KAUNO TECHNOLOGIJOS UNIVERSITETAS

MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

Antanas Šedbaras

ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ IR ALGORITMŲ ANALIZĖ

Magistro darbas

Vadovas
doc. dr. N. Listopadskis

KAUNAS, 2014

KAUNO TECHNOLOGIJOS UNIVERSITETAS

MATEMATIKOS IR GAMTOS MOKSLŲ FAKULTETAS

TAIKOMOSIOS MATEMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. N. Listopadskis
2014 06 05

ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ IR ALGORITMŲ ANALIZĖ

Taikomosios matematikos magistro baigiamasis darbas

Vadovas
.....doc. dr. N. Listopadskis
2014 06 01

Recenzentas
..... prof. dr. R. Butleris
2014 06 04

Atliko
FMMM 2 gr. stud.
.....A. Šedbaras
2014 05 30

KAUNAS, 2014

KVALIFIKACINĖ KOMISIJA

Pirmininkas:	Juozas Augutis, profesorius (VDU)
Sekretorius:	Eimutis Valakevičius, profesorius (KTU)
Nariai:	Jonas Valantinas, profesorius (KTU)
	Vytautas Janilionis, docentas (KTU)
	Kristina Šutienė, docentė (KTU)
	Zenonas Navickas, profesorius (KTU)
	Arūnas Barauskas, dr., direktoriaus pavaduotojas (UAB „Danet Baltic“)

Šedbaras A. Įrenginių paskirstymo uždavinių ir algoritmų analizė: Matematikos magistro darbas / vadovas dr. doc. N. Listopadskis; Taikomosios matematikos katedra, Matematikos ir Gamtos mokslų fakultetas, Kauno technologijos universitetas. – Kaunas, 2009. – 65 p.

SANTRAUKA

Šiame darbe nagrinėjami įrenginių paskirstymo uždaviniai, priklausantys kombinatorinio optimizavimo klasei, ir jų sprendimo algoritmai. Tai į kainą orientuoti uždaviniai: siekiama įrengti vieną ar daugiau naujų įrenginių, kurie patenkintų klientų poreikius su mažiausia kaina. Šie uždaviniai aktualūs pradedant ar plečiant verslą, kai ieškoma, kur geriausia būtų įrengti naujus įrenginius.

Darbe yra pateikiamos kombinatorinio optimizavimo, įrenginių paskirstymo ir algoritmų klasifikacijos. Tyrimui pasirinktas sandėlių paskirstymo uždavinys, kurio sprendimas atliekamas keturiais algoritmais: tiksliuoju brutalios jėgos; euristiniais: godžiuoju ir apkeitimo; ir-–euristiniu modeliuojamo atkaitinimo. Atsižvelgiant į algoritmų daromas paklaidas ir jų vykdymo laiką, pastebėta, kad brutalios jėgos algoritmas gerai veikia su mažos apimties uždaviniais, o su didesnės apimties uždaviniais – godusis algoritmas pateikia geresnius rezultatus ir per trumpesnį laiką, nei modeliuojamo atkaitinimo algoritmas.

Šedbaras A. Analysis of facility loction problems and algorithms: Master‘s work in mathematics / supervisor dr. prof. N. Listopadskis; Department of Applied mathematics,Faculty of Mathematics and Natural Sciences, Kaunas University of Technology. – Kaunas, 2014. – 65 p.

SUMMARY

The combinatorial optimization problem considered in this paper is facility location problem. Cost-oriented location of one or more than one new facilities out of a set of given locations to meet customer demands. Almost all real-world constraints, demands and guidelines can be considered in these models.

In this paper is overview of combinatorial optimization, facility location problems classification and methods used to solve them. There is also presented and realized one exact algorithm – Brute-force, and two heuristics: Greedy and Interchange, and meta-heuristics Simulated Annealing (SA). Analysis of these four algorithms is made. The results showed that Brutal - force is effective solving simple problems. While solving more complicated problems Greedy and Interchange is more effective than Simulated Annealing.

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

ĮVADAS

Kasdieniai, net ir patys menkiausi, žmonių pasirinkimai daugiau ar mažiau įtakoja tolimesnį jų gyvenimą: kurioje parduotuvėje apsipirkti ar, kaip tinkamai išdėstyti gaisrinius čiaupus patalpose. Renkantis parduotuvę sprendžiama kokiame rajone ji yra, ar lengvą ją pasiekti, kokiu transportu tai daryti, kokios išlaidos. Gaisrinio čiaupo parinkimui svarbu, kad jie būtų arčiausiai galimų gaisro židinių, kad jų būtų nei permažai, nei per daug. Sprendžiant šias, iš pirmo žvilgsnio nesusijusias, problemas, bet kiekvienu atveju tinkamai įvertinus visas aplinkybės galima suformuluoti įrenginių paskirstymo uždavinį.

Šie uždaviniai, kaip ir tvarkaraščių, keliaujančio pirklio uždaviniai, priklauso kombinatorinio optimizavimo klasei.

Darbo pradžioje pateikiama paskirstymo uždavinių komponentų aprašymas bei jų modifikacijos. Toliau, apžvelgus literatūrą ir ankstesnes šių uždavinių klasifikacijas, pateikiama diskrečių įrenginių paskirstymo uždavinių klasifikacija bei pasirinktas tiriamasis sandėlių paskirstymo uždavinys.

Taip pat, apžvelgiami šių uždavinių sprendimo algoritmai, kurie skirstomi į tris klases: tiksliuosius, euristinius ir meta-euristinius. Sandėlių paskirstymo uždavinio sprendimui realizuoti pasirinkti keturi algoritmai: tikslusis brutalios jėgos algoritmas, euristiniai godusis ir apkeitimo, pastarasis naudojamas, kaip godžiojo algoritmo pagerinimas, ir meta-euristinis, atsitiktinės paieškos modeliuojamo atkaitinimo algoritmas.

Tiriamojoje dalyje išsiaiškinta, kad norint pagerinti modeliuojamo atkaitinimo algoritmą tikslinga didinti iteracijų skaičių bei lėtinti proceso vėsimo greitį. Taip pat pastebėta, kad didėjantis įrenginių kiekis labiausiai paveikia brutalios jėgos algoritmo vykdymo laiką, kuris auga eksponentiškai, ir modeliuojamo atkaitinimo algoritmą, tačiau pastaruoju atveju išlaikoma tiesiška priklausomybė. Klientų skaičiaus pagausėjimas dari įtaką godžiojo algoritmo vykdymo laikui.

Atlikus lyginamąjį algoritmų tyrimą su paskirstymo uždaviniams pritaikytais duomenų failai iš J. E. Beasley tinklalapio [7], paaiškėjo, kad brutalios jėgos algoritmas visada randa optimalų sprendinį, tačiau, atsižvelgiant į vykdymo laiką, jis tinkamiausias mažos apimties uždaviniams. Su didelės apimties uždaviniais gerai susitvarkė euristinis godusis algoritmas, kai kuriais atvejais, apkeitimo algoritmas dar ir pagerindavo sprendinius, o skaičiavimo laikas nesiekdavo tūkstantosios sekundės dalies. Modeliuojamo atkaitinimo algoritmas davė prasčiausius rezultatus, tačiau vykdymo laikas buvo kur kas trumpesnis nei brutalios jėgos.

Darbo pabaigoje apibendrinami gauti rezultatai ir pateikiamos išvados bei rekomendacijos tolimesniems tyrimams.

1. TEORINĖ DALIS

1.1 KOMBINATORINIS OPTIMIZAVIMAS

Kombinatorinis optimizavimas yra aktuali tema taikomojoje matematikoje ir teoriniame programavime. Tai matematinio optimizavimo šeimai priklausanti uždavinių aibė, kurioje nagrinėjami uždaviniai yra diskretūs, o leistinoji aibė – baigtinė. Dėl pastarosios savybės visada egzistuoja vienintelis – optimalus sprendinys. Bet didžioji kombinatorinio optimizavimo uždavinių dalis, turi daug alternatyvų sprendinių, kuriuos reikia patikrinti norint surasti optimalų sprendinį. Pavyzdžiui, reikia rasti optimaliausią maršrutą tarp 50 miestų, tai galimų maršrutų yra $(50-2)!$ arba

12,413,915,592,536,072,670,862,289,047,373,375,038,521,486,354,677,760,000,000,000.

Todėl, išsami ir įprasta optimalaus sprendinio paieška ne visada yra įmanoma, nes paieškai atlikti gali prireikti daugybės skaičiavimų, o tai gali užtrukti labai ilgai.

Ši optimizavimo uždavinių klasė vadinama sveikaskaičio programavimo uždaviniais[9] [25].

Kombinatorinio optimizavimo uždavinys formaliai gali būti užrašomas kaip sistema (X/P/Y/F/extr), kur

X – leistinoji aibė (joje apibrėžta F ir P);

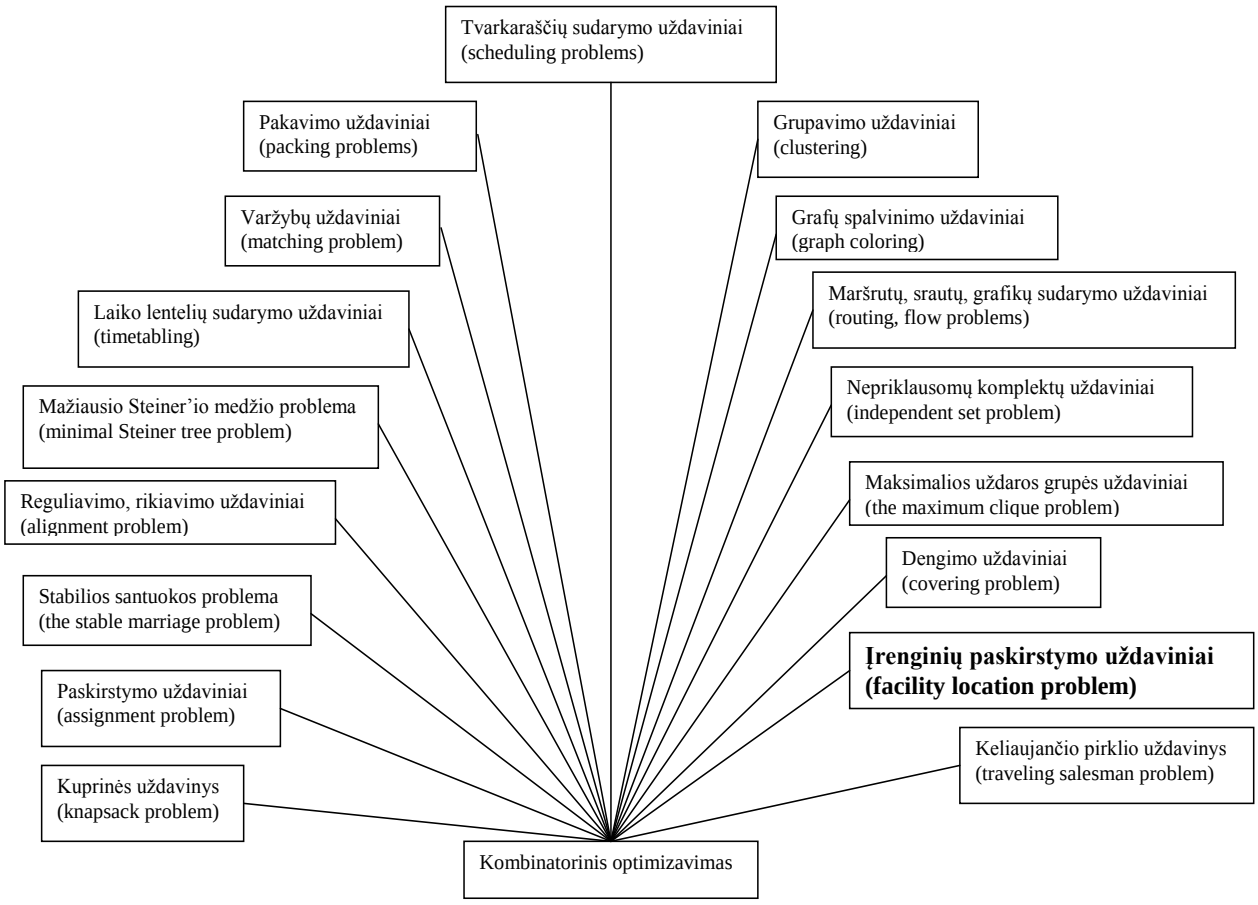
P – tinkamumo požymis;

Y – sprendinių aibė;

F - tikslo funkcija;

extr – ekstremumas (min arba max) [15], [20].

Uždaviniai, pagal savo formuluotę, leistiną sritį, tikslo funkciją bei apribojimus, yra skirstomi į klases, kurios pateiktos 1.1 paveiksle [5].



1.1 pav. Kombinatorinio optimizavimo uždavinių klasifikacija.

Šiame darbe analizuosime įrenginių paskirstymo uždavinių klasę, kurią plačiau aptarsime sekančiame skyrelyje.

1.2 ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ ANALIZĖ

Įrenginių paskirstymo uždaviniai yra aktualūs planuojant organizacijos įkūrimą ar jos plėtrą. Tokiu metu tarp alternatyvių vietų ieškoma, kur geriausia būtų įrengti patalpas su mažiausiomis investicijomis, bet tuo pačiu siekiama, kad jos būtų patrauklios ir lengvai pasiekiamos klientams [19].

Įrenginiais gali būti įvardijamos: sandėliavimo patalpos, kuriose laikomos prekės ir paskirstomos po parduotuves; gamyklos, kurias reikia aprūpinti žaliavomis ir iš jų išvežioti pagamintus gaminius; oro uostų terminalai, kurie aprūpina lėktuvus; gaisrinių punktai, skirti aptarnauti regioną ir t.t.

Netinkamas įrenginių išdėstymas gali turėti neigiamą įtaką organizacijos veiklai, pelnui ir pan. Tarkime, sandėliavimo patalpos paskirstomos neatsižvelgiant į parduotuvių skaičių regione, tai gali būti neveiksmingas jų išnaudojimą:

Kai regione daug sandėlių, bet mažai parduotuvių. Tokiu būdu prekės užsistovės sandėliuose;

Tačiau, jei yra daug parduotuvių, bet mažai sandėlių – galimas blogas parduotuvių aprūpinimas prekėmis ir tuščių sandėlių laikymas.

Dėl to į paskirstymo uždavinius yra įtraukiami įvairūs apribojimai ir reikalavimai. Atsižvelgiant į įvairius uždavinio kriterijus: įrenginių kiekį, jų pobūdį, išlaikymo kainą, darbo laiką, taip pat aptarnaujamus klientus, jų poreikius, atstumą nuo kliento iki įrenginio ir t.t. Paskirstymo uždaviniai gali būti formuojami, kaip vieno kriterijaus uždaviniai, pavyzdžiui, optimalios distancijos arba galima kombinuoti kelis kriterijus, pavyzdžiui, optimalios distancijos ir įrenginių aptarnavimo apribojimus ar įrenginių apribojimus ir optimalios kainos paieška. Taigi, uždavinio esmė surasti geriausią įrenginių paskirstymą su vienu ar keletu įrenginių, kurie gali turėti vieną ar keletą apribojimų [1].

Įrenginių paskirstymo uždaviniai priskiriami NP - sunkių uždavinių klasei, kur padidėjus uždavinio apimčiai labai išsiplečia sprendinių aibė [1][2] [19].

1.2.1 ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ KOMPONENTAI

Nagrinėjant įrenginių paskirstymo uždavinius susiduriama su savita terminologija, kuri charakterizuoja uždavinį. Įrenginiai, paklausa/klientai ir vieta/erdvė yra pagrindiniai komponentai aprašant įrenginių paskirstymo uždavinius. Tačiau įvairiuose uždaviniuose šie komponentai turi skirtingus vaidmenis ir kuria skirtingus uždavinio modelius. Šie komponentai plačiau pateikiami remiantis S. Nickel (2009)[19] ir S. Arfin (2010)[1].

Įrenginiai.

Įrenginiai apibrėžia objektus, kuriuos norima išdėstyti pasirinktoje srityje. Įrenginiai gali aptarnaujamo pobūdžio, kaip restoranai, picerijos, kirpyklos arba gamykliniai – fabrikai, sandėliai ir pan. Įrenginiai charakterizuojami pagal skaičių, tipą, kainą ir t.t.

Paskirstymo uždavinių modeliuose naujų įrenginių skaičius nurodo norimų pastatyti įrenginių skaičių. Skiriamos į dvi rūšis:

- Vieno įrenginio modelis. Vieno įrenginio modelyje norima įgyvendinti tik viena įrenginį iš galimų variantų. Tai gana paprastas uždavinys.
- Daugelio įrenginių modelyje pasirinktoje srityje norima įrengti daugiau nei vieną įrenginį. Šis modelis yra plačiausiai nagrinėjamas ir sudėtingesnis nei vieno įrenginio modelis.

Kitas svarbi įrenginių ypatybė tai jų tipas. Tipai apibrėžia skirtingas įrenginių galimybes ir skirtingą jų aptarnavimą. Įrenginių tipus pagal galimybes galima charakterizuoti, kaip apribotų ir begalinių galimybių objektus:

- jei įrenginys gali aptarnauti begalinį skaičių klientų, jis vadinamas begalinių galimybių įrenginiu;
- bet, jei įrenginys aptarnauja tik tam tikrą kiekį klientų, jis vadinamas apribotų galimybių įrenginiu.

Pagal aptarnavimą įrenginiai skirstomi į:

- vieno aptarnavimo, kur suteikiamos paslaugos yra tik vienos rūšies, kaip kosmetikos parduotuvė;
- daugelio aptarnavimo, kur suteikiamų paslaugų kiekis daugiau nei viena paslauga, kaip prekybos centrai.

Kita ypatybė – kaina. Ji apibrėžia įrenginio pastatymo, nuomos, priežiūros ir pan. kainą. Yra dviejų tipų kainos:

- fiksuota, kuri įrenginiams yra pastovi;
- kintanti, kuri priklauso nuo įvairių veiksnių.

Paklausa arba klientai.

Kitas uždavinio modelyje svarbus komponentas yra paklausa arba klientai. Tai objektai ar žmonės, kuriuos įrenginys turi aptarnauti ir/ar jiems teikti produkciją. Paklausą galima charakterizuoti pagal priklausomybę, kiekį, elgseną ir pasiekiamumą.

Paklausos priklausomybė apibrėžiama kaip:

- Vietos. Tarkime, parduotuvių mieste reikės daugiau nei kaime;
- Laiko. Klientus reikės aptarnauti kasdieną ar tai periodinis aptarnavimas, pagal nustatytą tvarkaraštį.

Paklausos kiekis nurodo, keliais produktais/paslaugomis yra susidomėję klientai:

- Vienu. Pavyzdžiui, masažo paslaugos.

- Keletu. SPA centras, kur atliekamos įvairios procedūros.

Pagal elgseną paklausa skirstoma į:

- Deterministinę. Klientų poreikiai aiškiai apibrėžti ir pastovūs.
- Tikimybinę. Galimi atsitiktiniai klientai, klientų poreikiai nėra pastovūs ir tiksliai apibrėžti.

Kliento ar įrenginio pasiekiamumas, taip pat įtakoja paklausą. Didesnis atstumas reikalauja daugiau išlaidų, kas gali sumažinti paklausą. Pasiekiamumą galima nurodyti, kaip:

- Tipinį atstumo matą: Euklido atstumą, kelio ilgį žemėlapyje, Manhateno atstumą ir t.t.
- Transportavimo kainą: nurodant, kiek kainuoja susisiekimas tarp kliento ir įrenginio.

Vieta arba erdvė.

Trečias paskirstymo modelio komponentas yra vieta arba erdvė. Šis komponentas apibrėžia, kur ir kaip bus paskirstomi įrenginiai erdvėje. Yra trys baziniai vietos tipai:

- Tolydus. Tai visi galimi taškai, t.y. naujas įrenginys gali būti pastatytas bet kurioje vietoje pasirinktame žemėlapyje.
- Tinklinis. Šiai bazei priklauso geležinkeli, kelių tinklai. Naujas įrenginys gali būti įrengtas tik prie pasirinkto tinklo.
- Diskretus. Naują įrenginio vietą galima pasirinkti tik iš apibrėžtų potencialių vietų, kurios atitinka duotus reikalavimus.

1.2.2 ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ KLASIFIKACIJA

Apjungus skirtingus paskirstymo modelius gaunamas daug realistiškesnis uždavinys, bet kartu ir sudėtingesnis. Pavyzdžiui, realiame pasaulyje parduotuvės yra skirtingų dydžių, tipų, gali aptarnauti įvairius klientų poreikius ir pan. Toks paprastų modelių jungimas sukuria labai įvairiapusišką uždavinių aibę. Sujungti uždaviniai turi tam tikras savybes ar požymius, pagal kuriuos yra įvairių bandymų suklasifikuoti įrenginių paskirstymo uždavinius.

Brendeau ir Chiu (1989) atliko įrenginių paskirstymo uždavinių klasifikaciją. Jie apžvelgė svarbesnius uždavinius ir bandė juos atskirti pagal skirtingus tipus arba susieti vieną su kitu. Taigi, jie suskirstė uždavinius į tris pagrindines klases pagal tikslą, sprendimo kintamuosius ir sistemos parametrus [3].

Klasifikavimas pagal tikslą buvo skiriant pagal tikslo funkcijos optimizavimas Sprendimo kintamieji buvo klasifikuojami pagal įrenginius, vietos aptarnavimą, įrenginių ar paslaugų skaičių ir t.t. Sistemos parametrai nurodydavo topologinę struktūrą, metriką, transportavimo laiką/kainą ir pan.

Church (1999) pateikė keturias pagrindines paskirstymo modelių klases pagal medianą, padengiamumą, apribojamumą ir konkurencingumą. Apribojamumas buvo skirstomas atsižvelgiant į kiekvieno įrenginio apribojimą ir reikalavimus. O konkurencingumo modeliai nurodė, kaip žmogus priimantis sprendimą, gali apsvarstyti visus potencialius įrenginius ir taip pakeisti savus įrenginius [12].

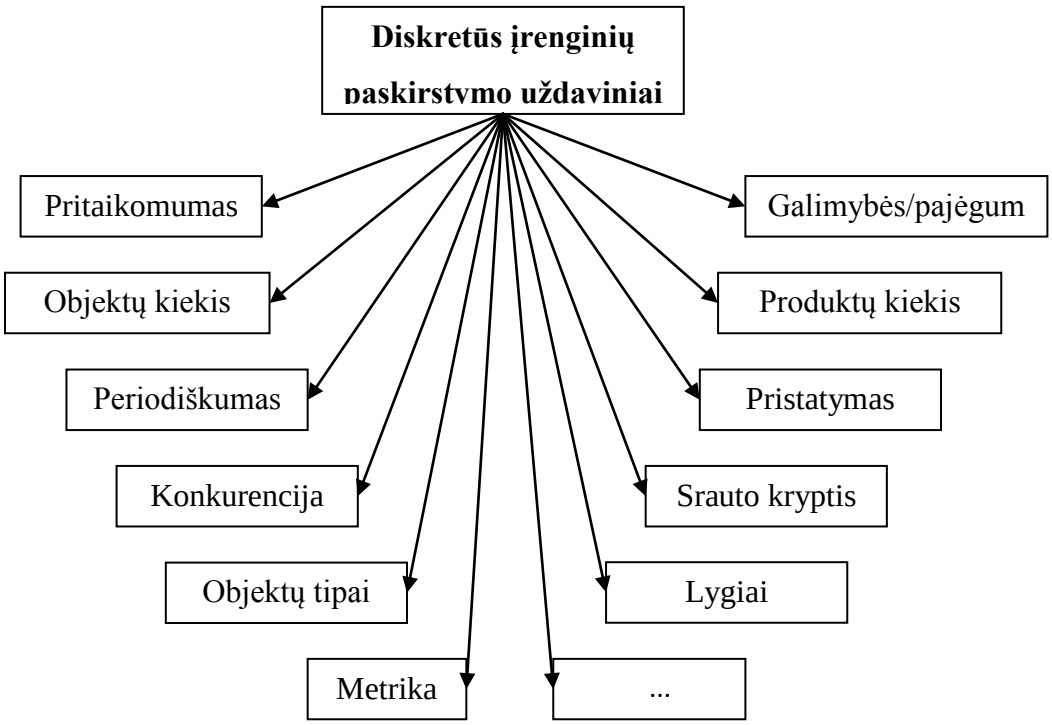
Daskin (1995) naudojo specializuotą klasifikacijos bazę pagal: vartojimą (viešas ar privatus sektorius), įrenginių kiekį (vienas ar daug įrenginių), įrenginių ar klientų vietą, elgseną (statinis ar dinaminis), veiklos laiką (vieno ar kelių periodų), sprendimo tipą (tikslus, euristinis ar metaeuristinis) ir jau egzistuojančių įrenginių veikimą.

Murray (2010) prie Daskin klasifikacijos pridėjo distancijos matą (Euklido, Manhateno, tinklų), įrenginių ar klientų vietos vaizdavimo tipą (taškas, linija, objektas), klientų padalijimą įrenginiams (vienodą, nereguliarų), aptarnavimo galimybes (vieno ar kelių produktų), įrenginių hierarchiją (vieno lygios ar kelių lygių) [18].

Šiandien šios klasifikacijos tampa dar sudėtingesnės.

Diskrečius įrenginių paskirstymo uždavinius galima suskirstyti pagal uždavinio modeliavimui parinktus parametrus (1.2 pav.)[19]:

- Pritaikomumas. Uždavinys skirtas privačiam arba viešam sektoriui;
- Objektų kiekis. Naudojamas vienas ar keli objektai;
- Periodiškumas. Statinis (vienas periodas) ar dinaminis(keli periodai) modelis;
- Konkurencija. Nauji objektai išdėstomi tarp jau egzistuojančių tokios pačios rūšies objektų ar ne;
- Objektų tipai. Regioniniai, centriniai ar pan.;
- Metrika. Naudojama metrika uždaviniui spręsti: Euklido, Manheteno ir t.t.;
- Lygiai. Vieno lygio ryšys, ar kelių lygiu(hierarchinis, nehierarchinis)
- Srauto tipai. Surankamasis ar dalijamasis;
- Pristatymas. Tik tarp lygių ar galia ir tame pačiame lygyje dalintis paslaugomis.
- Produktų kiekis. Vieno produkto ar kelėtos produktų sistema;
- Galimybės/pajėgumai. Neribotos apimties įrenginiai gali aptarnauti neribotą kiekį klientų, o ribotos apimtys - įrenginys gali aptarnauti tik nurodytą skaičių klientų.



1.2 pav. Diskrečių įrenginių paskirstymo uždavinių klasifikacija.

1.2.3 TIRIAMO UŽDAVINIO APRAŠYMAS

Šiame darbe tiriamas uždavinys literatūroje vadinamas sandėlių paskirstymo uždaviniu arba neapribotų įrenginių paskirstymo uždaviniu.

Sandėlių paskirstymo uždavinyje priimamos prielaidos, kad jis:

- Statinis;
- Vieno lygio;
- Neapribotas;
- Deterministinis;
- Vieno produkto;
- Tiesioginio pristatymo;
- Vieno tipo.

Uždavinyje duodama:

- Potencialių įrenginių aibė $I=\{1,..,n\}$;

- Klientų aibė $J=\{1,...,m\}$ su prekės poreikiu b_j, $j=1,...,m$;
- Transportavimo kaina t_{ij}, nurodanti, kiek kainuoja pasiekti i įrenginį nuo j kliento. Į šią kainą taip pat gali įeiti produkto kaina, jo saugojimas. Bendra j kliento aptarnavimo kaina i įrenginio yra: $c_{ij}=b_j*t_{ij}$. Taip gaunama transportavimo kainų matrica $C=(c_{ij})$, $i=1,...m$, $j = 1...n$;
- Fiksuota kaina f_i naujam įrenginiui i. Į šią kainą įeina pastatymo, priežiūros, operacijų fiksuotos kainos.

Sprendimo sudėtis:

- parenkamas įrenginių rinkinys $X=\{x_1,..x_p\}$. X yra I poaibis $X \subseteq I$, elementas $x_k \in X$ nurodo pasirinktą įrenginį.
- Parinktiems naujiems įrenginius reikia priskirti klientus. Klientas j priskiriamas tam įrenginiui i, kurio transportavimo kaina mažiausia $c_{ij} = \min\{ c_{ik} \mid x \in X \}$.

Matematinis modelis:

$$\min F(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{i=1}^{n} f_i y_i \qquad (1.1)$$

$$x_{ij} = \begin{cases} 1, \ jei \ \ i-as \ \ objektas \ \ \ aptarnauja \ \ \ j-ąjį \ klient ą \\ 0, kitu \ \ atveju \end{cases}$$

$$y_i = \begin{cases} 1, \ jei \ \ i \in X \\ 0, kitu \ \ atveju \end{cases} \qquad (1.2)$$

$$\sum_{j=1}^{m} x_{ij} = 1 \qquad \forall i \qquad (1.3)$$

$$x_{ij} \le y_j \qquad \forall i \ , \ \forall j \qquad (1.4)$$

Uždavinio tikslas minimizuoti (1.1) funkciją. Čia c_{ij} transportavimo kaina nuo i įrenginio iki j kliento, f_i i-tojo įrenginio kaina, x_{ij} nurodo ar j klientas priskirtas i įrenginiui, o y_i nurodo ar pasirinktas i įrenginys (1.2).

(1.3) nurodoma, kad kiekvienas klientas j priskiriamas tik vienam objektui i.

(1.4) klientas j gali būti priskirtas i objektui tik tada, kai jis priklauso parinkam rinkiniui X.

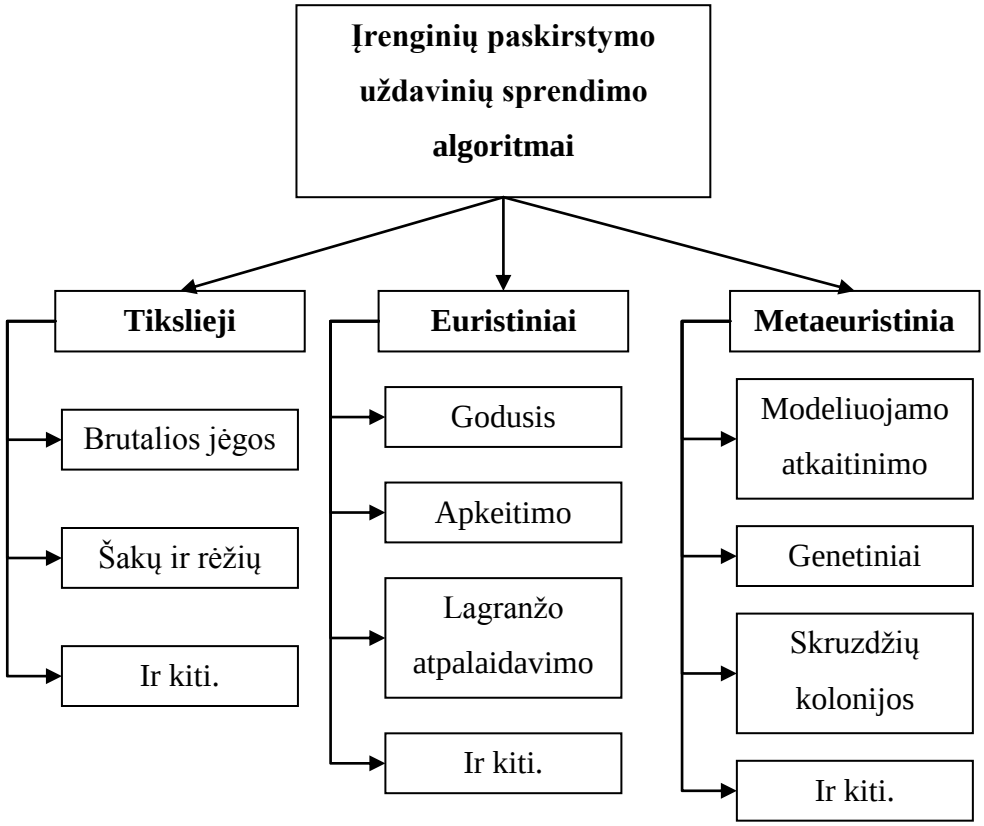
1.3 ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ ALGORITMŲ KLASIFIKACIJA

Kombinatorinio optimizavimo uždavinius išspręsti tiksliai galima tik eksponentinio sudėtingumo algoritmais. Tokiais, kaip brutalios jėgos (brute-force search) ar šakų ir rėžių algoritmais, kurie surandą optimalų sprendinį, tačiau jų vykdymo laikas eksponentiškai auga didėjant uždavinio apimčiai. Didelės apimties uždaviniams spręsti yra taikomi euristiniai ir meta–euristiniai algoritmai.

Euristiniai algoritmai siekia surasti aukštos kokybės, bet nebūtinai optimalų sprendinį. Algoritmo taisyklės nurodo sprendimą, kurį reikia priimti konkrečiose situacijose, tačiau euristiniai algoritmai yra pritaikomi tik tam tikriems konkretiems uždaviniams.[13] Euristiniai algoritmai skirti įrenginių paskirstymo uždaviniams spręsti yra: godusis algoritmas(greedy algorithm)[8], apkeitimo algoritmas (interchange algorithm), Lagranžo relaksacijos algoritmas (Lagrangian Relaxation)[27][11], kaimyninės paieškos (neighborhood search)[6][17], paieškos su draudimais(tabu search)[6][17].

Meta–euristiniai algoritmai apibrėžiami kaip tam tikro aukšto lygio abstrakčių nurodymų rinkiniai. Priešingai nei euristinių algoritmų, šie nurodymai formaliai aprašo kurios nors klasės uždavinių sprendimo idėją, principus. Šios klasės algoritmai gali išvengti lokalaus optimumo. Kiekvieną meta–euristini algoritmą sudaro kelios euristinių procedūrų, kurias galima realizuoti įvairiais budais, sekos. [13][1] Mera–euristiniai algoritmai skirti įrenginių paskirstymo uždaviniams spręsti yra: genetinis algoritmas (genetic algorithm)[1][16][21], modeliuojamas atkaitinimas (simulated annealing)[1][10], skruzdžių kolonijos optimizavimas (ant colony optimizacion)[22], kintamos aplinkos paieška (variable neigbourhood search).

1.3 paveiksle schematiškai pateikiama įrenginių paskirstymo uždavinių sprendimo algoritmų klasifikacija.



1.3 pav. Įrenginių paskirstymo uždavinių sprendimo algoritmai.

Sprendžiant įrenginių paskirstymo uždavinius, algoritmai ieško optimalaus įrenginių rinkinio $X^*=\{ x_1,...,x_k \}$, kuris duoda optimalų tikslo funkcijos sprendinį $S^*=F(X^*)$. Kadangi įrenginys gali būti panaudotas arba ne, t.y. dvi reikšmės 1 arba 0, ir įrenginių skaičius lygus n tai aibės galia $|X|=2^n$. Kaip minėjome ankščiau, tikslaus sprendinio paieška užtrunka labai ilgai: jei n yra didelis, pavyzdžiui, turime 25 įrenginių, tai galimų rinkinių bus 2^{25} = 33554432. Norint patikrinti visus rinkinius galima užtrukti daugiau nei valandą (programa Matlab, Intel(R) Core(TM) i3-3110M CPU @ 2.40 GHz, 8.00 GB RAM, 64-bit). O jei įrenginių skaičių padidinsime 1, rinkinių skaičius padvigubės ir užtruksime dvigubai ilgiau ir t.t.

Čia pasitelkiami euristiniai ir meta–euristiniai metodai, deja, jie ne visada randa optimalų sprendinį. Tačiau jie gali rasti rinkinį įrenginių, kurio sprendinys bus artimas optimaliam. Tokį rinkinį laikysime geriausiu ir žymėsime X^o, o sprendinį $S=F(X^o)$.

Algoritmų veikimo tikslumui įvertinimui skaičiuosime santykinę paklaidą (1.5) :

$$Pakaida = \frac{S - S^*}{S^*} \cdot 100 = \% \qquad (1.5)$$

Aptarti įrenginių paskirstymo uždavinių sprendimo metodų klasifikaciją bei jų specifiką. Toliau plačiau pateiksime darbe naudojamus algoritmus, pateiksime jų veikimo principus, skaičiavimo formules, programų pseudokodus ir kitą su jais susijusią informaciją.

1.3.1 BRUTALIOS JĖGOS ALGORITMAS

Darbe nagrinėjamuose uždaviniuose sprendinių skaičius yra baigtinis ir teoriškai juos galima išspręsti patikrinant visus galimus sprendinius. Tam pasirinktas tiksliųjų algoritmų klasei priklausantis brutalios jėgos algoritmą.

Šio algoritmo realizacija yra gana paprasta ir visada randa tikslų atsakymą, tačiau didėjant uždavinio apimčiai *n* sparčiai auga jo vykdymo laikas. Šio algoritmo idėja yra sugeneruoti visus galimus sprendimo atvejus ir nuosekliai juos tikrindamas ieškoti uždavinio optimalaus sprendinio *X** [23].

Struktūrizuotai jį galima pavaizduoti taip:

1 žingsnis..

1.1 žingsnis. Sugeneruojama visų galimų įrenginių rinkinių aibė $X = \{X_1,...,X_2^n\}$ *duotam uždaviniui;*

1.2 žingsnis. Apskaičiuojama tikslo funkcijos reikšmė su pirmu rinkiniu $S = F(X_1)$*;*

1.3 žingsnis. $i = 2$.

2 žingsnis.

2.1 žingsnis. Paimamas sekantis rinkinys X_i*;*

2.2 žingsnis. Apskaičiuojama tikslo funkcijos reikšmė su duotu rinkiniu $S' = F(X_i)$*;*

2.3 žingsnis. Jei $S' < S$*, tai priimame, kad radome geresnį sprendinį* $S = S'$.

2.4 žingsnis. $i = i + 1$*;*

2.5 žingsnis. Jei $i \leq 2^n$ *– grįžti į 2.1 žingsnį, kitaip 3 žingsnis*.

3 žingsnis.

3.1 žingsnis. Surastas optimalus sprendinys $S^* = S$.

Šio algoritmo programos kodas pateiktas C priede – C1 Brutalios jėgos algoritmo vykdymo kodas.

1.3.2 GODUSIS ALGORITMAS

Godusis – tai euristinis algoritmas, kuris kiekviename žingsnyje ieško lokalaus sprendinio, taip tikintis surasti globalųjį uždavinio sprendinį. Ši strategija ne visada pasiteisina ir daugelyje uždavinių godus algoritmas apskritai nesuranda optimalaus sprendinio.

Šiame darbe sprendžiami paskirstymo uždaviniai priskiriami NP- sunkių uždavinių klasei, kur didėjant uždavinio apimčiai eksponentiškai auga jo sudėtingumas ir optimalų sprendinį surasti darosi neįmanoma per gana greitą skaičiavimų laiką. Tokiu atveju ieškoma artimo optimaliam sprendinio, o godusis algoritmas yra tinkamas tam. [14] [19].

Algoritmas pasirenka geriausią sprendimą duotu metu. Pasirinkimas, padarytas godžiojo algoritmo, gali priklausyti nuo pasirinkimų, padarytų ligi šio momento, bet ne nuo būsimų pasirinkimų ar visų uždavinio pasirinkimų. Taip kartojamas vienas godus pasirinkimą po kito, mažinant kiekvieną duotą problemą į mažesnę. Kitaip tariant, godus algoritmas niekada neperžiūri savo pasirinkimų. [24] [4] [17] [24].

Godus algoritmas pradedamas su tuščia sprendinių aibe X ir ją pildo žingsnis po žingsnio $X=\{x_1,...,x_p\}$, čia x_i - į sprendinį įtrauktas įrenginys. Taip pridedant įrenginį siekiama sumažinti tikslo funkcijos reikšme kiek įmanoma labiau. Algoritmas baigiamas, kai joks naujas įrenginys nebepagerina sprendinio.

Tegul:

- J yra visų galimų įrenginių aibė;
- $X^z=\{ x_1,...,x_z \}$ apibrėžia z momentu surastą geriausią įrenginių rinkinį $x_1,...,x_z$;
- $I'=I\backslash X^z$ aibė įrenginių, kurie dar nėra įtraukti į geriausią rinkinį;
- $u_i^z = c_{i^* j} = \min\{ c_{ij} \mid i \in X^z \}$ tai mažiausia kaina, kuria aptarnaujamas j klientas i įrenginio.

Klientas j visada priskiriamas įrenginiui i^*, kurį pasiekti pigiausia.

Su rinkiniu X^z tikslo funkcija skaičiuojama :

$$F\left(X^z\right)=\sum_{j=1}^{m} c_{i^* j} + \sum_{i \in X^{zl}} f_i = \sum_{j=1}^{m} u_j^z + \sum_{i \in X^z} f_i \qquad \textbf{(1.6)}$$

Iteracijoje z godusis algoritmas nori pridėti naują įrenginį prie jau esamo X^{z-1} rinkinio. Taigi pridėjus naują įrenginį $i \in I'$ į rinkinį X^{z-1} gaunamas kainos pokytis:

$$\omega_i^z = F\left(X^{z-1}\right) - F\left(X^{z-1} \cup \{i\}\right) = \sum_{j=1}^{m} u_j^{z-1} - \sum_{j=1}^{m} \min\left\{u_j^{z-1}, c_{ij}\right\} - f_i = \\ = \sum_{i=1}^{m} \max\left\{0, u_j^{z-1} - c_{ij}\right\} - f_i = \Delta_i^z - f_i \quad (1.7)$$

Įvertinus visus galimus omega pokyčius, į rinkinį įtraukimas naujas įrenginys $i \in I'$, kurio kainos pokytis yra didžiausias, t.y. $\omega_i^z = \max_{k \in I'} \omega_k^z > 0$.

Omega taisyklė:

Kainos pokytis ω_i^z yra monotoniškai mažėjanti funkcija, t.y. $\omega_i^z \geq \omega_i^{z+1}$. Todėl, jei $k \in I' : \omega_k^z \leq 0$, tai *k* įrenginį pašaliname iš kandidatų sąrašo *I'*, nes kitoje iteracijoje skirtumas išliks neigiamas.[19]

Struktūrizuotai godųjį algoritmą galima pavaizduoti taip:

1 žingsnis. Iniciacija.

1.1 žingsnis. Sukuriame tuščią aibę X^0;

1.2 žingsnis. Apibrėžiame potencialių įrenginių aibę I' = I. I - visi uždavinio įrenginiai;

1.3 žingsnis. Apskaičiuojame transportavimo vektorių $u_j^0 = \max_{i=1,..,n} c_{ij}$, kur u_j yra maksimali kaina aptarnauti j klientą potencialiame įrenginyje i;

1.4 žingsnis. Iteracijų skaičius z = 1

2 žingsnis. Sprendinio paieška.

2.1 žingsnis. Tikriname ar potencialių įrenginių aibė I' netuščia, jei tuščia – stop, kitu atveju 2.2 žingsnis;

2.2 žingsnis. Jei su visais $i \in I'$ kainos pokytis yra neigiamas $\omega_i^z \leq 0$, stabdome skaičiavimą;

2.3 žingsnis. Surandame potencialų įrenginį $i \in I'$ tenkinantį $\omega_i^z = \max_{k \in I'} \omega_k^z > 0$;

2.4 žingsnis. Papildome X^z aibę naujuoju įrenginiu $X^z = X^{z-1} \cup \{i\}$;

2.5 žingsnis. Iš potencialių įrenginių sąrašo pašaliname rastą i įrenginį $I' = I' \setminus \{i\}$;

2.6 žingsnis. Jei $\omega_i^z \leq 0$, pašaliname i įrenginį, $I' = I' \setminus \{i\}$;

2.7 žingsnis. z = z + 1, grįžti į 2.1 žingsnį.

3 žingsnis.

Įrenginių rinkinys X^z yra geriausias rastas rinkinys $X^z = X^o$, o tikslo funkciją $S = F(X^o)$ yra euristinis sprendinys.

Šio algoritmo programos kodas pateiktas C priede – C2 Godžiojo algoritmo vykdymo kodas.

1.3.3 APKEITIMO ALGORITMAS

Apkeitimo algoritmas taip pat euristinis. Šio algoritmo tikslas pagerinti turimą sprendinį, taip sumažinant tikslo funkcijos reikšmę.

Algoritmo pradžioje duodamas rinkinys įrenginių, kurie laikomi optimalaus sprendinio rinkiniu. Tada pradinio rinkinio įrenginiai yra paeiliui apkeičiami su rinkiniui nepriklausančiais įrenginiais. Algoritmas baigiamas, kai joks įrenginių apkeitimas neduoda geresnio rezultato [17].

Apkeitimo algoritmas šiame darbe bus naudojamas, pagerinti godžiojo algoritmo gautą sprendinį. Šiam tikslui įgyvendinti pirmiausia uždavinį išspręsime godžiuoju algoritmu, o gautą įrenginių rinkinį panaudosime apkeitimo algoritme, kaip pradinį sprendinį.

Tarkime duotas pradinis rinkinys $X=\{x_1,...,x_p\}$. Apkeitimo algoritmu žingsnis po žingsnio apkeisime visus pradinio rinkinio įrenginius su potencialiais įrenginiais, taip tikėdamiesi pagerinti turimą sprendinį.

Jei x_k yra įrenginys, kuris priklauso rinkiniui X ir i nepriklauso X, t.y. $i \in I' = i \setminus X$, sudarome naują rinkinį $X' = X \setminus \{x_k\} \cup \{i\}$.

Jei su nauju įrenginių rinkiniu X' gaunamas geresnis sprendinys $X:F(X')<F(X)$, jį priimame (apkeičiame), t.y. $X=X'$.

Algoritmas sustabdomas, kai joks naujas apkeitimas nebepagerina sprendinio.

Apkeitimo algoritmu sprendinio pagerinimui galima naudoti dvi strategijas:

- Pirmas pagerinimas. Pradedama nuo pirmo įrenginio, jį vis pakeičiant nauju, potencialiu įrenginiu, kuris nepriklauso pradiniam rinkiniui. Tą patį kartojame su antru ir kitais įrenginiai. Baigiama, kai apkeitimas duoda geresnį sprendinį.
- Geriausias pagerinimas. Apskaičiuoti visus galimus apkeitimus ir realizuoti maksimalų tikslo funkcijos pakeitimą.

Tegul:

- $u_j = c_{ij*} = \min\{ c_{ij} \mid i \in X \}$ yra mažiausia kaina, kuri sumokama aptarnauti j klientą i įrenginyje iš X paskirstytų įrenginių.
- $u_j^k = \min \left\{ c_{ij} \mid i \in X, i \neq x_k \right\}$ mažiausia kaina aptarnauti j klientą i įrenginyje iš $X\setminus\{x_k\}$ paskirstytų įrenginių.

Tada kainos pokytis ω duotas (1.7) lygtyje su nauju įrenginiu $k \in X$ ir $i \in I'$ apskaičiuojamas taip[19]:

$$\omega_i^k = F(X) - F(X \setminus \{x_k\} \cup \{i\}) = \sum_{j=1}^{m} u_j - \sum_{j=1}^{m} \min\{u_j^k, c_{ij}\} + f_k - f_i =$$

$$= \sum_{j=1}^{m} (u_j - \min\{u_j^k, c_{ij}\}) + f_k - f_i = \Delta_i^k + f_k - f_i \quad (1.8)$$

Struktūrizuotai apkeitimo algoritmą galima pavaizduoti taip:

1 žingsnis. Iniciacija.

1.1 žingsnis. Įvedamas pradinis įrenginių rinkinys X={x1,...,xp} su p įrenginių;

1.2 žingsnis. Surandama potencialių įrenginių aibė $I' = I \setminus X$.

2 žingsnis.

Apskaičiuojamas vektorius $u_j = c_{ij*} = \min\{ c_{ij} \mid i \in X \}$ *visiems klientams j.*

3 žingsnis.

3.1 žingsnis. k = 1;

3.2 žingsnis. Apskaičiuojama $u_j^k = \min\{c_{ij} \mid i \in X, i \neq x_k\}$ *visiems klientams j;*

3.3 žingsnis. $i = 1, i \in I'$;

3.4 žingsnis. Jei $\omega_i^k > 0$ *vykdome apkeitimą* $X = X \setminus \{x_k\} \cup \{i\}$, *priešingu atveju einame į 3.7 žingsnis;*

3.5 žingsnis. Atnaujiname potencialių įrenginių sąrašą $I' = I \setminus X$;

3.6 žingsnis. Grįžtame į 2 žingsnį.

3.7 žingsnis. $i = i + 1$, *jei* $i \in I'$ *grįžtame į 3.3.1 žingsnį, priešingu atveju 3.8 žingsnis;*

3.8 žingsnis. $k = k + 1$, *jei* $k \leq p$ *grįžtame į 3.1 žingsnį, kitaip STOP.*

4 žingsnis.

Įrenginių rinkinys X^z *yra geriausias rastas rinkinys* $X^z = X^o$, *o tikslo funkciją* $S = F(X^o)$ *yra euristinis sprendinys.*

Šio algoritmo programos kodas pateiktas C priede – C3 Apkeitimo algoritmo vykdymo kodas.

1.3.4 MODELIUOJAMO ATKAITINIMO ALGORITMAS

Modeliuojamo atkaitimo algoritmas yra atsitiktinės paieškos meta–euristinis algoritmas skirtas spręsto įvairius kombinatorinio optimizavimo uždavinius.[1][2][26] Šis algoritmas pritaikomas spęsti ir šio darbo įrenginių paskirstymo uždavinius.

Algoritmo veikimo principas grindžiamas metalurgijoje naudojama metalų kristalų formavimu pamažu vėsinant įkaitusį metalą, taip metalas grūdinamas. Vėsinimo procedūros metu, kai temperatūra pamažu mažėja, sistemos energija natūraliai minimizuojasi. Atsitiktinis fizinės sistemos judėjimas, tikimybė specifinei sistemos konfigūracijai, priklauso nuo energijos ir temperatūros. Šia tikimybę apibrėžia Gibso lygtis[1]:

$$\rho = e^{\frac{E}{k^T}} \tag{1.9}$$

Čia ρ - tikimybė, k - Boltzmano konstanta, E - sistemos energija, T - temperatūra.

Pasinaudojęs (1.9) lygybe Kirkpatrick (1983) parodė, kaip atkaitinimo procesą galima panaudoti sprendžiant modeliuojamu atkaitinimu optimizavimo uždavinius[10].

Modeliuojamo atkaitimo algoritmas pradedamas su atsitiktiniu įrenginių rinkiniu *X*, kuris laikomas kaips geriausias X^o ir tikslo funkcijos reikšmė lygi $S = F(X^o)$. Kiekvienoje algoritmo iteracijos siekiama pagerinti turimą rinkinį *X*, tam atsitiktinai parenkant kaimyninį rinkinį *X'*. Jei su nauju rinkiniu *X'* gaunamas mažesnis sprendinys $S' = F(X')$, nei senasis $S' < S$, priimame jį $S = S'$ ir taip pat jis tampa geriausiu $X^o = X'$. Tačiau, jei naujas sprendinys *S'* nėra geresnis, mes naują rinkinį priimame tiks su tikimybe (1.10).

$$\rho = e^{\frac{\Delta s}{T}} \tag{1.10}$$

Ši lygybė yra panaši į Gibsono lygybe, kur $\Delta s = F(X) - F(X')$ yra tikslo funkcijos skirtumas tarp seno ir naujo sprendinio. *T* yra temperatūra, kaip ir Gibso lygtyje. Jei *T* yra aukštą yra didesnė tikimybė priimti blogesnį rinkinį *X'*, tačiau *T* artėjant prie 0 ši tikimybė mažėja. Dėl šios priežasties algoritmas pradedamas vykdyti su aukšta temperatūra, kad priėmus prastesnį įrenginių rinkinį pajudama iš rasto minimumo, kuris gali būti lokalus, ir taip galima pasiekti globalų minimumą.

Struktūriškai sprendinio priėmimą *n+1* iteracijoje galima pavaizduoti taip[2]:

$$S_{n+1} \leftarrow \begin{cases} S'_n, \; jei \;\; 0 < \Delta s; \\ S'_n, \; jei \;\; e^{\frac{\Delta s}{T_n}} > \rho; \\ S_n, kitu \;\; atveju \;\; . \end{cases} \tag{1.11}$$

Temperatūros vėsimo greitis δT nurodo, kaip greitai mažės pradinė temperatūra T_0 *i*-tojo vėsinimo metu $T_i = \delta T * T_{i-1}$. Toks vėsinimas yra proporcingas. δT gali kisti nuo 0 iki 1. Kuo reikšmė artimesnė 1 tuo vėsimas lėtesnis ir algoritmas atliks daugiau veiksmų, taip surasdamas tikslesnį sprendinį. Apžvelgtuose literatūros šaltiniuose δT parenkama reikšmė tarp 0.9 ir 0.99[1][2]. Taip pat

kiekvieno vėsimo metu atliekamas nurodytas vidinių iteracijų skaičius imax, kuris taip pat turi įtaką gaunamo sprendinio tikslumui.

Šiame algoritme labais svarbi procedūra yra kaimyninių įrenginių rinkinių paieška, tinkamai parinkus kaimyninį rinkinį, didesnė tikimybė rasti optimalų sprendinį. Todėl aptarsime šia procedūrą.

Kaimyniniai rinkiniai.

Įrenginių paskirstymo uždavinys galima įvardyti, kaip dvinarį: įrenginys gali veikti arba ne, t.y. 1 arba 0 atitinkamai. Atsižvelgiant į įrenginio būseną sudaromas sprendinio rinkinys *X*. Tarkime *X={1, 5, 8, 9}*, tai reiškia, kad veikiantys įrenginiai yra 1, 5, 8 ir 9 iš duotų *n* įrenginių. Atlikus pakeitimą rinkinyje *X* gaunamas naujas rinkinys *X'*.

Yra trys galimi rinkinių pakeitimo atvejai, kuriuos galima pateikti, kaip operacijos:

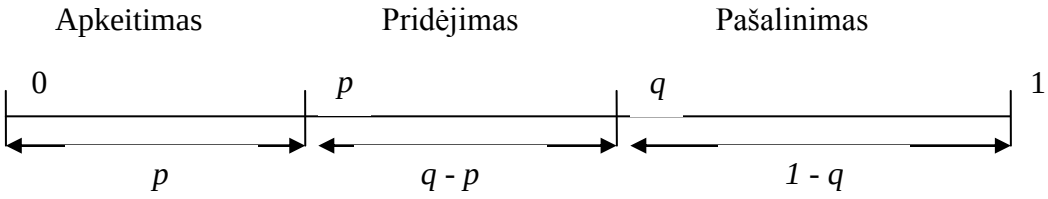
1. Apkeitimas. Vieną rinkinio *X* įrenginį x_i pakeičiant kitų $x_{j.,}$ čia $x_i \in X$, $x_j \notin X$;
2. Pridėjimas. Į turimą rinkinį X įtraukiamas naujas įrenginys $x_i \notin X$;
3. Pašalinimas. Iš rinkinio X pašalinamas įrenginys x_i;

Čia susiduriama su klausymu, o kurią operacija naudoti ir kuriuos elementus apkeisti, pridėti ar pašalinti.

Operacijos parinkimui įvesime du kintamuosius *p* ir *q*, $p, q \in [0,1],\ p < q$. Tai kintamieji, kuriuos keičiant bus galima padidinti ar sumažinti atitinkamos operacijos tikimybę. Tada atsitiktinai sugeneravus skaičių $\rho \in (0,1)$ žiūrima į kurios operacijos rėžį jis pataiko, šį operacijų rinkinį pažymėsime kaip funkcija *Kaimynas(p,q)* priklausančią nuo *p* ir *q* [2] (žr. 1.12).

$$\text{Kaimynas(p , q)} \leftarrow \begin{cases} \textit{Apkeitimas} . & (|X| = 1 \cap 0 < \rho < q) \cup (|X| > 1 \cap 0 < \rho < p) \\ \text{Pr}\ \textit{id ėdėjim} . & (|X| = 1 \cap q < \rho < 1) \cup (|X| > 1 \cap p < \rho < q) \\ \textit{Pašalinima}\ s. & (|X| = n) \cup (|X| < n \cap q < \rho < 1) \end{cases} \qquad (1.12)$$

Čia $\rho \in (0,1)$ atsitiktinai sugeneruotas skaičius, |*X*| - aibės *X* galia. Kaip matome, jei rinkinyje *X* turime tik vieną elementą mes galime atlikti *Apkeitimo* ir *Pridėjimo* operacijas su tikimybė *q* ir *1-q* atitinkamai. Jei rinkinyje *X* yra visi įrenginiai, galima tik *Pašalinimo* operacija. Tačiau, jei elementų yra daugiau nei 1 ir mažiau nei *n*, tai operacijos *Apkeisti, Pridėti* ir *Pašalinti* yra atliekamos su tikimybėmis *p, q-p* ir *1-q* atitinkamai (žr.).



1.4 pav. Funkcijos *Kaimynas(p, q)* operacijų parinkimo tikimybės.

Rinkinių elementų parinksimas atliekamas atsitiktine tvarka iš galimų variantų.

Struktūrizuotai modeliuojamo atkaitimo algoritmą galima pavaizduoti taip:

1 žingsnis. Iniciacija.

1.1 žingsnis. Sugeneruojame pradinį rinkinį X ir laikome jį geriausiu $X = X^o$, $S = F(X^o)$;

1.2 žingsnis. Parenkame pradinę temperatūrą T_0;

1.3 žingsnis. Nurodome absoliutinę temperatūrą $T_a = 0.001$;

1.4 žingsnis. Parenkame vėsimo tempą δT;

1.5 žingsnis. Parenkame maksimalų iteracijų skaičių imax;

1.6 žingsnis. Parenkamos p ir q reikšmės kaimyninių sprendinių paieškai.

2 žingsnis. Sprendinio gerinimas.

2.1 žingsnis. $T = T_a$;

2.2 žingsnis. Iteracijų skaičių nustatome i = 0;

2.3 žingsnis. Atsitiktinai sugeneruojame rinkinį X' iš kaimyninių rinkinių Kaimynas(p,q);

2.4 žingsnis. Apskaičiuojame tikslo funkcijos skirtumą $L = F(X') - F(X)$;

2.5 žingsnis. Jei $L \le 0$ *priimame naują rinkinį, kaip geriausią* $X^o = X'$ *ir* $S = F(X^o)$. *Priešingu atveju atsitiktinai sugeneruojame skaičių* $A \in (0,1)$, *jei* $A < \exp(-L/T)$, *tada* $X^o = X'$.

2.6 žingsnis. $i = i + 1$. *Jei* $i <$ *imax grįžtame į 2.3 žingsnį. Jei* $i =$ *imax einame į 2.7 žingsnį;*

2.7 žingsnis. $T = T * \delta T$. *Jei* $T > T_a$ *grįžtame į 2.2 žingsnį. Priešingu atveju STOP;*

3 žingsnis.

X^o *yra geriausias įrenginių rinkinys, o S yra meta-euristinis uždavinio sprendinys.*

Programos tekstas pateikiamas C priede – C4 Modeliuojamo atkaitinimo algoritmo vykdymo kodas

2. TIRIAMOJI DALIS IR REZULTATAI

Naudojamas kompiuteris su Windows 8.1 operacine sistema, resursai Intel(R) Core(TM) i3-3110M CPU @ 2.40 GHz procesorius, 8.00 GB RAM darbinė atmintis, 64-bit operacinės sistemos tipas.

Naudojantis sukurtomis programomis atliekamas tyrimas, kurio metu siekiama išsiaiškinti :

- metodų efektyvumą sprendžiant įrenginių paskirstymo uždavinius;
- metodų priklausomybė nuo parametrų;
- metodų skaičiavimo greitį;
- bei metodų daromas paklaidas.

Tyrimo metu vykdoma atsitiktinė duomenų generacija ir naudojama įrenginių paskirstymo uždaviniams pritaikyti duomenų failai iš J. E. Beasley tinklalapio [7]. Šiuose failuose pateikiamos potencialių įrenginių kainos, kiekvieno įrenginio aptarnaujamo kliento išlaidos bei tikslios optimalios uždavinio sprendinio reikšmės(žr. 2.1 lentelę).

2.1 lentelė.

Beasley tinklalapio failų duomenys

Failo pavadinimas	Dydis (n × m)	Optimalus sprendinys(S^*)
Cap71	16 × 50	932615,75
Cap72	16 × 50	977799,4
Cap73	16 × 50	1010641,45
Cap74	16 × 50	1034976,975
Cap101	25 × 50	796648,44
Cap102	25 × 50	854704,2
Cap103	25 × 50	893782,1125
Cap104	25 × 50	928941,75
Cap131	50 × 50	793439,562
Cap132	50 × 50	851495,325
Cap133	50 × 50	893076,712
Cap134	50 × 50	928941,75

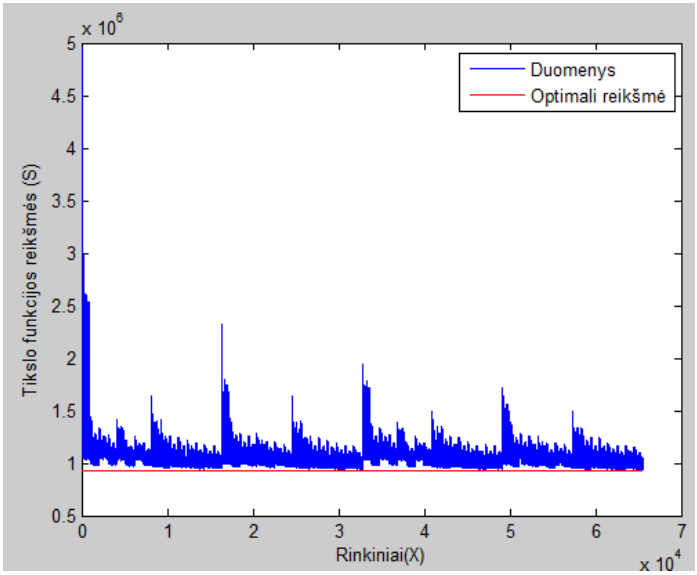
Dėl didelių apimčių, duomenų failai pateikiami skaitmeninių variantu pridėtame kompaktiniame diske, faile „duomenys“.

Modeliuojamo atkaitinimo algoritmas pagrįstas atsitiktine paieška ir priklauso nuo parinktų parametrų. Todėl prieš naudojamų algoritmų palyginimą buvo atliktas tyrimas, kurio metu tikrinama, kurie iš parametrų T_0, δT, *imax* ,*p* ir *q* turi didžiausią įtaką metodo tikslumui.

2.1 MODELIUOJAMO ATKAITINIMO ALGORITMO SPRENDINIO PRIKLAUSOMYBĖS NUO PARINKTŲ PARAMETRŲ TYRIMAS

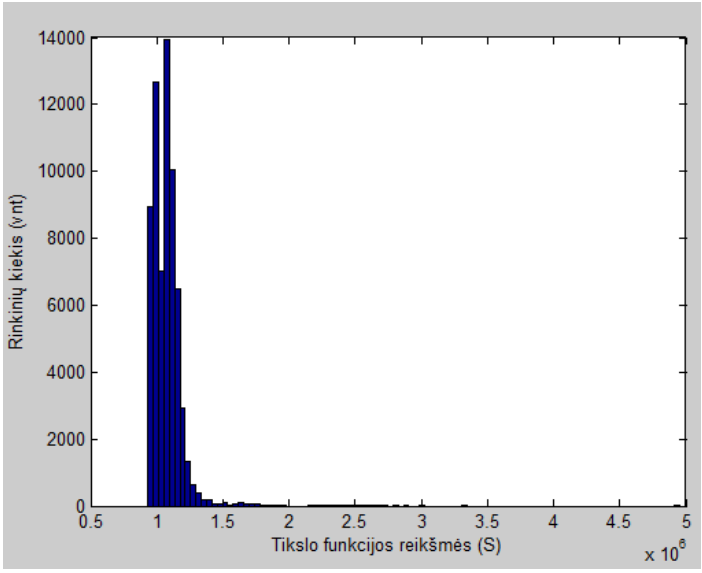
Pirmiausia apžvelgiama su kokiais duomenimis dirbame ir koks jų pasiskirstymas.

Tyrimui naudojamas duomenų failas cap71. Brutalios jėgos metodu apskaičiuotos visų galimų rinkinių reikšmes pateikiamos 2.1 paveiksle. Rinkiniai išdėstyti dvejetainio parinkimo didėjimo tvarka: pirmas rinkinys su pirmu įrenginiu, antras su antru įrenginių, trečias su pirmu ir antru įrenginiais... paskutinis rinkinys su visais įrenginiais. Kiekvienam rinkiniui skaičiuojama tikslo funkcijos reikšmė ir ji atvaizduojama. Gautos reikšmės vaizduojamos mėlyna spalva, o optimali reikšmė pavaizduota raudona liniją X^*=932615.75.



2.1 pav. Tikslo funkcijos priklausomybė nuo pasirinkto įrenginių rinkimo.

2.2 paveiksle pateikiama histograma, kurioje visos gautos tikslo funkcijos reikšmės (2.1 pav.) padalintos į 100 lygių dalių ir paskaičiuota, kiek rinkinių patenka į duotas sritis. Vieno histogramos stulpelio plotis apima 40244.488 Lt tikslo funkcijos pokyti, o t.y. 4.31% paklaida nuo optimalios reikšmės.

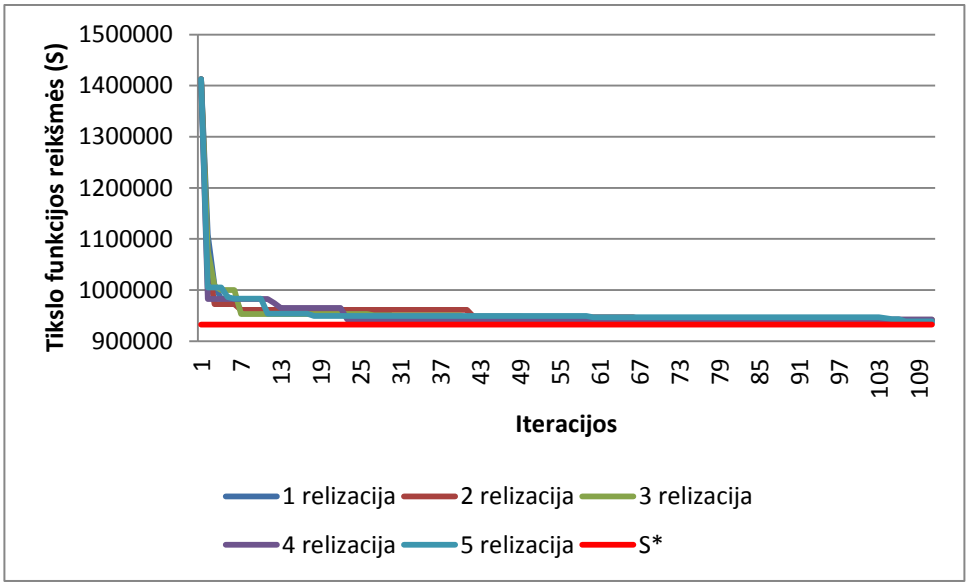


2.2 pav. Duomenų histograma.

Iš pateiktos histogramos matome, kad daugelio rinkinių reikšmės yra pasiskirsčiusios 0 – 32 % srityje, o rinkinių su labai dideliu sprendinių yra nedaug.

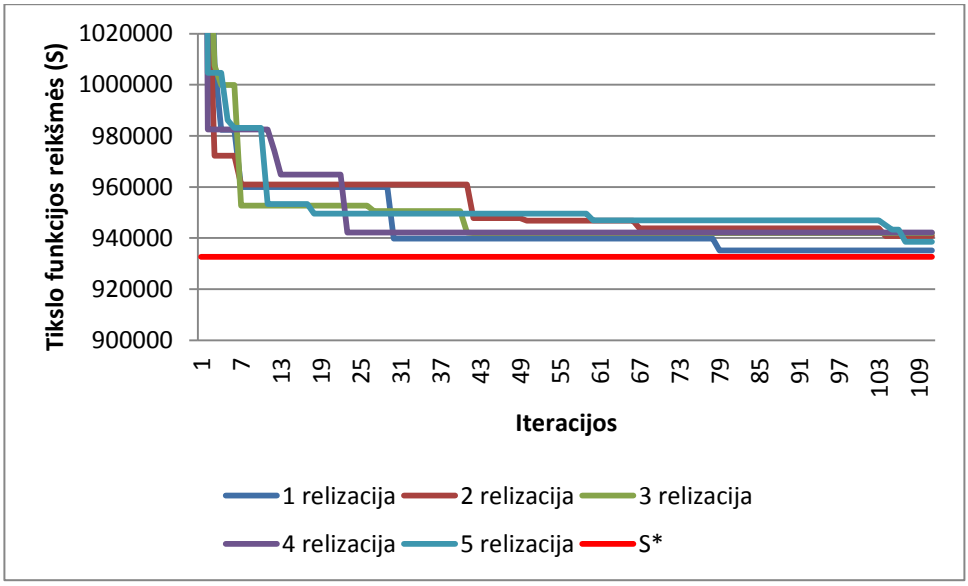
Žinant, kaip pasiskirstę duomenys, galima tiksliau vertinti algoritmo daromas paklaidas.

Kadangi modeliuojamo atkaitinimo algoritmas yra atsitiktinės paieškos metodas, tai pradėjęs spręsti uždavinį su tuo pačiu pradiniu rinkinių, gaunamas vis kitoks artėjimas prie optimalios reikšmės, kas ir pavaizduota 2.3 paveiksle.



2.3 pav. Optimalaus sprendinio paieška modeliuojamo atkaitinimo algoritmu.

2.4 paveiksle pateikiamas padidintas 2.3 paveikslo vaizdas.



2.4 pav. Optimalaus sprendinio paieška modeliuojamo atkaitinimo algoritmu.

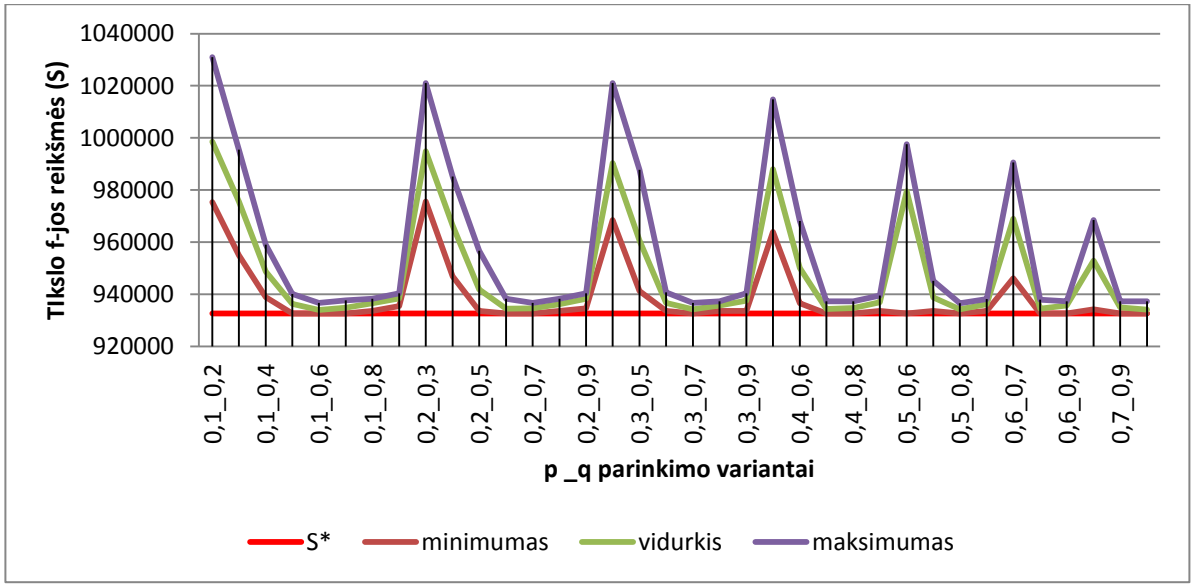
Dėl šios priežasties tyrimo metu skaičiavimus su tais pačias parametrais pakartojamas keletą kartų ir skaičiuojamos minimalios, maksimalios bei vidutines gaunamas algoritmo reikšmės.

2.1.1 PARAMETRŲ *p* IR *q* PARIKIMAS.

Nuo parametrų *p* ir *q* parikimo priklauso, koks veiksmas atliekamas su turimu įrenginių rinkiniu *X* ieškant kaimyninio sprendinio.

Priminsime, kad *Apkeitimo* operacijos tikimybė yra *p*, *Pridėjimo* operacijos *q-p* ir *Pašalinimo* operacijos *1-q*. Vizualiai tai pateikiama 1.4 paveiksle.

Eksperimentas atliekamas žingsnis po žingsnio keičiant p ir q reikšmes. Kadangi p < q, tai *p={0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8}*,o *q={ 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}*. Tais atvejais, kai *p* > *q*, skaičiavimai nevykdomi. Rezultatai pateikiami 2.5 paveiksle ir B priede su cap101 failu (B1 pav.). Grafike pateikiami *p* ir *q* rinkiniai žymimi *p_q (0.1_0.2 ir t.t.)*.



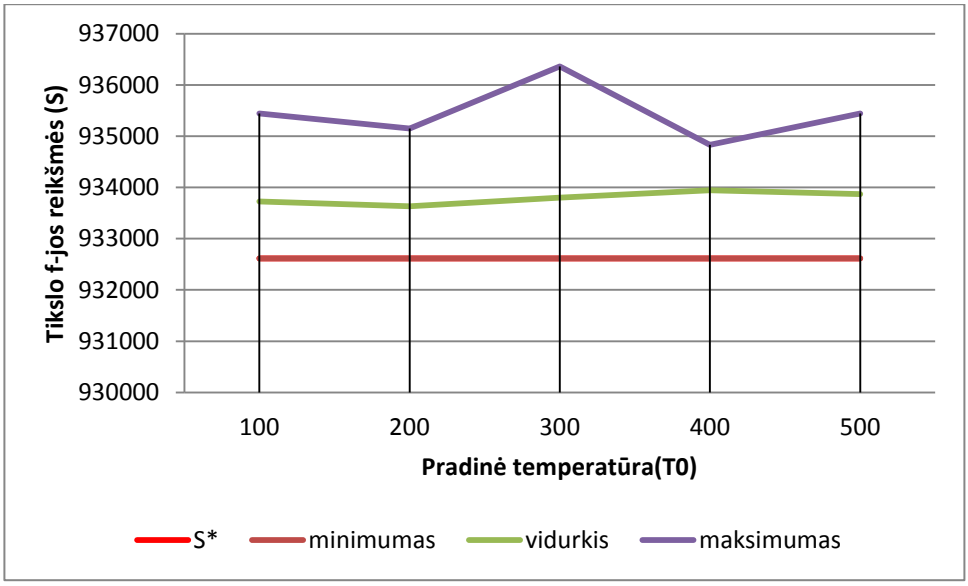
2.5 pav. Tikslo funkcijos priklausomybė nuo parinktų *p_q* rinkinių.

2.5 ir B1 paveiksluose pastebima, kad geriausi sprendiniai gaunami, kai *Pridėjimo* ir *Pašalinimo* operacijų tikimybės yra panašios arba *Pašalinimo* mažesnė, nei *Pridėjimo*, kitaip tariant, jei $q - p \approx 1 - q$, *arba* $q - p > 1 - q$.

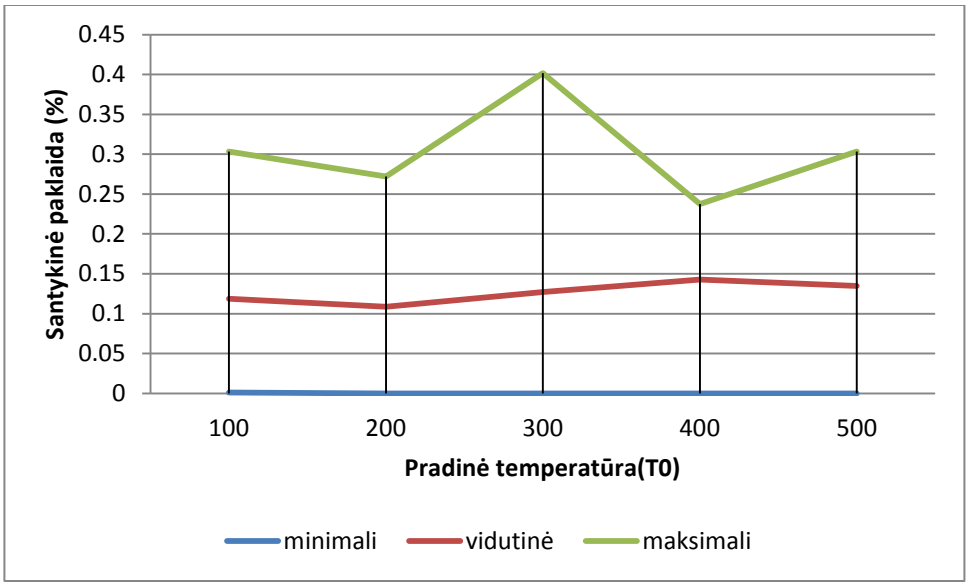
2.1.2 SPRENDINIO PRIKLAUSOMYBĖ NUO PARINKTOS PRADINĖS TEMPERATŪROS T_0.

Nuo temperatūros priklauso gauto naujo sprendinio priėmimas. Kuo aukštesnė temperatūra, tuo didesnė tikimybė priimti prastesnį sprendinį, tačiau blogesnio sprendinio priėmimas gali duoti galimybę pajudėti iš lokalaus minimumo, kas suteikia algoritmui pranašumą.

2.6 paveiksle pavaizduota tikslo funkcijos priklausomybė nuo pradinės temperatūros, o santykinė paklaida pateikta 2.7 paveiksle.

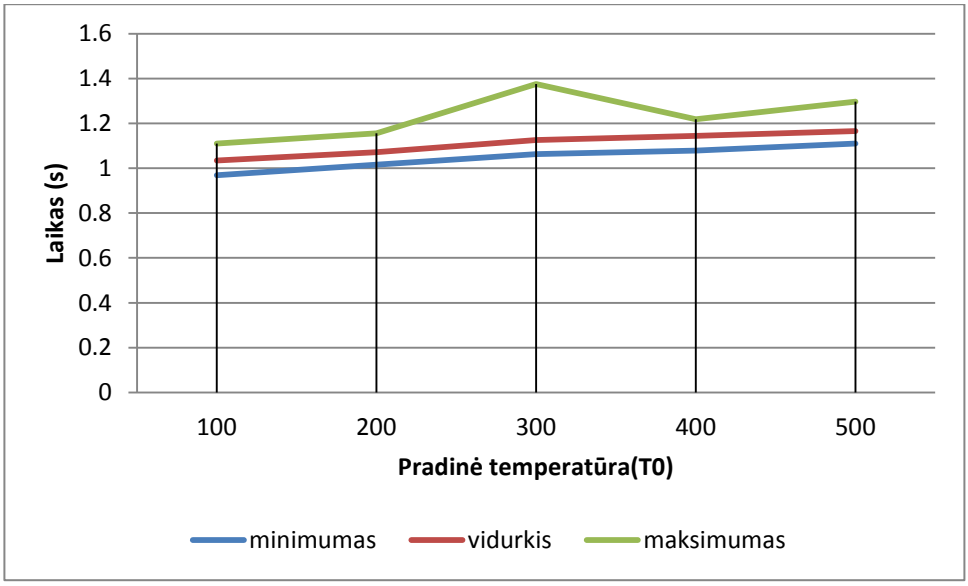


2.6 pav. Tikslo f-jos priklausomybė nuo parinktos pradinės temperatūros T_0.



2.7 pav. Gaunamų paklaidų priklausomybė nuo pradinės temperatūros T_0.

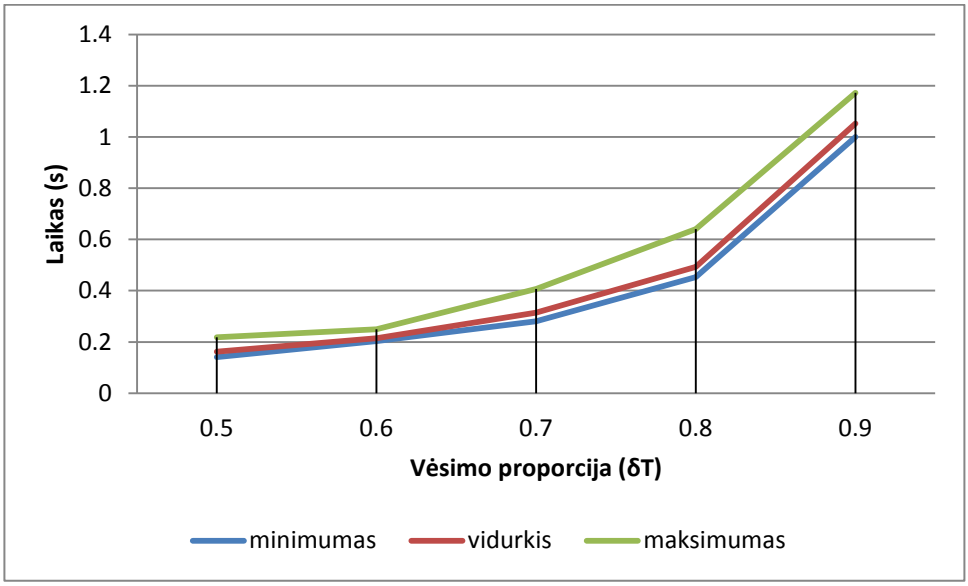
Temperatūrai didėjant sprendinio tikslumo gerėjimo tendencijos nesimato. Modeliuojamo atkaitinimo metodo vykdymo laiko priklausomybė nuo pradinės temperatūros pateikiama 2.8 paveiksle.



2.8 pav. Vykdymo laikas nuo T_0.

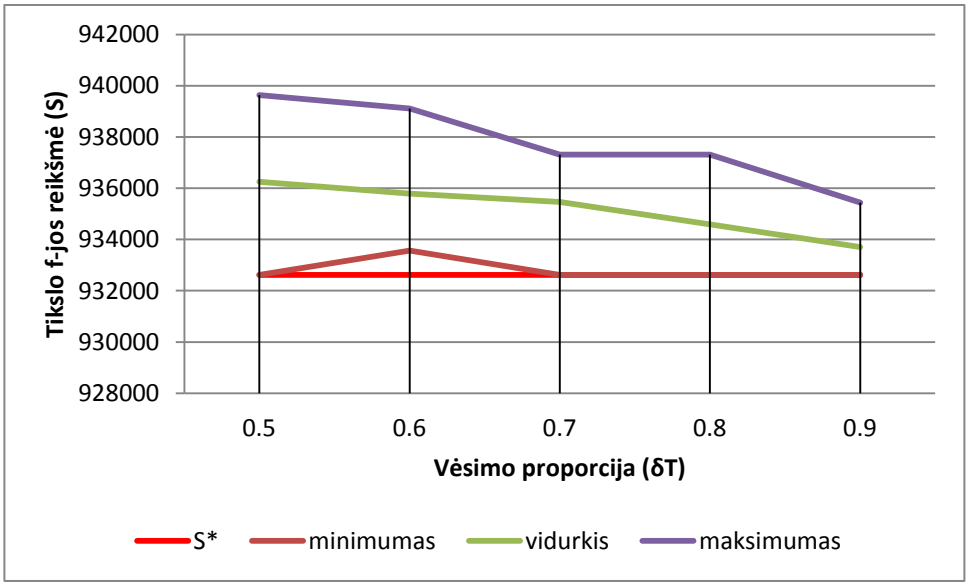
2.1.3 SPRENDINIO PRIKLAUSOMYBĖ NUO PARINKTO VĖSIMO PROPORCIJOS KOEFICIENTO δT.

Vėsimo proporcija koeficientas δT naudojamas mažinti temperatūrą. Šiame darbe naudojamas proporcingo vėsimo metodas, kurio, *i*-tojo vėsinimo momentu, temperatūra apskaičiuojama: $T_i = \delta T * T_{i-1}$. Taigi kuo δT mažesnis, tuo vėsimas vyksta greičiau, priešingu atveju – lėčiau (žr. 2.9 pav.).

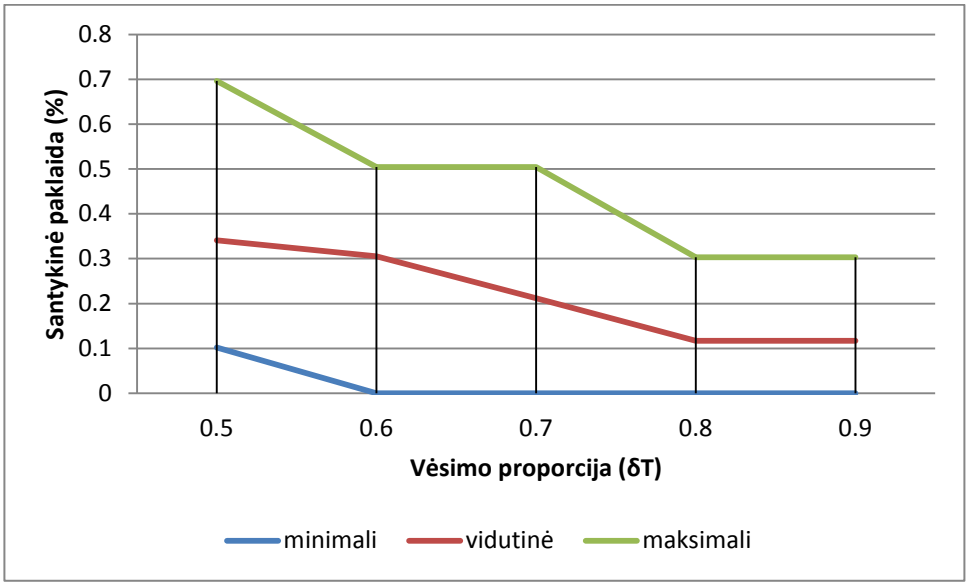


2.9 pav. Algoritmo vykdymo laiko priklausomybė nuo vėsimo proporcijos koeficiento δT.

Tačiau panagrinėjus, kaip kinta tikslo funkcijos reikšmė keičiant δT (žr. 2.10 pav.), pastebima, kad kuo didesnė δT, tuo labiau artėjama prie optimalios reikšmės. Santykinės paklaidos pateiktos 2.11 paveiksle.



2.10 pav. Tikslo f-jos priklausomybė nuo vėsimo proporcijos δT.

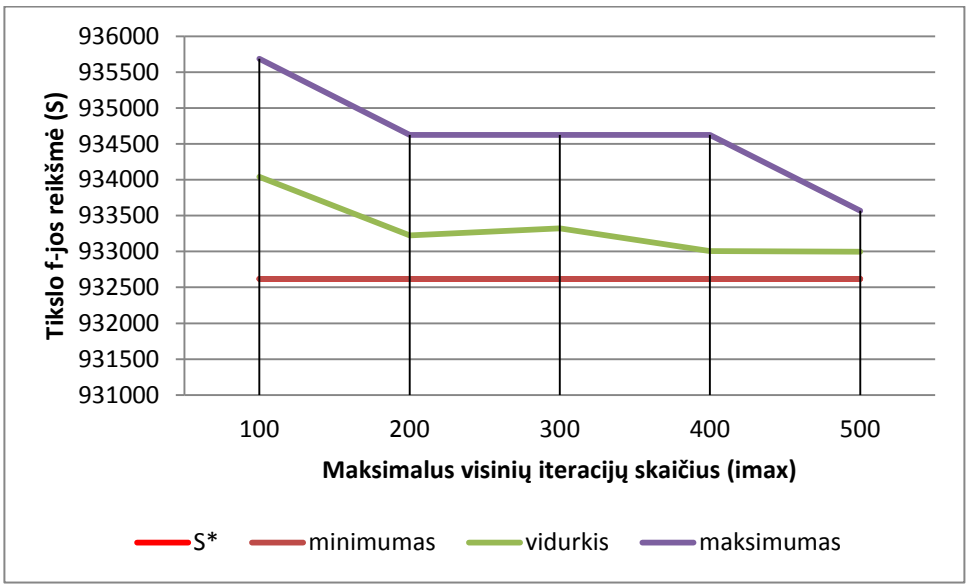


2.11 pav. Gaunamų paklaidų priklausomybė nuo vėsimo proporcijos δT.

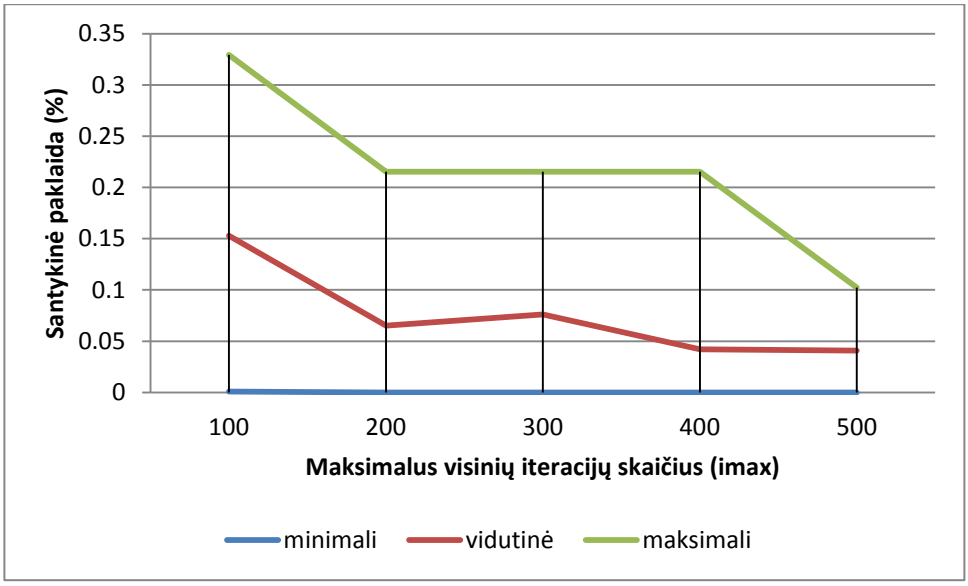
2.1.4 SPRENDINIO PRIKLAUSOMYBĖ NUO VIDINIŲ ITERACIJŲ SKAIČIAUS *imax*.

Vidinės iteracijos atliekamos tarp vėsinimo iteracijų. Kiekvieną kartą sumažinus temperatūrą ($T_i = \delta T * T_{i-1}$) modeliuojamo atkaitinimo algoritmas atlieka *imax* iteracijų siekiant pagerinti turimą sprendinį. Vidinės iteracijos metu ieškomas kaimyninis rinkinys naudojant funkciją *Kaimynas(p,q)*, kuri priklauso nuo jau aptartų parametrų *p* ir *q*, ir tikrinama ar gautas įrenginių rinkinys duoda geresnį sprendinį.

Taigi, jei atliekama daugiau iteracijų, tai gauname tikslesnį sprendinį(žr. 2.12 pav.). Gaunamos paklaidos pateikiamos 2.13 pav.

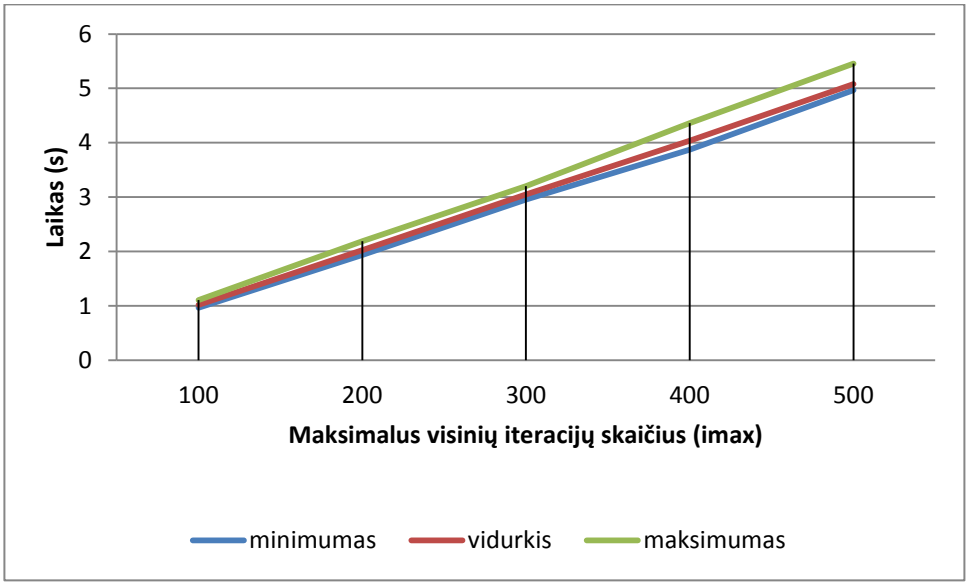


2.12 pav. Tikslo funkcijos priklausomybė nuo vidinių iteracijų *imax*.



2.13 pav. Gaunamų paklaidų priklausomybė nuo iteracijų skaičiaus *imax*.

Bet didinant iteracijų skaičių, proporcingai auga ir vykdymo laikas: jei iteracijų skaičių padvigubinsime, dvigubai pailgės skaičiavimo laikas (žr. 2.14 pav.)



2.14 pav. Algoritmo vykdymo laiko priklausomybė nuo maksimalaus vidinių iteracijų skaičiaus *imax*.

2.1.5 MODELIUOJAMO ATKAITINIMO ALGORITMO SPRENDINIO PRIKLAUSOMYBĖS NUO PARINKTŲ PARAMETRŲ APIBENDRINIMAS.

Aukščiau atliktais eksperimentais ištyrėme, kokia modeliuojamo atkaitinimo metodo parametrų daroma įtaka ieškant geriausio sprendinio, vertinant vykdymo laiką bei gaunamas santykines paklaidas.

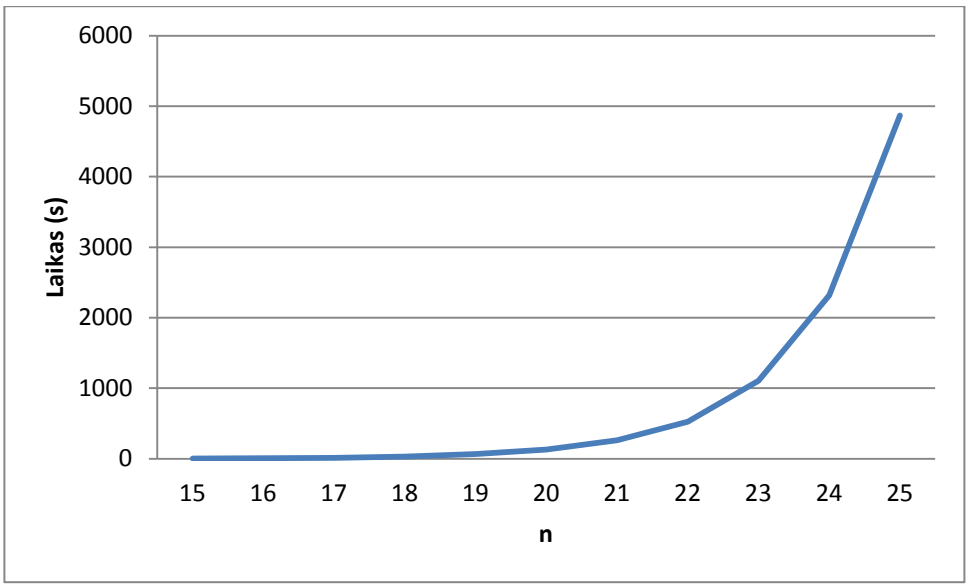
Apibendrinat eksperimentus galime teigti, kad:

- Funkcija *Kaimynas(p,q)* optimaliausiai veikia, kai *Pridėjimo* ir *Pašalinimo* operacijų tikimybės yra lygios arba *Pridėjimo* tikimybė didesnė.
- Pradinės temperatūros didinimas mažai įtakoja sprendinio tikslumą;
- Vėsimo proporcijai koeficientui δT artėjant prie 1 gaunami tikslesni sprendiniai, tačiau vykdymo laikas auga;
- Padidinus vidinių iteracijų skaičių *imax* santykinė paklaida mažėja, tačiau vykdymo laikas auga proporcingai padidintų iteracijų skaičiui.

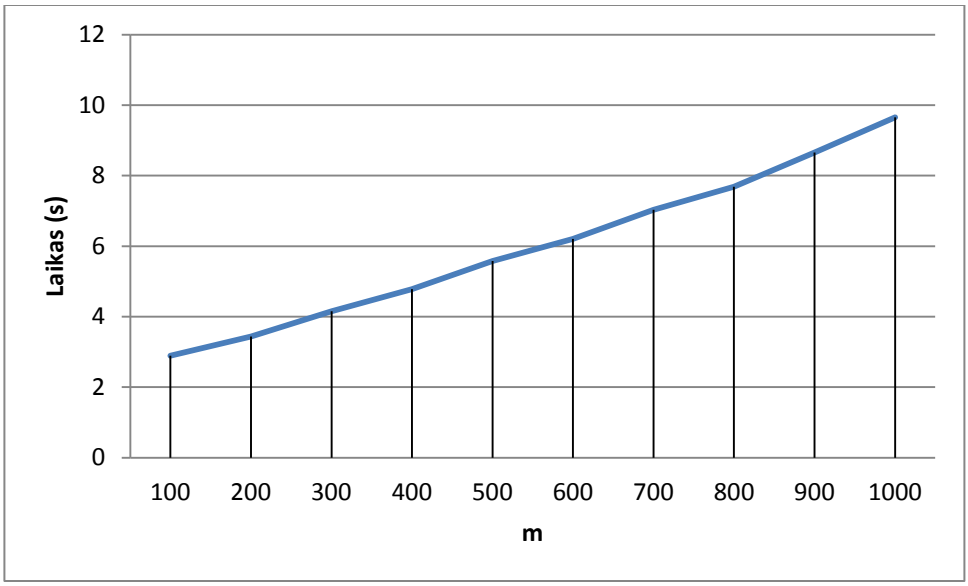
2.2 ALGORITMŲ SKAIČIAVIMO LAIKO PRIKLAUSOMYBĖ NUO UŽDAVINIO PARAMETRŲ *n* IR *m*

Siekiant išsiaiškinti, kokia yra pasirinktų algoritmų vykdomo laiko priklausomybė nuo įrenginių skaičiaus *n* ir klientų skaičiaus *m*, atliksime tyrimą, kurio metu naudojami atsitiktinai sugeneruoti duomenų failai.

Brutalios jėgos metodo vykdymo laiko priklausomybė nuo įrenginių skaičiaus $n=\{15,..,25\}$, kai klientų skaičius $m=100$ pateikiama 2.15 paveiksle. O vykdymo laiko priklausomybė nuo klientų skaičiaus $m=\{100,200,...,1000\}$,kai $n=15$ - 2.16 paveiksle. A priede rezultatai pateikti A1 lentelėje.



2.15 pav. Brutalios jėgos algoritmo skaičiavimo laiko priklausomybė nuo parametro *n*.



2.16 pav. Brutalios jėgos algoritmo skaičiavimo laiko priklausomybė nuo parametro *m*.

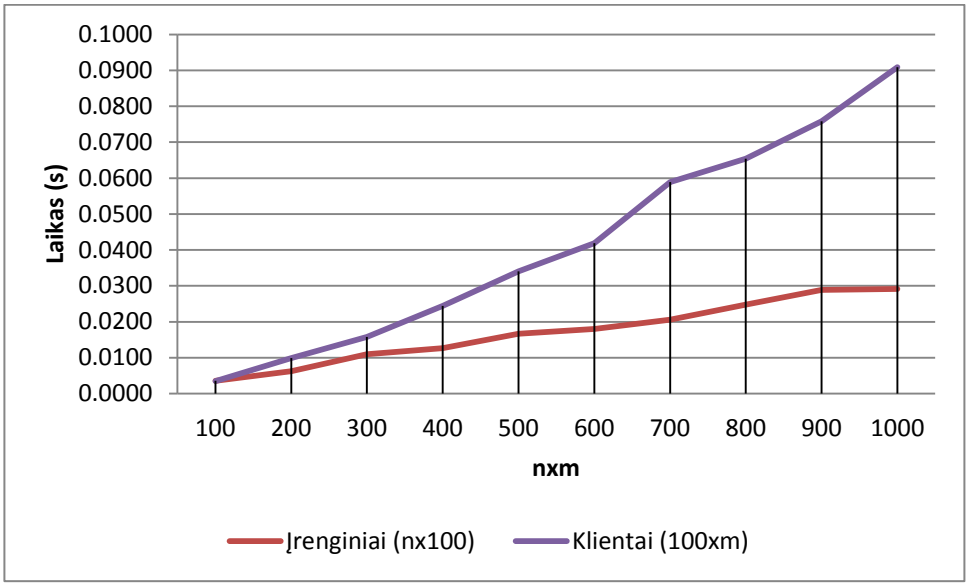
2.15 paveiksle matoma, kad didėjant įrenginių skaičiui, brutalios jėgos algoritmo vykdymo laikas auga eksponentiškai, tačiau didėjant klientų skaičiui gaunama tiesinė priklausomybė (2.16 pav.)

Toliau, tyrimas atliekamas fiksuojant vienas parametras ties 100 vienetų, o kitą parametrą keičiant nuo 100 iki 1000, intervalu - 100 vienetų. Eksperimento rezultatai nuo įrenginių skaičiaus didėjimo žymimas Įrenginiai(n*x*100), o nuo klientų skaičiaus – Klientai(100*x*m).

Su brutalios jėgos metodu šis bandymas neatliekamas, nes parinkti parametrų rinkiniai per dideli.

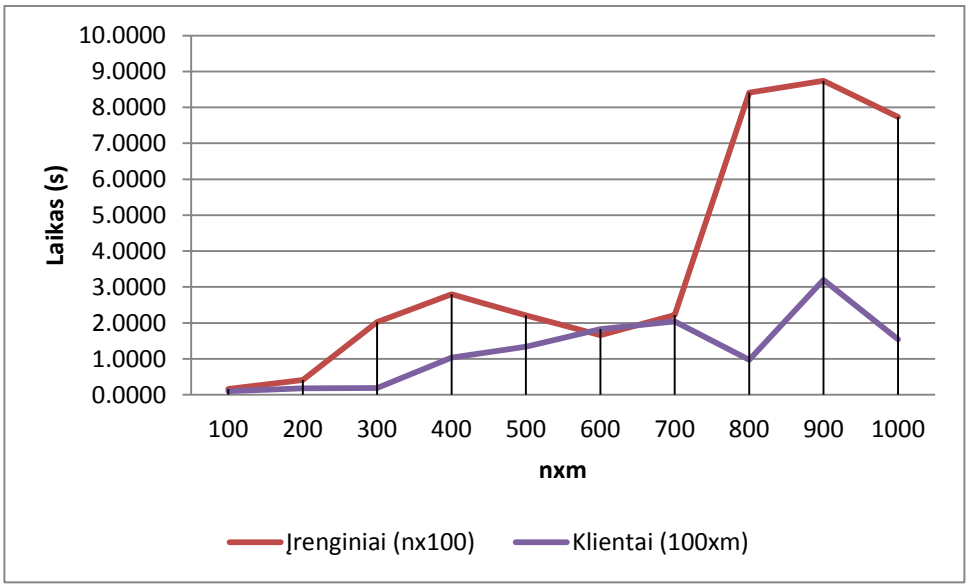
Godžiojo metodo gauti rezultatai pateikiami 2.17 paveiksle. Apkeitimo metodo - 2.18 paveiksle.

Modeliuojamo atkaitinimo metodo gauti rezultatai pateikiami 2.19 paveiksle. A priede atitinkamai pateikiamos lentelės A2, A3 ir A4.



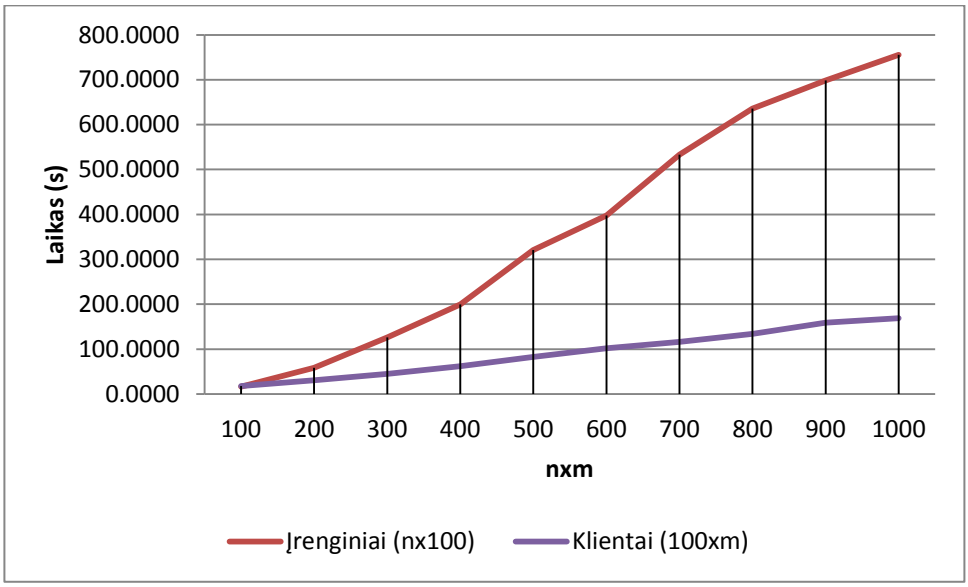
2.17 pav. Godžiojo algoritmo skaičiavimo laiko priklausomybė nuo klientų *m* ir įrenginių *n* skaičiaus

Godžiojo algoritmo vykdymo laikas tiesiškai priklausomas tiek nuo įrenginių skaičiaus *n*, tiek ir nuo klientų kiekio *m* padidėjimo. Tačiau laiko pokyti labiau įtakoja klientų skaičiaus padidėjimas. (2.17 pav.)



2.18 pav. Apkeitimo algoritmo skaičiavimo laiko priklausomybė nuo klientų *m* ir įrenginių *n* skaičiaus.

Apkeitimo algoritmo vykdymo laiką labiau įtakoja įrenginių skaičius *n*. Dėl kreivės netolydumo galima daryti prielaidą, kad šie pokyčiai atsiranda nuo atliekamų apkeitimų skaičiaus.



2.19 pav. Modeliuojamo atkaitinimo algoritmo skaičiavimo laiko priklausomybė nuo klientų *m* ir įrenginių *n* skaičiaus.

Modeliuojamo atkaitinimo algoritmo vykdymo laiko priklausomybė, kaip ir godžiojo algoritmo, yra tiesinė nuo abiejų parametrų, tik atvirkščiai nei godžiajam – atkaitinimo algoritmui ženkliai didesnę įtaką daro įrenginių skaičius *n* (žr. 2.17 ir 2.19 pav.).

2.3 ĮRENGINIŲ PASKIRSTYMO UŽDAVINIŲ SPRENDIMO METODŲ LYGINAMOJI ANALIZĖ

Šiame skyrelyje atliekamas palyginimas, kurie iš pasirinktų brutalios jėgos, godžiojo, apkeitimo ar modeliuojamo atkaitinimo metodų veikia tiksliau ir sparčiau sprendžiant įrenginių paskirstymo uždavinius. Tyrimui naudojami 2 skyriaus pradžioje aprašyti duomenų failai (žr. 2.1 lentelę).

Tiksliuoju brutalios jėgos metodu kiekvieną uždavinį spręsime tik kartą, nes šis metodas visada randą tikslų sprendinį ir vykdymo laikas priklauso tik nuo pasirinkto duomenų failo dydžio.

Taip pat kaip ir tiksliuoju metodu, euristinius godųjį ir apkeitimo algoritmus tam pačiam uždaviniui vykdysime po vieną kartą. Kitaip nei brutalios jėgos metodas, šie algoritmai ne visada randa tikslų sprendinį, tačiau visada tą patį sprendinį. Vykdymo laikas taip pat priklauso tik nuo duomenų failo dydžio (žr. 2.2 skyrelį).

Modeliuojamo atkaitinimo algoritme parinkti parametrai turi įtaką gaunamiems rezultatams. 2.1 skyrelyje buvo ištirta, kad vėsimo proporcija koeficiento δT ir maksimalus iteracijų skaičiaus *imax* kitimas turi džiausią įtaką gaunamiems rezultatams.

Su modeliuojamu atkaitinimu atliekami keturi eksperimentai: pirmo eksperimento metu skaičiuojama su vienu parametrų rinkiniu, tada, kitais trim eksperimentais, bandoma pagerinti gautus rezultatus. Eksperimentų parametrai pateikiami 2.2 lentelėje. Kiti parametrai: $T_0 = 100$, $p = 0.1$, $q = 0.6$.

2.2 lentelė.

Atliekamų eksperimentų parametrų pasirinkimas.

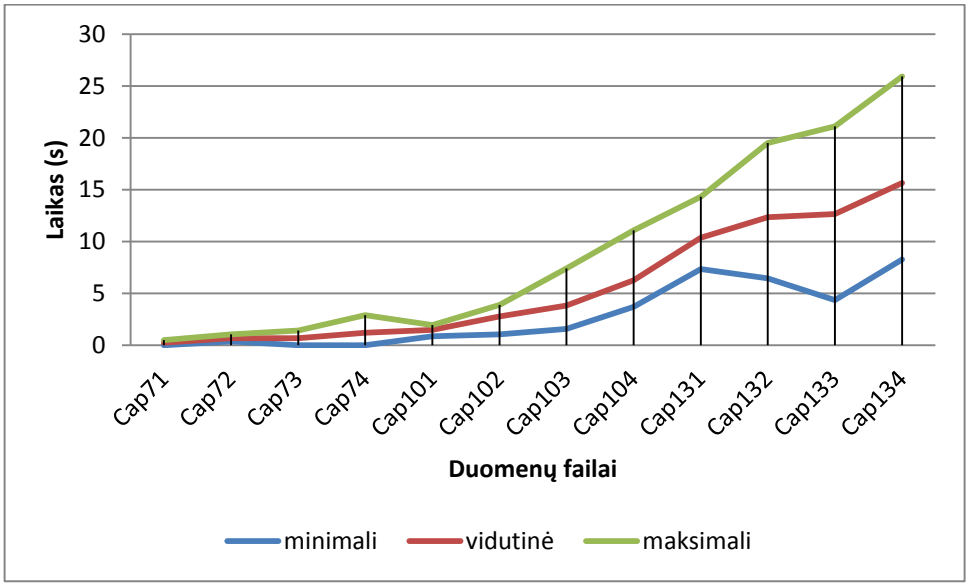
	δT	imax	Rezultatai
I eksperimentas	0,8	100	2.3 lentelė
II eksperimentas	0,9	100	B2 lentelė
III eksperimentas	0,8	400	B3 lentelė
IV eksperimentas	0,9	400	B4 lentelė

Pirmo eksperimento rezultatai pateikiami 2.3 lentelėje ir pavaizduojami grafiškai 2.20 paveiksle, kitų eksperimentų rezultatai pateikiami priede A5 – A7 lentelėse bei, atitinkamai, B2 - B4 paveiksluose. Vidutinės eksperimentų paklaidos ir vidutinis vykdymo laikas pateikiamas algoritmų palyginimo 2.4 ir 2.5 lentelėse.

2.3 lentelė.

Modeliuojamo atkaitinimo metodo rezultatai. I eksperimentas.

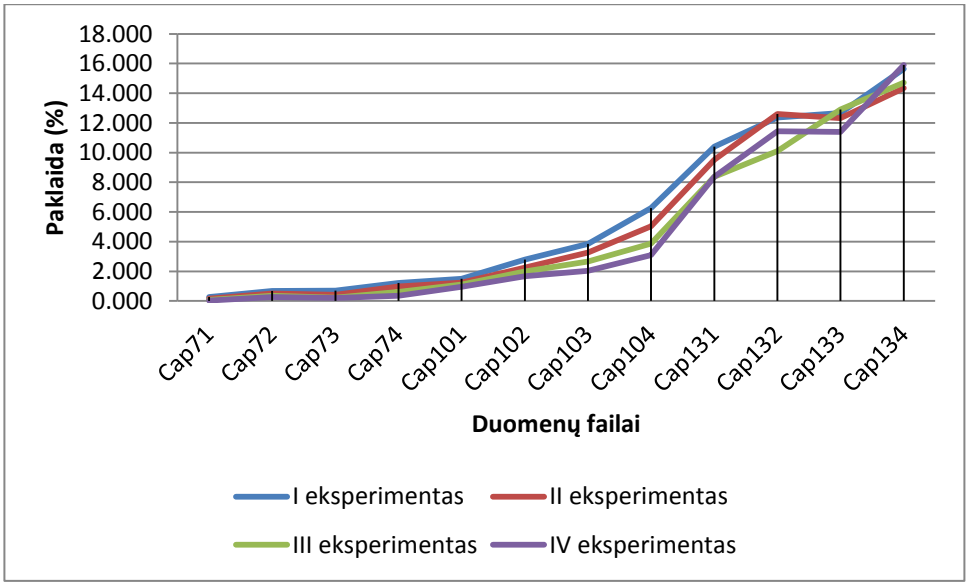
Pava-	Paklaida (%)			CPU laikas (s)
dinimas	minimali	vidutinė	maksimali	vidurkis
Cap71	0	0,267883	0,504023	0,462
Cap72	0,382435	0,681824	1,050038	0,464
Cap73	0	0,686194	1,429972	0,474
Cap74	0	1,220088	2,91767	0,466
Cap101	0,874656	1,485695	1,94602	0,705
Cap102	1,059824	2,798679	3,899389	0,708
Cap103	1,579203	3,845101	7,418317	0,725
Cap104	3,723673	6,263093	11,09085	0,710
Cap131	7,343953	10,3894	14,31309	1,666
Cap132	6,461524	12,35651	19,49972	1,699
Cap133	4,350957	12,66308	21,09977	1,711
Cap134	8,26357	15,63819	25,91181	1,733



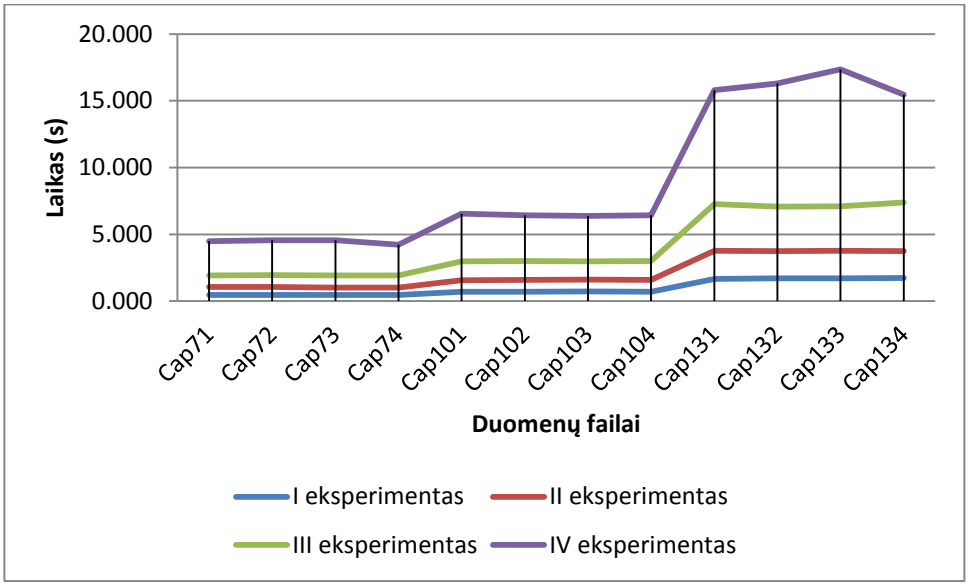
2.20 pav. I eksperimento paklaidų grafikas.

Iš gautų rezultatų, pateiktų grafike (2.20 pav. ir priede) pastebime, kad didėjant uždavinio apimčiai (*n)* auga gaunamo sprendinio paklaidos. Taip pat, intervalas tarp didžiausios ir mažiausios paklaidos, ženkliai padidėja.

Palyginimui, vykdytų keturių eksperimentų, pateikiamos vidutinės paklaidos (2.21 pav., 2.7 lentelė), ir vykdymo laikai (2.22 pav., 2.5 lentelė).



2.21 pav. Eksperimentų vidutinių paklaidų palyginimas.



2.22 pav. Eksperimentų vykdymo laiko palyginimas.

Pasirenkame I eksperimentą, kaip bazinį. Padidinus vėsimo koeficientą nuo $\delta T=0.8$ iki $\delta T=0.9$ (II eksperimentas) paklaidų pokytis yra nuo 0.1% (cap71-cap72) iki 1.3% (cap104), bet vykdymo laikas pailgėja ~2.2 karto.

Padidinus iteracijų skaičių nuo 100 iki 400 (III eksperimentas), tai paklaidų pokytis nuo 0.2% (cap71-cap72) iki 2.3% (cap141), bet vykdymo laikas išauga ~4.2 karto.

Jei abu pakeitimus atliksime iškarto (IV eksperimentas), tai gaunamas paklaidų pokytis nuo 0.2% (cap71-cap72) iki 3.2% (cap104), bet vykdymo laikas pailgėja ~10 kartų.

Brutalios jėgos, godžiojo ir apkeitimo algoritmais duomenų failai trasuot po vieną kartą, nes pakartotinis trasavimas nepakeičia gaunamų rezultatų.

2.4 lentelė.

Brutalios jėgos, godžiojo, apkeitimo ir modeliuojamo atkaitinimo algoritmų paklaidos.

Pava-dinimas	Dydis (n × m)	Brutalios jėgos	Godusis	Apkeitimo	Modeliuojamo atkaitinimo (eksperimentai)			
					I	II	III	IV
Cap71	16 × 50	0	0	0	0,268	0,164	0,081	0,020
Cap72	16 × 50	0	0,382	0,110	0,682	0,509	0,351	0,259
Cap73	16 × 50	0	0,182	0,182	0,686	0,411	0,214	0,191
Cap74	16 × 50	0	0	0	1,220	0,972	0,621	0,348
Cap101	25 × 50	0	0,108	0,108	1,486	1,225	1,102	0,955
Cap102	25 × 50	0	0,148	0	2,799	2,257	1,992	1,669
Cap103	25 × 50	0	0,139	0,139	3,845	3,272	2,664	2,032
Cap104	25 × 50	0	0	0	6,263	5,036	3,872	3,094
Cap131	50 × 50	*	0,108	0,108	10,389	9,514	8,377	8,376
Cap132	50 × 50	*	0,149	0	12,357	12,600	10,101	11,433
Cap133	50 × 50	*	0,114	0,114	12,663	12,321	12,899	11,402
Cap134	50 × 50	*	0	0	15,638	14,352	14,711	15,905

*- duomenų kiekis per didelis atlikti skaičiavimus.

2.5 lentelė.

Brutalios jėgos, godžiojo, apkeitimo ir modeliuojamo atkaitinimo algoritmų vykdymo laikas.

Pava-dinimas	Dydis (n × m)	Brutalios jėgos	Godusis	Apkeitimo	Modeliuojamo atkaitinimo (eksperimentai)			
					I	II	III	IV
Cap71	16 × 50	5,530	0,0005	0,0007	0,462	1,058	1,912	4,480
Cap72	16 × 50	5,420	0,0004	0,0012	0,464	1,069	1,942	4,560
Cap73	16 × 50	5,590	0,0003	0,0005	0,474	1,022	1,932	4,555
Cap74	16 × 50	5,530	0,0003	0,0005	0,466	1,012	1,913	4,214
Cap101	25 × 50	4929,310	0,0005	0,0015	0,705	1,569	2,977	6,555
Cap102	25 × 50	4985,630	0,0005	0,0024	0,708	1,584	3,013	6,426
Cap103	25 × 50	5025,230	0,0005	0,0009	0,725	1,620	2,975	6,389
Cap104	25 × 50	5031,540	0,0003	0,0008	0,710	1,593	3,005	6,440
Cap131	50 × 50	*	0,0009	0,0043	1,666	3,763	7,268	15,804
Cap132	50 × 50	*	0,0007	0,0079	1,699	3,742	7,070	16,303
Cap133	50 × 50	*	0,0006	0,0022	1,711	3,770	7,094	17,363
Cap134	50 × 50	*	0,0005	0,0012	1,733	3,749	7,374	15,460

*- duomenų kiekis per didelis atlikti skaičiavimus.

2.4 lentelėje matome, kad skaičiuojant tiksliuoju brutalios jėgos algoritmu visada randams optimalus sprendinys (paklaidos lygios 0), tačiau vykdymo laikas (2.5 lentelė) ir skaičiavimo apimtys

greitai auga didėjant uždavinio apimčiai: su 16 įrenginių skaičiuojama ~5.5s, o padidinus įrenginių skaičių iki 25 - net ~5000s (~83min arba 1.4h).

Euristinių algoritmų aprašyme teigiama, kad šių algoritmų tikslas per greitą laiką rasti aukštos kokybės, bet nebūtinai optimalų sprendinį. Iš gautų rezultatų matome, kad naudojamas godusis ir jį pagerinantis apkeitimo algoritmai labai gerai susitvarkė su duota užduotimi. Godžiojo algoritmo gaunamos paklaidos neviršija 0.15%, o kai kurias atvejais gaunamos optimalios sprendinys (žr. 2.4 lentelę, cap71, cap74, cap104, cap134 eilutės). Taip pat matome, kad apkeitimo algoritmas keliais atvejais pataiso godžiojo algoritmo gautus sprendinius (žr. 2.4 lentelę cap72,cap102, cap132 eilutės). Pastebimai mažas skaičiavimo laikas, lyginant su brutalios jėgos ir modeliuojamo atkaitinimo metodais, godžiojo algoritmu vykdymo laikas 10^{-4}, o apkeitimo - 10^{-3} sekundės dalys(žr. 2.5 lentelę).

Meta-euristinio modeliuojamo atkaitinimo algoritmo paklaidos (žr. 2.4 lentelę) akivaizdžiai yra didesnės, nei euristinių ir brutalios jėgos algoritmų paklaidos, tačiau modeliuojamo atkaitinimo vykdymo laikas yra kur kas mažesnis, nei brutalios jėgos laikas (žr. 2.5 lentelę).

3. PROGRAMINĖ REALIZACIJA IR INSTRUKCIJA VARTOTOJUI

Remiantis 1.3.1 - 1.3.4 skyreliuose pateiktų įrenginių paskirstymo uždavinių algoritmų pseudokodais ir aprašymais buvo sukurtos keturios programos. Jos realizuotos naudojant MathWorks kompanijos programa MATLAB®, 7.11.0.584(R2010b) versija, 64-bit operacinės sistemos tipu.

Programų kodai pateikti C priede.

Brutalios jėgos algoritmas vykdomas MATLAB komandų lauke įrašius komandą:

[A1, A2, A3] = bjega(C, f);

Programai reikia nurodyti transportavimo matricą C ir potencialių įrenginių kainų vektorių f. Atlikus skaičiavimus brutalios jėgos algoritmu gaunami atsakymai, kur:

A1 - įrenginių paskirstymo uždavinio sprendinys S*;

A2 - parinktų įrenginių sąrašas;

A3 - algoritmo vykdymo laikas.

Ši atsakymų struktūra išlaikoma su visais vykdomais algoritmais.

Godusis algoritmas vykdomas MATLAB komandų lauke įrašius komandą:

[A1, A2, A3] = godus(C, f);

Programai reikia nurodyti transportavimo matricą C ir potencialių įrenginių kainų vektorių f.

Apkeitimo algoritmas vykdomas MATLAB komandų lauke įrašius komandą:

[A1, A2, A3] = apkeitimas(X, C, f);

Programai reikia nurodyti pradinį potencialių įrenginių vektorių X, transportavimo matricą C ir potencialių įrenginių kainų vektorių f.

Modeliuojamo atkaitinimo algoritmas vykdomas MATLAB komandų lauke įrašius komandą:

[A1, A2, A3] = MA(X, C, f, T0, Ta, dT, imax, p, q);

- Programai reikia nurodyti:
- Pradinį potencialių įrenginių vektorių X;
- Transportavimo matricą C;
- Potencialių įrenginių kainų vektorių f;
- Pradinę vykdymo temperatūrą T0;
- Vesimo proporcijos koeficientą dT;
- Maksimalų vykdomų iteracijų skaičių imax;
- Ir tikimybių intervalo skirstymo p ir q reikšmes.

IŠVADOS

1. Įrenginių paskirstymo uždaviniai yra aktualūs įkuriant ar plečiant įmonę. Tinkamas įrenginių paskirstymas regione gali nulemti puikų įmonės vystymąsi.
2. Modeliuojamo atkaitinimo metodo skaičiavimo tikslumui didžiausią įtaką turi vėsimo proporcijos koeficientas δT ir maksimalus iteracijų skaičius *imax*. Algoritmo skaičiavimo laikui didesnę įtaką daro potencialių įrenginių *n* skaičiui, nei klientų skaičiaus *m,* tačiau modeliuojamo atkaitinimo algoritmo sprendinių intervalas, tarp rasto geriausio sprendinio ir blogiausio, platėja didėjant uždavinio apimčiai.
3. Brutalios jėgos metodas visada randa optimalų uždavinio sprendinį, bet vykdymo laiką labai įtakoja uždavinio apimtis.
4. Euristinio godžiojo algoritmo vykdymo laikas buvo trumpiausias sprendžiant sandėlių paskirstymo uždavinius, o sprendinio tikslumu nusileido tik brutalios jėgos metodui. Algoritmo skaičiavimo laikas labiau priklauso nuo klientų *m* skaičiui, nei potencialių įrenginių skaičiui *n*.
5. Godžiojo algoritmo sprendinio pagerinimui pasirinktas apkeitimo algoritmas, kai kuriais atvejai, pagerindavo godžiojo metodo sprendinius.

REKOMENDACIJOS

Darbe nagrinėjami tik keturi skirtingų klasių algoritmai, tačiau spręsti įrenginių paskirstymo uždavinius naudojama nemažai ir kitų algoritmų, tokių, kaip meta-euristiniai tabu paieška, skruzdžių kolonijos ir kiti. Taip pat, galima algoritmų hibridizacija, kai apjungiami keli algoritmai į vieną.

Pasirinktą sandėlių išdėstymo uždavinį galima modifikuoti įvedus apribojimus įrenginiams, nurodyti, kad įrenginiai gali teikti daugiau nei vieną paslaugą ir pan.

Dėl šių paminėtų priežasčių, lieka nemažai erdvės naujiems tyrimams, susijusiems su naujų algoritmų realizavimu, sudėtingesnių įrenginių paskirstymo uždavinių sprendimu.

PADĖKOS

Nuoširdžiai dėkoju savo magistro darbo vadovui doc. dr. Narimantui Listopadskiui už pagalbą rašant šį darbą, vertingus patarimus bei kantrybę.

Taip pat, dėkoju savo bendražygėms Erikai Butkutei ir Daliai Kitavičiūtei, už palaikymą, nes draugai iš duobės visada kapstosi kartu.

Dėkoju savo draugei Ievai už kantrybę ir palaikymą.

Šį darbą skiriu savo tėvams Aldonai ir Jonui už visokeriopą pagalbą.

SANTRUMPŲ IR TERMINŲ ŽODYNAS

I - įrenginių aibė

imax - maksimalus iteracijų skaičius;

J - klientų aibė

n - įrenginių skaičius;

m - klientų skaičius;

p - elementų parinkimo rėžis, nurodantis, kad su p tikimybė įvyks Apkeitimo operacija;

q - elementų parinkimo rėžis, nurodantis, kad su q-p tikimybė įvyks Pridėjimo operacija;

S^*- optimalus sprendinys;

S - geriausias sprendinys

T_0 -pradinė temperatūra;

T_a- absoliutinė temperatūra;

δT- vėsimo greitis;

X^* – optimalus rinkinys;

X^o – geriausias sprendinys;

imax - maksimalus iteracijų skaičius;

NAUDOTA LITERATŪRA

1. Arfin S., Location allocation problem using genetic algorithm and simulated annealing: a case study based on school in Enschede. Enschede, The Netherlands, p 95, 2010.
2. Aydin M. E., Yigit V. ir Fogarty T. C., Two Approaches to Simulated Annealing for Uncapacitated Facility Location Problems, South Bank University, London, UK.
3. Brendeau M. L. ir Chiu S. S., An overview of represenrative problems in location research", *Manadement science, vol 35*, pp 645-674, 1989.
4. Bumb A., Approximation Algorithms for Facility Location Problems, university of Twente, p 115, 2002.
5. Combinatorial Optimization [interaktyvus, žiūrėta 2014 01]. Prieiga per internetą: <http://www.mslevin.iitp.ru/CO.HTM>
6. Ghoch D., Neighborhood search heuristics for the uncapacitated facility location problem, *European Journal of Operational Research*, p 150–162, 2003.
7. Home page J E Beasley [interaktyvus, žiūrėta 2013 12]. Prieiga per internetą: < https://files.nyu.edu/jeb21/public/jeb/orlib/uncapinfo.html >
8. Jain K. Mahdian M. And ir kiti, Greedy Facility Location Algorithms Analyzed Using Dual Fitting with Factor-Revealing LP, *Journal of the ACM, Vol. 50, No. 6*, p. 795–824, 2003.
9. Karla Hoffman, Manfred Padberg. Combinatorial and Integer Optimization. George Mason University. Prieiga per internetą: <http://iris.gmu.edu/~khoffman/papers/newcomb1.html>
10. Kirkpatrick S., C. D. Gelatt ir M. P. Vecchi, Optimization by Simulated Annealing, *Science, New Series, Vol. 220, No. 4598*., p. 671–680, 1983.
11. Lagrangian Relaxation, *Chapter 12*, pp 195–208.
12. Li X. ans Gar-On Yeh A., Integration og genetic algorithms and GIS for optimas location search, *Internacional Journal of Geographical information systems, vol 19*, p 581–601, 2005
13. Listopadskis N., Kombinatorinio optimizavimo uždaviniai ir jų sprendimo algoritmai, [interaktyvus, žiūrėta 2014 03] prieiga per internetą: <http://www.fmf.lt/ft/studiju-programos/taikomoji-matematika/S-18508/straipsnis/Kombinatorinio-optimizavimo-uzdaviniai-ir-ju-sprendimo-algoritmai?p=1>

14. Mahdian M., Markakis E., Saberi A. ir Vazirani V., A Greedy Facility Location Algorithm Analyzed using Dual Fitting, USA, p 19. 2008.
15. Matematinis modeliavimas. Tiesinis programavimas. Aleksandro Stulginskio universitetas. [žiūrėta 2014 03] prieiga per internetą: <http://www.asu.lt/nm/failai/Matematinis_modeliavimas/25.htm>
16. Maric M., An efficient genetic algorithm for solving the multi-level uncapacitated facility location problem, *Computing and Informatics, Vol. 29,* p 183–201, 2010.
17. Mladenonic N., Labbe M. ir Hansen P., Solving the p-Center Problem with Tabu Search and Variable Neighborhood Search, *Networks, vol. 42(1),* p 48–64, 2003.
18. Murray A., Advances in location modeling: GIS linnkages and contributions, *Journal of Geographical systems, vol 12,* p. 335–354, 2010.
19. Nickel S., Facility Location and Strategic Supply Chain Management, *Chair of Business Administration Operations Research and Logistics,* p 119, 2009.
20. Stephen J. Wright, Optimization. Numerical Optimization Article free pass. the University of Wisconsin.. Prieiga per internetą: http://www.britannica.com/EBchecked/topic/430575/optimization
21. Tohyama H.,Ida K. and Matsueda J., A Genetic Algorithm for the Uncapacitated Facility Location Problem, *Electronics and Communications in Japan, Vol. 94, No. 5,* 2011.
22. Vlachos A., Ant colony system solving capacitaded location-allocation prblem on line, University of Piraeus, Greece, p 81–96, 2006.
23. Wikipedia The Free Encyclopedia [interaktyvus, žiūrėta 2014 01]. Prieiga per internetą: < http://en.wikipedia.org/wiki/Brute-force_search>
24. Wikipedia The Free Encyclopedia [interaktyvus, žiūrėta 2014 04]. Prieiga per internetą: < http://en.wikipedia.org/wiki/Greedy_algorithm >
25. Wikipedia The Free Encyclopedia [interaktyvus, žiūrėta 2013 04]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Combinatorial_optimization>
26. Wikipedia The Free Encyclopedia [interaktyvus, žiūrėta 2014 05]. Prieiga per internetą: <http://en.wikipedia.org/wiki/Simulated_annealing>
27. YunWua L., Zhanga X. S. ir Zhangb J. L., Capacitatedf acility location problem with general setup cost, *Computers & Operations Research 33,* p 1226–1241, 2006.

PRIEDAI

A PRIEDAS. LENTELĖS

A1 lentelė.

Brutalios jėgos vykdymo laiko priklausomybė nuo parametrų *n ir m*.

Įrenginiai (n)	Laikas (s)	Klientai (m)	Laikas (s)
15	2,7865	100	2,8906
16	6,1875	200	3,4375
17	12,9688	300	4,1563
18	27,7813	400	4,7813
19	64,8906	500	5,5781
20	131,4531	600	6,2031
21	262,9062	700	7,0313
22	525,8124	800	7,6875
23	1104,206	900	8,6563
24	2318,833	1000	9,6563
25	4869,549		

A2 lentelė.

Godžiojo metodo vykdymo laiko priklausomybė nuo parametrų *n ir m*.

Kintamojo reikšmė	Įrenginiai (nx100)	Klientai (100xm)
100	0,0036	0,0035
200	0,0062	0,0099
300	0,0110	0,0158
400	0,0127	0,0244
500	0,0167	0,0340
600	0,0180	0,0419
700	0,0206	0,0589
800	0,0248	0,0654
900	0,0289	0,0758
1000	0,0291	0,0909

A3 lentelė.

Apkeitimo metodo vykdymo laiko priklausomybė nuo parametrų *n ir m*.

Kintamojo reikšmė	Įrenginiai (nx100)	Klientai (100xm)
100	0,1635	0,1003
200	0,4141	0,1759
300	2,0240	0,1905
400	2,8022	1,0297
500	2,2118	1,3333
600	1,6599	1,8295
700	2,2146	2,0435
800	8,4070	0,9760
900	8,7364	3,1969
1000	7,7308	1,5456

A4 lentelė.

Modeliuojamo atkaitinimo metodo vykdymo laiko priklausomybė nuo parametrų *n ir m*.

Kintamojo reikšmė	Įrenginiai (nx100)	Klientai (100xm)
100	16,3773	17,6126
200	58,2969	30,4150
300	125,9479	45,2538
400	199,3955	62,3727
500	320,5839	82,8013
600	397,6163	102,2684
700	533,3152	116,2859
800	635,8738	134,2789
900	698,2188	158,8596
1000	755,5074	168,8894

A5 lentelė.

Modeliuojamo atkaitinimo metodo rezultatai. II eksperimentas.

Pava-dinimas	Paklaida (%)			CPU laikas (s)
	minimali	vidutinė	maksimali	vidurkis
Cap71	0	0,163554	0,431357	1,058
Cap72	0	0,508994	0,83083	1,069
Cap73	0,016496	0,410722	0,98387	1,022
Cap74	0	0,971619	2,185028	1,012
Cap101	0,558538	1,224723	1,903333	1,569
Cap102	1,298696	2,256791	3,323198	1,584
Cap103	1,85383	3,271685	4,927627	1,620
Cap104	0,607705	5,035921	9,525257	1,593
Cap131	5,997506	9,513557	13,2964	3,763
Cap132	6,232763	12,59994	20,38519	3,742
Cap133	5,555524	12,3209	22,74684	3,770
Cap134	7,654285	14,35202	20,58751	3,749

A6 lentelė.

Modeliuojamo atkaitinimo metodo rezultatai. III eksperimentas.

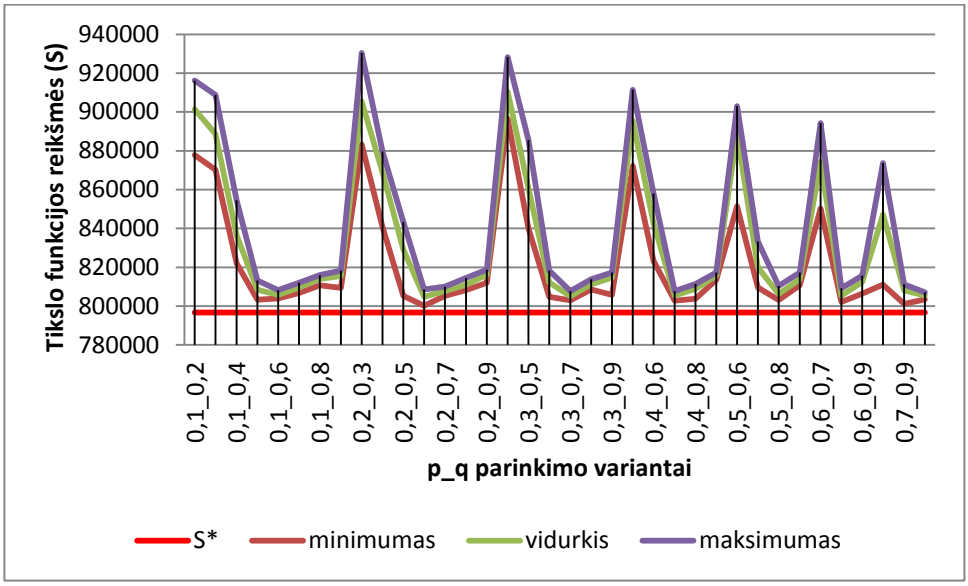
Pava-dinimas	Paklaida (%)			CPU laikas (s)
	minimali	vidutinė	maksimali	vidurkis
Cap71	0	0,081108	0,215182	1,912
Cap72	0	0,351073	0,654504	1,942
Cap73	0	0,214472	0,601291	1,932
Cap74	0	0,620705	1,560323	1,913
Cap101	0,783732	1,102431	1,49847	2,977
Cap102	0,975349	1,991581	3,192074	3,013
Cap103	0,635374	2,663834	4,061867	2,975
Cap104	1,553255	3,872068	6,683149	3,005
Cap131	6,199448	8,376775	10,54861	7,268
Cap132	4,926734	10,10135	14,22277	7,070
Cap133	4,149319	12,89918	21,27128	7,094
Cap134	4,208889	14,71129	24,06442	7,374

A7 lentelė.

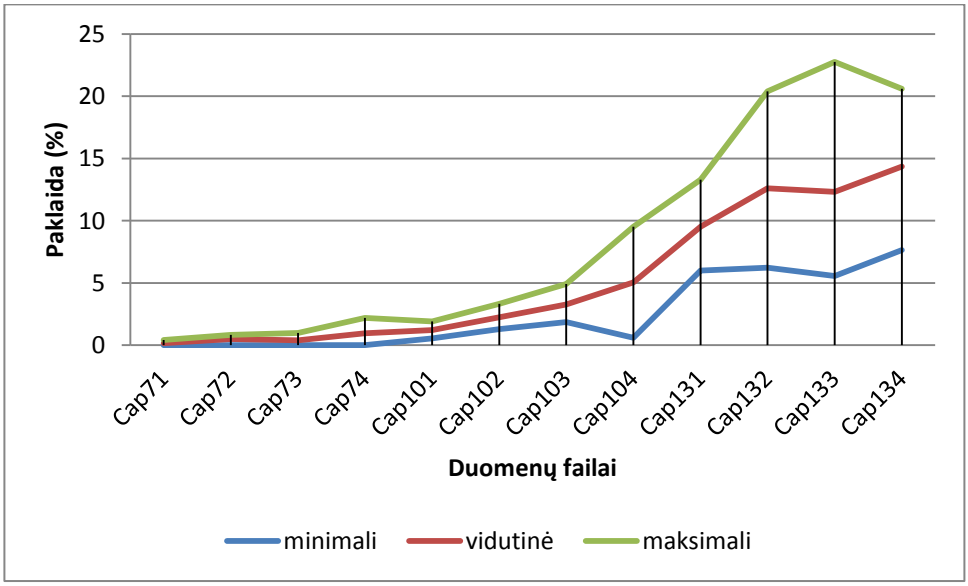
Modeliuojamo atkaitinimo metodo rezultatai. IV eksperimentas.

Pava-dinimas	Paklaida (%)			CPU laikas (s)
	minimali	vidutinė	maksimali	vidurkis
Cap71	0	0,02044	0,102202	4,480
Cap72	0	0,259091	0,664507	4,560
Cap73	0	0,19105	0,439608	4,555
Cap74	0	0,347621	0,924943	4,214
Cap101	0,432989	0,955087	1,293793	6,555
Cap102	0,934585	1,668816	2,560411	6,426
Cap103	0,715886	2,032413	3,732476	6,389
Cap104	1,416158	3,094137	5,652657	6,440
Cap131	6,46717	8,375907	10,48759	15,804
Cap132	5,484933	11,43331	17,14599	16,303
Cap133	2,374365	11,40195	17,39402	17,363
Cap134	5,005306	15,90484	24,78983	15,460

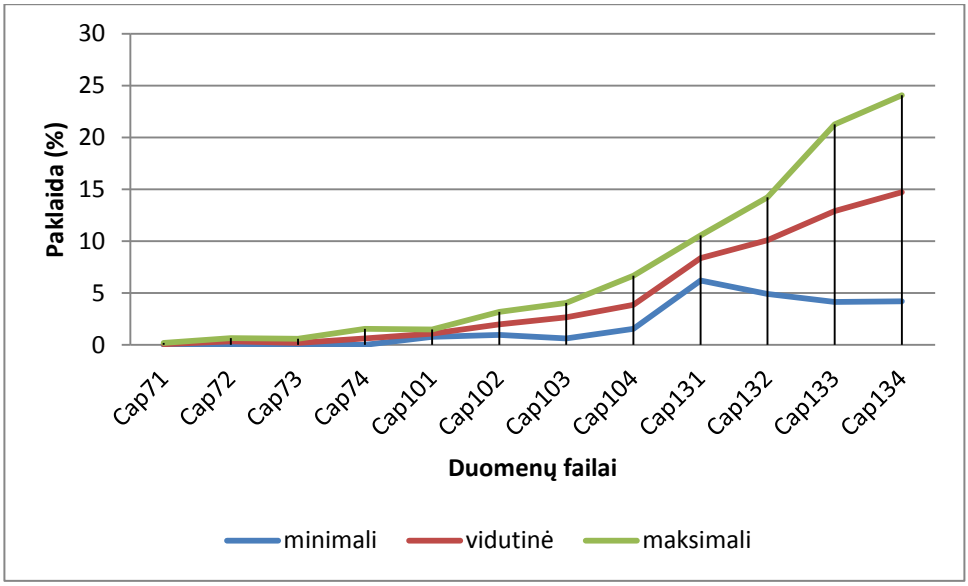
B PRIEDAS. PAVEIKSLAI



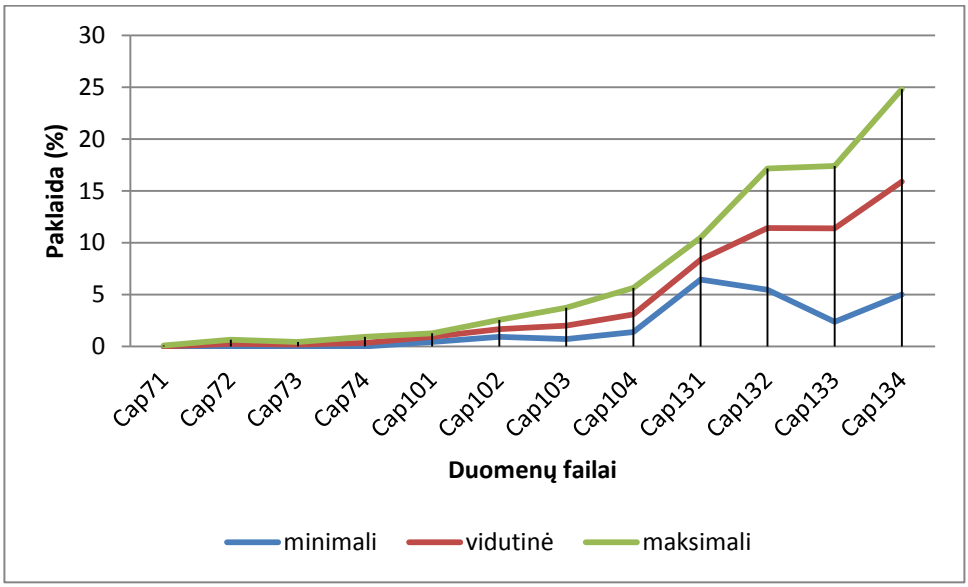
B1 pav. Tikslo funkcijos priklausomybė nuo parinktų *p_q* rinkinių (cap101).



B2 pav. II eksperimento paklaidų grafikas.



B3 pav. III eksperimento paklaidų grafikas.



B4 pav. IV eksperimento paklaidų grafikas.

C PRIEDAS. PROGRAMOS TEKSTAI

C1 Brutalios jėgos algoritmo vykdymo kodas.

```
function [x,y,e] = bjega(C,f)
if nargin < 2
    error('Būtina nurodyti transportavimo matricą C ir įrenginių kainų vektorių
f.');
end
t = cputime;
%nxm
m = length(f);
[n,m1] =  size(C);
if m~=m1
    error('Matricos C stulpelių skaičius turi būti lygus f ilgiui');
end
dFF2= ff2n(m);
F = zeros(1,2^m-1);
Y = 9999999999999999;
Z = 0;
for i = 2:2^m
    A = 0;
    G = 0;
    a = dFF2(i,:);
    for j = 1:m
        if a(j)==1
            if A==0
                A = C(:,j);
                G = f(j);
            else
                A = [A C(:,j)];
                G = G + f(j);
            end
        end
        F(i-1) = sum(min(A,[],2)) + G;
    end
    if F(i-1)<Y
        Y = F(i-1);
        Z = a;
    end
end
for i = 1:m
    if Z(i) ==1
        Z(i) = i;
    end
end
 x = min(F);
 y = Z;
 e = cputime-t;
end
```

C2 Godžiojo algoritmo vykdymo kodas.

```
function [x,y,e] = godus(C,f)
if nargin < 2
```

```
    error('Būtina nurodyti transportavimo matricą C ir įrenginių kainų vektorių
f.');
end
tic;
m = length(f);
[n,m1] =  size(C);
if m~=m1
    error('Matricos C stulpelių skaičius turi būti lygus f ilgiui');
end
X = zeros(1,m);
Y=0;
J = 1:m;
u = max(C,[],2);
F = sum(u);
while ~isempty(J(J>0))
    w = zeros(1,m);
    del = zeros(1,m);
    for j = 1:m
        if j==J(j)
            for i = 1:n
                d = max(0,u(i)-C(i,j));
                del(j) = del(j)+d;
            end
            w(j) = del(j) - f(j);
            if w(j)<=0
                J(j) = 0;
            end
        end
    end
    [W,I] = max(w);
    if W ~= 0;
        X(I) = I;
        if Y==0
            Y = C(:,I);
        else
            Y = [Y C(:,I)];
        end
        u = min(Y,[],2);
        J(I) = 0;
    else
        break
    end
    F = F - W;
end
x = F;
y = X;
e=toc;
end
```

C3 Apkeitimo algoritmo vykdymo kodas.

```
function [x,y,e] = apkeitimas(C,f,X)
if nargin < 3
    error('Būtina nurodyti transportavimo matricą C, įrenginių kainų vektorių f ir
pradinį įrenginių rinkinį X.');
end
tic;
```

```
m = length(f);
[n,m1] =  size(C);
m2 = length(X)
if m~=m1
    error('Matricos C stulpelių skaičius turi būti lygus f ilgiui');
elseif m~=m2
    error('f ir X dimensijos nelygios');
end
J = 1:m;
J1 = J;
Y = 0;
k = 0;
F = 0;
for j = 1:m
    if X(j)==j
        J1(j)=0;
        k = k +1;
        K(k) = j;
        if Y==0
            Y = C(:,j);
        else
            Y = [Y C(:,j)];
        end
    end
end
a=0;
b = 0;
u0 = min(Y,[],2);
l=1;
j=1;
while l < k +1
    w = zeros(1,m);
    del = zeros(1,m);
    Z = Y;
    j = 1;
    while j<m+1
        if j == J1(j)
            Z(:,l)=C(:,j);
            u = min(Z,[],2);
            del(j)= sum(u0 - u);
            w(j)= del(j) + f(K(l)) - f(j);
            if w(j)>0
                X(K(l))=0;
                J1(K(l))=K(l);
                K(l)=j;
                X(j) = j;
                J1(j) = 0;
                Y=Z;
                u0 = min(Y,[],2);
                a=1;
            end
        end
        if a==1
            b=1;
            a=0;
            break
        else
            j=j+1;
        end
```

```
    end
    if b==1;
        l = 1;
        b=0;
    else
        l = l+1;
    end
end
for i = 1:k
   F = F + f(K(i));
end
b = sum(u0);
G = F + b;
x = G;
y = X;
e=toc;
end
```

C4 Modeliuojamo atkaitinimo algoritmo vykdymo kodas

```
function [x,z,e] = MA(C,f,y,T0,Ta,dT,imax,p,q)
if nargin < 2
    error('Būtina nurodyti transportavimo matricą C ir įrenginių kainų vektorių
f');
elseif nargin ==2
    y1=length(f);
    y=zeros(1,y1);
    y(1)=1;
    T0=100;
    Ta=0.001;
    dT=0.9;
    imax=100;
    p=0.1;
    q=0.6;
elseif nargin < 9
    error('Pateikaimi duomanyes tokia tvarka: C,f,y,T0,Ta,dT,imax,p,q');
end
tic;
m = length(f);
[n,m1] =  size(C);
m2 = length(y)
if m~=m1
    error('Matricos C stulpelių skaičius turi būti lygus f ilgiui');
elseif m~=m2
    error('f ir X dimensijos nelygios');
end
if T0<0||Ta<0||dT<0||imax<0||p<0||q<0
    error('Visi paramertai turi būti teigiami');
elseif T0<Ta
    error('Turi būti T0 > Ta');
elseif dT>=1
    error('Turi būti dT < 1');
elseif p > q || p >1 ||q>1
    error('Turi būti 0 < p < q < 1');
end
Y = y;
a = y;
```

```
ss = tikslas(Y,C,f);
T = T0;
S = ss;
XX(1)=ss;
xx=2;
while T > Ta
    for i = 1:imax
       Y  = kaimynas(Y,p,q);
       sn = tikslas(Y,C,f);
       L = sn - ss;
       if L < 0
           ss = sn;
       else
           r = rand(1);
           A = exp(-L/T);
           if r<A
               ss = sn;
           end
       end
       if S> ss
           S = ss;
           a = Y;
       end
    end
    XX(xx)=ss;
    xx=xx+1;
    T = T * dT;
end
x = S;
z = a;
e = toc;
end
```

C5 Kaimyninių rinkinių paieškos algoritmo vykdymo kodas

```
function x = kaimynas(y,p,q)
X = y;
k = 0;
l = 0;
m = length(y);
L = zeros(1,m);
K = zeros(1,m);
for i = 1:m
    if y(i)>0
        k = k+1;
        K(k)=i;
    else
        l=l+1;
        L(l)=i;
    end
end
r = rand(1);
if (k == 1 && r<q && l>0)||(k>1 && r<p && l>0)
    r1 = randi(k);
    r2 = randi(l);
    X(K(r1)) = 0;
    X(L(r2)) = L(r2);
```

```
    x = X;
    z = 0;
    return
elseif(k == 1 && r>q && l>0)||(r>p && r<q && k<m && l>0)
    r2 = randi(l);
    X(L(r2)) = L(r2);
    x = X;
    z = 1;
    return
elseif k == m||(k>1 && r>q)
    r1 = randi(k);
    X(K(r1)) = 0;
    x = X;
    z = 2;
    return
end
end
```