VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Dalelių spiečių algoritmas

Grupinio projekto darbo ataskaita

Atliko: Paulina Podgaiska, Adriana Širokytė

VU el. p.: paulina.podgaiska@mif.stud.vu.lt
adriana.sirokyte@mif.stud.vu.lt

Vertino: Vyr. M. Darbuot., Dr. (HP), Julius Žilinskas

Vilnius
2025

UŽDUOTIS

Grupinio projekto užduotis pateikta 1 lentelėje.

1lentelė. Grupinio projekto užduotis

Dalelių spiečių algoritmas
1. Realizuokite algoritmą savo pasirinktomis programavimo priemonėmis.
2. Pademonstruokite programas ir rezultatus dėstytojui.
3. Aprašykite darbo eigą ir rezultatus ataskaitoje, pateikite išvadas, prieduose pateikite programų kodus.

ATASKAITA

1. ĮVADAS

Šiuo metu vienos iš populiariausių spiečių elgsena grįstų skaičiavimo metodologijų yra Ant Colony Optimization (ACO) ir Dalelių spiečiaus optimizacija (PSO). Abu metodai priklauso prie spiečiaus intelekto (angl. Swarm Intelligence) teorijų, kurios modeliuoja sprendimų paiešką remiantis gyvūnų grupių elgsena gamtoje.

Dalelių spiečiaus optimizacijos (PSO) algoritmas buvo pristatytas James Kennedy ir Russell Eberhart dar 1995 metais. Šį metodą išrado imituodami paukščių pulkų ir žuvių būrių elgseną ieškant maisto. Esminis PSO principas – kolektyvinė patirtis padeda geriau optimizuoti paiešką.

Lyginant su evoliuciniais algoritmais, tokiais kaip genetiniai algoritmai, dalelių spiečiaus optimizavimo (PSO) metodas pasižymi paprastesne struktūra, nes nereikalauja taikyti sudėtingų genetinių operacijų, tokių kaip kryžminimas ar mutacija. Be to, PSO yra efektyvesnis, ypač kai sprendžiamos dinamiškai kintančios arba netiesinės optimizavimo problemos. Vienas iš šio algoritmo pranašumų yra atminties mechanizmas, kuris leidžia kiekvienai dalelei atskirai prisiminti geriausią iki šiol rastą sprendimą, todėl paieška tampa labiau tikslinga ir prisitaikanti prie situacijos.

PSO algoritmo veikimo principas grindžiamas dalelių populiacijos kūrimu, kur kiekviena dalelė reprezentuoja vieną galimą sprendinį paieškos erdvėje. Kiekviena jų turi poziciją ir greitį, kurie lemia jos judėjimo kryptį. Dalelės nuolat atnaujina savo buvimo vietą, remdamosi ne tik savo asmenine patirtimi, bet ir geriausiu visos spiečiaus iki šiol rastu sprendimu. Tokiu būdu, sprendimo paieška grindžiama tiek individualiu prisitaikymu, tiek kolektyvine sąveika – tarsi paukščių pulkas ar žuvų būrys, judantis bendru tikslu.

Dėl šių savybių PSO plačiai taikomas įvairiose srityse. Jis efektyviai naudojamas valdymo sistemose, mašininio mokymosi problemose, signalų apdorojime bei netiesinių ir dinaminių optimizavimo problemų sprendimui. Skirtingai nei išsami paieška, kuri reikalauja patikrinti visus taškus paieškos erdvėje, PSO leidžia greičiau pasiekti artimą optimalų sprendimą, nes efektyviai pasinaudoja lokaliais minimumo taškais ir dalinasi informacija tarp dalelių.

PSO matematinis modelis susideda iš dviejų formulių

Greičio atnaujinimas:

$$v_i = W v_i + c_1 r_1 \left(P_{geriausias,i} - x_i\right) + c_2 r_2 \left(g_{geriausias} - x_i\right)$$

Pozicijos atnaujinimas:

$$x_i = x_i + v_i$$

Kur:

v_i – i-osios dalelės greitis
x_i – i-osios dalelės pozicija
$P_{geriausias,i}$ – i-osios dalelės individualus geriausias taškas
$r_1 \; ir \; r_2$ – atsitiktiniai skaičiai
$c_1 \; ir \; c_2$ – pagreičio koeficientai
W – inercija

$g_{geriausias}$ – globalus geriausias

PSO algoritmo veikimo principas pavaizduotas 1 pav.

Input: *PUT*, target path
Output: a test case covering target path
1 Initialize each particle and its velocity V and location X;
2 **repeat**
3 **for** *each particle* i **do**
4 Calculate the fitness value f_i of particle i;
5 **if** $f_i < f_{pbest_i}$ **then**
6 $pbest_i = X_i$;
7 **end**
8 **if** $f_i < f_{gbest}$ **then**
9 $gbest = X_i$;
10 **end**
11 Update X_i and V_i according to formula (3) and (4);
12 **end**
13 **until** *t*=*MAX or* $f_{gbest} = 0$;
14 Return $gbest$;

1 pav. PSO algoritmo veikimo logika

2. DARBO EIGA

Kadangi dalelių spiečiaus optimizacija yra tinkama taikyti įvairiems optimizavimo uždaviniams, ypač tais atvejais, kai sprendinių erdvė yra didelė, o tikslų sprendinį rasti sudėtinga, šis metodas buvo pasirinktas spręsti realaus pasaulio įkvėptą uždavinį. Įsivaizduokime situaciją: planuojama statyti naują universitetą Vilniuje, tačiau norima, kad jis būtų strategiškai patogioje vietoje. Universitetas turi būti kuo arčiau studentų bendrabučių, kad studentams būtų patogu jį pasiekti, bet kartu – pakankamai nutolęs nuo pramogų zonų ir triukšmingų miesto vietų, kad sudarytų tinkamą aplinką mokymuisi.

2.1. Uždavinio formulavimas

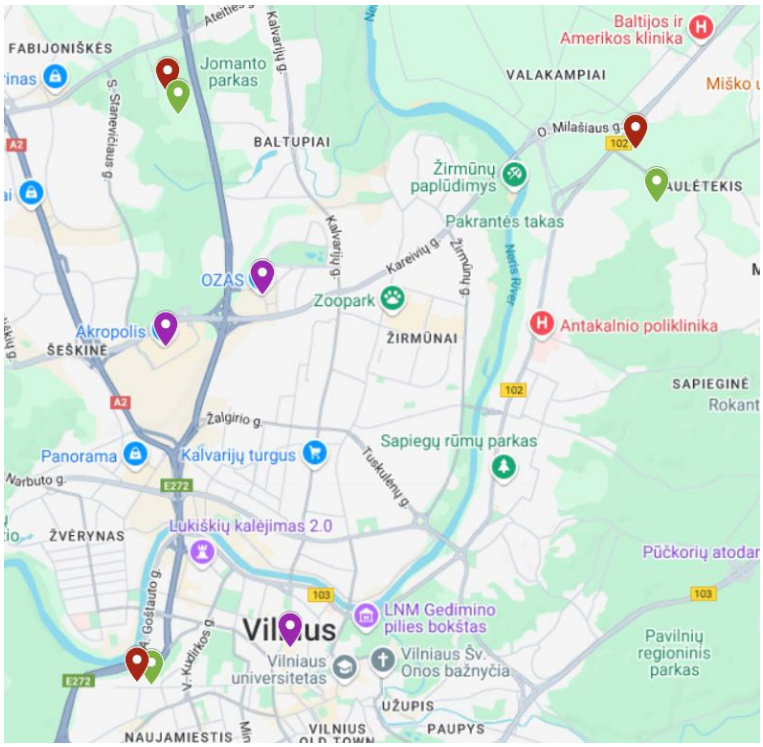
Ieškant optimalios vietos naujo universiteto statybai, iškeliami du pagrindiniai kriterijai: artumas prie studentų gyvenamųjų vietų ir atstumas nuo triukšmingų, dėmesį blaškančių zonų. Siekiant tai įgyvendinti, buvo pasirinktos trys vietos, laikomos patogios studentams: Baltupių bendrabutis, Saulėtekio alėjos bendrabučiai ir M. K. Čiurlionio bendrabutis. Šios vietos vertinamos kaip taškai, prie kurių universitetas turėtų būti kuo arčiau. Tuo tarpu triukšmingos ir pramoginės vietos, nuo kurių norima nutolti, yra: Akropolis, Ozas ir Vilniaus gatvė (Senamiestis). Taip pat peržvelgsime kaip algoritmas elgiasi su testiniais taškais.

Optimizavimo tikslas – surasti tašką žemėlapyje (koordinačių plokštumoje), kuris atitiktų šiuos kriterijus: būtų arti bendrabučių ir tuo pačiu – pakankamai toli nuo nurodytų pramoginių vietų. Taip pat siekiama įvertinti, ar PSO algoritmas sugeba pasiūlyti realiomis sąlygomis logišką vietą, ir kaip keičiasi rezultatai keičiant algoritmo parametrus.

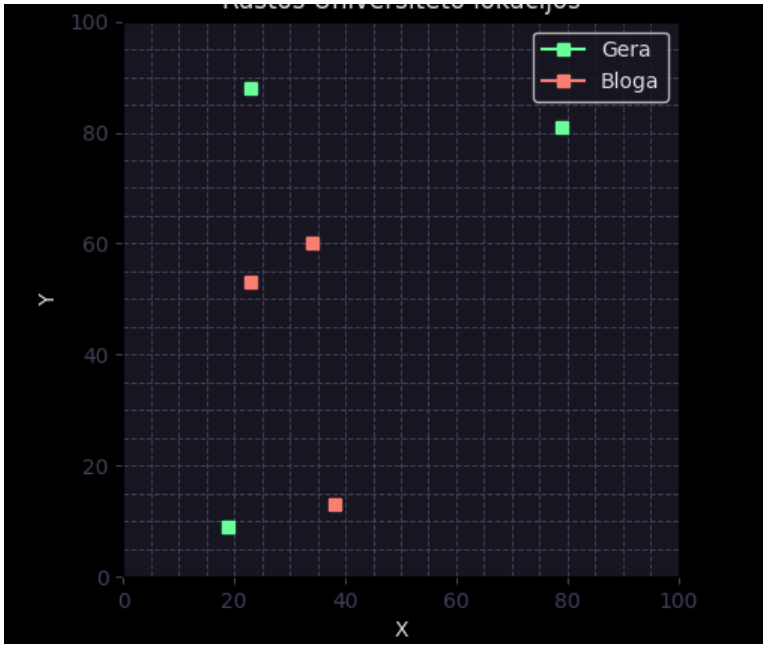
2.2. Uždavinio analizė ir pasiruošimas

Realiu atveju pasirinktos trys geros lokacijos: Baltupių bendrabutis, Saulėtekio alėjos bendrabučiai ir M. K. Čiurlionio bendrabutis „Google Maps“ pažymėti raudona spalva (žr. 2 pav.), o nepatogios vietos – violetine. Šis vaizdas yra vizualizuojamas naudojant Python programavimo

kalbą 3 pav., kur išsprendus uždavinį bus papildyta tinkamomis universiteto lokacijomis. Palyginimui turime „Google Maps“ jau žaliai atžymėtus esančius šalia universitetus.

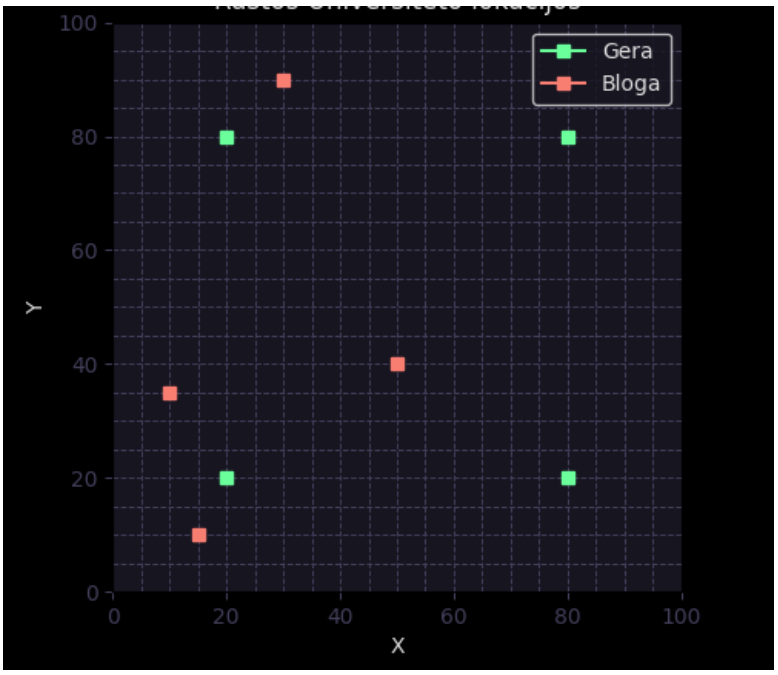


2 pav. Žemėlapis su pažymėtomis Vilniaus bendrabučių ir triukšmingų vietų lokacijomis



3 pav. Realių lokacijų vizualizavimas naudojant Python programavimo kalbą

4 pav. matomas testinis atvejis su pradinėmis gerosiomis ir blogosiomis koordinatėmis.



4 pav. Nepatogių lokacijų vizualizavimas naudojant Python programavimo kalbą

Norėdami rasti 3 optimalias universiteto lokacijas mūsų paieškos erdvėje, naudojant išsamią paiešką, reikėtų įvertinti net 166 616 670 000 galimų sprendinių, t. y. skaičiuoti atstumus tarp kiekvienos kombinacijos gerų ir blogų lokacijų. Tačiau su PSO gaunamas rezultatas per kelias sekundes.

2.3. Programos veikimas

Algoritmas stengiasi „pabėgti“ nuo blogų lokacijų ir „priartėti“ prie gerų. Dalelė – tai kandidatas į sprendimą. Dalelės pradžioje paskirstomos atsitiktinai visame plote, o po to jos iteratyviai keičia savo poziciją pagal trijų faktorių sumą:

```python
def naujas_greitis(self, global_best_pozicija, w_inerc, atmintis_ind, globali_patirtis):
    for x in range(len(self.pozicija)):
        inertia = w_inerc * self.greitis[x]
        atmintis = atmintis_ind * random.random() * (self.best_pozicija[x] - self.pozicija[x])
        globalus = globali_patirtis * random.random() * (global_best_pozicija[x] - self.pozicija[x])
        self.greitis[x] = inertia + atmintis + globalus
```

5 pav. Dalelių pozicijos pakeitimo principas Python kode

- inercijos svoris – reguliuoja, kiek dalelė laikosi savo ankstesnio greičio,
- atminties svoris – skatina sugrįžti į savo geriausią ankstesnę poziciją,
- globalios patirties svoris – skatina judėti link viso spiečiaus geriausio sprendimo.

Keičiant šiuos svorius, keičiasi ir algoritmo elgsena, nuo ko priklauso ir galutinis sprendimas, pvz., padidinus inercijos svorį, dalelės ilgiau nekeistų krypties ir taptų labiau inertiškos, lėčiau konverguodamos. Sumažinus globalią patirtį, jos silpniau atsižvelgtų į bendrą spiečiaus sprendimą ir būtų labiau savarankiškos.

Sprendimui rasti naudojama tinkamumo funkcija (fitness), kuri suformuluota taip, kad maksimizuotų mažiausią atstumą nuo universitetų iki artimiausios blogos vietos ir tuo pačiu minimizuotų didžiausią atstumą iki gerų vietų. Tai realizuojama funkcijoje:

$$fitness = min(min\_atstumas\_iki\_blogo) - max(min\_atstumas\_iki\_gero)$$

Čia:

- *min(min_atstumas_iki_blogo)* – mažiausias atstumas nuo blogų vietų iki dalelės,
- *max(min_atstumas_iki_gero)* – didžiausias atstumas nuo gerų vietų iki dalelės.

Pvz., jeigu vienas universitetas yra 20 metrų nuo artimiausios blogos lokacijos, o kitas 5 metrų, tai reikšmė min = 5. Jei geriausias atstumas iki gerų vietų yra 10 metrų, o blogiausias 30 metrų, tai reikšmė max = 30. Tokiu atveju fitness = 5 - 30 = -25, t. y., blogas rezultatas. Tikslas – gauti kuo didesnę fitness reikšmę.

Visas procesas vykdomas 1000 iteracijų. Galiausiai gaunama global_best_pozicija – tai koordinačių rinkinys, kuriame universitetai yra optimaliausios pozicijos. Rezultatai atvaizduojami vizualizacijoje, kurioje matomos visos geros ir blogos lokacijos bei optimizuotos universitetų vietos.

Apibendrinus, kiekviena dalelė kiekvienoje iteracijoje:

1. Apskaičiuoja naują greitį.
2. Atnaujina savo poziciją.
3. Įvertina savo tinkamumo funkciją (fitness).
4. Jei jos rezultatas geresnis nei bet kada anksčiau – įsimena jį.
5. Jei jos rezultatas yra geriausias iš visų dalelių – atnaujinamas bendras global_best.

Toks PSO pritaikymas leidžia lanksčiai ir efektyviai rasti geriausią universitetų vietą, atsižvelgiant į kompleksinius ir netiesioginius kriterijus, kuriuos būtų sunku apibrėžti tiesioginiais loginiais sąryšiais.

2.4. Tiriamieji eksperimentai

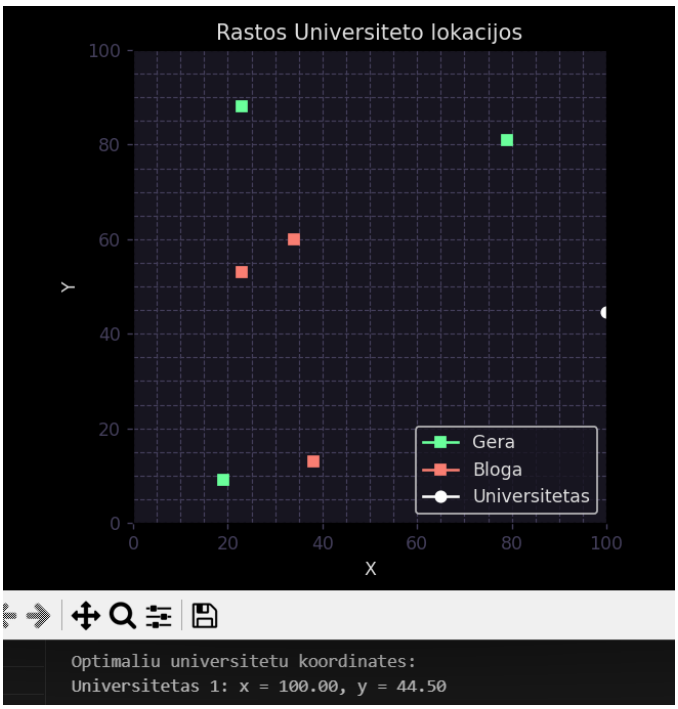
Tirsime, kaip keičiasi universitetų išdėstymas, norint optimizuoti 1, 2 ir 3 universitetų išdėstymą. Keičiant tokius parametrus kaip dalelių skaičius spiečiuje, iteracijų skaičius ir svorių reikšmės – inercijos, atminties bei globalios patirties – turėtų keistis optimizacijos rezultatai, sprendinių kokybė ir paieškos sparta. Skirtingos svorių reikšmės reguliuoja dalelių judėjimą: aukštesnis inercijos svoris skatina dalelių stabilumą ir ilgesnį kryptingą judėjimą, o didesnė globali patirtis – spartesnį priartėjimą prie geriausio bendro sprendimo.

Numatytieji parametrai:

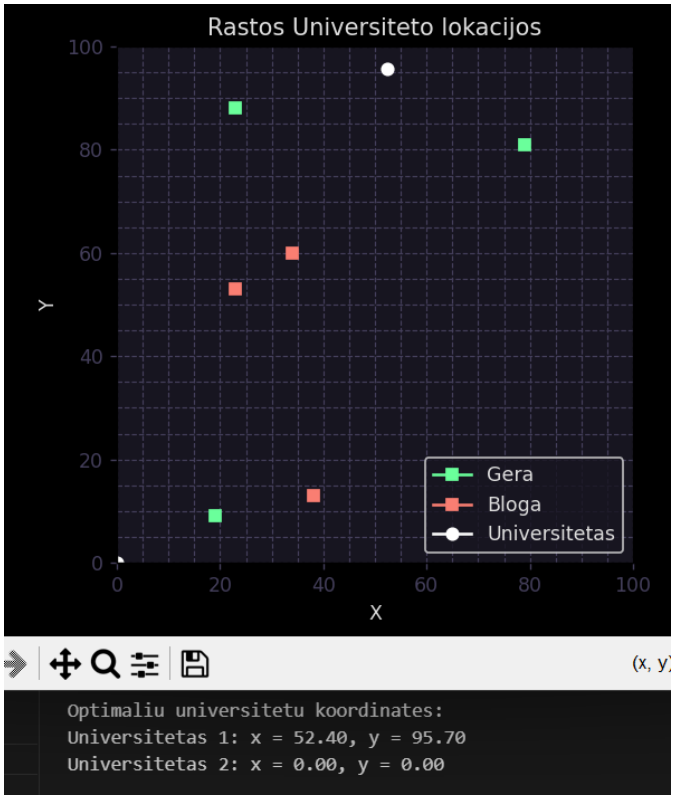
- Iteracijų skaičius: 1000
- Dalelių skaičius: 500
- Inercijos svoris: 0.7
- Atminties svoris: 1.5
- Globalios patirties svoris: 1.5

Buvo pasirinktos tokios numatytųjų parametrų reikšmės, nes jos užtikrina stabilų veikimą daugeliu atveju – tai buvo nustatyta bandymų būdu. Dideli iteracijų ir dalelių skaičiai reikalingi tam, kad algoritmo sprendiniai būtų tikslesni.

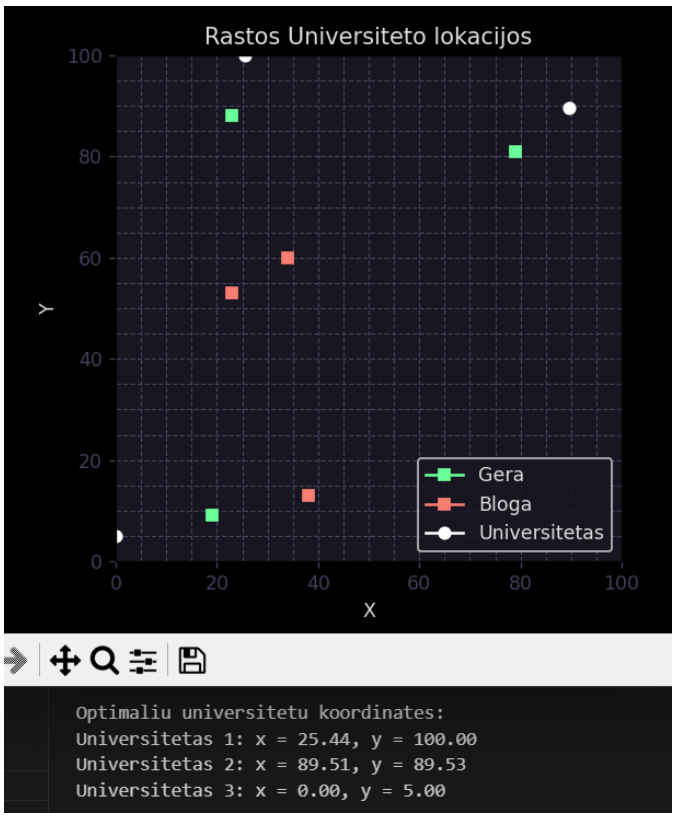
Rastos 1, 2 ir 3 optimalios universiteto lokacijos realiu atveju su numatytaisiais parametrais vizualizuojamos atitinkamai 6, 7 ir 8 pav. čia taip pat matomos ir optimalių universitetų koordinatės.



6 pav. Realaus atvejo 1 universiteto optimalios lokacijos vizualizacija ir koordinatės su numatytaisiais parametrais

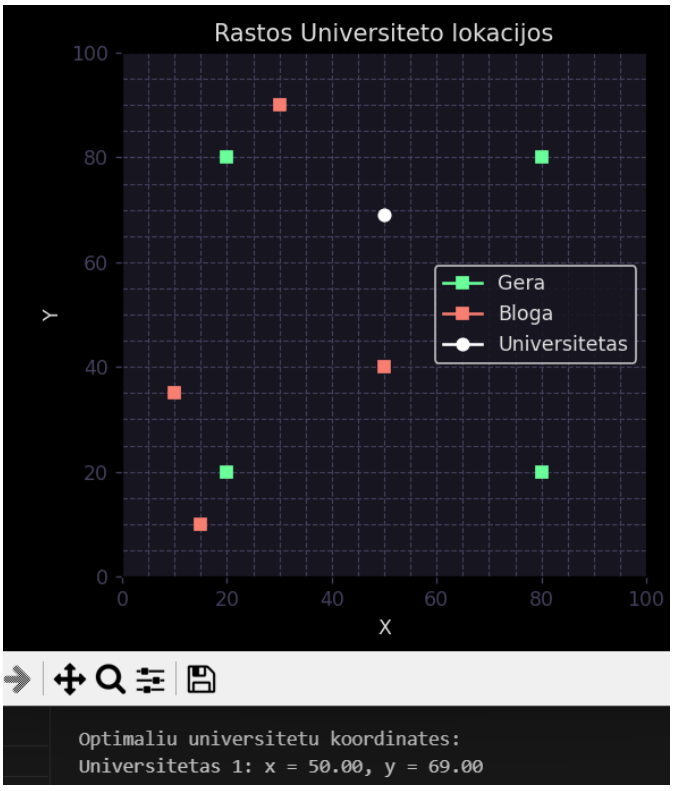


7 pav. Realaus atvejo 2 universitetų optimalių lokacijų vizualizacija ir koordinatės su numatytaisiais parametrais

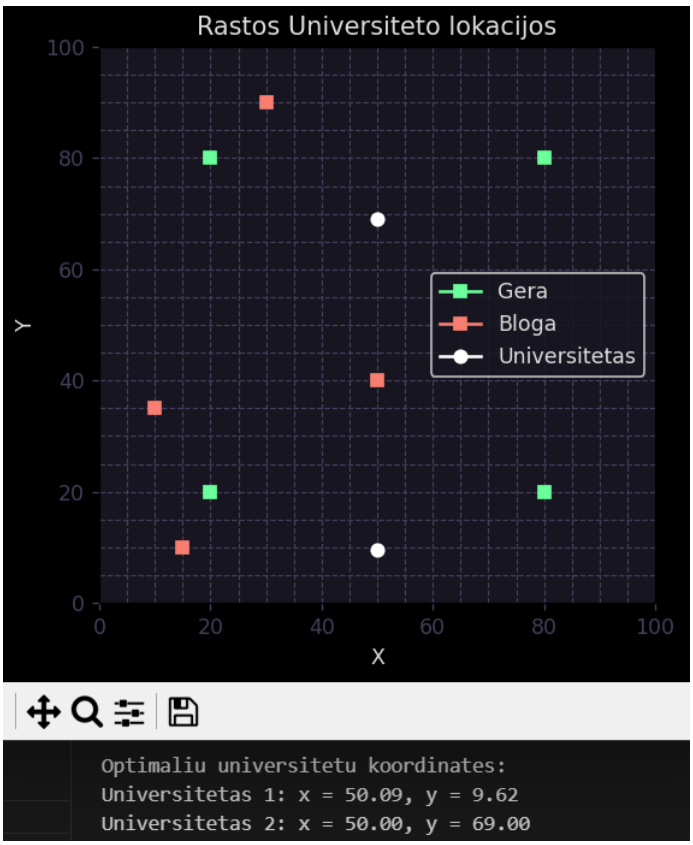


8 pav. Realaus atvejo 3 universitetų optimalių lokacijų vizualizacija ir koordinatės su numatytaisiais parametrais

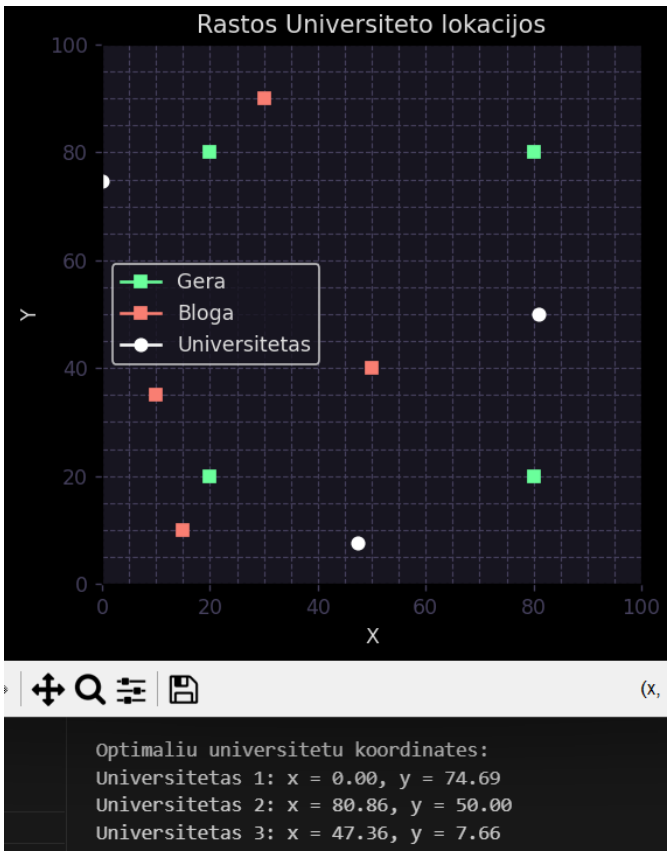
1, 2 ir 3 universitetų optimaliosios lokacijos testiniu atveju su numatytaisiais parametrais vizualizuojamos atitinkamai 9, 10 ir 11 pav. čia taip pat matomos ir optimalių universitetų koordinatės.



9 pav. Testinio atvejo 1 universiteto optimalios lokacijos vizualizacija ir koordinatės su numatytaisiais parametrais



10 pav. Testinio atvejo 2 universitetų optimalių lokacijų vizualizacija ir koordinatės su numatytaisiais parametrais



11 pav. Testinio atvejo 3 universitetų optimalių lokacijų vizualizacija ir koordinatės su numatytaisiais parametrais

Eksperimentų metu, siekiant įvertinti kiekvieno parametro įtaką atskirai, bus keičiamas tik vienas konkretus parametras (pvz., atminties svoris), o visi likusieji išliks tokie patys kaip numatytieji. Tokiu būdu užtikrinama, kad stebimi rezultato pokyčiai būtų nulemti vieno parametro, o ne jų kombinacijos. Tai padės įvertinti, kokie parametrai daro didžiausią įtaką galutiniams sprendiniams – optimalių universitetų lokacijų koordinatėms.

Naudojami testavimo parametrai:

- universitetų skaičius: 1, 2 ir 3, padės įvertinti, kaip didėjantis objektų skaičius veikia optimizacijos sudėtingumą ir sprendinių kokybę;
- iteracijų skaičius: 500, 2000, įvertins, kiek iteracijų reikia, kad algoritmas pasiektų sprendimą;
- dalelių skaičius: 100, 500 ir 1500, rodys, kaip spiečiaus dydis daro įtaką sprendimo stabilumui ir tikslumui;
- inercijos svoris: 0.3, 1.5, 2.5, valdys dalelių polinkį išlaikyti judėjimo kryptį;
- atminties svoris: 1.5, 3.0 ir 7.0, parodys, kiek dalelė yra linkusi grįžti į savo buvusią geriausią poziciją;
- globalios patirties svoris: 1.5, 3.0 ir 7.0, lems, kiek dalelė seka viso spiečiaus geriausią rezultatą.

Tokie parametrai parinkti, nes jie tiesiogiai lemia PSO algoritmo veikimą ir leidžia subalansuoti paieškos įvairovę ir konvergencijos greitį, siekiant rasti optimalų universitetų išdėstymą, kuris maksimaliai patenkina tiek artumo prie bendrabučių, tiek atstumo nuo triukšmingų zonų reikalavimus.

Siekiant užtikrinti rezultatų patikimumą, kiekvieno eksperimento metu, kai buvo tiriamas vienas konkretus parametras ir viena jo reikšmė, algoritmas buvo paleistas tris kartus. Tai atlikta tam, kad būtų galima patikrinti, ar gautas rezultatas yra stabilus ir pasikartojantis. Jei visi trys vykdymai duodavo tą patį rezultatą, buvo fiksuojamas pasikartojantis sprendinys. Priešingu atveju, kai rezultatai skyrėsi kiekvieną kartą, buvo skaičiuojamas gautų koordinačių vidurkis – tokie atvejai yra paryškinti lentelėse.

Pavyzdžiui, jei buvo analizuojamas 1 universiteto išdėstymo atvejis su 500 iteracijų, kodas buvo vykdomas tris kartus. Tuomet tas pats buvo daroma su 1 universitetu ir 2000 iteracijų. Taip buvo elgiamasi su visomis testuojamomis parametrų reikšmėmis.

2.4.1. Realus atvejis

Šiuo atveju buvo naudojami realūs taškai, atitinkantys faktines Vilniaus vietoves – bendrabučių (gerų lokacijų) ir triukšmingų zonų (blogų lokacijų) koordinatės parinktos pagal jų geografinį išsidėstymą mieste. Šis scenarijus leido įvertinti, ar algoritmas geba pasiūlyti logiškai pagrįstas universitetų vietas realiame kontekste. Eksperimentų rezultatai pateikti 2 lentelėje, 3 lentelėje, 4 lentelėje, 5 lentelėje ir 6 lentelėje.

2 lentelė. Realus atvejis – iteracijų skaičius

Universitetų skaičius	**Iteracijų skaičius**	**Koordinatės**
1	500	Universitetas 1: x = 100.00, y = 44.50
	2000	Universitetas 1: x = 100.00, y = 44.50
2	500	Universitetas 1: x = 0.00, y = 0.00

		Universitetas 2: x = 52.40, y = 95.70
	2000	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
3	500	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 19.11, y = 100.00** **Universitetas 3: x = 80.42, y = 91.05**
	2000	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 21.9, y = 100.00** **Universitetas 3: x = 77.14, y = 96.9**

3 lentelė. Realus atvejis – dalelių skaičius

Universitetų skaičius	**Dalelių skaičius**	**Koordinatės**
1	100	Universitetas 1: x = 100.00, y = 44.50
	500	Universitetas 1: x = 100.00, y = 44.50
	1500	Universitetas 1: x = 100.00, y = 44.50
2	100	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	500	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	1500	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
3	100	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 18.79, y = 100.00** **Universitetas 3: x = 79.64, y = 87.94**
	500	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 25.32, y = 100.00** **Universitetas 3: x = 87.39, y = 81.82**
	1500	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 26.50, y = 100.00** **Universitetas 3: x = 87.18, y = 95.16**

4 lentelė. Realus atvejis – inercijos svoris

Universitetų skaičius	**Inercijos svoris**	**Koordinatės**
1	0.3	Universitetas 1: x = 100.00, y = 44.50
	1.5	Universitetas 1: x = 100.00, y = 44.50
	2.5	Universitetas 1: x = 100.00, y = 44.50
2	0.3	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	1.5	Universitetas 1: x = 0.00, y = 0.00

		Universitetas 2: x = 52.08, y = 95.92
	2.5	**Universitetas 1: x = 0.00, y = 0.00** **Universitetas 2: x = 52.98, y = 94.49**
3	0.3	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 10.95, y = 98.18** **Universitetas 3: x = 86.89, y = 91.64**
	1.5	**Universitetas 1: x = 0.00, y = 5.08** **Universitetas 2: x = 24.21, y = 98.92** **Universitetas 3: x = 95.28, y = 76.79**
	2.5	**Universitetas 1: x = 0.00, y = 4.93** **Universitetas 2: x = 20.67, y = 100.00** **Universitetas 3: x = 67.94, y = 88.36**

5 lentelė. Realus atvejis – atminties svoris

Universitetų skaičius	**Atminties svoris**	**Koordinatės**
1	1.5	Universitetas 1: x = 100.00, y = 44.50
	3.0	Universitetas 1: x = 100.00, y = 44.50
	7.0	Universitetas 1: x = 100.00, y = 44.50
2	1.5	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	3.0	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	7.0	Universitetas 1: x = 0.00, y = 0.00 **Universitetas 2: x = 52.39, y = 95.67**
3	1.5	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 20.53, y = 100.00** **Universitetas 3: x = 84.68, y = 83.97**
	3.0	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 28.51, y = 100.00** **Universitetas 3: x = 79.9, y = 96.85**
	7.0	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 26.91, y = 100.00** **Universitetas 3: x = 78.98, y = 100.00**

6 lentelė. Realus atvejis – globalios patirties svoris

Universitetų skaičius	**Globalios patirties svoris**	**Koordinatės**
1	1.5	Universitetas 1: x = 100.00, y = 44.50
	3.0	Universitetas 1: x = 100.00, y = 44.50
	7.0	Universitetas 1: x = 100.00, y = 44.50

2	1.5	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	3.0	Universitetas 1: x = 0.00, y = 0.00 Universitetas 2: x = 52.40, y = 95.70
	7.0	Universitetas 1: x = 0.00, y = 0.00 **Universitetas 2: x = 52.94, y = 100.00**
3	1.5	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 19.31, y = 100.00** **Universitetas 3: x = 80.33, y = 90.04**
	3.0	**Universitetas 1: x = 0.00, y = 5.00** **Universitetas 2: x = 23.08, y = 100.00** **Universitetas 3: x = 82.24, y = 93.27**
	7.0	**Universitetas 1: x = 0.00, y = 0.00** **Universitetas 2: x = 25.43, y = 100.00** **Universitetas 3: x = 82.12, y = 100.00**

2.4.2. Testavimo atvejis

Testavimo atveju naudoti dirbtinai sukurti taškai, kurių išdėstymas nėra palankus optimaliam universitetų išdėstymui – jie specialiai parinkti taip, kad būtų išsidėstę netolygiai, kad išbandytų algoritmo gebėjimą ieškoti sprendimų sudėtingesnėse sąlygose. Eksperimentų rezultatai pateikti 7 lentelėje, 8 lentelėje, 9 lentelėje, 10 lentelėje ir 11 lentelėje.

7 lentelė. Nepatogus atvejis – iteracijos skaičius

Universitetų skaičius	**Iteracijų skaičius**	**Koordinatės**
1	500	Universitetas 1: x = 50.00, y = 69.00
	2000	Universitetas 1: x = 50.00, y = 69.00
2	500	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 49.65, y = 10.52**
	2000	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 50.06, y = 10.53**
3	500	**Universitetas 1: x = 0.00, y = 68.95** **Universitetas 2: x = 49.12, y = 5.61** **Universitetas 3: x = 86.48, y = 83.21**
	2000	**Universitetas 1: x = 0.00, y = 70.63** **Universitetas 2: x = 50.0, y = 4.58** **Universitetas 3: x = 98.75, y = 98.86**

8 lentelė. Nepatogus atvejis – dalelių skaičius

Universitetų skaičius	**Dalelių skaičius**	**Koordinatės**
1	100	Universitetas 1: x = 50.00, y = 69.00

	500	Universitetas 1: x = 50.00, y = 69.00
	1500	Universitetas 1: x = 50.00, y = 69.00
2	100	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 50.3, y = 10.2**
	500	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 49.93, y = 10.2**
	1500	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 50.29, y = 10.23**
3	100	**Universitetas 1: x = 0.00, y = 70.27** **Universitetas 2: x = 50.0, y = 4.58** **Universitetas 3: x = 98.63, y = 87.9**
	500	**Universitetas 1: x = 0.00, y = 72.31** **Universitetas 2: x = 48.24, y = 6.63** **Universitetas 3: x = 87.24, y = 61.81**
	1500	**Universitetas 1: x = 0.00, y = 72.13** **Universitetas 2: x = 47.36, y = 7.66** **Universitetas 3: x = 80.86, y = 50.00**

9 lentelė. Nepatogus atvejis – inercijos svoris

Universitetų skaičius	**Inercijos svoris**	**Koordinatės**
1	0.3	Universitetas 1: x = 50.00, y = 69.00
	1.5	Universitetas 1: x = 49.87, y = 68.57
	2.5	Universitetas 1: x = 50.58, y = 68.89
2	0.3	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 49.79, y = 10.11**
	1.5	**Universitetas 1: x = 56.62, y = 7.06** **Universitetas 2: x = 57.57, y = 71.63**
	2.5	**Universitetas 1: x = 51.46, y = 1.27** **Universitetas 2: x = 55.61, y = 71.55**
3	0.3	**Universitetas 1: x = 9.07, y = 66.66** **Universitetas 2: x = 16.37, y = 7.33** **Universitetas 3: x = 73.98, y = 67.83**
	1.5	**Universitetas 1: x = 1.47, y = 69.83** **Universitetas 2: x = 50.2, y = 4.58** **Universitetas 3: x = 100.0, y = 98.66**
	2.5	**Universitetas 1: x = 0.00, y = 92.43** **Universitetas 2: x = 49.14, y = 8.59** **Universitetas 3: x = 98.72, y = 76.33**

10 lentelė. Nepatogus atvejis – atminties svoris

Universitetų skaičius	**Atminties svoris**	**Koordinatės**
1	1.5	Universitetas 1: x = 50.00, y = 69.00
	3.0	Universitetas 1: x = 50.00, y = 69.00
	7.0	**Universitetas 1: x = 49.94, y = 68.73**
2	1.5	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 1: x = 49.91, y = 9.85**
	3.0	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 1: x = 49.76, y = 10.2**
	7.0	**Universitetas 1: x = 52.04, y = 7.43** **Universitetas 2: x = 53.00, y = 69.89**
3	1.5	**Universitetas 1: x = 0.00, y = 70.21** **Universitetas 2: x = 48.24, y = 6.63** **Universitetas 3: x = 87.24, y = 56.07**
	3.0	**Universitetas 1: x = 0.00, y = 70.19** **Universitetas 2: x = 50.0, y = 4.58** **Universitetas 3: x = 100.0, y = 87.31**
	7.0	**Universitetas 1: x = 0.00, y = 71.19** **Universitetas 2: x = 49.81, y = 4.58** **Universitetas 3: x = 100.0, y = 100.0**

11 lentelė. Nepatogus atvejis – globalios patirties svoris

Universitetų skaičius	**Globalios patirties svoris**	**Koordinatės**
1	1.5	Universitetas 1: x = 50.00, y = 69.00
	3.0	Universitetas 1: x = 50.00, y = 69.00
	7.0	**Universitetas 1: x = 49.99, y = 68.73**
2	1.5	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 49.69, y = 10.53**
	3.0	Universitetas 1: x = 50.00, y = 69.00 **Universitetas 2: x = 49.93, y = 10.18**
	7.0	**Universitetas 1: x = 49.06, y = 0.00** **Universitetas 2: x = 55.41, y = 71.94**
3	1.5	**Universitetas 1: x = 0.00, y = 72.18** **Universitetas 2: x = 48.24, y = 6.63** **Universitetas 3: x = 87.23, y = 66.51**
	3.0	**Universitetas 1: x = 0.00, y = 70.14** **Universitetas 2: x = 50.0, y = 4.58** **Universitetas 3: x = 100.0, y = 100.0**
	7.0	**Universitetas 1: x = 0.00, y = 68.29**

		Universitetas 2: x = 50.16, y = 4.93 **Universitetas 3: x = 100.00, y = 100.00**

3. REZULTATAI

Eksperimentų metu buvo išbandytas PSO algoritmas dviem skirtingais atvejais – su realiomis lokacijomis Vilniaus mieste ir su dirbtinai sukurtais, nepatogiai išdėstytais taškais. Analizuoti įvairūs veiksniai: universitetų skaičius, dalelių kiekis bei svorių reikšmės (inercijos, atminties ir globalios patirties).

Realaus atvejo rezultatai parodė, kad PSO algoritmas geba pateikti logiškai pagrįstas universitetų vietas, atsižvelgdamas į artumą prie bendrabučių ir atstumą nuo triukšmingų zonų. Rezultatai buvo stabilūs ir dažniausiai kartojosi visuose trijuose bandymuose, o jei ne – buvo apskaičiuotas vidurkis. Taip pat gauti sprendiniai buvo patikrinti su 2 pav. universitetų išdėstymu. Galima padaryti išvadą, kad pagal pasirinktus parametrus (bendrabučiai ir triukšmingos zonos), realiame gyvenime universitetai yra išdėstyti neoptimaliai.

Nepatogiu atveju, kai taškai išdėstyti netolygiai, pastebėta didesnė rezultatų variacija tarp bandymų. Buvo daugiau atvejų, kai gautos koordinatės reikšmingai skyrėsi, todėl tenka naudoti vidurkius. Tokie rezultatai parodo, kad sudėtingesnėmis sąlygomis algoritmas linkęs generuoti mažiau stabilius sprendinius.

Parametrų analizė parodė:

Kai optimizuojama daugiau universitetų (2 ar 3), rezultatai tampa mažiau stabilūs. Taip atsitiko dėl to, kad su didesniu universitetų kiekiu, sprendinių erdvė tampa didesnė ir sudėtingesnė, nes atsiranda daugiau taškų, daugiau galimų pozicijų.

Iteracijų skaičius reikšmingai neveikė rezultatų. Padidinus iteracijų kiekį iki 2000, reikšmingų koordinačių pokyčių neužfiksuota. Tai rodo, kad algoritmas konverguoja anksti ir per didelis iteracijų kiekis tik didina skaičiavimų trukmę, bet neduoda papildomos naudos rezultatui;

Dalelių skaičius turėjo didesnę įtaką nepatogaus atvejo sprendimų tikslumui ir stabilumui, ypač kai buvo 3 universitetai;

Didžiausią įtaką rezultatams darė svoriai – inercijos svoris, atminties svoris ir globalios patirties svoris. Kiekvienas iš jų skirtingai veikė galutines koordinates ir tinkamumo (fitness) funkcijos kvietimo kiekį:

- Inercijos svoris:

Kai inercija maža, dalelės dažnai keičia kryptį, todėl gali greitai konverguoti, bet taip pat gali įstrigti vietiniuose minimuose. Tai ir atsitiko, nes palyginus rezultatus su kitomis reikšmėmis, su 0.3 svoriu fitness funkcija buvo kviečiama daugiausiai. Inercija su 1.5 svoriu yra optimaliausia, nes išlaiko judėjimo kryptį, bet ne per ilgai, todėl greičiau suranda gerą sprendimą. Su didele inercija 2.5 dalelės ilgiau klaidžioja po paieškos erdvę, todėl konvergavimas sulėtėja, funkcija fitness kviečiama dažniau.

- Atminties svoris:

Eksperimentai parodė, kad atminties svoris taip pat darė reikšmingą įtaką sprendimų stabilumui.

Kai atminties svoris turėjo reikšmes 1.5 ir 3.0, sprendiniai buvo stabilesni, gautos koordinatės dažnai kartodavosi. Padidinus svorį iki 7.0, pastebėta, kad fitness funkcija buvo kviečiama rečiausiai, t. y. sprendimas randamas greičiau. Tačiau tokiais atvejais gauti sprendiniai buvo mažiau stabilūs – dalelės anksti "užsirakindavo" ties vienu sprendimu, kuris nebūtinai buvo globaliai geriausias. 5 lentelėje ir 10 lentelėje galima pastebėti, kad skirtingai nuo rezultatų su 1.5 ir 3.0 svoriais, su 7.0 globaliai geriausias taškas nebuvo pasiektas.

- Globalios patirties svoris:

Su mažu globalios patirties svoriu (1.5), algoritmas ilgesnį laiką tyrinėjo paieškos erdvę, o sprendiniai buvo įvairesni, palyginus su kitų reikšmių rezultatais. Tai naudinga, kai lokacijos yra nepatogiai išdėstytos arba yra daug triukšmingų zonų. Padidinus svorį iki 3.0, sprendimas dažnai buvo randamas greičiau, tačiau ne visais atvejais jis buvo optimalus. Per stiprus spiečiaus geriausio taško sekimas kartais lėmė, kad dalelės per greitai konvergavo, dar neįvertinus alternatyvių pozicijų. Todėl kai kuriais atvejais rezultatai buvo mažiau stabilūs. Kai svoris buvo dar padidintas iki 7.0, paieška tapo itin orientuota į bendrą geriausią tašką, todėl sprendinys buvo randamas sparčiausiai, tačiau rezultatai tapo mažiau įvairūs.

4. IŠVADOS

Atlikti eksperimentai leidžia padaryti tokias išvadas:

PSO pranašumas – greitas ir logiškas sprendimų radimas didelės apimties sprendinių erdvėje, kur išsami paieška būtų neefektyvi.
Algoritmas sėkmingai pritaikomas optimizuoti pastatų lokacijas tiek patogiomis, tiek sudėtingesnėmis situacijomis.
Didžiausią įtaką sprendiniui turėjo globalios patirties svoris, kuris labiausiai keitė optimizuotas koordinates.
Realių lokacijų atveju rezultatai dažniausiai buvo stabilūs ir pasikartojantys, o nepatogiu atveju – pastebėta daugiau nestabilių sprendimų, todėl lentelėse pateikti koordinačių vidurkiai.

5. GRUPĖS NARIŲ INDĖLIS Į DARBĄ

12 lentelėje pateikta informacija, kaip buvo paskirstytas darbas.

12 lentelė. Grupės narių indėlis į darbą

Grupės narys	**Atliktas darbas**
Adriana Širokytė	Suprogramuotas algoritmas, realaus ir nepatogaus atvejų tobulinimas; ataskaitoje aprašytas 2.4 poskyrius, 3, 4 skyriai.
Paulina Podgaiska	Suprogramuotas algoritmas; realaus ir nepatogaus atvejų idėja, programavimas; ataskaitoje aprašytas 1 skyrius, 2.1, 2.2, 2.3, 2.4 poskyriai.

PRIEDAS

Žemiau pateikiamas kodas su anksčiau aprašytu įgyvendintu algoritmu.

Realaus atvejo koordinatės:

```
geros_lokacijos = [
    [19, 9],
    [23, 88],
    [79,81]
]
blogos_lokacijos = [
    [23, 53],
    [34,60],
    [38,13]
]
```

Testinio atvejo koordinatės:

```
geros_lokacijos = [
    [20,20],
    [20,80],
    [80,20],
    [80,80]
]
blogos_lokacijos = [
    [15,10],
    [30,90],
    [10,35],
    [50,40]
]
```

Algoritmas:

```
import random
from plot_results import plot_results
import math

x_ribos = [0, 100]
y_ribos = [0, 100]
universitetu_sk = 3
daleliu_sk = 500
num_iterations = 1000
w_inerc = 0.7
atmintis_ind = 1.5
globali_patirtis = 1.5

geros_lokacijos = [
    [19, 9],
    [23, 88],
    [79,81]
]
blogos_lokacijos = [
    [23, 53],
    [34,60],
    [38,13]
]

def atstumas(p1, p2):
    return math.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

def fitness_fun(*universiteto_lokacijos):

    universiteto_lokacijos = zip(universiteto_lokacijos[0::2],
universiteto_lokacijos[1::2])
    atstumas_iki_gero = []
    atstumas_iki_blogo = []

    for univeras in universiteto_lokacijos:
```

```
        atstumas_iki_gero.append([atstumas(univeras, gera_l) for gera_l in
geros_lokacijos])
        atstumas_iki_blogo.append([atstumas(univeras, bloga_l) for bloga_l in
blogos_lokacijos])

    min_atstumas_iki_gero = [min(dists) for dists in zip(*atstumas_iki_gero)]
    min_atstumas_iki_blogo = [min(dists) for dists in zip(*atstumas_iki_blogo)]
    fitness = min(min_atstumas_iki_blogo) - max(min_atstumas_iki_gero)
    return fitness

class Particle:

    def __init__(self, bounds):
        self.pozicija = [random.uniform(bound[0], bound[1]) for bound in bounds]
        self.greitis = [random.uniform(-1, 1) for _ in bounds]
        self.best_pozicija = self.pozicija[:]
        self.best_fitness = fitness_fun(*self.pozicija)

    def naujas_greitis(self, global_best_pozicija, w_inerc, atmintis_ind,
globali_patirtis):
        for x in range(len(self.pozicija)):
            inertia = w_inerc * self.greitis[x]
            atmintis = atmintis_ind * random.random() * (self.best_pozicija[x] -
self.pozicija[x])
            globalus = globali_patirtis * random.random() * (global_best_pozicija[x] -
self.pozicija[x])
            self.greitis[x] = inertia + atmintis + globalus

    def naujas_pozicija(self, bounds):
        for x in range(len(self.pozicija)):
            self.pozicija[x] += self.greitis[x]
            self.pozicija[x] = max(bounds[x][0], min(self.pozicija[x], bounds[x][1]))
        fitness = fitness_fun(*self.pozicija)
        if fitness > self.best_fitness:
            self.best_fitness = fitness
            self.best_pozicija = self.pozicija[:]

class Swarm:

    def __init__(self, daleliu_sk, bounds, w_inerc=0.7, atmintis_ind=1.5,
globali_patirtis=1.5):
        self.w_inerc = w_inerc
        self.atmintis_ind = atmintis_ind
        self.globali_patirtis = globali_patirtis
        self.bounds = bounds
        self.daleles = [Particle(self.bounds) for _ in range(daleliu_sk)]
        self.global_best_pozicija = self.daleles[0].pozicija[:]
        self.global_best_fitness = self.daleles[0].best_fitness

    def optimize(self, iterations):
        for _ in range(iterations):
            for dalele in self.daleles:
                dalele.naujas_greitis(self.global_best_pozicija, self.w_inerc,
self.atmintis_ind, self.globali_patirtis)
                dalele.naujas_pozicija(self.bounds)
                if dalele.best_fitness > self.global_best_fitness:
                    print(f'Fit: {dalele.best_fitness}')
                    self.global_best_fitness = dalele.best_fitness
                    self.global_best_pozicija = dalele.best_pozicija[:]

bounds = [x_ribos, y_ribos] * universitetu_sk
swarm = Swarm(daleliu_sk, bounds, w_inerc, atmintis_ind, globali_patirtis)

swarm.optimize(num_iterations)
```

```
universiteto_lokacijos = [(swarm.global_best_pozicija[i],
swarm.global_best_pozicija[i+1]) for i in range(0, len(swarm.global_best_pozicija),
2)]

print("\nOptimaliu universitetu koordinates:")
for idx, loc in enumerate(universiteto_lokacijos, 1):
    print(f"Universitetas {idx}: x = {loc[0]:.2f}, y = {loc[1]:.2f}")

plot_results(x_ribos, y_ribos, geros_lokacijos, blogos_lokacijos,
universiteto_lokacijos)
```

plot_results.py :

```
import matplotlib.pyplot as plt

def plot_results(x_bounds, y_bounds, geros_lokacijos, blogos_lokacijos,
universiteto_lokacijos):
    fig, ax = plt.subplots()
    fig.patch.set_facecolor('#000000')
    ax.set_xlim(x_bounds[0], x_bounds[1])
    ax.set_ylim(y_bounds[0], y_bounds[1])
    ax.set_aspect('equal', adjustable='box')

    ax.set_facecolor('#171520')

    ax.tick_params(axis='both', colors='#45405c')
    ax.xaxis.label.set_color('#45405c')
    ax.yaxis.label.set_color('#45405c')
    plt.title("Rastos Universiteto lokacijos", color='#dfdfdf')

    ax.grid(True, which='both', linestyle='--', linewidth=0.7, color='#45405c')
    ax.minorticks_on()

    handles, labels = [], []

    def add_plot(x, y, marker, color, label, handles, labels):
        if label not in labels:
            labels.append(label)
            line, = ax.plot(x, y, marker=marker, markersize=6, color=color,
label=label)
            handles.append(line)
        else:
            ax.plot(x, y, marker=marker, markersize=6, color=color)

    for gera in geros_lokacijos:
        add_plot(gera[0], gera[1], 's', '#6AFF9B', 'Gera', handles, labels)

    for bloga in blogos_lokacijos:
        add_plot(bloga[0], bloga[1], 's', '#f97e72', 'Bloga', handles, labels)

    for universitetas in universiteto_lokacijos:
        add_plot(universitetas[0], universitetas[1], 'o', '#FFFFFF', 'Universitetas',
handles, labels)

    plt.xlabel("X", color='#dfdfdf')
    plt.ylabel("Y", color='#dfdfdf')
    ax.legend(handles=handles, loc='best', facecolor='#171520', edgecolor='#dfdfdf',
fontsize='medium', labelcolor='#dfdfdf')
    plt.show()
```