimas yra labai lėti. Parašius specializuotą serverį duomenis būtų galima saugoti tokioje struktūroje kurioje paieška veiktų greičiau. Taip pat duomenų rašymas būtų greitesnis. Problematiška sritis yra procesų sinchronizavimas. Keliems klientas vienu metu skaitant arba rašant duomenis gali susigadinti duomenys. Tokių problemų kilo net naudojant SQL duomenų bazę.

Išvados

Darbe buvo suformuluotas uždavinys sprendžiantis eismo imitavimo problemą. Taip pat parinkti duomenų formatai ir papildomi specifiniai parametrai reikalingi tokiam uždaviniui spręsti. Parinkti duomenys kurie bus renkami modeliavimo metu.

Aprašyti dėsniai, kurie yra pastebimi kelio pasirinkimo procese vykstant automobilių eismui. Remiantis šiais dėsniais buvo parinkti pavienio trumpiausio kelio radimo algoritmai ir suformuluoti nauji specifiniai eismo dalyvių apjungimo į visumą algoritmai.

Pasirinkta pagrindinė eismo savybė- kliūčių vengimas. Remiantis šia savybe buvo sudarytas ir realizuotas eismo dalyvių įtakos vienas kitam algoritmas. Sistema buvo sugeneruoti keli eismo modeliai. Juos lyginant matoma, kaip apkrova perkeliama gretimoms atkarpoms. Tai reiškia, kad kliūčių vengimo algoritmas yra veiksmingas.

Bendrai analizuojant gautą modelį, galima teigti, kad jis tinkamai mėgdžioja realų eismą. Modelis nei konverguoja, nei diverguoja, tačiau išlieka kintantis. Šie atsitiktiniai dydžiai pasiskirstę pagal normalųjį skirstinį. Šią savybę galima išnaudoti skaičiuojant įvairias statistikas, pavyzdžiui ribines apkrovas ir ilgalaikes prognozes. Pateikti keli galimi statistiniai skaičiavimai. Pabrėžtina, kad šios savybės buvo gautos generuojant vieną modelį ir nebūtinai galioti visiem galimiem modeliam. Taip pat generuojant yra gaunamos žymios paklaidos.

Sistema kuria yra generuojami modeliai yra tinkama lygiagretiesiems skaičiavimams. Apdorotinų duomenų kiekis yra visos Europos kelių lygmens. Trumpalaikis modelis tokia sistema gali būti sugeneruotas per „protingą laiką“. Tyrimo sumetimais modeliuojama buvo tik mažoje grafo dalyje.

Taip pat pasiūlytas algoritmas, kaip greitai perskaičiuoti pagrindinių kelių mazgų apkrovas. Tačiau tai nebuvo realizuota dėl laiko stokos. Taip pat šis algoritmas efektyvus tik modeliuojant visame grafe.

Naudota literatūra

[SS06] Presentation. Transit Node Routing based on Highway Hierarchies . Peter Sanders , Dominik Schultes . Institut für Theoretische Informatik – Algorithmik II Universität Karlsruhe (TH) New York City, November 16, 2006 http://algo2.iti.kit.edu/schultes/hwy/newYork.pdf

[Sch08] PhD Thesis. Route Planning in Road Networks . Dominik Schultes . von der Fakultät für Informatik der Universität Fridericiana zu Karlsruhe (TH) , 7. Februar 2008 http://algo2.iti.uka.de/schultes/hwy/schultes_diss.pdf

[Rim10] Bakalauro baigiamasis darbas. Greitųjų kelio paieškos žemėlapiuose algoritmų tyrimas. Raimondas Rimkus. Kauno Technologijos Universitetas. Kaunas. 2010.

[Rei03] Lecture: The Floyd-Warshall Algorithm. Erik Reinhard. University of Bristol Department of Computer Science. 2003. www.cs.ucf.edu/~reinhard/classes/cop3503-fall03/floyd.pdf

[Col08] Lecture notes. Statistical Modelling of Extreme Values . Stuart Coles and Anthony Davison. 2008 https://edit.ethz.ch/cces/projects/hazri/EXTREMES/talks/colesDavisonDavosJan08.pdf

[Wiki1] A* search algorithm, Wikipedia http://en.wikipedia.org/wiki/A*

Priedai

1. Klasių aprašai

db.py:

- Aprašytas duomenų bazės modelis
- Naudojama SQLAlchemy biblioteka
- Importuojant klasę, automatiškai sukuriama atitinkama struktūra duomenų bazėje

data.py:

- Lokali duomenų kopija greitesnei paieškai (išjungiama opcija)
- Bendriniai metodai duomenų paieškai ir apdorojimui

matrix.py:

- Metodai susiję su matricine duomenų struktūra
- Klasė neturi lokalios duomenų kopijos (opcijos naudą reiks ištirti)

finderX.py:

- Atitinkamo kelio paieškos algoritmo realizacija
- Visos klasės naudoja vienodai kviečiamą metodą find
- Metodui paduodama pradinio, bei tikslo kelių numeriai.
- Metodas grąžina kelių numerių masyvą kurie nurodo parinktą kelią ir laiką, per kurį tas kelias bus nuvažiuotas.

worker.py:

- Programa kuri leidžiama kiekvieno klasterio skaičiavimo vienete
- Pagal sudarytą darbų sąrašą atlieka skaičiavimus ir siunčia rezultatus

matrix_updater.py:

- Programa skirta grafą atitinkančios matricos pergeneravimui skaičiavimo eigoje.
- Modeliavime nenaudota dėl pasirinktos mažos grafo zonos bandymams

2. Pagalbinių klasių aprašai

genmatrix.py:

- Matricos generavimo modulis
- Netinkamas lygiagrečiam skaičiavimui su paieška

imp.py:

- Testinių duomenų iš tekstinio failo importavimo modulis
- Galima naudoti kaip pavyzdį duomenims importuoti iš kitų šaltinių

test.py:

- Skaičiavimo sesijos pavyzdys

genqueue.py:

- Eismo tvarkaraščio generatorius

prep.py:

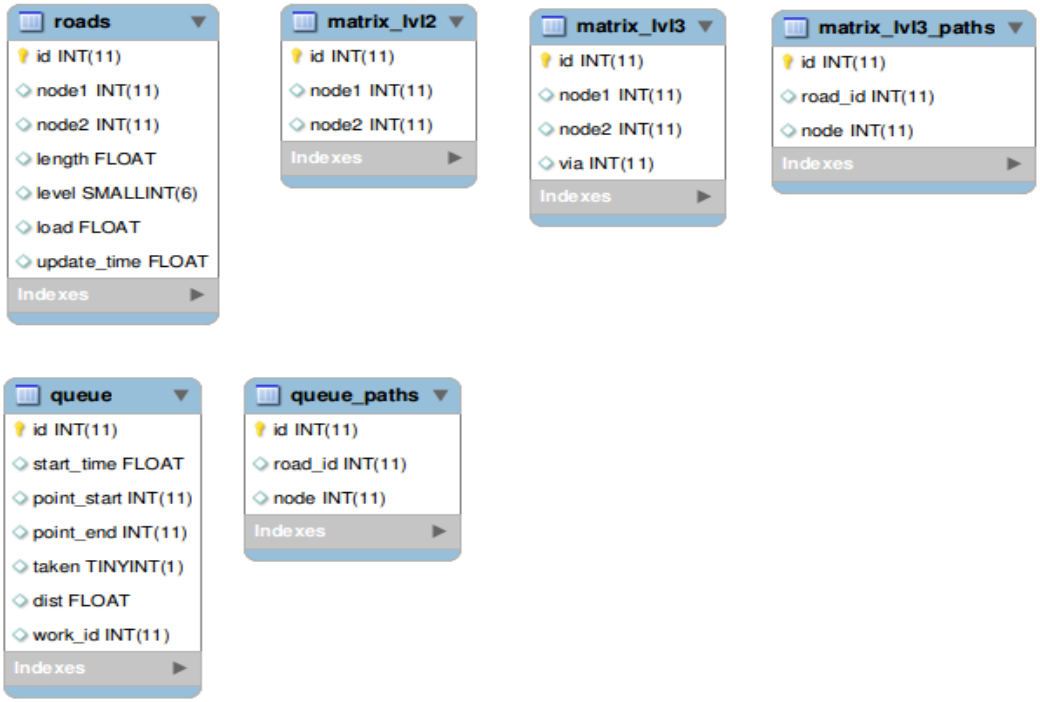
- Išvalo duomenų bazėje esančius skaičiavimo rezultatus
- Pažymi esamą eismo tvarkaraštį kaip neįvykdytą

.\loads*.py:

- Skriptai rezultatų pjūviams generuoti ir apdoroti

3. Duomenų bazės architektūra

Žemiau pateikta duomenų bazės schema.



25 pav. Duomenų bazės schema

roads:

- Pagrindinė duomenų lentelė. Saugomas pilnas kelių grafas.
- Duomenų bazės eilutė atitinka kelio atkarpą tarp dviejų sankryžų.
- node1 ir node2 – sankryžų kurias kelias jungia numeriai
- length – kelio atkarpos ilgis
- level – kelio atkarpos kokybinis lygmuo
- load – automobilių kiekis kelio atkarpoje
- update_time – laikas kuomet automobilių kiekis buvo paskaičiuotas

matrix_lvl2:

- Iš roads lentelės išrinktos kelio atkarpos, kurios priklauso aukščiausiam (antram) lygmeniui.
- Identiška struktūra roads lentelei, tik nesaugomas lygmuo ir atstumas.

matrix_lvl3:

- Iš aukščiausiojo lygmens atrinktų viršūnių, kurios transformuotos į matricą sąrašas.
- node1 ir node2 – viršūnių pora matricoje
- via – indeksas įrašo per kurį rekursiškai atkuriamas kelias

matrix_lvl3_paths:

- Saugomi sąrašai viršūnių per kurias pasiekiamos gretimos matricoje saugomos viršūnės.
- road_id – indeksas kelio kuriam skirtas šis kelias (foreign key)
- node_id – kelio viršūnės indeksas
- order – viršūnės eilės numeris kelyje

queue:

- Eismo tvarkaraščio lentelė
- start_time – išvykimo laikas
- point_start – išvykimo taškas
- point_end – tikslo taškas
- taken – elemento paėmimo į skaičiavimo sistemą požymis
- dist – rasto kelio atstumas
- work_id – užduočių komplekto id

queue_paths:

- Rastų kelių eismo dalyviams sąrašas
- rode_id – eismo dalyvio id
- node – kelio id sąraše

4. Sistemos paruošimas

Trumpa instrukcija kaip paruošti ir pasileisti sistemą:

1. Reikalinga Linux operacinė sistema. Windows sistemoje programa nebuvo bandoma. Galimos klaidos duomenų failų skaityme arba prisijungime į duomenų bazę.
2. Kompiuteris kuriame leidžiama sistema turi turėti 2GB RAM atminties. Dėl neefektyvaus atminties panaudojimo Python kalboje programai ir duomenų bazei veikti reikalinga apie 1GB atminties resursų.
3. Surašyti MySQL duomenų bazę, Python 2.7 interpretatorių su SQLAlchemy ir MySQL bibliotekomis.
4. Sukurti schemą duomenų bazėje (lentelės sukuriamos automatiškai) ir surašyti prisijungimo duomenis db.py faile.
5. Prie programos padėti Keliai_level2.txt duomenų failą ir jį suimportuoti imp.py programa. Sugeneruoti matricą genmatrix.py programa.
6. Šiuo metu sistemą galima naudoti pavienėms kelio paieškos užklausoms.
7. Eismo tvarkaraštis generuojamas genqueue.py programa.
8. Sudarytas tvarkaraštis gali būti vykdomas lokalioje mašinoje naudojant worker.py programą.
9. Norint lygiagretinti skaičiavimo procesą reikia:
    - 9.1 Nukopijuoti visas programas į kiekvieną kompiuterį
    - 9.2 db.py faile pakeisti prisijungimo duomenis į centrinę duomenų bazę
    - 9.3 Dedikuotame kompiuteryje paleisti matrixupdater.py programą (žingsnis reikalingas tik jei modeliavimui bus naudojamas visas grafas)
    - 9.4 Kiekviename skaičiavimo vienete paleisti tiek worker.py procesų, kiek norima išnaudoti to kompiuterio procesoriaus branduolių.
10. Galutiniai skaičiavimo rezultatai saugojamai duomenų bazėje.
11. Papildomos programos gali tikrinti duomenų bazėje esančius duomenis modeliavimo eigai stebėti.