Kauno technologijos universitetas

Informatikos fakultetas

Inžinerinis projektas

Duomenų lygiagretumo priemonių taikymas

Aistis Jakutonis

Studentas

Barisas Dominykas
Vasiljevas Mindaugas

Dėstytojai

Kaunas, 2025

Turinys

1. Užduotis

Išlygiagretinti skaitinių metodų antrojo namų darbo optimizavimo uždavinį:

Uždavinys 1.
Pagal pateiktą uždavinio sąlygą (6 lentelė) sudarykite tikslo funkciją ir išspręskite jį vienu iš gradientinių metodų (gradientiniu, greičiausio nusileidimo). Gautą taškų konfigūraciją pavaizduokite programoje, skirtingais ženklais pavaizduokite duotus ir pridėtus (jei sąlygoje tokių yra) taškus. Ataskaitoje pateikite pradinę ir gautą taškų konfigūracijas, taikytos tikslo funkcijos aprašymą, taikyto metodo pavadinimą ir parametrus, iteracijų skaičių, iteracijų pabaigos sąlygas ir tikslo funkcijos priklausomybės nuo iteracijų skaičiaus grafiką.

1 pav. Užduoties aprašymas

*** rekomenduojama $n \leq 20, m \leq 20$**

2 pav. Užduoties rekomendacija

Miestas išsidėstęs kvadrate, kurio koordinatės ($-10 \leq \boldsymbol{x} \leq 10,\ -10 \leq \boldsymbol{y} \leq 10$). Mieste yra **n** ($\boldsymbol{n \geq 3}$) vieno tinklo parduotuvių, kurių koordinatės yra žinomos (*Koordinatės gali būti generuojamos atsitiktinai, negali būti kelios parduotuvės toje pačioje vietoje*). Planuojama pastatyti dar **m** ($\boldsymbol{m \geq 3}$) šio tinklo parduotuvių. Parduotuvės pastatymo kaina (vietos netinkamumas) vertinama pagal atstumus iki kitų parduotuvių ir poziciją (koordinates). Reikia parinkti naujų parduotuvių vietas (koordinates) taip, kad parduotuvių pastatymo kainų suma būtų kuo mažesnė (naujos parduotuvės gali būti statomos ir už miesto ribos).

Atstumo tarp dviejų parduotuvių, kurių koordinatės $(\boldsymbol{x_1}, \boldsymbol{y_1})$ ir $(\boldsymbol{x_2}, \boldsymbol{y_2})$, kaina apskaičiuojama pagal formulę:

$$C(x_1, y_1, x_2, y_2) = \exp(-0.3 \cdot ((x_1 - x_2)^2 + (y_1 - y_2)^2))$$

Parduotuvės, kurios koordinatės $(\boldsymbol{x_1}, \boldsymbol{y_1})$, vietos kaina apskaičiuojama pagal formulę:

$$C^P(x_1, y_1) = \frac{x_1^4 + y_1^4}{1000} + \frac{\sin(x_1) + \cos(y_1)}{5} + 0.4$$

3 pav. Užduotis

Užduotis turi būti duomenų lygiagretumo užduotis – visiems duomenų rinkinio elementams taikoma ta pati operacija ir gaunamas rezultatas.

Galimybė lengvai keisti, kuris duomenų rinkinys apdorojamas ir kiek gijų ar procesų naudojama.

Naudojama priemonė: Python su joblib (python multiprocessing.pool)

Tikslas: gauti kuo geresnį pagreitėjimą kuo mažiau komplikuojant programos kodą.

2. Užduoties analizė ir sprendimo metodas

2.1. Sprendžiama problema

Šiame darbe sprendžiama optimizavimo problema, susijusi su naujų parduotuvių išdėstymu mieste, kuriame jau veikia tam tikras skaičius esamų parduotuvių. Užduoties tikslas – parinkti naujų parduotuvių koordinates taip, kad būtų minimizuota bendra tikslo funkcija, apimanti dvi sudedamąsias dalis: vietos (statybos) kainą kiekvienai naujai parduotuvei ir porines „kaimynystės“ kainas tarp parduotuvių. Vietos kaina modeliuojama kaip nelinijinė funkcija nuo parduotuvės koordinačių, o porinė kaina apibrėžiama kaip eksponentinė funkcija nuo atstumo tarp dviejų parduotuvių – tiek tarp naujų ir esamų, tiek tarp pačių naujų. Taip gaunama sudėtinga, nelinijinė, daugiamatė optimizavimo užduotis su keliais lokaliais ekstremumais, kuri natūraliai tinkama gradientiniams metodams ir duomenų lygiagretumo tyrimui.

2.2. Sprendimo būdas

Užduočiai spręsti pasirinktas gradientinis metodas su fiksuotu žingsniu. Pirmiausia analitiškai išvestos tikslo funkcijos dalinės išvestinės naujų parduotuvių koordinatėms, iš kurių sudaromas gradiento vektorius. Kiekvienoje iteracijoje apskaičiuojamas tikslo funkcijos gradientas esamame taške ir atliekamas atnaujinimas pagal formulę $x_{k+1} = x_k - \alpha \nabla F(x_k)$, kur α – pastovus žingsnio dydis. Skaičiavimai realizuoti naudojant „Python“ kalbą ir „NumPy“ biblioteką, kad būtų efektyviai apdorojamos vektorinės ir matricinės operacijos. Teisingumui patikrinti papildomai apskaičiuojamas skaitinis gradientas (baigtinių skirtumų metodu) ir palyginamas su analitiniu bei lygiagrečiu gradientais – jų normų ir didžiausio komponentų skirtumo reikšmės patvirtina, kad implementuotos formulės yra korektiškos.

2.3. Lygiagretumo priemonės pritaikymas

Duomenų lygiagretumas pritaikytas gradiento skaičiavimui naujoms parduotuvėms, naudojant „joblib“ bibliotekos funkciją *Parallel* su *delayed*. Kiekvienai naujai parduotuvei j atskirai apskaičiuojama jos gradiento dalis *grad_for_j(j)*, kuri apima vietos kainos išvestinę, porinių kainų su esamomis parduotuvėmis ir porinių kainų su kitomis naujomis parduotuvėmis indėlius. Šios vienodos operacijos kiekvienam duomenų rinkinio elementui (kiekvienai naujai parduotuvei) atliekamos lygiagrečiai atskiruose gijose, o gautas dalines išvestines programa sujungia į bendrą gradiento vektorių. Lygiagretumo stipris valdomas parametru *n_jobs* gradiento metodo funkcijoje: pakeitus šią reikšmę galima lengvai pasirinkti, kiek gijų bus naudojama skaičiavimams, ir tokiu būdu tirti vykdymo laiko priklausomybę nuo duomenų rinkinio dydžio ir naudojamų gijų skaičiaus.

3. Testavimas ir programos vykdymo instrukcija

3.1. Testavimo rezultatai

Automatinis gradiento tikrinimo testas:

Programos pradžioje automatiškai palyginami trys gradiento variantai pradiniame taške x_0: nuoseklus analitinis (gradient_seq), lygiagretus (gradient_parallel) ir skaitinis gradientas (numeric_grad). Išvestyje matyti, kad:

- analitinio ir lygiagretaus gradiento normos sutampa iki mašininio tikslumo *(||g_seq||₂ = 3.853616, ||g_par||₂ = 3.853616*);
- skaitinio gradiento norma praktiškai tokia pati (||g_num||₂ = 3.853617);
- maksimalus skirtumas tarp analitinio ir lygiagretaus gradiento komponentų yra ~$2.78 \cdot 10^{-17}$, kas iš esmės yra apvalinimo paklaida;
- maksimalus skirtumas tarp analitinio ir skaitinio gradiento yra ~$6.33 \cdot 10^{-7}$, t.y. daug mažesnis už pasirinktą slenkstį 10^{-5}.

Remiantis šiuo testu galima teigti, kad:

1. analitinės gradiento formulės įgyvendintos teisingai;
2. lygiagreti gradiento versija (joblib.Parallel) duoda tą patį rezultatą kaip ir nuosekli;
3. tiek nuoseklus, tiek lygiagretus gradientai suderinami su skaitiniu baigtinių skirtumų įvertinimu.

Be to, gradientinio metodo eigos metu F reikšmė sumažėjo nuo 13.116381 iki 0.564584 (pagerėjimas 12.551797), kas rodo, kad gradientas naudojamas teisinga kryptimi ir funkcija sėkmingai minimizuojama.

```
=== Gradiento tikrinimas pradiniame taške x0 ===
||g_seq||_2  = 3.853616e+00
||g_par||_2  = 3.853616e+00
||g_num||_2  = 3.853617e+00
max|g_seq - g_par|  = 2.775558e-17
max|g_seq - g_num|  = 6.334379e-07
max|g_par - g_num|  = 6.334379e-07
Jei šie maksimalūs skirtumai yra maži (pvz. < 1e-5), gradientas įgyvendintas teisingai.
```

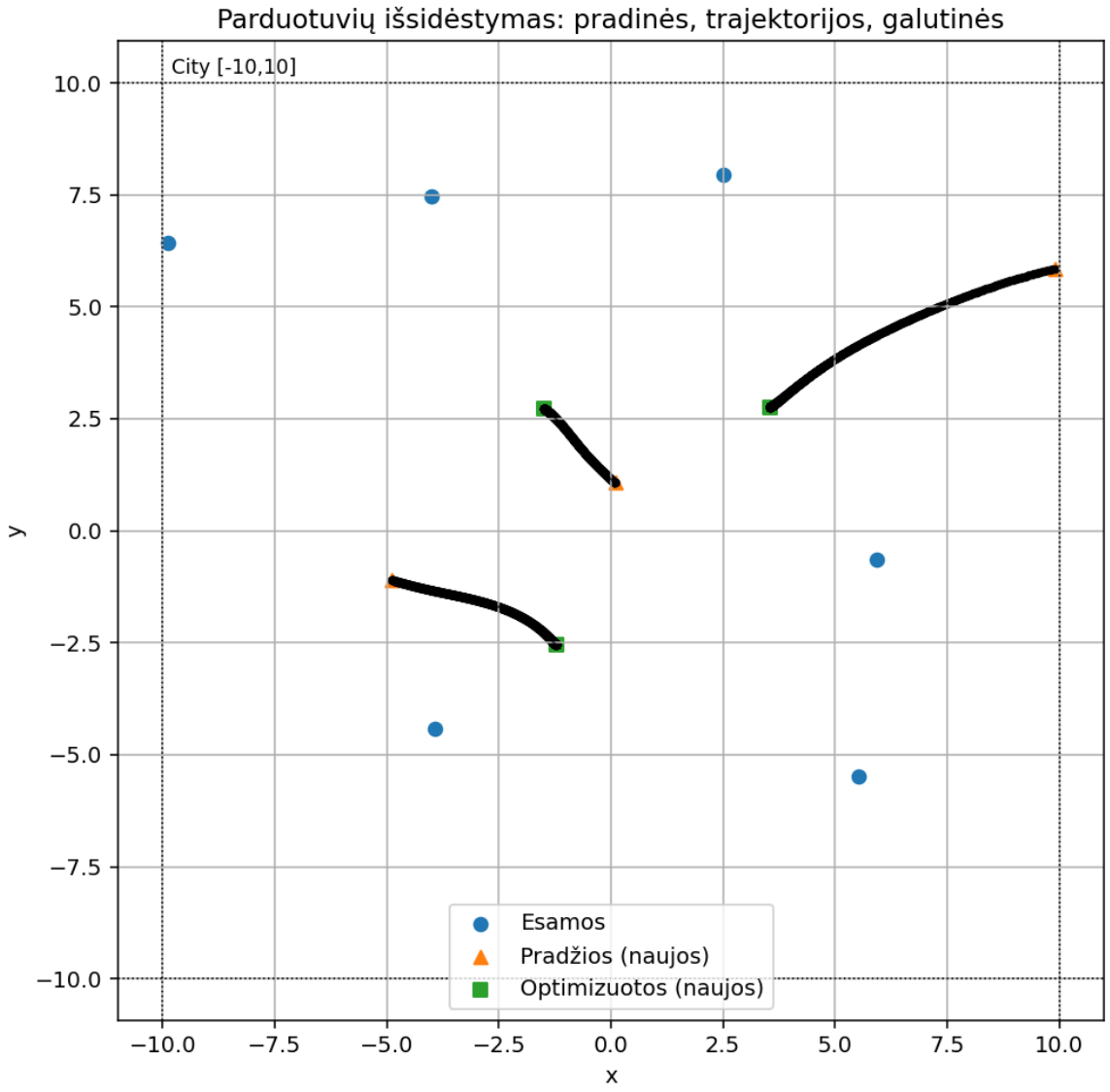
4 pav. Automatinių testų rezultatai

```
n (esamos) = 6, m (naujos) = 3
Parametrai: step=0.01, tol=1e-6, max_iter=3000, n_jobs=4
Iteracijų sk.: 3000
Sustabdymo priežastis: max_iter
F: pradžia=13.116381, pabaiga=0.564584 (pagerėjimas 12.551797)
```

5 pav. Automatinių testų pagerėjimo rezultatai

Rankiniai testai:

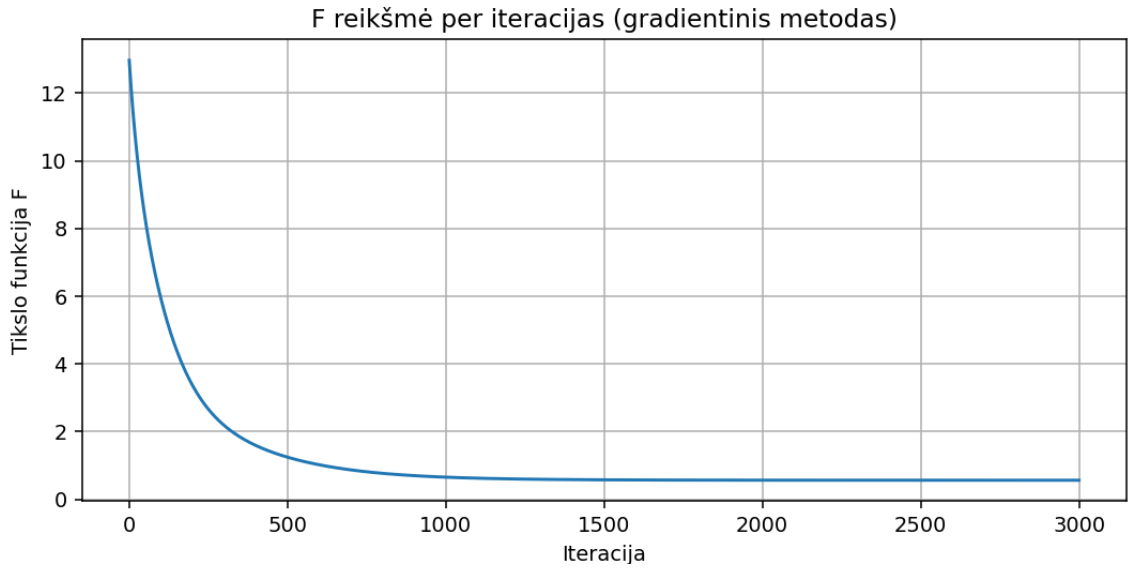
Grafike pavaizduotas esamų ir naujų parduotuvių išsidėstymas plokštumoje bei naujų parduotuvių judėjimo trajektorijos optimizavimo metu. Mėlynais taškais pažymėtos esamos parduotuvės, oranžiniais trikampiais – pradinės naujų parduotuvių vietos, žaliais kvadratais – optimizuotos naujų parduotuvių koordinatės. Juodos linijos su taškais rodo, kaip kiekviena nauja parduotuvė juda per iteracijas. Iš grafiko matyti, kad naujos parduotuvės palaipsniui artėja prie „geresnių“ vietų ir jų trajektorijos yra tolygios, be staigių šuolių. Tai atitinka gradientinio metodo logiką – taškai juda kryptimi, kuria tikslo funkcija F labiausiai mažėja.



6 pav. Parduotuvių koordinatės bei trajektorijos grafikas

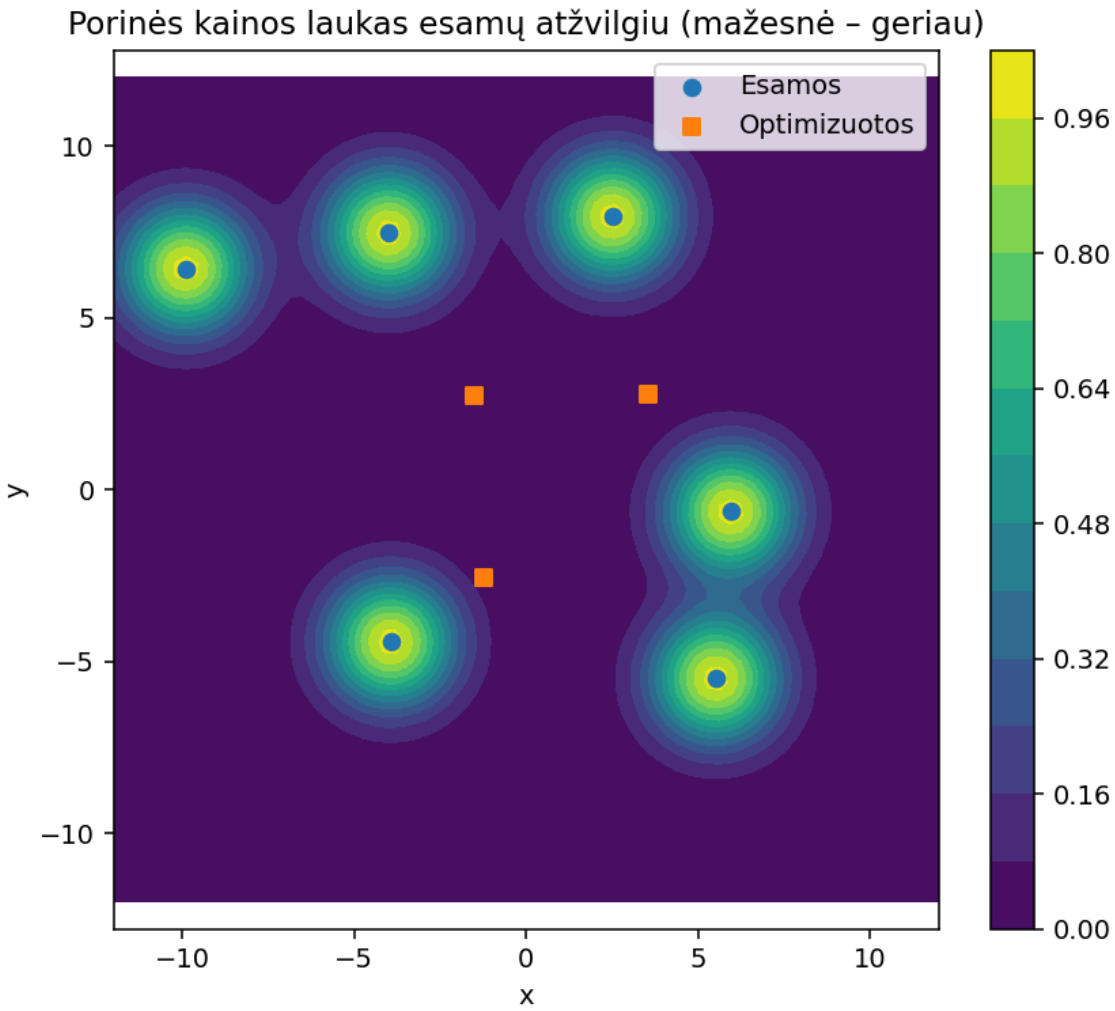
Grafike pavaizduota tikslo funkcijos F priklausomybė nuo iteracijos numerio naudojant gradientinį metodą. Matyti, kad pradžioje F ≈ 13.1, o didėjant iteracijų skaičiui funkcijos reikšmė greitai mažėja ir galiausiai nusistovi ties maždaug 0.56. Kreivė yra monotoninė (F nepadidėja) ir artėja prie tam tikro stabilaus lygio, kas rodo, jog algoritmas sėkmingai randa lokalią minimalią tikslo

funkcijos reikšmę. Šis grafikas vizualiai patvirtina, kad gradientas skaičiuojamas teisinga kryptimi ir pasirinktas žingsnis step=0.01 yra pakankamai mažas stabiliai konvergencijai.



7 pav. Tikslo funkcijos priklausomybės nuo iteracijų skaičiaus grafikas

Grafike pavaizduotas porinės kainos laukas esamų parduotuvių atžvilgiu. Spalvomis (heatmap) parodyta porinės funkcijos $\sum_i \text{Cost\_pair}(p, q_i)$ reikšmė skirtinguose plokštumos taškuose – tamsesnės spalvos atitinka mažesnę kainą (geresnes vietas). Mėlynais taškais čia vėl pažymėtos esamos parduotuvės, o oranžiniais kvadratais – optimizuotos naujų parduotuvių vietos. Iš šio grafiko matyti, kad naujos parduotuvės išsidėsto tokiose srityse, kur porinė kaina yra mažesnė, bet jos nėra „prilipusios“ prie esamų parduotuvių ar viena kitos. Tai parodo, kad algoritmas suranda kompromisą tarp vietos kainos ir porinių sąveikų kainos, kaip ir numatyta tikslo funkcijos apibrėžime.



8 pav. Kainų šilumos žemėlapis

3.2. Instrukcija

3.2.1. Reikalavimai

Programai paleisti reikia:

- Python 3 (rekomenduojama 3.9 ar naujesnė versija);
- šių bibliotekų:
  - `numpy`
  - `matplotlib`
  - `pandas`
  - `joblib`

Jei jų nėra, jas galima įdiegti komandinėje eilutėje:

pip install numpy matplotlib pandas joblib

3.2.2. Programos paleidimas iš komandinės eilutės

Išsaugoti visą programos kodą, pavyzdžiui, faile **`main.py`**.

Atidaryti terminalą / „Command Prompt“ tame kataloge, kuriame yra failas `main.py`.

Paleisti komandą:

python main.py

Paleidus programą:

- pirmiausia konsolėje bus atspausdintas **gradiento tikrinimo rezultatas** (automatinis testas);
- po to – **optimizacijos santrauka** ir lentelės su esamų, pradinių ir optimizuotų parduotuvių koordinatėmis;
- tuomet programa parodys pranešimą `Paspausk Enter, kad sugeneruotum grafikus...` – reikia paspausti Enter;
- po to bus atidaryti trys grafikai:
    - parduotuvių išsidėstymas ir trajektorijos,
    - tikslo funkcijos F reikšmė per iteracijas,
    - porinės kainos lauko žemėlapis.

3.2.3. Programos paleidimas PyCharm aplinkoje

Atidaryti **PyCharm** ir sukurti naują projektą (arba naudoti esamą).

Projekte sukurti naują Python failą, pvz. **`main.py`**, ir į jį įklijuoti programos kodą.

Įsitikinti, kad projekto interpreteris turi įdiegtas `numpy`, `matplotlib`, `pandas`, `joblib` bibliotekas (jei reikia, jas galima įdiegti per PyCharm „Python Packages“ arba projekto terminalą).

Viršuje pasirinkti `main.py` kaip „Run Configuration“ ir paspausti **Run**.

Programos veikimas toks pats kaip ir paleidus iš terminalo:

- „Run“ lange matyti gradiento tikrinimo ir optimizacijos rezultatai,
- kai pasirodo tekstas `Paspausk Enter, kad sugeneruotum grafikus...`, reikia paspausti Enter,
- grafikai atsiranda atskiruose languose.

3.2.4. Parametrų keitimas

Norint atlikti papildomus eksperimentus, galima keisti:

- **`n`** **ir** **`m`** – esamų ir naujų parduotuvių skaičių;

- **`step`** – gradientinio metodo žingsnio dydį;
- **`n_jobs`** – lygiagrečiai naudojamų gijų skaičių.

Pakeitus šiuos parametrus ir vėl paleidus programą, galima stebėti, kaip kinta konvergencija, vykdymo laikas ir gauti grafikai.

4. Vykdymo laiko kitimo tyrimas

4.1. Pirmas tyrimas

Duomenų rinkiniai:

```
datasets = [
   ("S1", 6, 3),
   ("S2", 20, 10),
   ("S3", 50, 20),
   ("S4", 100, 30),
   ("S5", 100, 40),
   ("S6", 100, 50),
   ("S7", 200, 60),
   ("S8", 200, 70),
]
```

Rezultatų laikai atliekant uždavinį nuosekliai:

```
=== Vykdymo laiko matavimai ===
S1: n=   6, m=   3, vidutinis laikas ~ 0.338 s
S2: n=  20, m=  10, vidutinis laikas ~ 2.682 s
S3: n=  50, m=  20, vidutinis laikas ~ 11.431 s
S4: n= 100, m=  30, vidutinis laikas ~ 30.771 s
S5: n= 100, m=  40, vidutinis laikas ~ 44.125 s
S6: n= 100, m=  50, vidutinis laikas ~ 58.902 s
S7: n= 200, m=  60, vidutinis laikas ~ 121.336 s
S8: n= 200, m=  70, vidutinis laikas ~ 146.742 s
```

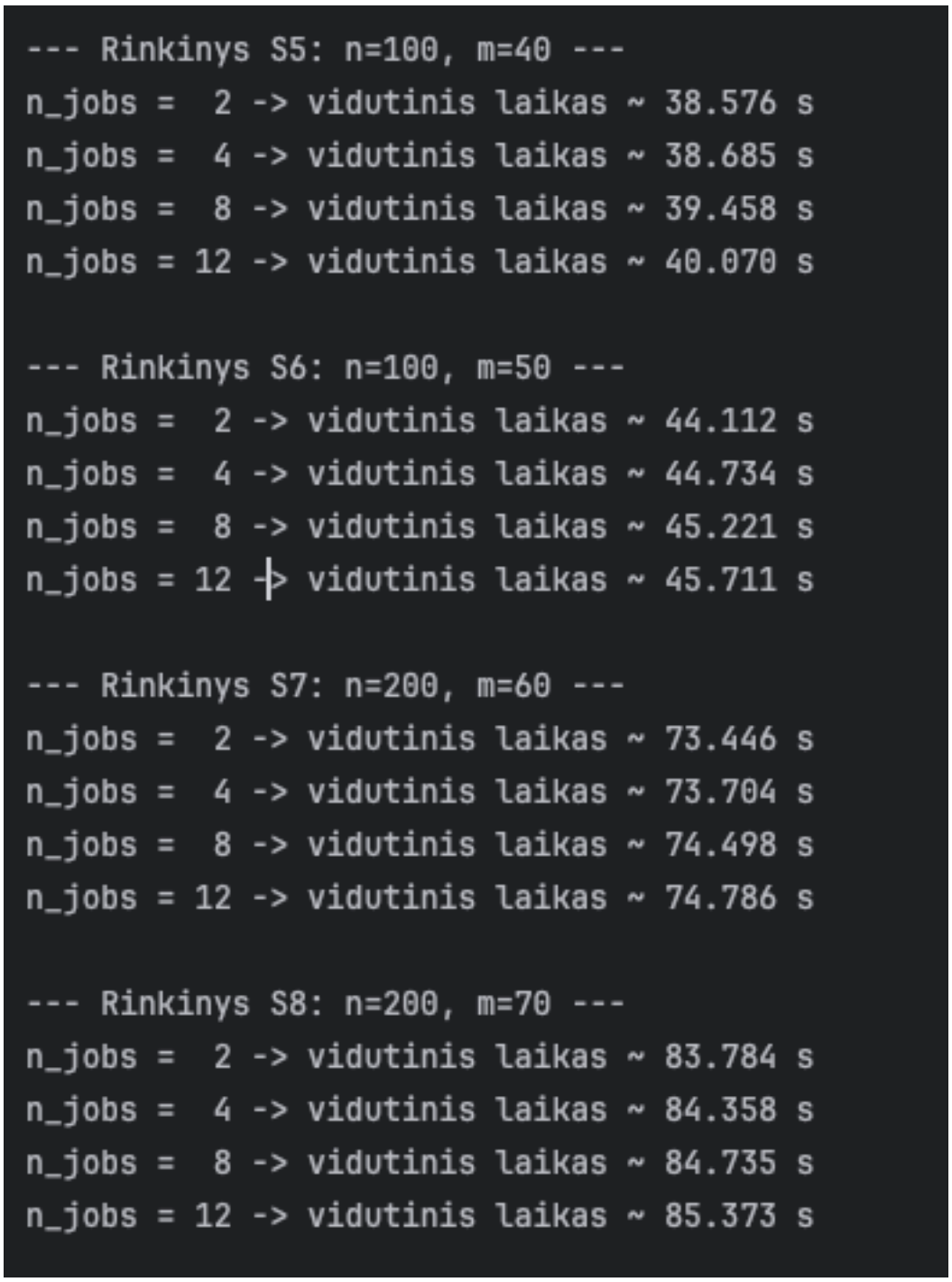
9 pav. Nuoseklaus vykdymo laiko matavimai

Pirmajame tyrime buvo tiriamas programos vykdymo laikas, kai optimizavimo uždavinys sprendžiamas nuosekliai, naudojant tik vieną giją (n_jobs = 1). Buvo sudaryti aštuoni skirtingos apimties duomenų rinkiniai – nuo labai mažo (S1, kur turiu 6 esamas ir 3 naujas parduotuves) iki didelio (S8, kur esamų ir naujų parduotuvių skaičius gerokai didesnis). Kiekvienam rinkiniui gradientinis metodas buvo paleidžiamas kelis kartus, matuojamas vykdymo laikas ir skaičiuojamas vidurkis. Rezultatai parodė, kad vykdymo laikas didėja augant duomenų rinkinio dydžiui: mažiausias rinkinys apdorojamas labai greitai (milisekundžių eilės), o didžiausiems rinkiniams laikas tampa žymiai ilgesnis. Tai atitinka teorinį sudėtingumą – kiekvienoje iteracijoje reikia suskaičiuoti porines sąveikas tarp naujų ir esamų parduotuvių, taip pat tarpusavio sąveikas tarp visų naujų parduotuvių, todėl skaičiavimų kiekis auga bent kvadratiškai nuo parduotuvių skaičiaus.

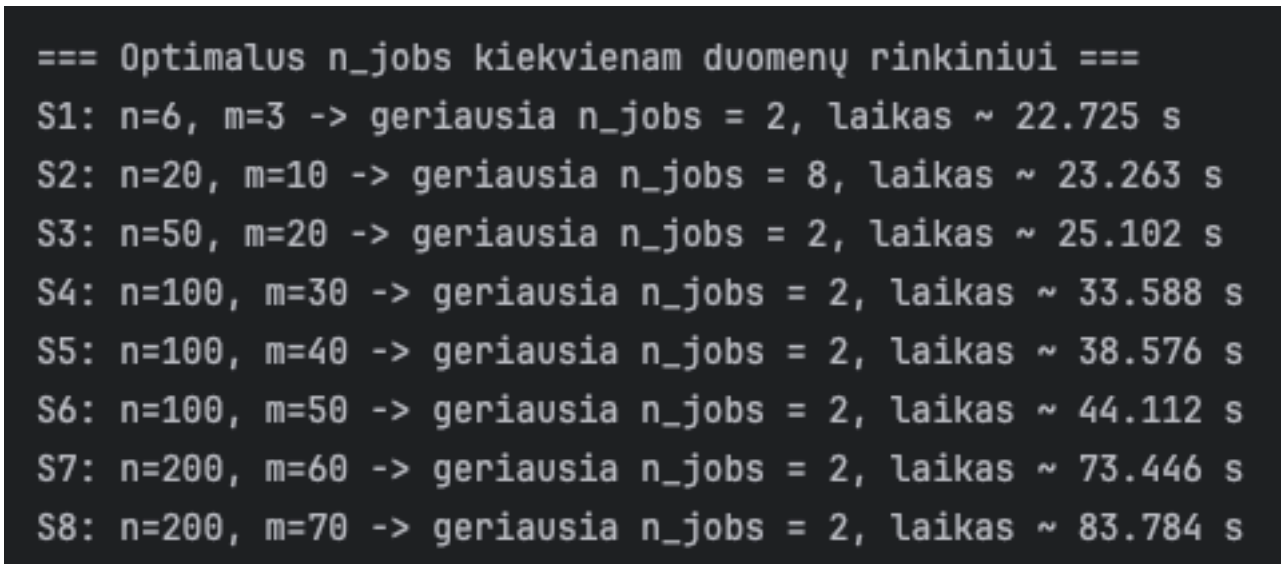
4.2. Antras tyrimas

```
--- Rinkinys S1: n=6, m=3 ---
n_jobs =  2 -> vidutinis laikas ~ 22.725 s
n_jobs =  4 -> vidutinis laikas ~ 23.181 s
n_jobs =  8 -> vidutinis laikas ~ 24.311 s
n_jobs = 12 -> vidutinis laikas ~ 25.050 s

--- Rinkinys S2: n=20, m=10 ---
n_jobs =  2 -> vidutinis laikas ~ 23.860 s
n_jobs =  4 -> vidutinis laikas ~ 23.722 s
n_jobs =  8 -> vidutinis laikas ~ 23.263 s
n_jobs = 12 -> vidutinis laikas ~ 23.532 s

--- Rinkinys S3: n=50, m=20 ---
n_jobs =  2 -> vidutinis laikas ~ 25.102 s
n_jobs =  4 -> vidutinis laikas ~ 25.274 s
n_jobs =  8 -> vidutinis laikas ~ 25.707 s
n_jobs = 12 -> vidutinis laikas ~ 26.042 s

--- Rinkinys S4: n=100, m=30 ---
n_jobs =  2 -> vidutinis laikas ~ 33.588 s
n_jobs =  4 -> vidutinis laikas ~ 33.795 s
n_jobs =  8 -> vidutinis laikas ~ 34.608 s
n_jobs = 12 -> vidutinis laikas ~ 35.059 s
```

10 pav. Pirmų keturių duomenų rinkinių lygiagretūs laiko matavimai



11 pav. Paskutinių keturių duomenų rinkinių lygiagretaus vykdymo laiko matavimai



12 pav. Optimalus gijų skaičius kiekvienam duomenų rinkiniui

Antrajame tyrime buvo analizuojama, kaip vykdymo laikas priklauso nuo naudojamų gijų skaičiaus. Tam pačiam duomenų rinkiniui gradientinis metodas buvo paleistas su skirtingomis n_jobs reikšmėmis (pvz., 2, 4, 8, 12), naudojant „joblib“ bibliotekos Parallel ir delayed mechanizmą su prefer="threads". Kiekvienu atveju buvo atliekami keli paleidimai ir skaičiuojamas vidutinis laikas. Nors teoriškai didesnis gijų skaičius turėtų sutrumpinti skaičiavimus, praktikoje matyti, kad vykdymo laikas tarp skirtingų n_jobs reikšmių kinta nedaug, o kai kuriais atvejais net šiek tiek padidėja. Taip yra dėl to, kad gradiento skaičiavimo užduotys yra gana smulkios, o „joblib“ lygiagretinimo valdymo kaštai (užduočių dalinimas, rezultatų surinkimas, gijų sinchronizavimas) sudaro reikšmingą bendro laiko dalį. Todėl didinant gijų skaičių lygiagrečiai atliekamo skaičiavimo nauda beveik kompensuojama papildomais valdymo kaštais.

4.3. Trečias tyrimas

Trečiajame tyrime buvo lyginami nuoseklaus ir lygiagretaus vykdymo laikai, siekiant įvertinti realų pagreitėjimą ir jo atitikimą Amdalo dėsniui. Nuoseklus variantas (n_jobs = 1) buvo laikomas atskaitos tašku, o lygiagretūs variantai su skirtingais n_jobs leido apskaičiuoti pagreitėjimą kaip T_nuoseklus / T_lygiagretus. Gautas pagreitėjimas pasirodė esantis nedidelis – kai kuriais atvejais lygiagretus sprendimas buvo tik nežymiai greitesnis arba netgi lėtesnis už nuoseklų. Tai rodo, kad nuosekliai vykdomos programos dalies (tikslo funkcijos, valdymo logikos, „joblib“ vidinio darbo) dalis yra gana didelė, o lygiagrečiai skaičiuojama tik dalis gradiento. Pagal Amdalo dėsnį, jei reikšminga kodo dalis negali būti lygiagrečiai vykdoma, maksimalus teorinis pagreitėjimas yra ribotas, kad ir kiek gijų būtų pridėta. Šiame darbe būtent taip ir gavosi: dėl didelio nuoseklių ir valdymo operacijų kiekio skaičiavimai nesugeba pilnai išnaudoti daugialypio procesoriaus galimybių, todėl optimalus gijų skaičius yra palyginti nedidelis, o labai didelis n_jobs nebeduoda naudos.

5. Išvados

Atlikus darbą buvo sukonstruota ir įgyvendinta optimizavimo užduotis, kurioje reikia parinkti naujų parduotuvių koordinates, minimizuojant sudėtinę tikslo funkciją, apimančią vietos kainą ir porines sąveikas tarp parduotuvių. Išanalizavus uždavinį buvo parinktas gradientinis metodas su fiksuotu žingsniu, o tikslo funkcijai išvestos analitinės dalinės išvestinės. Skaitinis gradiento skaičiavimas (baigtinių skirtumų metodu) patvirtino, kad analitinė ir lygiagreti gradiento versijos įgyvendintos teisingai, nes gauti rezultatai sutampa iki mašininio tikslumo ribų. Grafikai parodė, kad tikslo funkcija monotoniškai mažėja ir algoritmas nuveda naujų parduotuvių koordinates į tokias sritis, kuriose porinė kaina yra mažesnė, kartu išlaikant kompromisą tarp vietos ir sąveikos kainų.

Lygiagretumo eksperimentai parodė, kad nors „joblib“ leidžia gana paprastai pritaikyti duomenų lygiagretumą ir keisti gijų skaičių vienu parametru, realus pagreitėjimas yra ribotas. Dėl to, kad gradiento skaičiavimas išskaidomas į gana smulkias užduotis, o Parallel kvietimai atliekami daug kartų, dominuoja lygiagretinimo valdymo kaštai, o ne pats skaičiavimas. Tai gerai iliustruoja Amdalo dėsnis: esant dideliai nuosekliai kodo daliai, maksimalus pasiekiamas pagreitėjimas net ir su daug gijų yra nedidelis. Darbo metu buvo praktiškai įsitikinta, kad vien lygiagretaus vykdymo „įjungimas“ automatiškai negarantuoja didelio pagreitėjimo – būtina vertinti, kokia kodo dalis iš tikrųjų lygiagrečiai skaičiuojama ir kiek tai kainuoja valdymo prasme.