VILNIAUS UNIVERSITETAS

ALGIRDAS LANČINSKAS

# ATSITIKTINĖS PAIEŠKOS GLOBALIOJO OPTIMIZAVIMO ALGORITMŲ LYGIAGRETINIMAS

Daktaro disertacija

Fiziniai mokslai
Informatika (09 P)

Vilnius, 2013

Disertacija rengta 2009–2013 metais Vilniaus universitete

**Mokslinis vadovas:**
prof. dr. Julius Žilinskas (Vilniaus universitetas, fiziniai mokslai, informatika – 09 P)

# Padėka

*Nuoširdžiai dėkoju moksliniam darbo vadovui prof. dr. Juliui Žilinskui už nuoširdų ir atsakingą vadovavimą, vertingas konsultacijas ir nuolatinį skatinimą tobulėti.*

*Dėkoju Vilniaus universiteto Matematikos ir informatikos instituto direktoriui prof. habil. dr. Gintautui Dzemydai už suteiktas sąlygas doktorantūros studijoms; disertacijos recenzentams prof. habil. dr. Kaziui Kazlauskui ir doc. dr. Olgai Kurasovai už vertingas pastabas ir patarimus; Vilniaus universiteto Matematikos ir Informatikos instituto Sistemų analizės ir Atpažinimo procesų skyrių kolektyvams už bendradarbiavimą, pagalbą ir palaikymą; Lietuvos edukologijos universiteto Informatikos katedros kolektyvui už pagalbą ir moralinį palaikymą; Lietuvos valstybiniam studijų fondui, Lietuvos mokslų akademijai ir asociacijai InfoBalt už finansinę paramą doktorantūros studijų metu.*

*Nuoširdžiai dėkoju visiems draugams, artimiesiems ir šeimos nariams už supratingumą, meilę ir paramą.*

*Dėkoju visiems, tiesiogiai ir netiesiogiai prisidėjusiems prie šio darbo rengimo.*

*Algirdas Lančinskas*

# Reziumė

Optimizavimo uždaviniai yra aktualūs įvairiose mokslo ir pramonės srityse. Paprastai efektyviausiai sprendžiami uždaviniai, turintys tam tikras savybes, tokias kaip tikslo funkcijų tiesiškumas, iškilumas, diferencijuojamumas ir pan. Tačiau ne visi praktikoje pasitaikantys optimizavimo uždaviniai tenkina šias savybes, o kartais iš vis negali būti išreiškiami adekvačia matematine išraiška. Be to, optimizavimo uždaviniai, kurių tikslo funkcijos netenkina iškilumo savybės, gali turėti daug lokalių sprendinių, kurių geriausiojo radimas yra sudėtingas uždavinys. Dėl šių priežasčių yra populiarūs atsitiktinės paieškos optimizavimo metodai, kurių įvairovė apima tiek optimizavimą pagal vieną kriterijų, tiek daugiakriterį optimizavimą.

Disertacijoje pagrindinis dėmesys skiriamas atsitiktinės paieškos optimizavimo algoritmams ir jų lygiagretinimui. Viena pagrindinių šių algoritmų sudedamųjų dalių yra atsitiktinių skaičių generatoriai, kurių savybės gali įtakoti optimizavimo kokybę. Be to, sprendžiant optimizavimo uždavinius lygiagrečiųjų skaičiavimų sistemose yra aktualu užtikrinti, kad skirtingi procesoriai generuotų skirtingas atsitiktinių skaičių sekas. Ši problema ypač aktuali naudojant bendrosios atminties lygiagrečiųjų skaičiavimų sistemas. Todėl yra tiriami ir lyginami skirtingi atsitiktinių skaičių sekų generavimo metodai bei jų naudojimas lygiagrečiųjų skaičiavimų sistemose. Yra siūlomi keli atsitiktinių skaičių sekų pradinių reikšmių parinkimo metodai, taikytini sekų generavimui bendrosios atminties lygiagrečiųjų skaičiavimų sistemose.

Disertacijoje yra nagrinėjamas dalelių spiečiaus optimizavimo algoritmas, siūloma jo modifikacija, grįsta paieškos srities mažinimu. Modifikuota algoritmo versija yra taikoma erdvėlaivių skrydžių trajektorijų optimizavimo uždaviniams spręsti lygiagrečiųjų skaičiavimų sistemose.

Taip pat yra nagrinėjami daugiakriterio optimizavimo atsitiktinės paieškos algoritmai. Yra siūlomas lokaliojo daugiakriterio optimizavimo algoritmas, gautas modifikuojant lokaliojo optimizavimo pagal vieną kriterijų algoritmą. Gautas lokaliojo optimizavimo algoritmas apjungiamas su daugiakriterio optimizavimo genetiniu algoritmu taip sudarant hibridinį globaliojo daugiakriterio optimizavimo algoritmą. Siūlomo algoritmo kokybė vertinama sprendžiant įvairius daugiakriterio optimizavimo uždavinius.

Yra siūlomos kelios daugiakriterio optimizavimo genetinio algoritmo lygiagretinimo strategijos, grįstos sprendinių rangavimo lygiagretinimu. Siūlomų strategijų efektyvumas eksperimentiniu būdu tiriamas sprendžiant konkuruojančių objektų vietos parinkimo uždavinį didelio našumo lygiagrečiųjų skaičiavimų sistemose.

# Abstract

Optimization problems are important in various fields of research and industry. Normally it is easier to solve optimization problems having some specific properties such as linearity, convexity, differentiability, etc. of objective function. However there are a lot of practical problems that do not satisfy such properties or even cannot be expressed in an adequate mathematical form. Moreover non-convex optimization problems may have several local solutions, and finding the best one is a hard task. Due to these reasons it is popular to use random search optimization methods for solving various optimization problems.

Dissertation is focused on random search algorithms and their parallelization. One of the main components of such algorithms is random numbers' generator which properties can impact the quality of optimization. It is important to ensure the generation of different random numbers' sequences when using random search algorithms in parallel computing systems. It is especially important in shared memory parallel computing systems. Therefore, several random numbers' generators and their applicability to parallel computing systems are investigated. Several strategies to choose seed value of random numbers' sequence are proposed and investigated with respect to application in shared memory parallel computing systems.

The modification of Particle Swarm Optimization algorithm, based on reduction of the search area, is proposed and investigated. Modified version of the algorithm is applied to solve Multiple Gravity Assist Problem using parallel computing systems.

The local multi-objective optimization algorithm, based on a single agent stochastic search strategy, is presented. The proposed algorithm is incorporated into multi-objective optimization genetic algorithm, thus developing a hybrid algorithm for global multi-objective optimization. The developed algorithm is experimentally investigated by solving various global multi-objective optimization problems.

Several strategies to parallelize multi-objective optimization genetic algorithm are proposed. The proposed strategies, based on parallelization of Pareto ranking of solutions, are experimentally investigated by solving multi-objective competitive facility location problem on high performance computing systems.

# Žymėjimai ir santrumpos

| | |
|---|---|
| $d$ | funkcijos kintamųjų skaičius; |
| $\mathbb{R}^d$ | $d$-matė realiųjų skaičių erdvė; |
| $x_i$ | $i$-tasis funkcijos kintamasis; |
| $\mathbf{x}$ | funkcijos kintamųjų vektorius; |
| $f(\mathbf{x})$ | tikslo funkcija; |
| $f_i(\mathbf{x})$ | daugiakriterio optimizavimo uždavinio $i$-tasis kriterijus; |
| $F(\mathbf{x})$ | daugiakriterio optimizavimo uždavinio kriterijų vektorius; |
| $D$ | leistinoji paieškos sritis; |
| $x_i^{LB}$ | mažiausia leistina $i$-tojo kintamojo reikšmė; |
| $x_i^{UB}$ | didžiausia leistina $i$-tojo kintamojo reikšmė; |
| SASS | vieno agento stochastinė paieška (angl. *Single Agent Stochastic Search*); |
| MOSASS | daugiakriterė vieno agento stochastinė paieška (angl. *Multi-Objective Single Agent Stochastic Search*); |
| PSO | dalelių spiečiaus optimizavimas (angl. *Particle Swarm Optimization*); |
| GA | genetinis algoritmas (angl. *Genetic Algorithm*); |
| NSGA-II | rikiavimu pagal nedominuojamumą grįstas genetinis algoritmas (angl. Non-dominated Sorting Genetic Algorithm); |
| NSGA/LS | rikiavimu pagal nedominuojamumą grįstas genetinis algoritmas su lokaliąja paieška (angl. *Non-dominated Sorting Genetic Algorithm with Local Search*); |
| $\pi_c$ | kryžminimo operacijos tikimybė; |
| $\pi_m$ | mutavimo operacijos tikimybė; |

| | |
|---|---|
| MGA | erdvėlaivių skrydžių trajektorijų optimizavimas (angl *Multiple Gravity Assist*); |
| CFL | konkuruojančių objektų vietos parinkimas (angl. *Competitive Facility Location*); |
| PASG | pseudo atsitiktinių skaičių generatorius; |
| LCG | tiesinis kongruentinis atsitiktinių skaičių generatorius; |
| $P^{(k)}$ | populiacija / spiečius $k$-tojoje algoritmo iteracijoje; |
| $N$ | populiacijos / spiečiaus dydis; |
| $P*$ | tikrasis Pareto frontas; |
| $\tilde{P}$ | Pareto fronto aproksimacija; |
| $p$ | procesorių skaičius; |
| $S_p$ | lygiagrečiojo algoritmo spartinimo koeficientas; |
| $E_p$ | lygiagrečiojo algoritmo efektyvumo koeficientas. |

# Turinys

# Iliustracijų sąrašas

# Įvadas

## Tyrimų sritis ir darbo aktualumas

Optimizavimo uždaviniai moksle ir praktikoje kyla dar nuo antikinių laikų. Yra pasiūlyta daug įvairių optimizavimo metodų, skirtų spręsti optimizavimo uždaviniams, kylantiems įvairiose srityse, tokiose kaip chemija, biologija, biomedicina, operacijų tyrimas ir pan.

Paprastai efektyviausiai sprendžiami uždaviniai, turintys tam tikras savybes, tokias kaip tikslo funkcijų tiesiškumas, iškilumas, diferencijuojamumas ir pan. Tačiau ne visi praktikoje kylantys optimizavimo uždaviniai tenkina šias savybes, o kartais iš vis negali būti išreiškiami adekvačia matematine išraiška. Be to, optimizavimo uždaviniai, kurių tikslo funkcijos netenkina iškilumo savybės, gali turėti daug lokalių sprendinių, kurių geriausiojo radimas yra sudėtingas uždavinys. Dėl šių priežasčių yra populiarūs atsitiktinės paieškos optimizavimo metodai, kurių įvairovė apima tiek optimizavimą pagal vieną kriterijų, tiek daugiakriterį optimizavimą.

Šiuolaikinės technologijos lygiagrečiųjų skaičiavimų srityje leidžia išspręsti daug skaičiavimo resursų reikalaujančius optimizavimo uždavinius. Aktuali problema yra optimalus uždavinio suskaidymas į viena nuo kitos nepriklausomas užduotis, siekiant optimaliai paskirstyti darbą procesoriams ir minimizuoti duomenų perdavimo kaštus. Sprendžiant optimizavimo uždavinius atsitiktinės paieškos algoritmais yra aktualu užtikrinti, kad skirtingi procesoriai generuotų skirtingas atsitiktinių skaičių sekas. Ši problema ypač aktuali naudojant bendrosios atminties lygiagrečiųjų skaičiavimų sistemas.

Todėl šio darbo tyrimų sritis yra atsitiktinės paieškos globaliojo optimizavimo algoritmai, jų lygiagretinimas ir taikymas praktikoje pasitaikantiems uždaviniams spręsti.

## Tyrimų objektas

Disertacijos tyrimų objektas yra:

- atsitiktinės paieškos globaliojo optimizavimo algoritmai;
- lygiagrečiųjų skaičiavimų sistemos;
- atsitiktinių skaičių sekų generavimo metodai.

## Darbo tikslas ir uždaviniai

Darbo tikslas – modifikuoti esamus ir pasiūlyti naujus atsitiktinės paieškos globaliojo optimizavimo lygiagrečiuosius algoritmus, siekiant efektyvesnio optimizavimo uždavinių sprendimo. Siekiant šio tikslo buvo sprendžiami tokie uždaviniai:

- apžvelgti esamus atsitiktinės paieškos globaliojo optimizavimo metodus ir algoritmus bei jų lygiagretinimo būdus, išskirti tiriamų algoritmų grupę;
- ištirti atsitiktinių skaičių generavimo lygiagrečiųjų skaičiavimų sistemose galimybes ir metodus;
- siekiant efektyvesnio optimizavimo uždavinių sprendimo, modifikuoti ir išlygiagretinti tiriamus algoritmus, atsižvelgiant į sprendžiamą uždavinį, techninę ir programinę įrangą;
- eksperimentiniu būdu ištirti pasiūlytų algoritmų efektyvumą;
- gautus rezultatus palyginti su kitais rezultatais, gautais gerai žinomais atsitiktinės paieškos algoritmais;
- ištirti pasiūlytų algoritmų taikymo galimybes sprendžiant įvairių tipų uždavinius.

## Mokslinis darbo naujumas

Pasiūlyta dalelių spiečiaus optimizavimo algoritmo modifikacija erdvėlaivių skrydžių trajektorijų optimizavimui, grįsta leistinosios paieškos srities mažinimu. Ištirtos kelios informacijos apsikeitimo tarp kompiuterių dalelių spiečiaus optimizavimo lygiagrečiajame algoritme strategijos.

Pasiūlytas lokaliojo daugiakriterio optimizavimo algoritmas, grįstas vieno agento stochastinės paieškos strategija. Pasiūlytas lokaliojo optimizavimo algoritmas įkomponuotas į gerai žinomą globaliojo daugiakriterio optimizavimo genetinį algoritmą, taip gaunant hibridinį globaliojo daugiakriterio optimizavimo algoritmą.

Pasiūlyta sprendinių vertinimo (rangavimo) daugiakriterio optimizavimo algoritmuose lygiagretinimo strategija. Pasiūlytos strategijos pagrindu išlygiagretintas daugiakriterio optimizavimo genetinis algoritmas, skirtas tiek paskirstytosios, tiek bendrosios atminties lygiagrečiųjų skaičiavimų sistemoms. Algoritmo efektyvumas eksperimentiniu būdu ištirtas sprendžiant daugiakriterio optimizavimo uždavinius, skaičiavimams naudojant iki 2048 procesorių.

## Autoriaus dalyvavimas mokslinėse programose

Autorius dalyvauja vykdant:

- Lietuvos mokslo tarybos pagal Visuotinės dotacijos priemonę finansuojamą projektą „Kompiuterinių metodų, algoritmų ir įrankių efektyviam sudėtingos geometrijos biojutiklių modeliavimui ir optimizavimui sukūrimas“ (2011–2015);

- Lietuvos mokslo tarybos finansuojamą Mokslininkų grupių projektą „Neiškilos daugiakriterės optimizacijos metodai ir algoritmai“ (2012–2014).

Autorius nuo 2010 m. rugsėjo 24 d. iki 2010 m. gruodžio 23 d. studijavo Almerijos universitete (Ispanija), pagal studentų mainų programą ERASMUS.

Autorius nuo 2011 m. gegužės 18 d. iki 2011 m. rugpjūčio 10 d. stažavosi Edinburgo superskaičiavimų centre (The Edinburgh Parallel Computing Centre, EPCC, Jungtinė Karalystė), remiamas Europos komisijos finansuojamo projekto HPC-Europa.

Autorius 3 kartus (2011, 2012 ir 2013 metais) po dvi savaites, pagal COST programos veiklą IC0805 „Open European Network for High-Performance Computing on Complex Environments“, stažavosi Almerijos universitete, Ispanijoje.

## Ginamieji disertacijos teiginiai

1. Tinkamas leistinosios srities siaurinimas gali ženkliai padidinti dalelių spiečiaus optimizavimo algoritmo efektyvumą sprendžiant erdvėlaivių trajektorijų optimizavimo uždavinį.

2. Pasiūlytas hibridinis daugiakriterio globaliojo optimizavimo algoritmas efektyviai sprendžia eksperimentiniam tyrimui naudotus daugiakriterio optimizavimo testo uždavinius.

3. Pasiūlytos daugiakriterio genetinio algoritmo modifikacijos ir lygiagretinimo strategijos leidžia efektyviai spręsti optimizavimo uždavinius didelio našumo lygiagrečiųjų skaičiavimų sistemose.

4. Pasiūlyti algoritmai efektyviai sprendžia erdvėlaivių skrydžių trajektorijų optimizavimo ir konkuruojančių objektų vietos parinkimo uždavinius.

## Darbo rezultatų aprobavimas

Pagrindiniai disertacijos rezultatai paskelbti 6 mokslinėse publikacijose: 4 straipsniai periodiniuose recenzuojamuose moksliniuose leidiniuose; 2 straipsniai konferencijų pranešimų medžiagoje.

Disertacijos rezultatai pristatyti šiose tarptautinėse ir nacionalinėse konferencijose ir seminaruose:

- International Conference of Young Scientists. 2010 m. balandžio 29–30 d., Šiauliai, Lietuva.

- IEEE International Conference on Cluster Computing, Workshop on High-Performance Computing on Complex Environments. 2010 m. rugsėjo 20–24 d., Heraklionas, Kreta, Graikija.

- 1-oji jaunųjų mokslininkų konferencija Fizinių ir technologijos mokslų tarpdalykiniai taikymai. 2011 m. vasario 8 d., Vilnius, Lietuva.

- 4-oji Lietuvos jaunųjų mokslininkų konferencija Operacijų tyrimai versle, inžinerijoje ir informacinėse technologijose. 2011 m. rugsėjo 30 d., Kaunas, Lietuva.

- Third World Congress on Nature and Biologically Inspired Computing (NaBIC 2011). 2011 m. spalio 19–21 d., Salamanka, Ispanija.

- 2nd Workshop of COST IC0805 Open European Network for High-Performance Computing on Complex Environments. 2012 m. sausio 25–27 d., Timišoara, Rumunija.
- Antroji jaunųjų mokslininkų konferencija Tarpdalykiniai tyrimai fiziniuose ir technologiniuose moksluose. 2012 m. vasario 14 d., Vilnius, Lietuva.
- 4-asis tarptautinis seminaras Duomenų analizės metodai programų sistemoms. 2012 m. gruodžio 6–8 d., Druskininkai, Lietuva.
- 25th European Conference on Operational Research (EURO 2012). 2012 m. liepos 8–11 d., Vilnius, Lietuva.
- COST projekto IC0805 veiklos darbo grupių susitikimas. 2013 m. balandžio 15 d., Madridas, Ispanija.

## Disertacijos apimtis

Disertaciją sudaro įvadas, 3 skyriai ir bendrosios išvados. Papildomai disertacijoje pateikta: naudotų žymėjimų ir santrumpų sąrašas, iliustracijų sąrašas ir literatūros sąrašas. Bendra disertacijos apimtis yra 106 puslapiai, kuriuose pateikta 37 paveikslai, 10 lentelių ir 4 algoritmai. Disertacijoje remtasi 77 literatūros šaltiniais.

# 1 skyrius

# Atsitiktinės paieškos optimizavimo metodai

## 1.1 Optimizavimo metodų apžvalga

Optimizavimo uždaviniai moksle ir praktikoje kyla dar nuo antikinių laikų. Nors optimalumo terminą Leibnicas įvedė 1710 metais, dar senieji graikai Euklidas ir Aleksandrijos Heronas, įrodinėdami, kad optinėse sistemose šviesa renkasi trumpiausią kelią, sprendė optimizavimo uždavinį.

### 1.1.1 Optimizavimas pagal vieną kriterijų

Matematikoje ir informatikoje optimizavimas siejamas su geriausio pagal tam tikrą kriterijų elemento parinkimu iš galimos kandidatų aibės.

Matematiškai optimizavimo uždavinys apibrėžiamas kaip funkcijos $f(\mathbf{x})$ minimalios reikšmės paieška:

$$f_{min} = \min_{\mathbf{x} \in D} f(\mathbf{x}). \tag{1.1}$$

Čia funkcija $f : D \to \mathbb{R}^d$ yra vadinama *tikslo funkcija*, $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ – tikslo funkcijos *kintamųjų vektorius*, $d$ – kintamųjų skaičius, o $D \subset \mathbb{R}^d$ – parametrų reikšmių *leistinoji sritis*.

Analogiškai apibrėžiamas maksimalios tikslo funkcijos reikšmės paieškos uždavinys:

$$f_{max} = \max_{\mathbf{x} \in D} f(\mathbf{x}). \tag{1.2}$$

Nemažindami bendrumo, toliau nagrinėsime minimalios tikslo funkcijos reikšmės paieškos atvejį.

Parametrų vektorius $\mathbf{x}^* \in D$, toks, kad

$$f(\mathbf{x}^*) = f_{min} \tag{1.3}$$

yra vadinamas tikslo uždavinio *optimaliu sprendiniu*. Sprendinys $\mathbf{x}^*$, kuris yra optimalus savo aplinkoje, yra vadinamas *lokaliuoju optimaliu sprendiniu*, o sprendinys, kuris yra optimalus visoje leistinojoje srityje $D$, yra vadinamas *globaliuoju optimaliu sprendiniu*. Lokaliojo arba globaliojo optimumo paieška yra vadinama atitinkamai *lokaliuoju optimizavimu* arba *globaliuoju optimizavimu*.

Literatūroje [1, 2, 3, 4] yra siūloma įvairių globaliojo optimizavimo metodų. Monografijoje [5] siūloma tokia globaliojo optimizavimo metodų klasifikacija:

- tiesioginiai metodai:
  - atsitiktinės paieškos metodai,
  - apibendrinti nusileidimo metodai,
  - grupavimo metodai;
- netiesioginiai metodai:
  - metodai, aproksimuojantys lygio aibes,
  - metodai, aproksimuojantys tikslo funkciją;
- metodai, užtikrinantys sprendinio tikslumą:
  - padengimo metodai.

Detaliau aptarsime atsitiktinės paieškos metodus. Šie metodai gali būti adaptyvūs – atsižvelgiantys į anksčiau atliktų bandymų rezultatus, ir neadaptyvūs – neatsižvelgiantys į ankstesnių bandymų rezultatus.

Paprasčiausia neadaptyvi atsitiktinė paieška yra Monte-Karlo metodas, kuriuo galimų sprendinių koordinatės generuojamos kaip nepriklausomi atsitiktiniai dydžiai. Atsitiktinai sugeneruoti galimi sprendiniai gali būti naudojami kaip pradiniai taškai lokaliojo optimizavimo algoritmams. Tokie metodai yra paprasti, tačiau neefektyvūs – tikimybė rasti globalųjį optimumą artėja į 1, kai bandymų skaičius artėja į begalybę. Kita vertus, šie algoritmai yra paprastai realizuojami, todėl neretai yra naudojami optimizavimo uždavinių savybių nustatymui: globaliųjų ir lokaliųjų optimumų skaičiui nustatyti, leistinosios srities apibrėžimui ir pan. Taip pat tokie metodai yra lengvai lygiagretinami.

Adaptyvūs atsitiktinės paieškos metodai atsižvelgia į anksčiau atliktų bandymų rezultatus. Šių optimizavimo metodų klasei priklauso

- evoliuciniai skaičiavimai:
  - genetiniai algoritmai [6, 7, 8];
  - evoliuciniai algoritmai [9, 10];
- spiečiaus sumanumo strategijos:
  - dalelių spiečiaus optimizavimas [11, 12, 13];
  - skruzdžių kolonijos optimizavimas [14, 15];
  - bičių algoritmai [16];
- atkaitinimo modeliavimo metodai.

Atsitiktinės paieškos metodai yra populiarūs dėl paprasto realizavimo ir mažų reikalavimų tikslo funkcijai – tereikia užtikrinti galimybę apskaičiuoti tikslo funkcijos reikšmes leistinosios srities taškuose.

Optimizavimo metodai nagrinėti ir Lietuvoje apgintose disertacijose [17, 18, 19]. Tačiau jose nebuvo naginėjami atsitiktinės paieškos globaliojo optimizavimo lygiagretieji algoritmai.

### 1.1.2 Daugiakriteris optimizavimas

Praktikoje pasitaikančiuose optimizavimo uždaviniuose dažnai būna daugiau negu vienas vertinimo kriterijus. Dažniausiai kriterijai būna prieštaringi vienas kitam – mažinant vieno kriterijaus reikšmę, didinama kito kriterijaus reikšmė.

Matematiškai daugiakriterio optimizavimo uždavinys, turintis $d$ kintamųjų ir $m$ *kriterijų vektorių*

$$F(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \tag{1.4}$$

yra apibrėžiamas kaip visų kriterijų minimizavimo uždavinys:

$$F_{min} = \min_{\mathbf{x} \in D} F(\mathbf{x}). \tag{1.5}$$

### 1.1.3 Dominuojamumo sąryšis

Dėl kriterijų prieštaringumo, dažniausiai yra neįmanoma rasti tokio uždavinio sprendinio $\mathbf{x}^*$, kuris būtų vienintelis ir geriausias pagal visus kriterijus:

$$f_i(\mathbf{x}^*) = \min_{\mathbf{x} \in D} f_i(\mathbf{x}), \tag{1.6}$$

čia $i = 1, 2, \ldots, m$. Sprendinys, geriausias pagal vieną kriterijų, gali būti ne pats geriausias (arba net blogiausias) pagal kitą kriterijų. Todėl sprendinių (kintamųjų reikšmių vektorių) palyginimui yra naudojamas *dominuojamumo sąryšis*.

Sakoma, kad sprendinys $\mathbf{x}_1$ *dominuoja* sprendinį $\mathbf{x}_2$ (žymima $\mathbf{x}_1 \succ \mathbf{x}_2$), jei bet kuriam kriterijui $f_i$, $i = 1, 2, \ldots, m$ galioja

$$f_i(\mathbf{x}_1) \leq f_i(\mathbf{x}_2)$$

ir egzistuoja bent vienas kriterijus $f_j$, $j \in (1, 2, \ldots, m)$, toks, kad

$$f_j(\mathbf{x}_1) < f_j(\mathbf{x}_2).$$

Jei $\mathbf{x}_1 \succ \mathbf{x}_2$, tai sprendinys $\mathbf{x}_1$ yra vadinamas sprendinio $\mathbf{x}_2$ *dominatoriumi*, o sprendinys $\mathbf{x}_2$ laikomas *dominuojamu* sprendinio $\mathbf{x}_1$. Sprendinio dominatorių skaičius vadinamas jo *dominuojamumo rangu*.

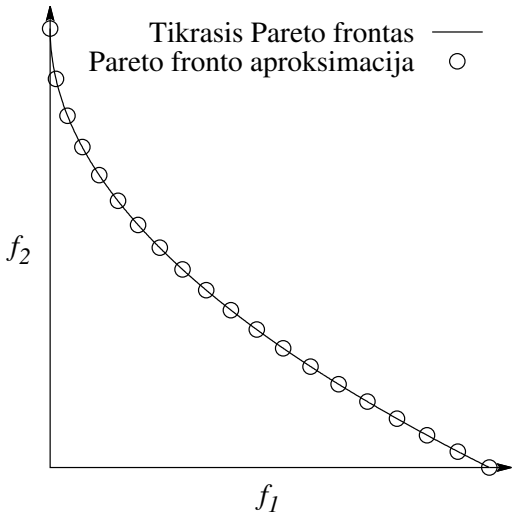
Sprendinys, neturintis nė vieno dominatoriaus yra vadinamas *nedominuojamu* arba *Pareto optimaliu* [20], pavadintu pagal italų sociologo ir ekonomisto Vilfredo Pareto pavardę.

Jei $\mathbf{x}_1 \not\succ \mathbf{x}_2$ (sprendinys $\mathbf{x}_1$ nedominuoja sprendinio $\mathbf{x}_2$) ir $\mathbf{x}_2 \not\succ \mathbf{x}_1$, tai abu sprendiniai laikomi nepalyginamais (angl. *Indifferent*). Toks sąryšis žymimas $\mathbf{x}_1 \sim \mathbf{x}_2$.

Dominuojamumo sąryšis pasižymi tranzityvumo savybe: jei $\mathbf{x}_1 \succ \mathbf{x}_2$ ir $\mathbf{x}_2 \succ \mathbf{x}_3$, tai $\mathbf{x}_1 \succ \mathbf{x}_3$.

Taigi, sprendžiant daugiakriterio optimizavimo uždavinį, ieškoma ne vieno sprendinio, kuris būtų geriausias pagal visus kriterijus, bet aibės nedominuojamų (kompromisinių) sprendinių. Šių sprendinių aibė yra vadinama *Pareto aibe*, o juos atitinkančių kriterijų vektorių aibė – *Pareto frontu*.

Pareto fronto radimas neretai būna sudėtingas ir imlus laikui uždavinys. Kita vertus sprendžiant praktikoje pasitaikančius uždavinius ne visada būtina rasti visą Pareto frontą, bet aibę nedominuojamų sprendinių, kurie reprezentuotų tikrąjį Pareto frontą. Paprastumo dėlei kriterijų vektorius toliau vadinsime taškais. Pareto fronto aproksimacijos 20 taškų aibe iliustracija pateikta 1.1 paveiksle, kuriame tolydžia kreive pažymėtas tikrasis Pareto frontas, o apskritimais – jį aproksimuojantys taškai.




1.1 pav.: Pareto fronto aproksimacija

### 1.1.4 Sprendinių vertinimo matai

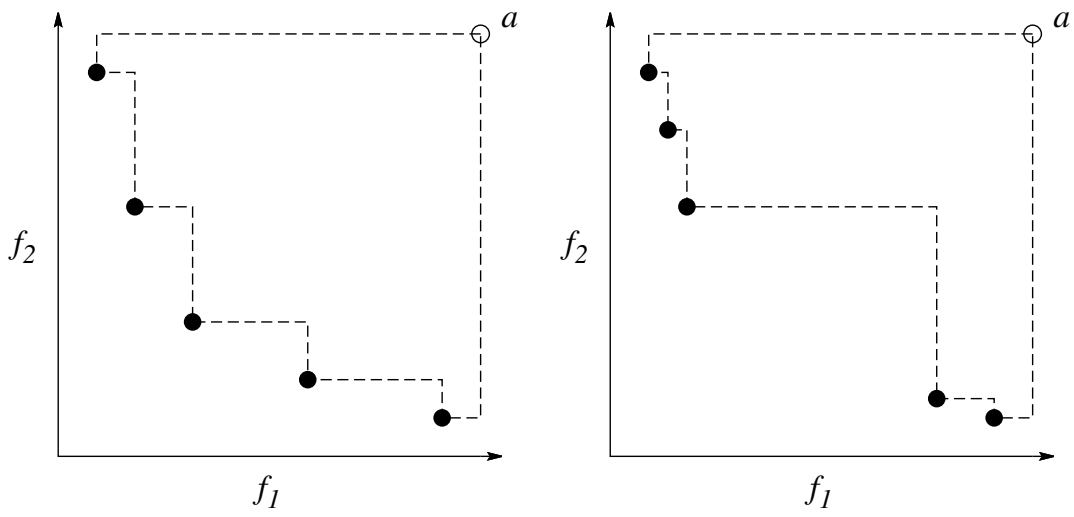
Skirtingais optimizavimo algoritmais gaunamos skirtingos (tikslesnės arba mažiau tikslios) tikrojo Pareto fronto $P^*$ aproksimacijos $\tilde{P}$, o jų vertinimui siūlomi įvairūs matai.

- *Pareto fronto aproksimacijos dydis* (angl. *Pareto Size*, PS). Šis matas parodo, kiek nedominuojamų sprendinių sudaro Pareto fronto aproksimaciją.

- *Hipertūris* (angl. *Hypervolume*, HV). Šis matas apibrėžia hipertūrį (dviejų kriterijų atveju – plotą), kurį apibrėžia tikrąjį Pareto frontą aproksimuojantys taškai ir nurodytas *atramos taškas* (angl. *Reference Point*). Šis matas atspindi Pareto fronto aproksimacijos dydį, atstumą iki tikrojo Pareto fronto ir taškų išsidėstymo tolygumą – kuo tikslesnė Pareto fronto aproksimacija ir kuo tolygiau išsidėstę aproksimuojantys taškai, tuo bus didesnė hipertūrio vertė. Hipertūrio matas iliustruotas 1.2 paveiksle, kuriame matyti, kad tolygesnės (kairėje) aproksimacijos hipertūrio mato vertė akivaizdžiai didesnė už mažiau tolygios (dešinėje) aproksimacijos hipertūrio mato vertę.

- *Pareto frontų persidengimas* (angl. *Coverage Metric*, C) [21]. Šis matas parodo, kiek Pareto frontą aproksimuojančių taškų yra iš tikrojo Pareto fronto. Matas yra išreiškiamas formule

$$C(\tilde{P}, P^*) = |\{\mathbf{x} \in \tilde{P} : \ \mathbf{x} \sim \mathbf{x}', \ \forall \mathbf{x}' \in P^*\}|. \tag{1.7}$$




1.2 pav.: Pareto fronto aproksimacijos hipertūris

Mato reikšmės gali kisti intervale $[0, |\tilde{P}|]$. $C(\tilde{P}, P^*) = 0$ reiškia, kad nė vienas iš aproksimacijos $\tilde{P}$ taškų nepriklauso tikrajam Pareto frontui $P^*$, ir atvirkščiai – $C(\tilde{P}, P^*) = |\tilde{P}|$ reiškia, kad visi fronto aproksimacijos $\tilde{P}$ taškai priklauso tikrajam Pareto frontui $P^*$.

- *Apibendrintas atstumas* (angl. *Inverted Generational Distance*, IGD) [22]. Šis matas parodo vidutinį atstumą nuo tikrojo Pareto fronto taško iki artimiausio taško aproksimacijoje. IGD yra išreiškiamas formule

$$IGD(\tilde{P}, P^*) = \frac{1}{|P^*|} \sum_{\mathbf{x}' \in P^*} \min_{\mathbf{x} \in \tilde{P}} \parallel F(\mathbf{x}') - F(\mathbf{x}) \parallel . \tag{1.8}$$

  Kadangi siekiama rasti aproksimaciją, kuri geriau reprezentuotų tikrąjį Pareto frontą, tai mažesnė IGD vertė reikškia tikslesnę aproksimaciją.

- *Pasiskirstymo įvertis* (angl. *Spread*, $\Delta$). Šis matas parodo, Pareto fronto aproksimacijos taškų išsidėstymo tolygumą. Šis matas yra paremtas atstumų tarp gretimų taškų skaičiavimu ir skirtas dvimačių Pareto fronto aproksimacijų vertinimui. Darbe [22] siūloma šio mato modifikacija, skirta uždavinių, turinčių daugiau kriterijų, Pareto frontams vertinti. Matas išreiškiamas formule

$$\Delta = \frac{\sum_{i=1}^{m} d(\mathbf{e}_i, \tilde{P}) + \sum_{\mathbf{x} \in \tilde{P}} |d(\mathbf{x}, \tilde{P}) - \bar{d}(P^*, \tilde{P})|}{\sum_{i=1}^{m} d(\mathbf{e}_i, \tilde{P}) + |P^*| \bar{d}(P^*, \tilde{P})}, \tag{1.9}$$

čia $\mathbf{e}_i = (e_1, e_2, \ldots, e_d)$ – kintamųjų reikšmių vektorius, toks, kad

$$f_i(\mathbf{e}_i) = \min_{\mathbf{x} \in D} f_i(\mathbf{x}), \;\; i = 1, 2, \ldots, m \tag{1.10}$$

$$d(\mathbf{x}, \tilde{P}) = \min_{\mathbf{y} \in \tilde{P}, \;\mathbf{y} \neq \mathbf{x}} ||F(\mathbf{x}) - F(\mathbf{y})||^2, \tag{1.11}$$

$$\bar{d}(P^*, \tilde{P}) = \frac{1}{|P^*|} \sum_{\mathbf{x} \in P^*} d(\mathbf{x}, \tilde{P}). \tag{1.12}$$

Jei Pareto fronto aproksimaciją sudaro tolygiai pasiskirstę taškai, įskaitant taškus $F(\mathbf{e}_i)$, tai $\Delta = 0$.

## 1.2 Optimizavimo metodų taikymai

### 1.2.1 MGA uždavinys

MGA (angl. *Multiple Gravity Assist*) yra erdvėlaivių skrydžių tarpplanetinių trajektorijų planavimo uždavinys, iš optimizavimo pusės nagrinėtas įvairiuose literatūros šaltiniuose [23, 24]. Uždavinys yra aktualus erdvėlaivių misijų, kurių tikslas yra pasiekti tam tikrą dangaus kūną arba įskristi į jo orbitą, planavimui. Kadangi dangaus kūnų išsidėstymas kinta laike, svarbu yra įvertinti misijos pradžios laiką.

Tarkime, kad turime $n + 1$ dangaus kūnų seką $B_0, B_1, \ldots, B_n$, čia $B_0$ – dangaus kūnas, iš kurio bus pradedama misija (dažniausiai – žemė). Misijos pradžios laiką pažymėkime $t_0$, o $T_i$, $i = 1, 2, \ldots, n$ – kelionės tarp dangaus kūnų $B_{i-1}$ ir $B_i$ trukmę. Žinant šias reikšmes, galima paskaičiuoti dangaus kūno $B_i$ poziciją $p_i$ laiko momentu

$$t_0 + \sum_{j=1}^{j=i} T_i, \tag{1.13}$$

o tuo pačiu įvertinti reikiamą erdvėlaivio greitį įskrendant į dangaus kūno $B_i$ orbitą. Erdvėlaivio skrydžio trajektorija reguliuojama keičiant erdvėlaivio greitį dangaus kūnų orbitose. Greičio pokytį dangaus kūno $B_i$ orbitoje žymėsime $\Delta v_i$. Žinant, kad greičio pakeitimui reikalingos energijos sąnaudos yra proporcingos greičio pokyčio moduliui, galime įvertinti skrydžio

trajektorijos reguliavimui reikalingas energijos sąnaudas:

$$\sum_{i=0}^{i=n} |\Delta v_i|. \tag{1.14}$$

Kadangi dangaus kūnų išsidėstymas kinta laike, tai (1.14) išraiška apibrėžtos energijos sąnaudos priklausys nuo misijos pradžios laiko ir numatytų tarpplanetinių skrydžių trukmių.

Taigi MGA optimizavimo uždavinys yra rasti tokias vektoriaus $(t_0, T_1, T_2, \ldots, T_n)$ reikšmes, kad (1.14) išraiškoje pateiktos energijos sąnaudos būtų minimalios.

### 1.2.2 Objektų vietos parinkimo uždavinys

Šiame ir tolesniuose skyreliuose *objektu* (angl. *Facility*) laikysime įvairius pramonės ir verslo objektus, teikiančius klientams paslaugas ar prekes konkrečioje geografinėje srityje. Tokiais objektais gali būti laikomi prekybos centrai, įvairių paslaugų tiekimo taškai, susisiekimo taškai (autobusų ar traukinių stotys, oro uostai) ir pan.

Tokių objektų vietų parinkimas yra strategiškai svarbus įmonėms, konkuruojančioms tarpusavyje dėl užimamos rinkos dalies. Literatūroje [25, 26, 27] yra siūloma įvairių konkuruojančių objektų vietos parinkimo (angl. *Competitive Facility Location*, CFL) uždavinio modelių. Dauguma siūlomų modelių yra susiję su naujos įmonės įsitraukimu į rinką srityje, kurioje turi konkuruoti su jau esančiomis įmonėmis. Naujos įmonės tikslas yra užimti kuo didesnę rinkos dalį, atsižvelgiant į potencialių klientų elgseną renkantis juos aptarnaujantį objektą.

Mažiau literatūroje nagrinėjami modeliai, susiję su rinkoje esančios įmonės plėtra. Tarkime, kad rinkoje yra dvi įmonės $A$ ir $B$, turinčios po atitinkamai $n_A$ ir $n_B$ objektų. Įmonė $A$ nori padidinti savo rinkos dalį, įkuriant $n_X$ naujų objektų aibę $X$. Naujų objektų vietos turi būti parenkamos taip, kad užimtų kuo didesnę rinkos dalį, pritraukiant konkuruojančios įmonės $B$ klientus. Tačiau nauji objektai $X$ gali pritraukti ir tuos klientus, kurie yra aptarnaujami kitų įmonės $A$ objektų, tokiu būdu iššaukiant taip vadinamą *kanibalizmo* (angl. *Cannibalization*) efektą, kuris plačiau tirtas darbe [28]. Vadinasi besiplečianti įmonė $A$ turi spręsti daugiakriterį optimizavimo uždavinį su dviem kriterijais: maksimizuoti naujų objektų $X$ rinkos dalį ir minimizuoti kanibalizmo efektą.

Galimų klientų aibė yra suskirstoma į poaibius pagal geografinę padėtį

ir suvedami į taip vadinamus *paklausos taškus* (angl. *Demand Points*) [29], pavyzdžiui regionus, savivaldybes, miestus ar pan. Paklausos taškų aibę žymėsime $I$.

Visi konkrečiam paslaugos taškui priklausantys klientai renkasi patrauk-liausią objektą, kai kiekvieno objekto $j \in A \cup B \cup X$ patrauklumas paklausos taškui $i \in I$ įvertinamas pagal formulę

$$a_{ij} = \frac{q_j}{1 + d_{ij}}, \tag{1.15}$$

čia $q_j$ – $j$-tojo objekto kokybės parametras, o $d_{ij}$ – atstumas tarp $j$-jo objekto ir $i$-tojo paslaugos taško. Tokiu būdu klientai objektą renkasi atsižvelgdami į objekto kokybę ir atstumą iki objekto. Atskiru atveju, esant vienodoms kokybės $q_j$, $j \in A \cup B \cup X$ vertėms, klientų pasirinkimas tampa grįstas vien tik atstumu – klientai visada renkasi artimiausią objektą.

Klientų elgsenos modelis, kai visi paklausos taško klientai renkasi pa-traukliausią objektą yra vadinamas *binariniu* (angl. *Binary*) ir yra vienas plačiausiai naudojamų klientų elgsenos modelių [30, 31, 32]. Kitas gerai žinomas klientų elgsenos modelis yra *proporcinis* (angl. *Proportional*) [33, 34, 35]. Šiame modelyje visi konkrečiam paklausos taškui priklausantys klientai pasiskirsto tarp visų rinkoje esančių objektų, proporcingai objektų patrauklumo vertėms.

Priklausomai nuo paieškos srities, objektų vietos parinkimo uždaviniai gali būti diskretieji arba tolydieji. Pirmuoju atveju naujų objektų vietos parenkamos iš numatytos diskrečios kandidatų aibės $L$. Šiuo atveju paieškos sritį sudaro visi galimi aibės $L$ poaibiai $X$, tokie, kad $|X| = s$, čia $s$ – objektų, kuriems norima parinkti vietas, skaičius. Tuo tarpu tolydžiuoju atveju, naujų objektų vietos parenkamos pagal geografines koordinates (ilguma; platuma), o paieškos sritį sudaro leistinų koordinačių aibė, kuri paprastai būna tolydi.

Įvairiems objektų vietos parinkimo uždaviniams spręsti taikomi įvairūs daugiakriterio optimizavimo metodai, tarp jų ir atsitiktinės paieškos. Ville-gas [36] taikė genetinį optimizavimo algoritmą (NSGA-II) uždaviniui su dviem kriterijais: maksimizuoti įmonės "Colombian Coffee" rinkos dalį, minimizuojant kavinių aptarnavimo kaštus. Huapu [37] ir Liao [38] tai-kė evoliucinę strategiją (SPEA algoritmą) dviejų kriterijų logistikos centrų optimalių vietų parinkimo uždavinių sprendimui.

## 1.3 Atsitiktinės paieškos algoritmai

### 1.3.1 Vieno agento stochastinė paieška

Vieno agento stochastinė paieška (angl. *Single Agent Stochastic Search*, SASS) yra iteracinis algoritmas skirtas vieno kriterijaus funkcijų optimizavimui, kuris buvo pasiūlytas [39]. Algoritmas yra grįstas stochastine paieška geriausio žinomo sprendinio $\mathbf{x}$ aplinkoje ir yra pradedamas nuo pradinio sprendinio, kuris parenkamas atsitiktinai arba sistemingai, naudojant tam tikrą informaciją. Naujas galimas sprendinys $\mathbf{x}'$ generuojamas pagal formulę

$$\mathbf{x}' = \mathbf{x} + \xi, \tag{1.16}$$

čia $\xi \leftarrow (\xi_1, \xi_2, \ldots, \xi_d)$ – atsitiktinių skaičių vektorius, sugeneruotas pagal Gauso skirstinį:

$$\xi_i \sim \mathcal{N}(b_i, \sigma(x_i^{UB} - x_i^{LB})), \tag{1.17}$$

Čia $b_i$ – Gauso skirtinio vidurkis, $\sigma$ – standartinis nuokrypis, $i = 1, 2, \ldots, d$, $d$ – uždavinio kintamųjų skaičius, o $x_i^{UB}$ ir $x_i^{LB}$ – $i$-tojo kintamojo reikšmių apatinis ir viršutinis rėžiai. Pradinėmis vidurkių $\mathbf{b} = (b_1, b_2, \ldots, b_d)$ ir standartinio nuokrypio reikšmėmis yra priskiriama atitinkamai $b_i = 0$ ir $\sigma = \sigma^{UB}$. Čia $\sigma^{UB}$ – didžiausia leistina standartinio nuokrypio reikšmė, nurodoma kaip algoritmo įvesties parametras.

Jei sugeneruotasis sprendinys $\mathbf{x}'$ nepagerina tikslo funkcijos reikšmės, tai jis yra keičiamas kitu galimu sprendiniu $\mathbf{x}''$, priešingu geriausio žinomo sprendinio $\mathbf{x}$ atžvilgiu:

$$\mathbf{x}'' = \mathbf{x} - \xi. \tag{1.18}$$

Jei kuris nors iš sugeneruotų galimų sprendinių $\mathbf{x}'$ ir $\mathbf{x}''$ pagerina tikslo funkcijos reikšmę, tai iteracija laikoma sėkminga ir vidurkio $\mathbf{b}$ reikšmės atnaujinamos pagal formulę

$$b_i \leftarrow \begin{cases} 0,2 \cdot b_i + 0,4\xi_i, & \text{jei} \quad f(\mathbf{x}') < f(\mathbf{x}), \\ b_i - 0,4\xi_i, & \text{jei} \quad f(\mathbf{x}'') < f(\mathbf{x}). \end{cases} \tag{1.19}$$

Jei nė vienas iš sugeneruotų galimų sprendinių nepagerina tikslo funkcijos reikšmės, tai iteracija laikoma nesėkminga ir vidurkio $\mathbf{b}$ reikšmės atnaujinamos pagal formulę

$$b_i \leftarrow 0,5b_i. \tag{1.20}$$

Optimizavimo metu yra stebimi pasikartojančių sėkmingų ir nesėkmingų

iteracijų skaičiai: atitinkamai $scnt$ ir $fcnt$. Jei yra atliekama daugiau nei $Scnt$ iš eilės pasikartojančių sėkmingų iteracijų, tai standartinis nuokrypis $\sigma$ yra padidinamas:

$$\sigma \leftarrow \sigma \cdot ex, \tag{1.21}$$

čia $ex > 1$ – standartinio nuokrypio didinimo koeficientas.

Analogiškai, jei atliekama daugiau nei $Fcnt$ pasikartojančių nesėkmingų iteracijų, tai standartinis nuokrypis $\sigma$ yra sumažinamas:

$$\sigma \leftarrow \sigma \cdot ct, \tag{1.22}$$

čia $ct < 1$ – standartinio nuokrypio mažinimo koeficientas.

Jei standartinio nuokrypio $\sigma$ mažinimo arba didinimo metu pažeidžiamas leistinieji $\sigma$ reikšmės rėžiai $[\sigma^{LB}, \sigma^{UB}]$, tai

$$\sigma \leftarrow \sigma^{UB}. \tag{1.23}$$

Standartinio nuokrypio didinimo, mažinimo koeficientų reikšmės, mažiausia ir didžiausia galimos standartinio nuokrypio reikšmės, bei dydžiai $Scnt$ ir $Fcnt$ yra nurodomi kaip algoritmo įvesties parametrai. Literatūroje [39] rekomenduojamos tokios pradinės parametrų reikšmės: $ex = 2$, $ct = 0.5$, $Scnt = 5$, $Fcnt = 3$, $\sigma^{LB} = 10^{-5}$, $\sigma^{UB} = 1.0$.

SASS algoritmas pateiktas 1 algoritme.

### 1.3.2 Dalelių spiečiaus optimizavimas

Dalelių spiečiaus optimizavimo (angl. *Particle Swarm Optimization*, PSO) algoritmas pasiūlytas 1995 metais Eberhart ir Kennedy darbuose [11, 13]. PSO priklauso spiečiaus sumanumo algoritmų grupei ir yra skirtas vieno kriterijaus funkcijų globaliajam optimizavimui. Algoritmas yra grįstas socialiniu individų elgesiu: *spiečius* (aibė) *dalelių* (galimų sprendinių) keičia savo pozicijas uždavinio leistinojoje srityje, atsižvelgiant į geriausią žinomą asmeninę poziciją ir geriausią poziciją, aptiktą visų spiečiaus dalelių.

Tarkime, kad turime $N$ dalelių pradinį spiečių

$$S^{(0)} = (\mathbf{x}_1^{(0)}, \mathbf{x}_2^{(0)}, \ldots, \mathbf{x}_N^{(0)}). \tag{1.24}$$

Spiečiaus dalelių pozicijos gali būti parenkamos atsitiktinai leistinojoje srityje arba sistemingai, remiantis ankstesnių uždavinio sprendimų rezultatais

**1 algoritmas:** SASS

**procedure** SASS($\mathbf{x}$, $Scnt$, $Fcnt$, $ex$, $ct$, $\sigma^{LB}$, $\sigma^{UB}$)
  $\mathbf{b}_0 \leftarrow (0, \ldots, 0)$; $scnt \leftarrow 0$; $fcnt \leftarrow 0$;
  $\sigma_0 \leftarrow \sigma^{UB}$
  $k = 1$;
  **while** sustojimo sąlyga nepatenkinta **do**
    $\sigma^{(k)} \leftarrow \begin{cases} \sigma^{(k-1)} \cdot ex & \text{jei } scnt > Scnt, \\ \sigma^{(k-1)} \cdot ct & \text{jei } fcnt > Fcnt, \end{cases}$
    **if** $\sigma^{(k)} < \sigma^{LB}$ **then**
      $\sigma^{(k)} \leftarrow \sigma^{UB}$;
    **end if**
    Generuojamas atsitiktinių skaičių vektorius $\xi^{(k)}$ pagal (1.17);
    $\mathbf{x}' = \mathbf{x}^{(k-1)} + \xi^{(k)}$;
    **if** $f(\mathbf{x}') < f(\mathbf{x}^{(k-1)})$ **then**
      $\mathbf{x}^{(k)} \leftarrow \mathbf{x}'$;
      $\mathbf{b}^{(k)} \leftarrow 0.2\mathbf{b}^{(k-1)} + 0.4\xi^{(k)}$;
      $scnt \leftarrow scnt + 1$;
      $fcnt \leftarrow 0$;
    **else**
      $\mathbf{x}'' = \mathbf{x}^{(k-1)} - \xi^{(k)}$;
      **if** $f(\mathbf{x}'') < f(\mathbf{x}^{(k-1)})$ **then**
        $\mathbf{x}^{(k)} \leftarrow \mathbf{x}''$;
        $\mathbf{b}^{(k)} \leftarrow \mathbf{b}^{(k-1)} - 0.4\xi^{(k)}$;
        $scnt \leftarrow scnt + 1$;
        $fcnt \leftarrow 0$;
      **else**
        $\mathbf{x}^{(k)} \leftarrow \mathbf{x}^{(k)}$;
        $\mathbf{b}^{(k)} \leftarrow 0.5\mathbf{b}^{(k-1)}$;
        $fcnt \leftarrow fcnt + 1$;
        $scnt \leftarrow 0$;
      **end if**
    **end if**
    $k \leftarrow k + 1$;
  **end while**
  **return x**
**end procedure**

ar kita informacija. Kiekvienai spiečiaus dalelei

$$\mathbf{x}_i^{(0)} = (x_{i1}^{(0)}, x_{i2}^{(0)}, \ldots, x_{id}^{(0)}),$$

yra žinomas jos judėjimo greitis

$$\mathbf{v}_i^{(0)} = (v_{i1}^{(0)}, v_{i2}^{(0)}, \ldots, v_{id}^{(0)}),$$

geriausia asmeninė pozicija $\mathbf{p}_i$, ir geriausia visų spiečiaus dalelių pozicija $\mathbf{p}^g$ $(i = 1, 2, \ldots, N)$.

$k$-toji spiečiaus būsena $S^{(k)}$ sudaroma keičiant dalelių pozicijas $k-1$-joje spiečiaus būsenoje $S^{(k-1)}$:

$$\begin{aligned} &\mathbf{v}_i^{(k)} = w\mathbf{v}_i^{(k-1)} + c_1 r_1(\mathbf{p}_i - \mathbf{x}_i^{(k-1)}) + c_2 r_2(\mathbf{p}^g - \mathbf{x}_i^{(k-1)}), \\ &\mathbf{x}_i^{(k)} = \mathbf{x}_i^{(k-1)} + \mathbf{v}_i^{(k)}, \\ &i = 1, \ldots, N, \end{aligned} \tag{1.25}$$

čia $w$ – inercijos momentas, $r_1$ ir $r_2$ – atsitiktiniai skaičiai iš intervalo $[0, 1]$. Konstantos $c_1$ ir $c_2$ yra atitinkamai savęs pasikliovimo (angl. *Self-Confidence*) ir spiečiaus pasikliovimo (angl. *Swarm-Confidence*) koeficientai – didesnė $c_1$ reikšmė reiškia didesnį pasikliovimą geriausia asmenine pozicija, o didesnė $c_2$ reikšmė – didesnį pasikliovimą geriausia pozicija, rasta visų spiečiaus dalelių [40]. Literatūroje [40] siūloma naudoti $c_1 = c_2 = 2$, kad sandaugų $c_1 r_1$ ir $c_2 r_2$ vidurkiai būtų lygūs 1.

Kai nauja spiečiaus būsena $S^{(k)}$ sudaryta, apskaičiuojamos visų dalelių tikslo funkcijos reikšmės ir atnaujinama informacija apie geriausias asmenines pozicijas $\mathbf{p}_i$ ir geriausią visų spiečiaus dalelių poziciją $\mathbf{p}^g$.

Analogiškai keičiant spiečiaus $S^{(k)}$ dalelių pozicijas sudaroma kita spiečius būsena $S^{(k+1)}$. Taip gaunamas iteracinis procesas, pateiktas 2 algoritme, kuris tęsiamas kol nepatenkintas pabaigos kriterijus.

Pabaigos kriterijai gali būti įvairūs:

- įvykdytas numatytas tikslo funkcijų skaičiavimų arba iteracijų skaičius;
- pasiektas numatytas skaičiavimo laikas;
- pakankamai maži tikslo funkcijos ar kintamųjų reikšmių pokyčiai;
- ir pan.

Dalelių judėjimo greičių vektorių reikšmės yra ribojamos apatiniais ir viršutiniais rėžiais: atitinkamai

$$\mathbf{v}^{LB} = (v_1^{LB}, v_2^{LB}, \ldots, v_d^{LB}) \quad \text{ir} \quad \mathbf{v}^{UB} = (v_1^{UB}, v_2^{UB}, \ldots, v_d^{UB}).$$

Paprastai viršutiniais dalelių judėjimo greičio rėžiais parenkamas

$$\mathbf{v}^{max} = \mu \cdot (\mathbf{x}^{UB} - \mathbf{x}^{LB}), \tag{1.26}$$

čia $\mathbf{x}^{LB}$ ir $\mathbf{x}^{UB}$ – kintamųjų reikšmių apatiniai ir viršutiniai rėžiai, $\mu \in (0, 1]$. Dažnai naudojama $\mu = 0.5$ [12].

**2 algoritmas:** Dalelių spiečiaus optimizavimas

**procedure** PSO($N$, $w$, $c_1$, $c_2$, $\mathbf{x}^{LB}$, $\mathbf{x}^{UB}$)
  Generuojamas pradinis dalelių spiečius $S^{(0)}$;
  **for all** $i = 1, 2, \ldots, N$ **do**
    $\mathbf{p}_i \leftarrow \mathbf{x}_i^{(0)}$;
  **end for**
  $\mathbf{p}^g \leftarrow \mathbf{p}_i : f(\mathbf{p}_i) = \min\{f(\mathbf{p}_1), f(\mathbf{p}_2), \ldots, f(\mathbf{p}_N)\}$;
  $k \leftarrow 1$;
  **while** sustojimo sąlyga nepatenkinta **do**
    **for all** $i = 1, 2, \ldots, N$ **do**
      Generuojama pozicija $\mathbf{x}_i^{(k)}$ pagal (1.25) formulę;
      **if** $f(\mathbf{x}_i^{(k)}) < f(\mathbf{p}_i)$ **then**
        $\mathbf{p}_i \leftarrow \mathbf{x}_i^{(k)}$;
      **end if**
    **end for**
    $\mathbf{p}^g \leftarrow \mathbf{p}_i : f(\mathbf{p}_i) = \min\{f(\mathbf{p}_1), f(\mathbf{p}_2), \ldots, f(\mathbf{p}_N)\}$;
    $k \leftarrow k + 1$;
  **end while**
  **return** $\mathbf{p}^g$;
**end procedure**

### 1.3.3 Genetinis algoritmas

Genetiniai algoritmai (angl. *Genetic Algorithms*, GA) yra globaliojo optimizavimo algoritmai, grįsti Darvino teorija apie gamtoje vykstančią evoliuciją [3, 4, 41, 42]. Genetiniai algoritmai nėra nauja mokslo šaka – gilesni moksliniai tyrinėjimai vykdomi nuo 1985 metų, kuomet Pitsburge (JAV) buvo surengta pirmoji tarptautinė konferencija apie genetinius algoritmus.

Genetinis algoritmas simuliuoja biologinę individų evoliuciją ir principą „išgyvena tik labiausiai prisitaikantys“ [1]. Algoritmas pradedamas nuo $N$ dydžio atsitiktinių sprendinių (individų) populiacijos $P^{(k-1)}$, vadinamos *tėvų populiacija*. Čia $k$ – algoritmo iteracijos numeris, pradžioje lygus 1. Iš populiacijos $P^{(k-1)}$ parenkami du sprendiniai

$$\mathbf{x}_1 = (x_{11}, x_{12}, \ldots, x_{1d}) \tag{1.27}$$

ir

$$\mathbf{x}_2 = (x_{21}, x_{22}, \ldots, x_{2d}). \tag{1.28}$$

Kiekvienas sprendinys gali būti išrinktas su tikimybe, proporcinga jo tikslo funkcijos reikšmei – didesnę tikimybę būti išrinktu turi sprendiniai, kurių tikslo funkcijos reikšmės yra geresnės (minimizavimo atveju – mažesnės, maksimizavimo atveju – didesnės).

Išrinkti sprendiniai yra *kryžminami*, taip gaunant du naujus sprendinius, vadinamus *palikuonimis*:

$$\mathbf{x}_1' = (x_{11}, \ldots, x_{1(c-1)}, x_{2c}, \ldots, x_{2d}) \tag{1.29}$$

ir

$$\mathbf{x}_2' = (x_{21}, \ldots, x_{2(c-1)}, x_{1c}, \ldots, x_{1d}). \tag{1.30}$$

čia $c \in \{2, 3, \ldots, d\}$ – *kryžminimo taškas* (angl. *Crossover Point*). Kryžminimas atliekamas su tikimybe $\pi_c \in [0, 1]$. Jei kryžminimas nevykdomas, tai palikuonimis laikomos sprendinių $\mathbf{x}_1$ ir $\mathbf{x}_2$ kopijos. Sprendinių $\mathbf{x}_1$ ir $\mathbf{x}_2$ kryžminimo su tikimybe $\pi_c$ operaciją žymėsime $Crossover(\mathbf{x}_1, \mathbf{x}_2, \pi_c)$.

Gautiems palikuonims taikomas *mutavimas* (angl. *Mutation*) – nežymūs palikuonių parametrų reikšmių pakeitimai. Kiekvienas parametras mutuojamas su tikimybe $\pi_m$, vadinama *mutavimo dažniu* (angl. *Mutation Rate*). Sprendinio $\mathbf{x}_1$ mutavimą dažniu $\pi_m$ žymėsime $Mutation(\mathbf{x}_1, \pi_m)$.

Po mutavimo gauti sprendiniai, pažymėkime juos atitinkamai $\mathbf{x}_1''$ ir $\mathbf{x}_2''$, išsaugomi palikuonių populiacijoje $Q^{(k-1)}$.

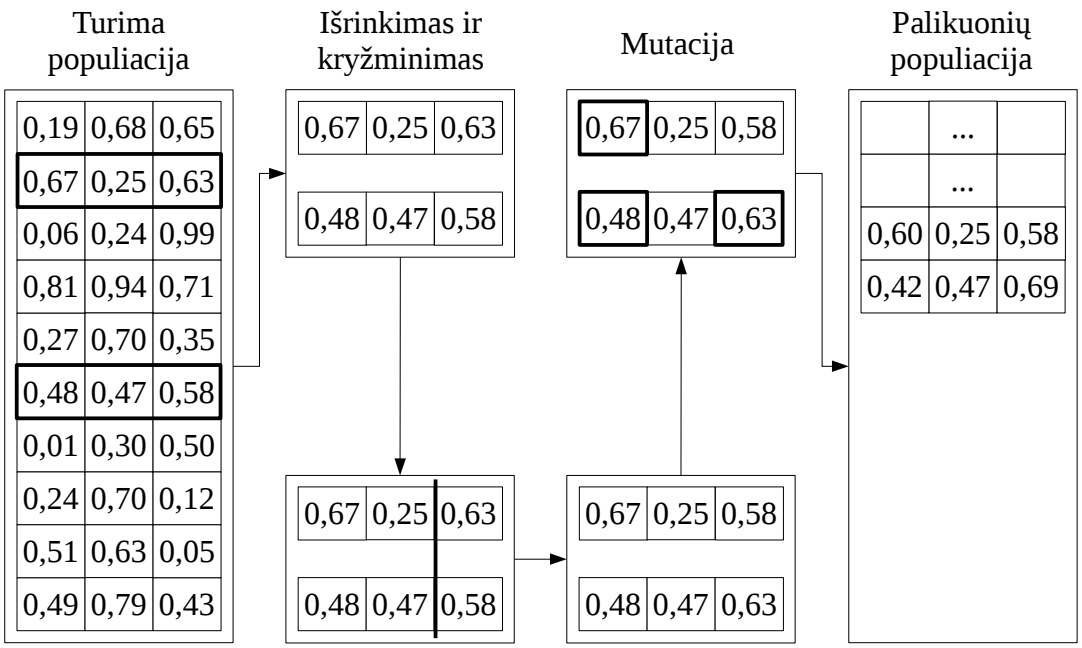
Ši palikuonių generavimo procedūra pavaizduota 1.3 paveiksle. Procedūra kartojama kol sugeneruojama $N$ dydžio palikuonių populiacija. Iš abiejų populiacijų $P^{(k-1)}$ ir $Q^{(k-1)}$ sudaroma nauja populiacija $P^{(k)}$, parenkant $N$ geriausias tikslo funkcijų reikšmes turinčius sprendinius.

Analogiškai generuojama kita palikuonių populiaciją $Q^{(k)}$, naudojant populiacijos $P^{(k)}$ sprendinius. Tokiu būdu gaunamas iteracinis procesas, pateiktas 3 algoritme, kuris tęsiamas kol nepatenkintas pabaigos kriterijus.

Pabaigos kriterijai gali būti įvairūs:

- įvykdytas numatytas tikslo funkcijų skaičiavimų arba generacijų skaičius;
- pasiektas numatytas skaičiavimo laikas;
- pakankamai maži tikslo funkcijos ar kintamųjų reikšmių pokyčiai;
- ir pan.

GA efektyvumas priklauso nuo populiacijos dydžio: naudojant labai mažą populiaciją, galimas konvergavimas į lokalųjį minimumą; naudojant didelę populiaciją, uždavinio sprendimas gali užtrukti dėl didelio tikslo funkcijų skaičiavimų skaičiaus.




1.3 pav.: Genetinių operacijų schema

**3 algoritmas:** Genetinis algoritmas

**procedure** GA($N$, $\pi_c$, $\pi_m$)
    Generuojama pradinė sprendinių populiacija $P^{(0)}$;
    $k \leftarrow 1$;
    **while** sustojimo sąlyga nepatenkinta **do**
        $Q^{(k-1)} \leftarrow \emptyset$;
        **for all** $i = 1, 2, \ldots, N/2$ **do**
            Iš populiacijos $P^{(k-1)}$ parenkami 2 sprendiniai: $\mathbf{x}_1$ ir $\mathbf{x}_2$;
            $\{\mathbf{x}'_1, \mathbf{x}'_2\} \leftarrow Crossover(\mathbf{x}_1, \mathbf{x}_2, \pi_c)$;
            $\mathbf{x}''_1 \leftarrow Mutation(\mathbf{x}'_1, \pi_m)$;
            $\mathbf{x}''_2 \leftarrow Mutation(\mathbf{x}'_2, \pi_m)$;
            $Q^{(k-1)} \leftarrow Q^{(k-1)} \cup \{\mathbf{x}''_1, \mathbf{x}''_2\}$;
        **end for**
        $P^{(k)} \leftarrow N$ geriausių $P^{(k-1)} \cup Q^{(k-1)}$ sprendinių;
        $k \leftarrow k + 1$;
    **end while**
    **return** Geriausias rastas sprendinys;
**end procedure**

### 1.3.4 NSGA-II algoritmas

NSGA (angl. *Non-dominated Sorting Genetic Algorithm*) algoritmas yra vienas pirmųjų evoliucinių daugiakriterio optimizavimo algoritmų, kurį 1995

**4 algoritmas:** NSGA-II

**procedure** NSGA($N$, $\pi_c$, $\pi_m$)
  Generuojama pradinė populiacija $P^{(0)}$;
  $k \leftarrow 1$;
  **while** sustojimo sąlyga nepatenkinta **do**
    $Q^{(k-1)} \leftarrow \emptyset$
    **for all** $i = 1, 2, \ldots, N/2$ **do**
      Iš populiacijos $P^{(k-1)}$ parenkami 2 sprendiniai: $\mathbf{x}_1$ ir $\mathbf{x}_2$;
      $\{\mathbf{x}'_1, \mathbf{x}'_2\} \leftarrow Crossover(\mathbf{x}_1, \mathbf{x}_2, \pi_c)$;
      $\mathbf{x}''_1 \leftarrow Mutation(\mathbf{x}'_1, \pi_m)$;
      $\mathbf{x}''_2 \leftarrow Mutation(\mathbf{x}'_2, \pi_m)$;
      $Q^{(k-1)} \leftarrow Q^{(k-1)} \cup \{\mathbf{x}''_1, \mathbf{x}''_2\}$;
    **end for**
    $R \leftarrow P^{(k-1)} \cup Q^{(k-1)}$;
    Skaičiuojami populiacijos $R$ sprendinių dominuojamumo rangai;
    $P^{(k)} \leftarrow N$ mažiausiai dominuojamų populiacijos $R$ sprendinių;
    $k \leftarrow k + 1$;
  **end while**
  **return** nedominuojami populiacijos $P^{(k-1)}$ sprendiniai;
**end procedure**

metais pasiūlė Srinivas ir Deb [43]. Algoritmas buvo taikomas įvairiems daugiakrierio optimizavimo uždaviniams spręsti [44, 45]. Antroji algoritmo versija – NSGA-II buvo pasiūlyta autorių Deb ir kt. darbe [46].

Kaip ir GA (žr. 1.3.3 poskyrį), NSGA-II algoritmas pradedamas nuo pradinės atsitiktiniu būdu parinktos $N$ sprendinių populiacijos $P^{(0)}$.

Nauja $N$ palikuonių populiacija $Q^{(k-1)}$ sudaroma taikant genetines operacijas (kryžminimą ir mutaciją) populiacijos $P^{(k-1)}$ individams.

Abi populiacijos $P^{(k-1)}$ ir $Q^{(k-1)}$ apjungiamos į vieną populiaciją

$$R^{(k-1)} = P^{(k-1)} \cup Q^{(k-1)}, \tag{1.31}$$

kurios individai surikiuojami pagal dominuojamumo rangus.

Nauja populiacija $P^{(k)}$ sudaroma iš populiacijos $R^{(k-1)}$ išrenkant mažiausiai dominuojamus sprendinius. Jeigu reikia rinktis tarp dviejų, vienodus rangus turinčių sprendinių, parenkamas tas, kuris yra mažiau „apgaubtas“ kitų to paties rango sprendinių [46].

Analogiškai generuojama nauja palikuonių populiacija $Q^{(k)}$, tėvų populiacija laikant $P^{(k)}$. Tokiu būdu gaunamas iteracinis procesas, pateiktas 4 algoritme, kuris tęsiamas kol nepatenkintas pabaigos kriterijus.

## 1.4 Atsitiktinių skaičių generavimas

Atsitiktinių skaičių generavimas yra taikomas įvairiose srityse: lošimuose, statistikoje, kriptografijoje, kompiuteriniame imitavime ir pan. Atsitiktinių skaičių generavimas taip pat yra ir neatsiejama atsitiktinės paieškos optimizavimo algoritmų dalis.

Pagal generavimo metodą, atsitiktiniai skaičiai skirstomi į dvi klases:

- tikrieji atsitiktiniai skaičiai (angl. *True Random Numbers*);
- pseudo atsitiktiniai skaičiai (angl. *Pseudo Random Numbers*).

Pirmuoju atveju generavimui yra naudojama speciali techninė įranga, kuri pasitelkia įvairius fizinius procesus, tokius kaip temperatūros, garso, radijo triukšmai.

Antruoju atveju atsitiktinių skaičių sekos sudaromos matematiniais metodais. Iš tikrųjų šie skaičiai nėra visiškai atsitiktiniai, kadangi jie yra išskaičiuojami, todėl yra vadinami *pseudo atsitiktiniais skaičiais*. Dauguma pseudo atsitiktinių skaičių generatorių (PASG) reikalauja pradinės reikšmės, nuo kurios bus pradedamas kitų skaičių generavimas.

### *Tiesinis kongruentinis generatorius*

Literatūroje [47, 48] yra pasiūlyta įvairių PASG, tačiau vienu pirmųjų yra laikomas *tiesinis kongruentinis generatorius* (angl. *Linear Congruential Generator*, LCG), kurį 1951 metais pasiūlė Lehmer [49]. Algoritmas generuoja neneigiamų sveikųjų skaičių sekas. LCG yra apibrėžiamas rekursija

$$x_{k+1} = (ax_k + c) \bmod m, \tag{1.32}$$

čia $a$ – daugiklis, $c$ – prieaugis, $m$ – modulis. Kaip ir daugelis PASG, taip ir LCG reikalauja pradinės skaičių sekos reikšmės $x_0$. LCG gali sugeneruoti ne daugiau kaip $m$ skirtingų skaičių seką, o maksimalią skirtingų skaičių seką sugeneruos tada ir tik tada, jei [47]:

- $c$ ir $m$ yra tarpusavyje pirminiai;
- $a - 1$ dalinasi iš visų pirminių $m$ daliklių;
- $a - 1$ dalinasi iš 4, jei $m$ dalinasi iš 4.

LCG algoritmas yra naudojamas įvairiose programavimo aplinkose, tokiose kaip Visual C/C++, Borland C/C++, Borland Delphi ir pan. – esminis naudojimo skirtumas yra parametrų $a$, $c$ ir $m$ reikšmių parinkimas.

### *Mersenne Twister generatorius*

Kitas PASG algoritmas yra taip vadinamas *Mersenne Twister* (MT), kurį 1998 metais pasiūlė Matsumoto ir Nishimura [50]. Algoritmas yra grįstas tiesine rekursija

$$\mathbf{x}_{k+n} = \mathbf{x}_{k+m} \oplus (\mathbf{x}_k^u \mid \mathbf{x}_{k+1}^l)A. \tag{1.33}$$

Čia $\mathbf{x}_{k+n}$ – $w$ dydžio dvejetainis vektorius, koduojantis $k+n$-ąjį sekos skaičių, $A$ – $w \times w$ dydžio dvejetainė matrica, $1 \leq m \leq n$, $\mathbf{x}_k^u$ – $w-r$ aukščiausių skaičiaus $\mathbf{x}_k$ bitų, $\mathbf{x}_{k+1}^l$ – $r$ žemiausių skaičiaus $\mathbf{x}_{k+1}$ bitų ($1 \leq r \leq w$). Pavyzdžiui, jei

$$\begin{aligned} \mathbf{x}_k &= (x_{w-1}^k, x_{w-2}^k, \ldots, x_{r+1}^k, x_r^k, x_{r-1}^k, \ldots, x_0^k), \\ \mathbf{x}_{k+1} &= (x_{w-1}^{k+1}, x_{w-2}^{k+1}, \ldots, x_{r+1}^{k+1}, x_r^{k+1}, x_{r-1}^{k+1}, \ldots, x_0^{k+1}), \end{aligned} \tag{1.34}$$

tai

$$\begin{aligned} \mathbf{x}_k^u &= (x_{w-1}^k, x_{w-2}^k, \ldots, x_{r+1}^k, x_r^k), \\ \mathbf{x}_{k+1}^l &= (x_{r-1}^{k+1}, x_{r-2}^{k+1}, \ldots, x_0^{k+1}). \end{aligned} \tag{1.35}$$

Skaičius $\mathbf{x}_k^u|\mathbf{x}_{k+1}^l$ sudaromas apjungiant skaičių $\mathbf{x}_k^u$ ir $\mathbf{x}_{k+1}^l$ bitus:

$$\mathbf{x}_k^u|\mathbf{x}_{k+1}^l = (x_{r-1}^{k+1}, x_{r-2}^{k+1}, \ldots, x_0^{k+1}, x_{w-1}^k, x_{w-2}^k, \ldots, x_{r+1}^k, x_r^k). \tag{1.36}$$

Simbolis $\oplus$ žymi griežtos loginės sudėties operaciją (XOR).

### *Mother of All generatorius*

Marsaglia [51] 2005 metais pasiūlė daugybos su perkėlimu metodą, kitaip dar žinomą pavadinimu „Mother of All“. Algoritmas paremtas rekursija

$$x_n = (ax_{n-r} + c_{n-1}) \; mod \; m, \tag{1.37}$$

čia $c_n = \lfloor ax_{n-r} + \frac{c_{n-1}}{m} \rfloor$, $n \geqslant r$, $n, r \in \mathbb{N}$.

### *Atsitiktinių skaičių sekų vertinimo kriterijai*

Vienas pagrindinių atsitiktinių skaičių sekų kokybės vertinimo būdų yra Kolmogorovo-Smirnovo testas (K-S) [52], kuris nusako kaip turima atsitiktinių skaičių seka atitinka norimą statistinį skirstinį. K-S įvertina maksimalų atstumą tarp vertinamos skaičių sekos ir norimo teorinio skirstinio tikimybinio pasiskirstymo funkcijų. Atsitiktinių skaičių sekos $x_0, x_1, \ldots, x_n$

tikimybinio pasiskirstymo funkcija $F_n$ yra apibrėžiama

$$F_n(x) = \frac{1}{n} \sum_{i=1}^{n} I_{x_i \leq x}, \tag{1.38}$$

čia

$$I_{x_i \leq x} = \begin{cases} 1, & \text{jei } x_i \leq x, \\ 0, & \text{kitu atveju.} \end{cases} \tag{1.39}$$

Atsitiktinių skaičių sekos ir tolygiojo skirstinio tikimybinio pasiskirstymo funkcijos $F_n(x)$ ir $F(x)$ yra iliustruotos 1.4 paveiksle.

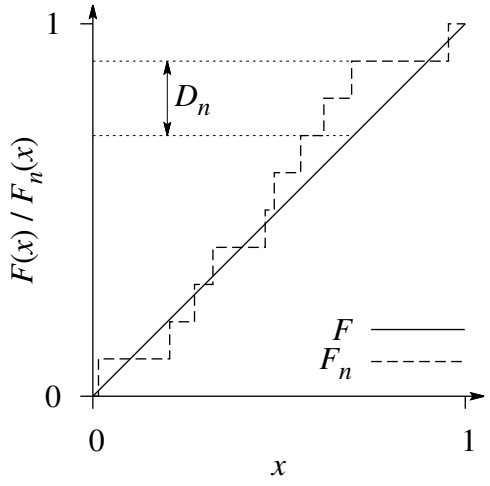
K-S skirstinio atitikimo įvertis yra išreiškiamas

$$D_n = \sup_x |F_n(x) - F(x)|. \tag{1.40}$$

Kitas svarbus PASG kokybės matas yra skaičių pasikartojimas sugeneruotoje sekoje. Šis matas gali būti išreiškiamas skirtingų skaičių procentu nuo visų sekoje esančių skaičių. Pavyzdžiui, jei 100 atsitiktinių skaičių sekoje du skaičiai sutampa, o visi kiti skirtingi, tai galima teigti, kad 99 % sekoje esančių skaičių yra skirtingi.

Trečias atsitiktinių skaičių sekos kokybės matas yra sugeneruotų sekų tarpusavio koreliacija. Tarkime, kad turime $k$ atsitiktinių skaičių sekų $X_1, X_2, \ldots, X_k$. Tada sekų tarpusavio koreliacija gali būti vertinama koreliacijų koeficiento maksimalia absoliutine reikšme

$$\max_{1 \leq i < j \leq k} |r_{X_i X_j}|, \tag{1.41}$$




1.4 pav.: Kolmogorovo-Smirnovo statistika

čia $r_{X_iX_j}$ – sekų $X_i$ ir $X_j$ koreliacijos koeficientas.

Dar vienas disertacijoje tiriamas PASG kokybės matas yra generavimo trukmė. Šis matas gali būti įvertinamas generuojant pakankamai ilgas atsitiktinių skaičių sekas ir matuojant algoritmo vykdymo laiką. Skirtingai nuo aukščiau pateiktų PASG vertinimo matų, generavimo trukmės matas priklauso nuo kompiuterio kurio vykdomas generavimas.

## 1.5 Lygiagretieji skaičiavimai

Lygiagretieji skaičiavimai yra viena labiausiai vystomų mokslo ir technologijų sričių. Lygiagrečiųjų skaičiavimų sistemas turi dauguma Lietuvos ir Europos universitetų. Yra nemažai literatūros lygiagrečiųjų skaičiavimų tema, tačiau lietuvių kalba parašytų leidinių yra nedaug. Vienas jų – R. Čiegio vadovėlis [53]. Kita vertus šioje srityje Lietuvoje apginta nemažai disertacijų [54, 55, 56, 57, 58, 59, 60, 61]. Tačiau lygiagretieji atsitiktinės paieškos globaliojo optimizavimo algoritmai jose nebuvo tiriami.

Lygiagretieji skaičiavimai remiasi prielaida, kad sprendžiamas uždavinys gali būti suskaidomas į viena nuo kitos nepriklausomas užduotis, kurių vykdymas padalinamas procesoriams. Tokiu atveju gali būti mažinamos uždavinio sprendimui reikalingos laiko sąnaudos. Galimas laiko sąnaudų sumažinimas labai priklauso nuo lygiagretinamo algoritmo specifikos: vieniems algoritmams sprendimo trukmė gali būti trumpinama daugiau, o kai kurių algoritmų visai neįmanoma lygiagretinti (pavyzdžiui Fibonači sekos narių skaičiavimas).

Algoritmų lygiagretinimo kokybė vertinama dviem pagrindiniais matais: spartinimo ir efektyvumo koeficientais. Spartinimo koeficientas $S_p$ parodo kiek kartų sutrumpėjo algoritmo vykdymo trukmė naudojant $p$ procesorių, lyginant su nuosekliojo algoritmo vykdymo trukmė. Spartinimo koeficientas išreiškiamas formule

$$s_p = \frac{T_0}{T_p}, \tag{1.42}$$

čia $T_0$ – greičiausio žinomo nuosekliojo algoritmo vykdymo trukmė, o $T_p$ – lygiagrečiojo algoritmo vykdymo trukmė, naudojant $p$ procesorių. Paprastai $1 \leq s_p \leq p$, tačiau anomalijų atveju gali būti ir $s_p < 1$ arba $s_p > p$. Lygiagrečiojo algoritmo didžiausias galimas spartinimo koeficientas gali būti įvertintas pagal nuosekliosios (nelygiagretinamos) algoritmo dalies dydį. Amdalo dėsnis [62] teigia, kad nepriklausomai nuo procesorių skaičiaus ly-

giagrečiojo algoritmo didžiausias galimas spartinimo koeficientas

$$S_{max} = \frac{1}{\alpha}, \tag{1.43}$$

čia $\alpha \in [0, 1]$ – lygiagrečiojo algoritmo nuoseklioji dalis.

Algoritmo efektyvumo koeficientas $e_p$ parodo, kaip efektyviai yra išnaudojami procesoriai ir yra išreiškiamas formule

$$e_p = \frac{s_p}{p}. \tag{1.44}$$

Paprastai $0 \leq e_p \leq 1$, o didesnė $e_p$ reikšmė reiškia geresnį procesorių išnaudojimą.

Lygiagrečiųjų skaičiavimų sistemos gali būti skirstomos į dvi pagrindines grupes: paskirstytosios atminties ir bendrosios atminties. Bendrosios atminties sistemos turi vieną bendrą atminties bloką, pasiekiamą visiems procesoriams. Todėl vieno procesoriaus į atmintį įrašyta informacija gali būti laisvai pasiekiama kito procesoriaus. Tuo tarpu paskirstytosios atminties sistemose kiekvienas procesorius turi atskirą atminties bloką, kuris prieinamas tik jam pačiam. Šiuo atveju procesoriai bendrauja vienas kitam siųsdami pranešimus.

Šiuolaikinės lygiagrečiųjų skaičiavimų sistemos dažnai yra hierarchinės – bendrosios atminties posistemiai, sujungti į paskirstytosios atminties sistemą.

## 1.6 Pirmojo skyriaus apibendrinimas

Skyriuje supažindinama su optimizavimo sąvoka, pateikiami aktualūs apibrėžimai. Apžvelgiami populiarūs atsitiktinės paieškos globaliojo optimizavimo algoritmai, kurie yra populiarūs dėl paprasto realizavimo ir mažų reikalavimų tikslo funkcijai. Daugiau dėmesio skiriama lokaliosios paieškos strategijai, grįstai vieno agento stochastine paieška (SASS), bei populiacija grįstiems genetiniam ir dalelių spiečiaus optimizavimo algoritmams.

Taip pat supažindinama su daugiakriterio optimizavimo sąvokomis ir pagrindiniais apibrėžimais. Apžvelgiamas vienas populiariausių atsitiktinės paieškos globaliojo daugiakriterio optimizavimo algoritmų – NSGA-II.

Supažindinama su dviem disertacijoje sprendžiamais optimizavimo uždaviniais – erdvėlaivių skrydžių trajektorijų optimizavimo (MGA) ir konkuruojančių objektų vietos parinkimo (CFL).

Supažindinama su lygiagrečiųjų skaičiavimų ir algoritmų lygiagretinimo

sąvokomis. Aprašomi disertacijoje aktualūs lygiagrečiųjų algoritmų vertinimo matai: spartinimas ir efektyvumas.

Apžvelgiami populiariausi atsitiktinių skaičių sekų generatoriai, bei atsitiktinių skaičių sekų vertinimo kriterijai, aktualūs lygiagretiesiems globaliojo optimizavimo algoritmams.

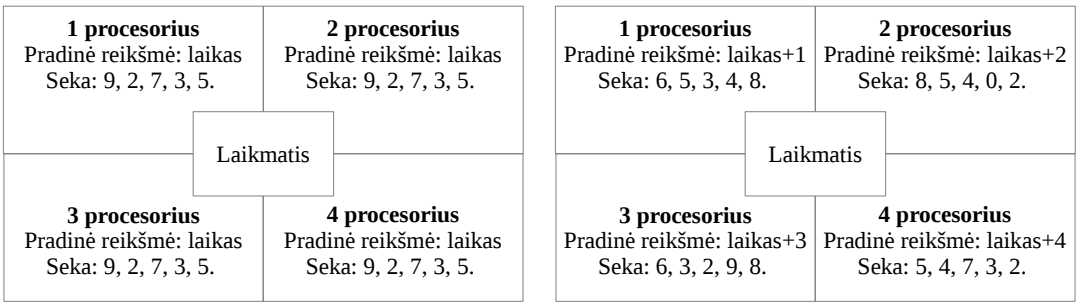
# 2 skyrius

# Algoritmų modifikavimas ir lygiagretinimas

Šiame skyriuje siūlomos atsitiktinių skaičių sekų pradinės reikšmės parinkimo strategijos, taikytinos sekų generavimui bendrosios atminties lygiagrečiųjų skaičiavimų sistemose [A1], dalelių spiečiaus optimizavimo algoritmo modifikacija, grįsta paieškos srities mažinimu [A2] bei hibridinis globaliojo daugiakriterio optimizavimo algoritmas, gautas apjungiant lokaliosios ir globaliosios paieškos strategijas [A3, A4]. Nagrinėjamos daugiakriterio optimizavimo genetinio algoritmo lygiagretinimo galimybės ir siūlomos kelios sprendinių vertinimo genetiniuose algoritmuose lygiagretinimo strategijos [A4].

## 2.1 Pradinės PASG reikšmės parinkimas

Kaip buvo minėta 1.4 skyrelyje, dauguma PASG algoritmų reikalauja pradinės reikšmės (arba kelių reikšmių), nuo kurios pradedamas skaičių sekos generavimas. Naudojant tą pačią pradinę reikšmę generuojamos identiškos skaičių sekos, todėl pradinės reikšmės parinkimas yra svarbus siekiant generuoti kelias nekoreliuojančias skaičių sekas. Tokia pradine sekos reikšme dažnai yra naudojamas esamas laikas, kadangi skirtingu laiko momentu pradėtos generuoti sekos turės skirtingas pradines reikšmes ir bus generuojamos skirtingos skaičių sekos. Tačiau toks sprendimas nėra priimtinas skaičių generavimui bendrosios atminties lygiagrečiųjų skaičiavimų sistemose, kuriose visi procesoriai naudoja bendrą laikmatį. Tokiu atveju visi procesoriai įgis vienodas pradines reikšmes ir generuos identiškas skaičių sekas. Ši problema iliustruota 2.1 paveiksle, kuriame pavaizduotos dvi situacijos:




2.1 pav.: Pseudo atsitiktinių skaičių sekų generavimas lygiagrečiųjų skaičiavimų sistemose

1. keturi bendrosios atminties procesoriai vienu metu pradeda generuoti atsitiktiniu skaičių sekas pradine reikšme naudodami esamą laiką (kairėje);

2. tie patys procesoriai vienu metu pradeda generuoti atsitiktinių skaičių sekas pradine reikšme laikydami esamo laiko ir procesoriaus numerio (ID) sumą (dešinėje).

Pirmuoju atveju visi procesoriai įgyja vienodas pradines reikšmes, todėl generuoja identiškas skaičių sekas. Tuo tarpu antruoju atveju, kiekvienas procesorius skaičių sekos generavimą pradeda nuo skirtingos pradinės reikšmės ir lygiagrečiai sugeneruojamos 4 skirtingos skaičių sekos.

Pasiūlysime kelias pradinių PASG reikšmių parinkimo strategijas, taikytinas lygiagrečiuosiuose algoritmuose:

S1. Procesoriaus vidinio laikrodžio laiko vertė ($Time$) padauginta iš to procesoriaus numerio (ID):

$$seed_{S1} = Time \times ID. \tag{2.1}$$

S2. Procesoriaus vidinio laikmačio pertraukimų, įvykdytų nuo programos vykdymo pradžios, skaičius ($Tics$), padaugintas iš procesoriaus ID:

$$seed_{S2} = Tics \times ID. \tag{2.2}$$

S3. Reikiamas kiekis skirtingų pradinių reikšmių sugeneruojamas naudojant nuoseklų PASG, kurios padalinamos atitinkamiems procesoriams.

S4. Procesoriaus vidinio laikmačio vertė ($Time$), laikmačio pertraukimų

skaičius ($Tics$) ir procesoriaus ID yra kombinuojami pagal formulę

$$\begin{aligned} seed_{S4} = (Time \bmod 32768) \text{ rotr } (ID \bmod 31) + \\ (Tics \bmod 32768) \text{ rotr } (ID \bmod 31), \end{aligned} \tag{2.3}$$

čia „rotr“ reiškia skaičiaus, dvejetainėje skaičiavimo sistemoje, pasukimą į dešinę per nurodytą skaičių pozicijų. Pavyzdžiui,

$$00001111_2 \text{ rotr } 3 = 11100001_2. \tag{2.4}$$

S5. Procesoriaus vidinio laikmačio vertė ($Time$), laikmačio pertraukimų skaičius ($Tics$) ir procesoriaus ID yra kombinuojami pagal formulę

$$seed_{S5} = 2 \cdot Time \cdot Tics \cdot 2^{id \bmod 15} \bmod 2^{30} - 1. \tag{2.5}$$

## 2.2 Paieškos srities siaurinimas PSO algoritme

Šiame poskyryje siūloma dalelių spiečiaus (PSO) algoritmo (žr. 1.3.2 poskyrį) modifikacija [A2], grįsta prielaida, kad mažesnėje paieškos srityje lengviau randamas uždavinio globalusis sprendinys.

Srities mažinimas atliekamas apibrėžiant naują pradinės paieškos srities $D$ posritį $\tilde{D} \subset D$. Tarkime, kad pradinė paieškos sritis

$$D = [x_1^{LB}, x_1^{UB}] \times [x_2^{LB}, x_2^{UB}] \times \ldots \times [x_d^{LB}, x_d^{UB}], \tag{2.6}$$

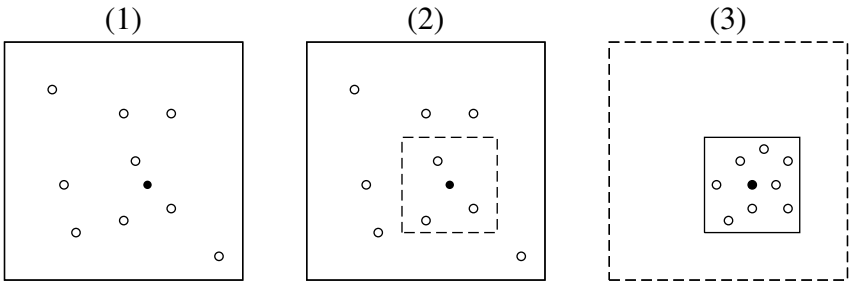
čia $x_1^{LB}, x_2^{LB}, \ldots, x_d^{LB}$ ir $x_1^{UB}, x_2^{UB}, \ldots, x_d^{UB}$ – atitinkamai mažiausios ir didžiausios leistinosios kintamųjų $x_1, x_2, \ldots, x_d$ reikšmės, $d$ – kintamųjų skaičius. Nauja paieškos sritis

$$\tilde{D} = [\tilde{x}_1^{LB}, \tilde{x}_1^{UB}] \times [\tilde{x}_2^{LB}, \tilde{x}_2^{UB}] \times \ldots \times [\tilde{x}_d^{LB}, \tilde{x}_d^{UB}] \tag{2.7}$$

apskaičiuojama pagal formulę

$$\begin{aligned} \tilde{x}_i^{LB} &= p_i^g - \delta_i, \\ \tilde{x}_i^{UB} &= p_i^g + \delta_i, \\ \delta_i &= c_r(x_i^{UB} - x_i^{LB}), \end{aligned} \tag{2.8}$$

čia $p_i^g$ – geriausio žinomo sprendinio $i$-tojo parametro reikšmė, $c_r \in (0, 1)$ –




2.2 pav.: Paieškos srities siaurinimas PSO algoritme

paieškos srities mažinimo koeficientas, $i = 1, 2, \ldots, d$. Mažinimo koeficientas $c_r$ nurodo naujosios srities dydį – kuo mažesnė $c_r$ reikšmė, tuo mažesnė nauja paieškos sritis $\tilde{D}$. Kadangi $\tilde{D}$ turi būti srities $D$ posritis, turi galioti

$$x_i^{LB} \leq \tilde{x}_i^{LB} < \tilde{x}_i^{UB} \leq x_i^{UB}. \tag{2.9}$$

Tuo atveju, jei kuriam nors $j \in \{1, 2, \ldots, d\}$ galioja $\tilde{x}_j^{LB} \leq x_j^{LB}$, tai $j$-ojo kintamojo mažiausia leistinoji reikšmė naujojoje paieškos srityje prilyginama to paties kintamojo mažiausiai leistinajai reikšmei pradinėje paieškos srityje: $\tilde{x}_j^{LB} \leftarrow x_j^{LB}$. Analogiškai elgiamasi ir tuo atveju, jei $\tilde{x}_j^{UB} \geq x_j^{UB}$. Tuomet $j$-ojo kintamojo didžiausia leistinoji reikšmė naujojoje paieškos srityje prilyginama to paties kintamojo didžiausiai leistinajai reikšmei pradinėje paieškos srityje: $\tilde{x}_j^{UB} \leftarrow x_j^{UB}$.

Paieškos srities mažinimas vykdomas po numatyto skaičiaus $E_G$ tikslo funkcijų skaičiavimų. Naujoje paieškos srityje atsitiktiniu būdu sugeneruojamas naujas dalelių spiečius, kuris bus naudojamas tolesniam optimizavimui. Paieškos srities mažinimo procedūra iliustruota 2.2 paveiksle.

## 2.3 MOSASS algoritmas

Šiame poskyryje nagrinėjami daugiakriterio optimizavimo algoritmai: siūloma SASS algoritmo (žr. 1.3.1 skyrelį), skirto vieno kriterijaus funkcijų lokaliajam optimizavimui, modifikacija, pritaikant algoritmą daugiakriteriam lokaliajam optimizavimui [A3]. Modifikuota algoritmo versija vadinama Daugiakritere vieno agento stochastine paieška (angl. *Multi-objective Single Agent Stochastic Search*, MOSASS).

MOSASS algoritmas, kaip ir jo pirmtakas SASS, turi tuos pačius įvesties parametrus: pradinį sprendinį $\mathbf{x}$, standartinio nuokrypio didinimo ir mažinimo koeficientus $ex$ ir $ct$, viršutinį ir apatinį standartinio nuokrypio rėžius

$\sigma_{LB}$ ir $\sigma_{UB}$, bei parametrus $Scnt$ ir $Fcnt$. Taip pat MOSASS algoritmui nurodomi ir du nauji įvesties parametrai: archyvas $A$, kuriame bus saugomi rasti nedominuojami sprendiniai ir maksimalus archyvo dydis $N_A$. Algoritmo vykdymo pradžioje archyvas $A$ – tuščias, tačiau galima jam priskirti jau žinomus nedominuojamus sprendinius.

Algoritmas pradedamas nuo naujo sprendinio $\mathbf{x}'$ generavimo pagal (1.16) formulę. Toliau elgiamasi priklausomai nuo sugeneruoto sprendinio $\mathbf{x}'$ ir kitų žinomų sprendinių dominuojamumo sąryšio:

- jei $\mathbf{x}' \succ \mathbf{x}$, tai sprendinys $\mathbf{x}$ yra pakeičiamas naujuoju sprendiniu $\mathbf{x}'$ ir pradedama nauja iteracija.
- jei $\mathbf{x}'$ nedominuoja savo pirmtako $\mathbf{x}$, bet tuo pačiu nėra dominuojamas nei sprendiniu $\mathbf{x}$, nei bet kuriuo archyve esančiu sprendiniu,

$$\nexists \mathbf{y} \in \{\mathbf{x}\} \cup A \mid \mathbf{y} \succ \mathbf{x}', \tag{2.10}$$

  tai archyvas papildomas naujuoju sprendiniu, pašalinami tie archyvo sprendiniai, kuriuos dominuoja $\mathbf{x}'$,

$$A \leftarrow \left(A \cup \{\mathbf{x}'\}\right) \setminus \{\mathbf{y} \in A : \mathbf{x}' \succ \mathbf{y}\} \tag{2.11}$$

  ir pradedama nauja iteracija.
- jei $\mathbf{x}'$ yra dominuojamas bent vienu iš žinomų sprendinių,

$$\exists \mathbf{y} \in \{\mathbf{x}\} \cup A \mid \mathbf{y} \succ \mathbf{x}', \tag{2.12}$$

  tai $\mathbf{x}'$ yra atmetamas ir priešingas sprendinys $\mathbf{x}''$ (žr. (1.18) formulę) yra tiriamas analogišku būdu.

Jei iteracijos metu randamas sprendinys, dominuojantis $\mathbf{x}$ arba archyvas yra papildomas, tai iteracija laikoma sėkminga, priešingu atveju – nesėkminga.

Jei archyvo dydis pasiekia ribą $N_A$, tai pusė jo narių yra pašalinami, analogiškai kaip algoritme NSGA-II algoritme [63].

Vidurkio ir standartinio nuokrypio parametrai yra dinamiškai keičiami kaip ir algoritme SASS (žr. 1.3.1 skyrelį).

Naujo sprendinio parinkimo strategija gali turėti didelę įtaką atsitiktinės paieškos kokybei. Tiek SASS, tiek ir MOSASS algoritme naujas sprendinys generuojamas prie sprendinio $\mathbf{x}$ pridedant tokio paties dydžio atsitiktinių reikšmių vektorių. Vadinasi tikimybė, kad naujas sprendinys $\mathbf{x}'$ (arba $\mathbf{x}''$)

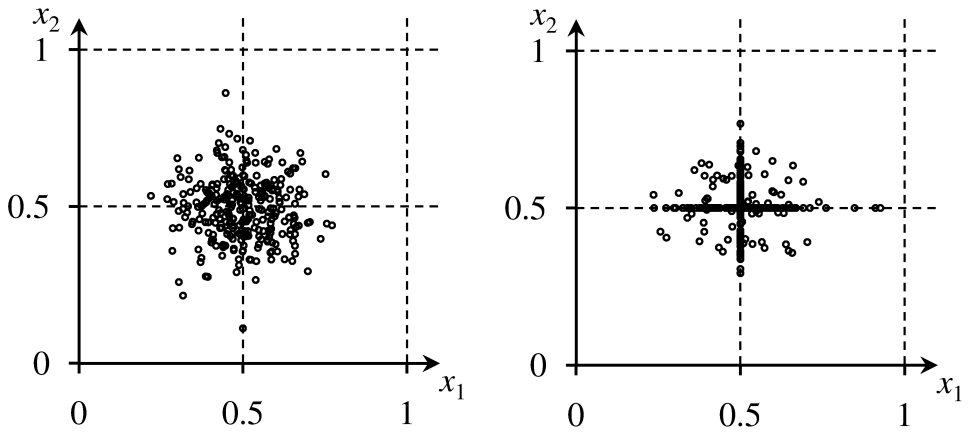
skirsis nuo savo pirmtako $\mathbf{x}$ pagal visus parametrus, yra artima 1. Tačiau neretai tereikia pakeisti tik vieno ar kelių parametrų reikšmes, kad gautasis sprendinys dominuotų savo pirmtaką ar, bent jau, būtų nedominuojamas kitų žinomų sprendinių. Tai ypatingai aktualu vėlyvesnėse algoritmo stadijose, kai turima gan tiksli Pareto fronto aproksimacija. Siekiant patvirtinti (arba paneigti) šią hipotezę, MOSASS algoritmas modifikuojamas įvedant tikimybę naujo sprendinio generavime – kiekviena sprendinio $\mathbf{x}$ koordinatė keičiama su tikimybe $\pi_m$:

$$\xi_i = \begin{cases} N(b_i, \sigma_i), & \text{jei} \quad r < \pi_m, \\ 0, & \text{kitu atveju.} \end{cases} \tag{2.13}$$

Čia $r$ atsitiktinis skaičius iš intervalo $[0, 1]$, o $\pi_m \in (0, 1]$ – koordinatės keitimo tikimybė. Didesnė $\pi_m$ reikšmė reiškia didesnę koordinatės keitimo tikimybę.

Modifikuota MOSASS algoritmo versija toliau bus vadinama MOSASS/P. Esminis skirtumas tarp abiejų generavimo metodų yra iliustruotas 2.3 paveiksle, kur pavaizduoti labiausiai tikėtini nauji sprendiniai $\mathbf{x}'$ algoritmuose MOSASS (kairėje) ir MOSASS/P (dešinėje). Čia laikoma, kad $x_i = 0.5$, $p = 0.5$, $\sigma = 0.1$, $b_i = 0$, $i = 1, 2$.

Kartu su koordinačių keitimo tikimybės įvedimu atsiranda ir tikimybė, kad nė vienas sprendinys nebus pakeistas, t.y. $\mathbf{x}' = \mathbf{x}$. Tokiu atveju naujo sprendinio generavimo procedūra pakartojama.




2.3 pav.: Sprendinio generavimas MOSASS ir MOSASS/P algoritmuose

### *Hibridinis globaliojo optimizavimo algoritmas*

Lokaliojo daugiakriterio optimizavimo algoritmas (MOSASS arba MOSASS/P) gali būti taikomas kaip lokaliosios paieškos strategija globaliojo daugiakriterio optimizavimo algoritme, pavyzdžiui, NGSA-II (žr. 1.3.4 poskyrį). Vienas būdų tai padaryti – nedominuojamus sprendinius, rastus algoritmu NSGA-II naudoti pradiniais MOSASS sprendiniais. Lokalioji paieška gali būti vykdoma NSGA-II algoritmo pabaigoje arba kelis kartus, tarkime, kas $E_G$ tikslo funkcijų perskaičiavimų. Tokiu būdu sudaromas memetinis algoritmas, kurį vadinsime NSGA-II/LS arba NSGA-II/LSP, priklausomai nuo naudojamo lokaliosios paieškos algoritmo: MOSASS ar MOSASS/P.

Siūlomas hibridinis algoritmas pradedamas nuo NSGA-II algoritmo iteracijų vykdymo: iš $N$ individų tėvų populiacija $P^{(k-1)}$ sudaroma palikuonių populiacija $Q^{(k-1)}$, o iš populiacijų sąjungos $P^{(k-1)} \cup Q^{(k-1)}$ atrenkama nauja populiacija $P^{(k)}$, kuri bus naudojama kitoje iteracijoje (žr. 4 poskyrį). Čia $k$ – algoritmo iteracijos numeris. Kiekvienos iteracijos metu reikia įvykdyti $N$ tikslo funkcijų reikšmių skaičiavimų, vadinasi po kiekvienos iteracijos atnaujinamas funkcijų reikšmių skaičiavimų skaitliukas $E$ (algoritmo pradžioje $E = 0$):

$$E \leftarrow E + N. \tag{2.14}$$

Jei $E \geq E_G$, tai vykdoma lokalioji paieška. Iš turimos sprendinių populiacijos, pažymėkime ją $P'$, lokaliajam optimizavimui parenkamas $k$ sprendinių poaibis $P_L$. Pirmiausia poaibiui $P_L$ priskiriami visi populiacijos $P'$ nedominuojami sprendiniai. Jei $|P_L| > k$ tai $|P_L| - k$ atsitiktinai parinktų individų yra pašalinami iš $P_L$. Priešingu atveju, jei $|P_L| < k$, tai pridedama $k - |P_L|$ atsitiktinai parinktų dominuojamų $P$ individų. Kiekvienas poaibyje $P_L$ esantis sprendinys yra lokaliai optimizuojamas algoritmu MOSASS arba MOSASS/P. Kiekvieno sprendinio optimizavimui skiriama po $E_L$ tikslo funkcijos reikšmių skaičiavimų.

Lokaliojo optimizavimo algoritmas grąžina aibę nedominuojamų sprendinių. Kadangi yra optimizuojami visi aibėje $P_L$ esantys sprendiniai, tai bendras lokaliojo optimizavimo rezultatas bus $|P_L|$ nedominuojamų sprendinių aibių. Visos šios aibės, kartu su populiacija $P$, yra apjungiamos į vieną bendrą populiaciją $R$, iš kurios yra išrenkamas $N$ mažiausiai dominuojamų sprendinių poaibis, kuris bus naudojamas kaip tėvų populiacija kitoje iteracijoje. Sprendinių išrinkimui naudojama ta pati strategija kaip NSGA-II algoritme (žr. 1.3.4 poskyrį).

## 2.4 NSGA-II algoritmo taikymas CFL uždaviniui spręsti

Šiame skyrelyje aptariamas algoritmo NSGA-II (žr. 1.3.4 skyrelį) taikymas konkuruojančių objektų vietos parinkimo (CFL) (žr. 1.2.2 skyrelį) uždaviniui spręsti.

Pagal uždavinio specifiką, nauji objektai gali būti lokalizuojami skirtinguose paklausos taškuose, kurie gali būti parenkami iš baigtinės kandidatų aibės $L = \{l_1, l_2, \ldots, l_r\}$. Vadinasi uždavinio paieškos sritis $D$ gali būti aprašoma kaip aibė visų $L$ poaibių $X$, tokių, kad $|X| = s$:

$$D = \{X \subset L : |X| = s\}, \tag{2.15}$$

čia $s$ – uždavinio kintamųjų skaičius (skaičius objektų, kuriems norima parinkti vietas).

Pagal klasikinio NSGA-II algoritmo schemą (žr. 1.3.4 poskyrį) algoritmas pradedamas nuo pradinės $N$ dydžio tėvų populiacijos $P^{(0)}$ generavimo, šiuo atveju, $N$ skirtingų aibės $L$ poaibių, tenkinančių (2.15) nelygybę, parinkimo.

Kiekvienas palikuonių populiacijos $Q^{(k-1)}$ individas $X^{(3)}$ generuojamas kryžminant du tėvų populiacijos $P^{(k-1)}$ individus, pažymėkime juos $X^{(1)}$ ir $X^{(2)}$. Čia $k$ – algoritmo iteracijos numeris. Kryžminimas atliekamas atsitiktiniu būdu parenkant $s$ individų poaibį

$$X^{(3)} \subset (X^{(1)} \cup (X^{(2)} \setminus X^{(1)})). \tag{2.16}$$

Individų kryžminimas yra atliekamas su iš anksto nurodyta tikimybe $\pi_c$. Tuo atveju, jei individų kryžminimas nevykdomas, nauju individu $X^{(3)}$ priimamas tėvų populiacijos individas $X^{(1)}$.

Antrasis palikuonių populiacijos individo generavimo etapas yra mutavimas. Kiekvienas individo $X^{(3)}$ kintamasis $x_i^{(3)}$, $i = 1, 2, \ldots, s$ yra pakeičiamas į

$$l \in (L \setminus x_i^{(3)}), \tag{2.17}$$

su iš anksto nurodyta tikimybe $\pi_m$.

Nauja kintamojo reikšmė $l$ gali būti parenkama atsitiktinai iš visų galimų kandidatų $L$, tačiau tokia strategija gali skatinti gana chaotišką mutavimą. Siekiant labiau sistemingo mutavimo, reikšmė $l$ gali būti parenkama iš ma-

žesnės aibės

$$L^{(h)} = \{l \in L \setminus x_i^{(3)} : d(l, x_i^{(3)}) \leq d^{(h)}\}, \tag{2.18}$$

čia $d(l, x_i^{(3)})$ – atstumas tarp paklausos taškų, kuriuos reprezentuoja $l$ ir $x_i^{(3)}$, o $d^{(h)}$ – atstumas tarp paklauso taško $x_i^{(3)}$ ir $h$-tojo artimiausio paklausos taško. Kitaip tariant, paklausos taškas, kurį reprezentuoja $x_i^{(3)}$ keičiamas kitu, atsitiktiniu būdu parinktu iš $h$ artimiausių paklausos taškų. Parametro $h$ reikšmė gali būti nuo 1 iki $|L| - s - 1$.

### *Lokaliosios paieškos strategija*

Pasiūlysime lokaliosios paieškos strategiją, skirtą dagiakriteriams CFL uždaviniams spręsti. Nors siūloma strategija gali būti taikoma ir daugelio kriterijų CFL uždaviniams spręsti, nemažindami bendrumo toliau nagrinėsime dviejų kriterijų CFL uždavinius.

Siūloma lokaliosios paieškos strategija yra paremta kaimyninio sprendinio atsitiktine paieška. Strategija yra skirta patikslinti turimą Pareto fronto aproksimaciją $\tilde{P}$, tačiau gali būti pradedama ir nuo atsitiktinio sprendinio. Kiekviena algoritmo iteracija vykdoma tokiais etapais:

1. Iš turimos Pareto fronto aproksimacijos $\tilde{P}$ atsitiktiniu būdu parenkamas sprendinys $X$.

2. Generuojamas naujas kaimyninis sprendinys $X'$ naujų objektų vietas parenkant iš aibės sudarytos pagal (2.18) formulę. Kiekvienas sprendinio $X$ kintamasis keičiamas su tikimybe $1/d$, čia $d$ – uždavinio kintamųjų skaičius. Tuo atveju, jei nė vienas kintamasis nebuvo pakeistas ($X = X'$), naujo sprendinio generavimas pakartojamas iš naujo.

3. Skaičiuojamos sprendinio $X'$ tikslo funkcijų reikšmės ir nustatomas dominuojamumo sąryšis tarp $X'$ ir visų sprendinių, esančių aibėje $\tilde{P}$.

4. Jei $X'$ nėra dominuojamas nė vieno sprendinio iš $\tilde{P}$, tai aibė $\tilde{P}$ yra papildoma naujuoju sprendiniu $X'$, o tie aibės $\tilde{P}$ sprendiniai, kuriuos dominuoja $X'$ – pašalinami:

$$\tilde{P} \leftarrow (\tilde{P} \cup \{x'\}) \setminus \{y \in \tilde{P} : x' \succ y\}. \tag{2.19}$$

   Jei aibės $\tilde{P}$ dydis pasiekia maksimalią ribą $N_P$, tai aibė $\tilde{P}$ yra sumažinama, pašalinant pusę sprendinių, labiausiai „apgaubtų" to paties rango sprendinių, analogiškai kaip NSGA-II algoritme (žr. 1.3.4 poskyrį).

5. Jei sustojimo kriterijus patenkintas, tai darbas baigiamas. Priešingu atveju, einama į 1-ąjį žingsnį.

Yra keturi algoritmo įvesties parametrai: pradinė Pareto fronto aproksimacija $\tilde{P}$, maksimalus aproksimacijos dydis $N_A$, algoritmo sustojimo kriterijus ir parametras $h$.

Sustojimo kriterijai gali būti įvairūs: maksimalus iteracijų skaičius, tikslo funkcijų reikšmių perskaičiavimų skaičius, maksimalus algoritmo vykdymo laikas ir pan.

## 2.5 Lygiagretieji algoritmai

### 2.5.1 Lygiagretusis PSO algoritmas

Kiekvienos generacijos metu PSO algoritmas perskaičiuoja visų populiacijos individų tikslo funkcijos reikšmes, todėl uždavinio sprendimas gali užtrukti, jei tikslo funkcijos reikšmių skaičiavimas reikalauja daug laiko sąnaudų. Kita vertus, PSO, kaip ir daugelis kitų populiacija paremtų optimizavimo algoritmų, gali būti nesudėtingai pritaikomas lygiagrečiųjų skaičiavimų sistemoms.

Kadangi didžioji skaičiavimo resursų dalis tenka tikslo funkcijos reikšmių skaičiavimui, natūraliausia PSO algoritmo lygiagretinimo strategija – skirtingų populiacijos individų tikslo funkcijų reikšmių skaičiavimo padalinimas procesoriams.

Tarkime, kad skaičiavimams yra naudojama $p$ procesorių. Tuomet minėtas lygiagretusis PSO algoritmas gali būti aprašomas tokiais žingsniais:

1. Visi $p$ procesorių generuoja dalelių populiacijos $P$ dalį $P_i$ taip, kad

$$|\bigcup_{i=1}^{p} P_i| = N, \tag{2.20}$$

   čia $i$ – procesoriaus ID, $i = 1, 2, \ldots, p$.

2. Visi $p$ procesoriai skaičiuoja atitinkamos populiacijos dalies $P_i$ sprendinių tikslo funkcijos reikšmes.

3. Visi procesoriai apsikeičia informacija apie geriausią rastą sprendinį.

4. Jei algoritmo sustojimo sąlyga patenkinta, tai einama į 7-tąjį žingsnį. Priešingu atveju einama į 5-ąjį žingsnį.

5. Kiekvienas $i$-tasis procesorius, turėdamas informaciją apie geriausią visų procesorių rastą sprendinį $\mathbf{p}^g$, atnaujina populiacijos $P_i$ dalelių pozicijas.

6. Einama į 2-ąjį žingsnį.

7. Baigiamas algoritmo darbas, uždavinio sprendiniu laikant geriausią visų procesorių rastą sprendinį.

Kiekvienos populiacijos dalies $P_i$ dydis gali skirtis, priklausomai nuo $i$-tojo procesoriaus našumo – homogeninėse lygiagrečiųjų skaičiavimų sistemose visiems procesoriams priskiriamos vienodo dydžio populiacijos dalys:

$$P_i = P_j, \tag{2.21}$$

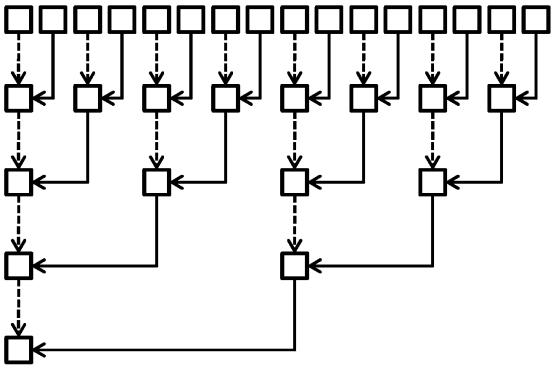
čia $i, j = 1, 2, \ldots, p$. Tuo tarpu heterogeninėse skaičiavimo sistemose populiacijos dalies $P_i$ dydis priklauso nuo $i$-tojo procesoriaus našumo – našesniems procesoriams priskiriamos didesnės populiacijos dalys, mažiau našiems – atitinkamai mažesnės dalys.

Kiekvienos generacijos metu visi skaičiavimus vykdantys procesoriai turi apsikeisti informacija apie geriausią rastą sprendinį: poziciją bei tikslo funkcijos reikšmę. Tačiau dažnas procesorių komunikavimas gali būti vienu iš faktorių, labiausiai įtakojančiu lygiagrečiojo algoritmo spartinimo koeficientą [64]. Todėl tinkamo komunikavimo tarp procesorių strategijos parinkimas gali būti svarbus sudarant lygiagretųjį algoritmą [65, 66].

Aptarsime 4 komunikavimo tarp procesorių lygiagrečiajame PSO algoritme strategijas: sinchroninę centralizuotą (sC), visi visiems (VV), hierarchinės sklaidos (HS) ir asinchroninę centralizuotą (aC).

**Sinchroninė centralizuota strategija**. Vienas procesorius yra išskiriamas kaip procesorius-šeimininkas, kuris be jam priskiriamo skaičiavimo darbo papildomai kontroliuoja informacijos mainus. Visi procesoriai-darbininkai, baigę iteracijai skirtą darbą, siunčia informaciją apie geriausią rastą sprendinį (koordinates ir tikslo funkcijos reikšmę) procesoriui-šeimininkui. Procesorius-šeimininkas, baigęs iteracijai skirtą darbą, surenka procesorių-darbininkų siunčiamą informaciją, identifikuoja geriausią sprendinį ir paplatina visiems procesoriams-darbininkams.

**Visi visiems**. Kiekvienas procesorius siunčia informaciją apie geriausią rastą sprendinį visiems kitiems procesoriams ir surenka analogišką informaciją iš visų procesorių. Kiekvienas procesorius iš surinktų sprendinių atsirenka geriausią, kuris bus naudojamas kaip $\mathbf{p}^g$ kituose žingsniuose.



2.4 pav.: Hierarchinė informacijos surinkimo iš procesorių strategija

**Hierarchinė sklaida**. Duomenys $p$ procesoriams padalinami pagal 2.4 paveiksle pavaizduotą schemą. Visi $p$ procesorių yra padalinami į dvi grupes po $p/2$ procesorių: procesorius-siuntėjus ir procesorius-gavėjus. Procesoriai, kurių ID yra nelyginiai, priskiriami procesoriams-gavėjams, o procesoriai, kurių ID yra lyginiai, – procesoriams-siuntėjams. Visi procesoriai-siuntėjai siunčia informaciją apie geriausią žinomą sprendinį $\mathbf{p}_i^g$ (čia $i$ – procesoriaus ID) procesoriui-gavėjui, kurio ID yra vienetu mažesnis. Tuo tarpu procesoriai-gavėjai, kurių ID yra nelyginiai, išsaugo siunčiamą informaciją kaip $\mathbf{p}_{(r)}^g$ ir atnaujina informaciją apie geriausią žinomą sprendinį

$$\mathbf{p}_i^g = \begin{cases} \mathbf{p}_{(r)}^g, & \text{jei} \quad f(\mathbf{p}_{(r)}^g) < f(\mathbf{p}_i^g), \\ \mathbf{p}_i^g, & \text{jei} \quad f(\mathbf{p}_{(r)}^g) \geq f(\mathbf{p}_i^g). \end{cases} \tag{2.22}$$

Toliau, visi procesoriai-gavėjai analogiškai suskirstomi į dvi grupes: procesorius-siuntėjus (procesoriai, kurių ID yra $3, 7, 11, \ldots$) ir procesorius-gavėjus (procesoriai, kurių ID yra $2, 5, 9, \ldots$), ir kartojama analogiška duomenų persiuntimo procedūra. Galiausiai $\log_2 p$-tame žingsnyje procesoriams-siuntėjams ir procesoriams-gavėjams priklausys po vieną procesorių – procesorių kurio ID yra 1 ir procesorių, kurio ID yra $p/2 + 1$.

Naudojant hierarchinės sklaidos strategiją, duomenys iš $p$ kompiuterių surenkami naudojant $\log_2 p$ persiuntimo operacijų. Tiek pat operacijų reikalauja ir duomenų paskleidimas. Tuo tarpu, naudojant anksčiau aprašytas strategijas, duomenų surinkimas ir persiuntimas reikalauja po $p-1$ persiuntimo operacijų.

Visos aukščiau aprašytos strategijos yra sinchroninės – visi kompiuteriai pradeda generaciją turėdami naujausią informaciją apie geriausią rastą sprendinį.

Toliau aptarsime asinchroninę strategiją, negarantuojančią, kad kiekvienas kompiuteris pradėdamas iteraciją turės naujausią informaciją, tačiau sumažinančią duomenų perdavimo kaštus.

**Asinchroninė šeimininko-darbininko**. Ši strategija yra panaši į sinchroninę darbininko-šeimininko strategiją, tik šiuo atveju procesorius-šeimininkas neturi skaičiavimo darbo – atlieka tik duomenų mainų kontrolės funkciją. Procesorius-darbininkas, baigęs generacijai skirtą darbą persiunčia informaciją apie geriausią jam žinomą sprendinį ir laukia informacijos apie procesoriui-šeimininkui žinomą geriausią sprendinį. Procesorius-šeimininkas priima informaciją, atsižvelgdamas į gautus duomenis atnaujina jam žinomą informaciją ir išsiunčią ją atgal, procesoriui-darbininkui, laukiančiam atnaujintos informacijos.

### 2.5.2 Lygiagretusis NSGA-II

NSGA-II algoritmas gali būti skaidomas į 3 pagrindines dalis (žr. 2.5 paveikslą): tikslo funkcijų reikšmių skaičiavimai; sprendinių rangavimas; kitos operacijos. Dauguma literatūroje [67, 68, 69, 70] aprašomų NSGA-II algoritmo lygiagretinimo strategijų remiasi pirmosios dalies lygiagretinimu – tikslo funkcijų reikšmių skaičiavimų paskirstymu procesoriams.

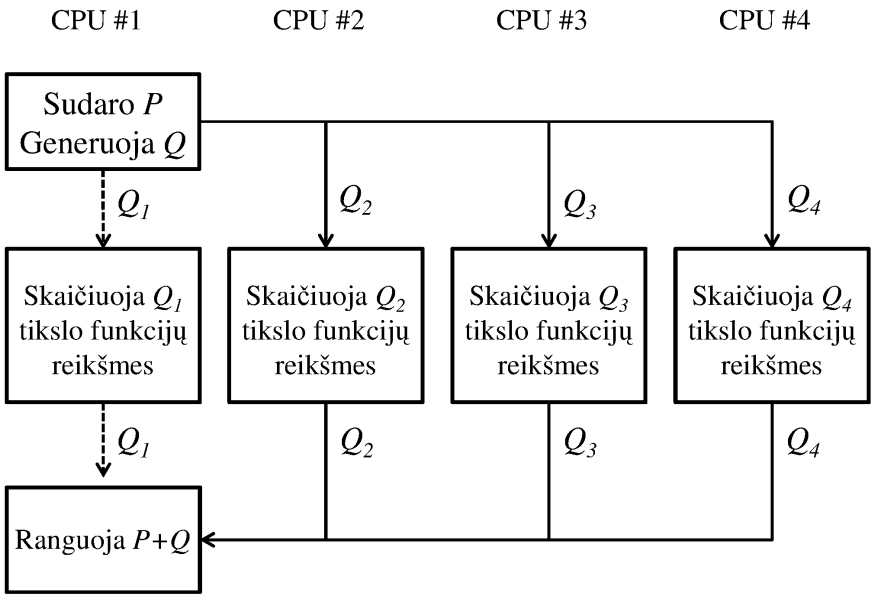
Kiekvienos generacijos metu procesorius-šeimininkas sugeneruoja palikuonių populiacija $Q$, suskaido ją į $p$ poaibių $Q_1, Q_2, \ldots, Q_p$ taip, kad

$$\bigcup_{i=1}^{p} Q_i = Q, \tag{2.23}$$

kuriuos padalina procesoriams-darbininkams, vieną pasilikdamas sau. Kiekvienas $i$-tasis procesorius apskaičiuoja poaibio $Q_i$ individų tikslo funkcijų reikšmes ir siunčia rezultatus procesoriui-šeimininkui. Šeimininkas, baigęs savo darbo dalį, surenka rezultatus iš visų darbininkų, sujungia populiacijas $P$ ir $Q$ į vieną populiaciją $R$, paskaičiuoja sprendinių dominuojamumo rangus ir pašalina pusę sprendinių – tuos, kurių dominuojamumo rangai didžiausi. Šio algoritmo, kuris toliau bus vadinamas ParNSGA/PE, schema pavaizduota 2.6 paveiksle.

| (1) | (2) | (3) |
|---|---|---|
| Tikslo funkcijų reikšmių skaičiavimai | Sprendinių rangavimas | Kitos operacijos |

2.5 pav.: Trys pagrindinės NSGA-II algoritmo dalys




2.6 pav.: Algoritmo ParNSGA/PE schema

Nors ši strategija yra paprastai realizuojama, ji pasiteisina tik sprendžiant uždavinius, kuriuose tikslo funkcijų reikšmių skaičiavimas reikalauja daug laiko, lyginant su kitomis algoritmo operacijomis. Pavyzdžiui, jei funkcijų reikšmių skaičiavimai sudaro 99 % viso algoritmo vykdymo laiko, tai, pagal Amdalo dėsnį (žr. (1.43) formulę), didžiausias galimas algoritmo spartinimo koeficientas būtų 100, nepriklausomai nuo skaičiavimams naudojamų procesorių. Tačiau, jei funkcijų skaičiavimams užima tik 95 % arba 90 % visos algoritmo vykdymo trukmės, didžiausias galimas algoritmo spartinimo koeficientas būtų atitinkamai 20 ir 10, nepriklausomai nuo naudojamų procesorių skaičiaus.

### *Sprendinių rangavimo lygiagretinimas*

Kita NSGA-II algoritmo dalis, reikalaujanti sąlyginai daug skaičiavimo resursų yra sprendinių rangavimas. Todėl šios dalies optimizavimas ir lygiagretinimas gali ženkliai padidinti algoritmo spartinimo koeficientą.

Kiekvienoje NSGA-II algoritmo iteracijoje reikia suranguoti $2N$ dydžio populiacijos $R$ individus. Populiacija $R$ yra sudaryta iš dviejų $N$ dydžio populiacijų $P$ ir $Q$. Vadinasi reikia suskaičiuoti kiek dominatorių turi kiekvienas populiacijos $P$ individas tarp populiacijų $P$ ir $Q$ individų, ir kiek dominatorių turi kiekvienas populiacijos $Q$ individas tarp tų pačių populiacijų $P$ ir $Q$ individų. Vektorių, aprašantį aibės $P$ individų dominatorių

aibėje $Q$ skaičius, pažymėkime

$$r(P,Q) = (r_1, r_2, \ldots, r_{|P|}), \tag{2.24}$$

čia

$$r_i = |\{\mathbf{y} \in Q : \mathbf{y} \succ \mathbf{x}_i, \mathbf{x}_i \in P\}|, \tag{2.25}$$

o $i = 1, 2, \ldots, |P|$. Analogiškos prasmės bus ir žymėjimų $r(Q,P)$, $r(P,P)$ bei $r(Q,Q)$. Tada visų aibės $P$ individų rangus aprašo vektorių suma

$$r(P,P) + r(P,Q), \tag{2.26}$$

o visų aibės $Q$ individų rangus aprašo vektorių suma

$$r(Q,Q) + r(Q,P). \tag{2.27}$$

Skaičiuojant $r(P,Q)$, turi būti nustatomas kiekvieno aibės $P$ individo dominuojamumo sąryšis su kiekvienu aibės $Q$ individu. Vadinasi turi būti atliekama $N^2$ dominuojamumo sąryšio nustatymo operacijų. Tiek pat operacijų turi būti atlikta ir skaičiuojant $r(Q,P)$. Tuo tarpu vektorių $r(P,P)$ ir $r(Q,Q)$ skaičiavimui reikia po $N(N-1)$ dominuojamumo sąryšio nustatymo operacijų, kadangi nebūtina lyginti individo su juo pačiu. Vadinasi visos aibės $P \bigcup Q$ rangavimui reikia atlikti

$$2N^2 + 2N(N-1) = 2N(2N-1) \tag{2.28}$$

dominuojamumo sąryšio nustatymo operacijų.

**Teiginys**. Jei sprendinys $\mathbf{x}$ dominuoja sprendinį $\mathbf{y}$, tai sprendinio $\mathbf{x}$ rangas bus griežtai mažesnis už sprendinio $\mathbf{y}$ rangą.

Teiginio įrodymas seka iš dominavimo sąryšio tranzityvumo savybės. Jo teisingumą, dviejų tikslo funkcijų atveju, iliustruoja 2.7 paveikslas.

Iš teiginio seka, kad didžiausią rangą turinčio sprendinio pašalinimas iš populiacijos nepakeis likusių sprendinių rangų (žr. 2.7 paveikslą). Vadinasi, vektoriaus $r(P,P)$ skaičiavimas yra būtinas tik pirmoje NSGA-II algoritmo generacijoje, rangavimą vykdant pirmą kartą. Kitose generacijose $r(P,P)$ skaičiavimas yra nebūtinas, nes populiacija $P$ yra sudaroma iš populiacijos $R$ pašalinant didžiausius rangus turinčius individus. Tokiu būdu dominuojamumo sąryšio nustatymo operacijų skaičius, reikalingas atlikti $2N$ dydžio populiacijos $R$ individų rangavimui gali būti sumažinamas $N(N-1)$ dominuojamumo sąryšio nustatymo operacijų.

Remiantis šia teiginio išvada yra siūloma sprendinių rangavimo lygia-

gretinimo strategija. Populiacija $P$ yra paplatinama visiems procesoriams, taip, kad kiekvienas procesorius turėtų po pilną populiacijos $P$ kopiją. Populiacija yra platinama naudojant hierarchinę strategiją (žr. 2.8 pav.). Tokiu būdu populiaciją galima perduoti $p$ skaičiavimo mazgams $\log_2 p$ žingsniais.

Toliau $i$-tasis procesorius generuoja atitinkamą populiacijos $Q$ poaibį $Q_i$ taip, kad

$$Q = \bigcup_i Q_i, \tag{2.29}$$

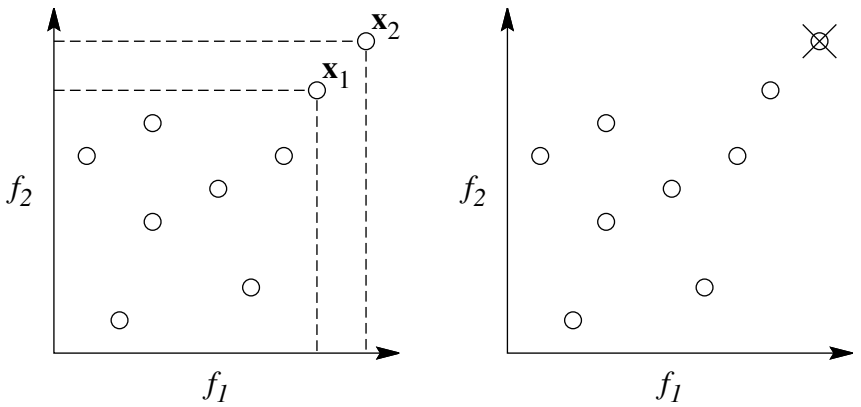
ir paskaičiuoja tikslo funkcijų reikšmes sugeneruotuose taškuose. Kai visi skaičiavimo mazgai baigia savo darbą, šeimininkas surenka duomenis iš darbininkų. Duomenų surinkimui taip pat naudojama hierarchinė strategija (žr. 2.9 pav.), analogiška duomenų skleidimo strategijai.

Populiacijos $Q$ surinkimo metu yra vykdomas dalinis sprendinių rangavimas. Prieš atliekant pirmąjį siuntimo žingsnį, kiekvienas procesorius paskaičiuoja vektorių sumą

$$r_Q = r(Q_i, P) + r(Q_i, Q), \tag{2.30}$$

čia $i$ – procesoriaus numeris. Visi procesoriai su lyginiais ID siunčia populiaciją $Q_i^{(1)}$, kartu su vektorių suma $r_Q$, skaičiavimo mazgui, kurio ID yra $i-1$. Priimantysis skaičiavimo mazgas gautą populiaciją išsaugo kaip $Q'_i$ ir paskaičiuoja $r(Q_i, Q'_i)$, $r(Q'_i, Q_i)$, apjungia abi populiacijas $Q_i^{(1)}$ ir $Q_{i+1}^{(1)}$ į vieną dvigubo dydžio populiaciją

$$Q_i \leftarrow Q_i \cup Q'_i, \tag{2.31}$$




2.7 pav.: Teiginio apie sprendinių nedominuojamumo rangus iliustracija

| 1 etapas | 2 etapas | ... | $\log_2 p$-as etapas |
|---|---|---|---|
| $1 \to 2$ | $1 \to 3$ | ... | $1 \to (p/2+1)$ |
| | $2 \to 4$ | ... | $2 \to (p/2+2)$ |
| | | ... | ... |
| | | ... | $(p/2) \to p$ |

2.8 pav.: Duomenų padalinimo procesoriams schema

ir siunčia kitam procesoriui, laikantis hierarchinės strategijos (žr. 2.9 pav.).

Paskutiniame etape šeimininkas turi pilną suranguotų individų populiaciją $Q$. Papildomai šeimininkas turi paskaičiuoti $r(P, Q)$, kad visų individų rangavimas būtų baigtas.

Šią sprendinių rangavimo strategiją žymėsime HR-1, o lygiagretųjį algoritmą, naudojantį HR-1 strategiją, žymėsime ParNSGA/HR-1.

Lygiagretinimo strategijos schema pavaizduota 2.10 paveiksle.

Pirmajame duomenų surinkimo etape procesoriai turi atlikti

$$\frac{N}{p}\left(\frac{N}{p}-1\right)+\frac{N^2}{p} \tag{2.32}$$

dominuojamumo sąryšio nustatymo operacijų ir persiųsti

$$\frac{d(N+m)}{p}+\frac{N}{p} \tag{2.33}$$

dydžio vektorių. Antrajame etape skaičiavimo mazgai turi atlikti

$$2\left(\frac{N}{p}\right)^2 \tag{2.34}$$

dominuojamumo sąryšio nustatymo operacijų ir persiųsti

$$2\frac{d\,(N+m)}{p}+\frac{N}{p} \tag{2.35}$$

dydžio vektorių.

| 1 etapas | ... | $\log_2 p$-1-as etapas | $\log_2 p$-as etapas |
|---|---|---|---|
| $(p/2+1) \to 1$ | ... | $3 \to 1$ | $2 \to 1$ |
| $(p/2+2) \to 2$ | ... | $4 \to 2$ | |
| ... | ... | | |
| $p \to (p/2)$ | ... | | |

2.9 pav.: Duomenų surinkimo iš procesorių schema

Kiekviename etape dominuojamumo sąryšio nustatymo operacijų skaičius didėja 4 kartus, o perduodamų duomenų kiekis 2 kartus. Po paskutinio duomenų surinkimo etapo, šeimininkas turi papildomai atlikti $N^2$ dominuojamumo sąryšio nustatymo operacijų.

Visame duomenų perdavimo procese kiekvienas skaičiavimo mazgas turi atlikti

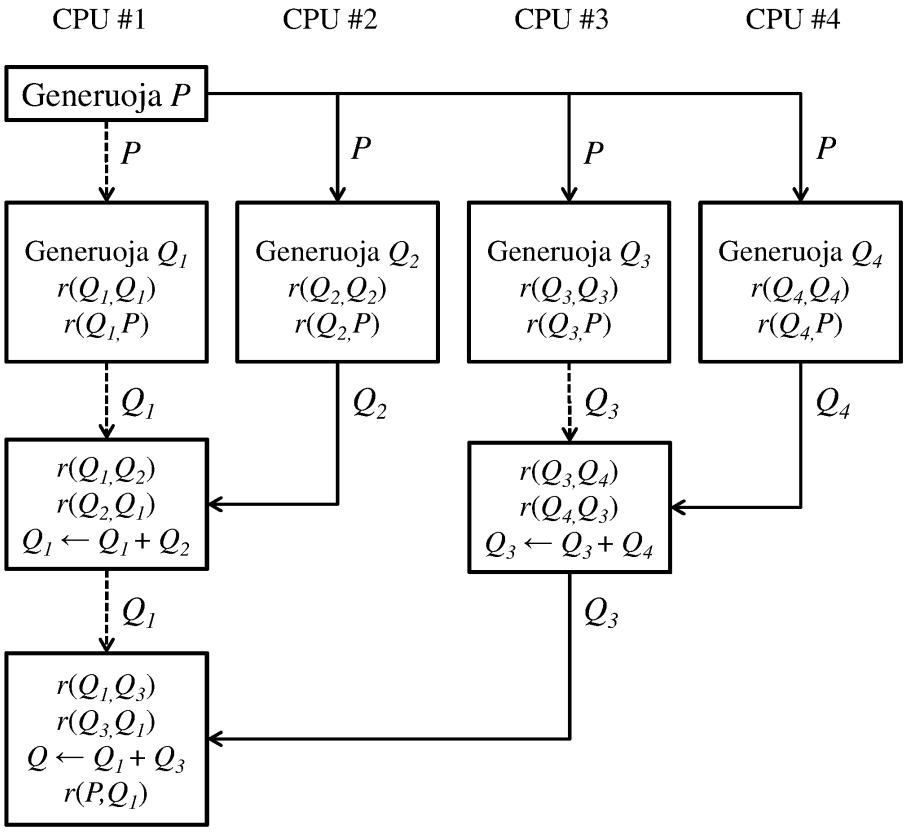
$$\left(\frac{N}{p}\right)^2 + \frac{N}{p} + \frac{N^2}{p} + 2\left(\frac{N}{p}\right)^2 \left(1 + 2 + 4 + 8 + \ldots + \frac{p^2}{4}\right) + N^2 \qquad (2.36)$$

dominuojamumo sąryšio nustatymo operacijų ir persiųsti

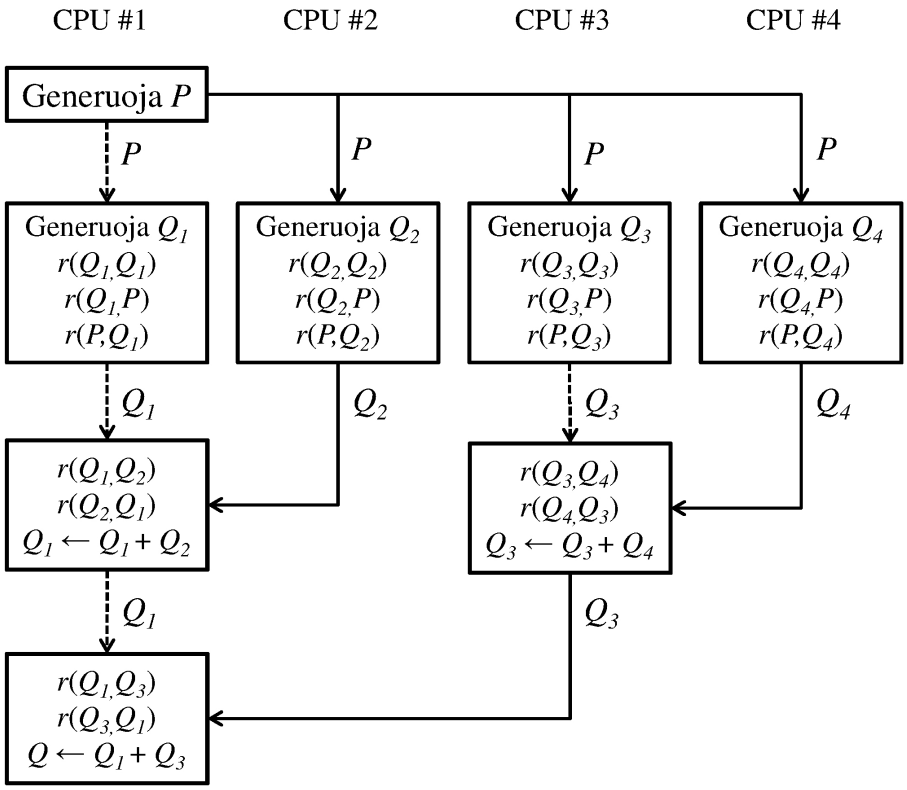
$$\left(d\,(N+m) + \frac{N}{p}\right)(1 + 2 + 4 + \ldots + \log_2 p) \qquad (2.37)$$

realiųjų skaičių.

Pastarojoje strategijoje visi procesoriai turi laukti kol šeimininkas paskaičiuos $r(P,Q)$ paskutiniame informacijos surinkimo etape. Šios procesorių prastovos galima išvengti kiekvienam procesoriui papildomai paskai-




2.10 pav.: Lygiagrečiojo algoritmo ParNSGA/HR-1 schema




2.11 pav.: Lygiagrečiojo algoritmo ParNSGA/HR-2 schema

čiuojant $r(P, Q_i)$, prieš pradedant duomenų surinkimą. Tas reikalauja

$$\frac{N^2}{p} \tag{2.38}$$

dominuojamumo sąryšio nustatymo operacijų. Nors procesoriai turi papildomai persiųsti $N$ dydžio vektorių, saugantį populiacijos $P$ individų dominatorių kiekius, tačiau informacijos surinkimo pabaigoje, šeimininkui nereikia atlikti $N^2$ dominuojamumo sąryšio nustatymo operacijų.

Šią sprendinių rangavimo strategiją žymėsime HR-2, o lygiagretųjį algoritmą, naudojantį HR-2 strategiją, žymėsime ParNSGA/HR-2. Strategijos HR-2 schema yra pavaizduota 2.11 paveiksle.

Strategijose HR-1 ir HR-2 procesoriai persiunčia visą sugeneruotą palikuonių populiaciją, nors generacijos pabaigoje dalis jų yra atmetami. Siekiant optimizuoti perduodamų duomenų kiekį, dalis naujų sprendinių gali būti atmetami dar prieš pradedant informacijos surinkimo procedūrą. Paskaičiavus $r(Q_i, P)$ pirmajame informacijos surinkimo etape, yra žinoma kiek kiekvienas naujas sprendinys turi dominatorių populiacijoje $P$. Kadangi paskutinėje generacijos stadijoje bus pašalinami sprendiniai, kurie turi daugiausiai dominatorių, tai nauji sprendiniai, populiacijoje $P$ turintys

daugiau dominatorių už didžiausią populiacijos $P$ individo rangą, bus pašalinti generacijos pabaigoje. Todėl tokie sprendiniai gali būti atmesti dar prieš pradedant duomenų persiuntimą. Lygiagretųjį algoritmą, įgyvendintą pagal šią strategiją, toliau vadinsime ParNSGA/HR-R.

Nors sprendinių rangavimo lygiagretinimas algoritmuose ParNSGA/HR-1, ParNSGA/HR-2 ir ParNSGA/HR-R nereikalauja didelių komunikacijos kaštų, tačiau procesoriai-darbininkai kurį laiką turi būti be darbo, kol šeimininkas baigs sprendinių rangavimą ir sugeneruos naują populiaciją. Šios procesorių prastovos gali būti nereikšmingos naudojant kelis arba keliasdešimt procesorių, kita vertus tokios prastovos gali sudaryti sąlyginai didelę laiko dalį naudojant kelis šimtus ar tūkstančius procesorių. Palyginimui aptarsime dar vieną sprendinių rangavimo lygiagretinimo strategiją, kuri leidžia išvengti minėtų prastovų, tačiau reikalauja papildomų kaštų duomenų perdavimui. Strategija realizuojama pagal šią schemą:

(1) procesorius-šeimininkas paplatina tėvų populiaciją $P$ visiems procesoriams-darbininkams, naudojant hierarchinę duomenų skleidimo tinkle schemą (žr. 2.8 paveikslą).

(2) Visi procesoriai (įskaitant ir šeimininką) generuoja atitinkamą palikuonių populiacijos $Q$ dalį $Q_i$ taip, kad

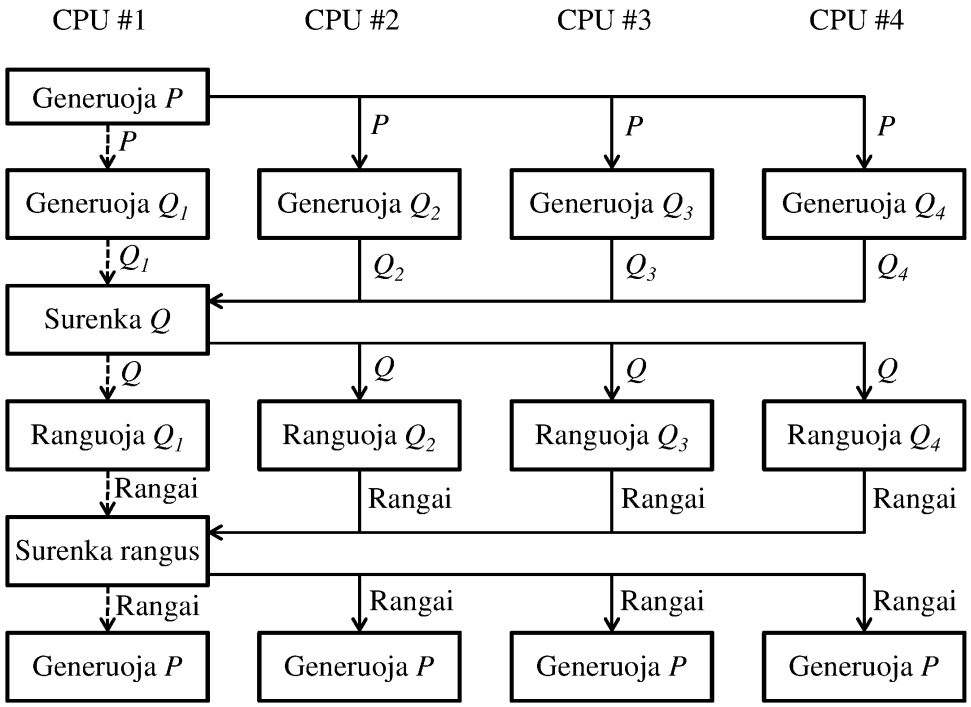
$$\bigcup_{i=1}^{p} Q_i = Q \tag{2.39}$$

ir paskaičiuoja tikslo funkcijų reikšmes sugeneruotuose taškuose. Čia $p$ – procesorių skaičius, o $i$ – kompiuterio ID, $i = 1, 2, \ldots, p$.

(3) Procesorius-šeimininkas surenka iš procesorių-darbininkų visus poaibius $Q_i$, suformuoja vaikų populiaciją $Q$ ir paplatina visiems procesoriams-darbininkams.

(4) Visi procesoriai apskaičiuoja palikuonių populiacijos atitinkamo poaibio $Q_i$ individų rangus: $r(Q_i, Q)$, $r(Q_i, P)$ ir $r(P, Q_i)$.

(5) Procesorius-šeimininkas surenka iš procesorių-darbininkų visą informaciją apie populiacijų $P$ ir $Q$ individų rangus, paskaičiuoja galutines kiekvieno individo rango vertes ir paplatina jas darbininkams. Po šio etapo visi procesoriai turės pilnas populiacijas $P$ ir $Q$ su informacija apie kiekvieno individo rango vertes.




2.12 pav.: Lygiagrečiojo algoritmo ParNSGA/DR schema

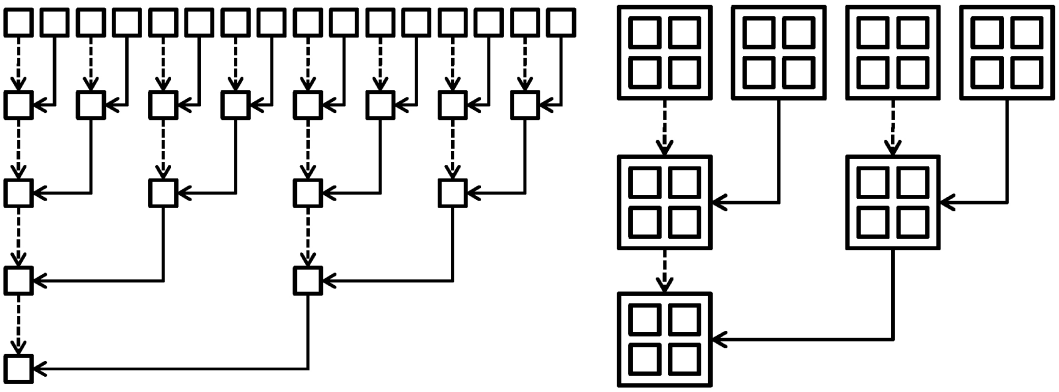
(6) Visi procesoriai suformuoja naują tėvų populiaciją $P$ iš populiacijos $P \cup Q$ pašalindami didžiausius rangus turinčius individus, ir pradeda naują generaciją (grįžta į 2-ąjį žingsnį).

Šią sprendinių rangavimo strategiją žymėsime DR, o lygiagretųjį algoritmą, naudojantį DR strategiją, žymėsime ParNSGA/DR. Algoritmo schema pavaizduota 2.12 paveiksle.

### *Hibridinis MPI-OpenMP algoritmas*

Visos aukščiau aprašytos NSGA-II algoritmo lygiagretinimo strategijos orientuotos į paskirstytosios atminties lygiagrečiųjų skaičiavimų sistemas, kuriose informacija tarp procesorių perduodama siunčiant žinutes. Siekiant išlaikyti lygiagrečiojo algoritmo elgseną tokią kaip ir nuosekliojo NSGA-II, duomenų apsikeitimas tarp procesorių turi būti atliekamas po kiekvienos iteracijos. Vadinasi, kiekvienos generacijos metu, visi procesoriai, vienu ar kitu būdu turi apsikeisti žinutėmis, kad turėtų naujausią informaciją. Tačiau dažnas procesorių komunikavimas žinutėmis didina duomenų perdavimo kaštus, kas neigiamai įtakoja lygiagretaus algoritmo spartinimo koeficientą.

Siekiant optimizuoti duomenų perdavimo kaštus, žinučių perdavimo modelis gali būti keičiamas bendravimu per bendrąją atmintį modeliu. Kita



2.13 pav.: Informacijos surinkimas iš paskirstytosios (*kairėje*) ir bendrosios (*dešinėje*) atminties procesorių

vertus, bendrosios atminties skaičiavimų sistemų naudojimo galimybes riboja aparatinė skaičiavimo įranga. Todėl NSGA-II lygiagretinimui yra siūloma hibridinė strategija, apjungianti abu – paskirstytosios ir bendrosios atminties modelius. Skirtingai nuo anksčiau aprašytų strategijų, vietoj $p$ procesorių, komunikuojančių žinučių perdavimu, yra naudojama $p_1$ procesorių grupių po $p_2$ bendrosios atminties procesorių, taip, kad $p = p_1 p_2$. Kiekvienoje iš $p_1$ grupių visi $p_2$ procesoriai tarpusavyje komunikuoja per bendrąją atmintį, o tarp procesorių grupių informacija persiunčiama žinutėmis.

Tardami, kad $p_1 = p_2 = 4$, duomenys iš visų 16 procesorių gali būti surenkami dviem etapais, kuomet naudojant vien tik paskirstytosios atminties procesorius, duomenų surinkimas reikalauja 4 etapų (žr. 2.13 pav.).

## 2.6 Antrojo skyriaus apibendrinimas

1. Pasiūlytos 5 atsitiktinių skaičių sekų pradinių reikšmių parinkimo strategijos, naudotinos sekų generavimui bendrosios atminties lygiagrečiųjų skaičiavimų sistemose.

2. Pasiūlytas daugiakriterio lokaliojo optimizavimo algoritmas MOSASS, gautas modifikuojant lokaliojo optimizavimo pagal vieną kriterijų algoritmą SASS. Pasiūlyta lokaliojo optimizavimo strategija įkomponuota į globaliojo daugiakriterio optimizavimo genetinį algoritmą taip sudarant memetinį daugiakriterio optimizavimo algoritmą.

3. Pasiūlytas daugiakriterio optimizavimo genetinio algoritmo NSGA-II taikymas konkuruojančių objektų vietos parinkimo uždaviniui spręsti.

4. Pasiūlyta dalelių spiečiaus optimizavimo algoritmo modifikacija, grįsta paieškos srities siaurinimu, ir aptartos 4 algoritmo lygiagretinimo

strategijos.

5. Pasiūlytas sprendinių rangavimo evoliuciniuose daugiakriterio optimizavimo algoritmuose metodas ir 4 sprendinių rangavimo lygiagretinimo strategijos.

6. Pasiūlyta mutavimo genetiniame algoritme ir lokaliojo optimizavimo strategijos, taikytinos sprendžiant konkuruojančių objektų vietos parinkimo uždavinį.

# 3 skyrius

# Eksperimentiniai tyrimai

Šiame skyriuje aprašomi atsitiktinių skaičių generatorių ir 2.1 poskyryje pasiūlytų pradinių atsitiktinių skaičių sekos reikšmių parinkimo metodų eksperimentiniai tyrimai, pateikiami gauti rezultatai [A1]. Taip pat aprašomi 2.2 poskyryje siūlomos dalelių spiečiaus optimizavimo algoritmo modifikacijos [A2] ir 2.3 poskyryje siūlomų globaliojo daugiakriterio optimizavimo algoritmų [A3, A6] eksperimentiniai tyrimai ir pateikiami gauti rezultatai. Pateikiami 2.5.2 poskyryje siūlomų NSGA-II algoritmo lygiagretinimo strategijų eksperimentinių tyrimų aprašymai ir aptariami gauti rezultatai [A4, A5]. Skyriaus pabaigoje pateikiamas skyriaus apibendrinimas ir formuluojamos išvados.

## 3.1 Eksperimentinis PASG tyrimas

Šiame poskyryje aptariami įvairių atsitiktinių skaičių sekų generavimo metodų ir 2.1 poskyryje pasiūlytų atsitiktinių skaičių sekų pradinių reikšmių parinkimo būdų eksperimentiniai tyrimai.

Pirmuoju tyrimu buvo tiriami 7 PASG, kurie buvo vertinami pagal 4 kriterijus: Kolmogorovo-Smirnovo skirstinio atitikimo kriterijų (K-S), skaičių pasikartojimą sekoje, koreliaciją tarp sugeneruotų sekų ir generavimo laiką (žr. 1.4 poskyrį).

Generavimo trukmės įvertinimui buvo generuojama 40 sekų po $10^8$ skaičių, tolygiai pasiskirsčiusių intervale $[0, 1]$, ir vertinama vidutinė generavimo trukmė. Kitų kriterijų vertinimui buvo generuojama 100 sekų po $10^4$ skaičių, taip pat tolygiai pasiskirsčiusių intervale $[0, 1]$. Visiems generatoriams buvo priskiriamos vienodos pradinės reikšmės. Generatoriui LCG buvo naudojami tokie parametrai: $a = 7777$, $c = 101$ ir $m = 32771$ (žr. 1.4 poskyrį).

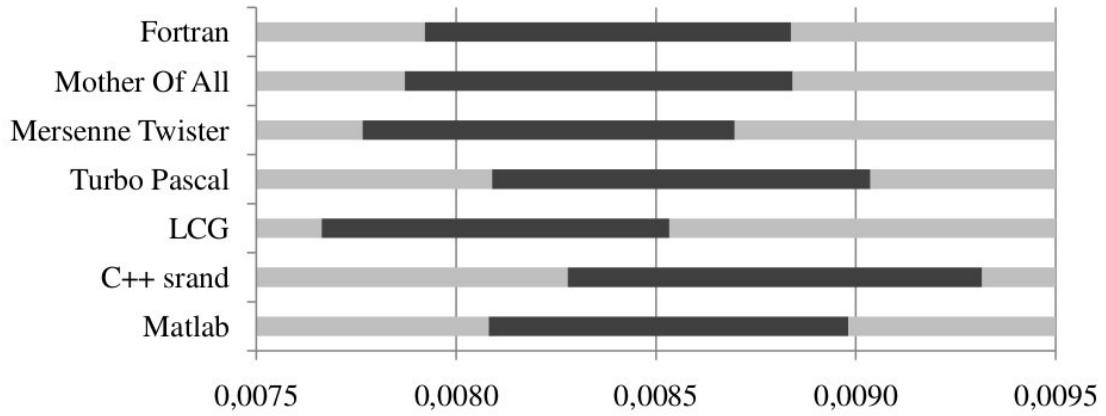
3.1 lentelė: Vidutinės PASG kokybės matų vertės

| PASG | Skirtingų skaičių (%) | K-S vertė | Koreliacijos koeficientas | Generavimo trukmė (s.) |
|---|---|---|---|---|
| Matlab | 99,996 | 0,00853 | 0,0047 | 2,105 |
| Visual C++ | 86,156 | 0,00879 | 0,0050 | 4,374 |
| LCG | 100,00 | 0,00809 | 0,0023 | 3,124 |
| Turbo Pascal | 92,725 | 0,00856 | 0,0030 | 43,75 |
| Mersenne Twister | 99,995 | 0,00823 | 0,0098 | 10,25 |
| Mother Of All | 99,994 | 0,00836 | 0,0032 | 9,274 |
| FORTRAN | 99,983 | 0,00838 | 0,0158 | 2,002 |

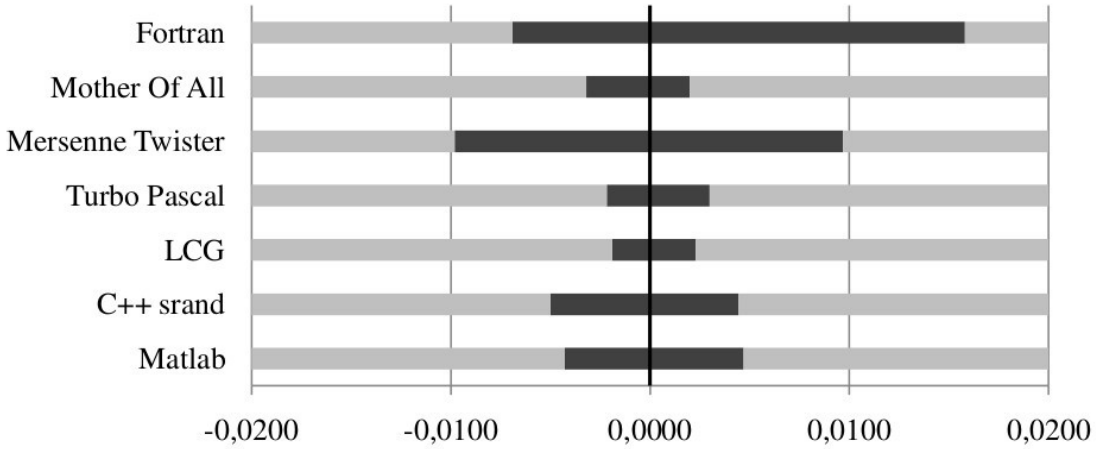
Eksperimento rezultatai pateikti 3.1 lentelėje, kurioje matyti, kad visose LCG generuotose sekose visi skaičiai buvo skirtingi. Tai reiškia, kad nė vienoje generuotų sekų nė vienas skaičius nepasikartojo. Mažiausios vidutinės K-S ir koreliacijos koeficientų vertės taip pat buvo sekų, generuotų LCG. Greičiausiai sekas generavo FORTRAN, tačiau sugeneravo labiausiai koreliuojančias sekas.

3.1 paveiksle pavaizduoti K-S verčių pasikliautinieji intervalai, kai pasikliovimo lygmuo yra 0,95. Paveiksle matyti, kad K-S vertė bus mažesnė už 0,0095 su tikimybe 0,95 visiems tirtiems PASG, tačiau geriausi pasikliatinųjų intervalų įverčiai yra LCG generatoriaus.

3.2 paveiksle pavaizduoti intervalai tarp mažiausių ir didžiausių (išskyrus 5 % didžiausią absoliutinę vertė turinčias reikšmes) koreliacijos koeficientų reikšmių. Paveiksle matyti, kad geriausios (arčiausiai nulinių) koreliacijos koeficientų reikšmės yra LCG generuotų sekų. Labiausiai išsibarsčiusios koreliacijos koeficientų vertės yra sekų, generuotų naudojant Fortran ir Matlab.




3.1 pav.: K-S verčių pasikliautinieji intervalai kai pasikliovimo lygmuo 0.95




3.2 pav.: Minimalios ir maksimalios atsitiktinių skaičių sekų koreliacijos koeficientų reikšmės

3.2 lentelė: Vidutinės sekų koreliacijų koeficientų reikšmės

| Pradinės reikšmės parinkimo metodas | Vidutinė koreliacijos koeficiento reikšmė |
|---|---|
| S1 | -0,0002512 |
| S2 | 0,0001007 |
| S3 | 0,0000388 |
| S4 | -0,0000034 |
| S5 | -0,0000022 |

Antruoju tyrimu buvo vertinami 2.1 poskyryje pasiūlyti pradinių atsitiktinių skaičių sekų reikšmių parinkimo būdai. Tyrimo metu buvo generuojama 100 sekų po 100 pseudo atsitiktinių skaičių, naudojant 100 lygiagrečių procesų, naudojančių bendrą laikmatį. Sugeneruotos sekos buvo vertinamos pagal tarpusavio koreliacijos koeficientą.

Tyrimo rezultatai, gauti naudojant skirtingus pradinių reikšmių parinkimo metodus, pateikti 3.2 lentelėje, kurioje matyti, kad geriausios (artimiausios nuliui) koreliacijų reikšmės buvo gautos naudojant pradinių reikšmių parinkimo metodus S4 ir S5 (turintys daugiausiai papildomų parametrų).

## 3.2 NSGA-II/LS ir NSGA-II/LSP algoritmų tyrimas

Šiame poskyryje aptariami eksperimentinių tyrimų, atliktų siekiant ištirti 2.3 poskyryje pasiūlytų hibridinių algoritmų NSGA-II/LS ir NSGA-II/LSP kokybę, rezultatai. Tyrimui buvo naudojami 26 daugiakriterio optimizavimo testo uždaviniai, kurių sąrašas pateiktas 3.3 lentelėje. Lentelėje patei-

kiamas kiekvieno testo uždavinio pavadinimas, kintamųjų ir kriterijų kiekiai. Kintamųjų ir kriterijų kiekiai parinkti pagal atitinkamuose literatūros šaltiniuose [21, 46, 71, 72, 73, 74, 75] pateiktas rekomendacijas.

3.3 lentelė: Daugiakriterio optimizavimo testo uždaviniai

| Uždavinys | Kintamųjų | Kriterijų |
|---|---|---|
| ZDT1 [21] | 30 | 2 |
| ZDT2 [21] | 30 | 2 |
| ZDT3 [21] | 30 | 2 |
| ZDT4 [21] | 10 | 2 |
| ZDT6 [21] | 30 | 2 |
| DTLZ1 [46] | 6 | 2 |
| DTLZ2 [46] | 12 | 2 |
| DTLZ3 [46] | 6 | 2 |
| DTLZ4 [46] | 12 | 2 |
| DTLZ7 [46] | 12 | 2 |
| UP1 [71] | 30 | 2 |
| UP2 [71] | 30 | 2 |
| UP3 [71] | 30 | 2 |
| UP4 [71] | 30 | 2 |
| UP7 [71] | 30 | 2 |
| UP8 [71] | 30 | 3 |
| UP10 [71] | 30 | 3 |
| KNO1 [72] | 2 | 2 |
| OKA1 [72] | 2 | 2 |
| OKA2 [72] | 3 | 2 |
| VLMOP2 [72] | 2 | 2 |
| Kursawe [73] | 3 | 2 |
| DMM [74] | 2 | 2 |
| VLMOP3 [72] | 2 | 3 |
| Comet [46] | 3 | 3 |
| SPH [75] | 10 | 3 |

Pirmuoju eksperimentiniu tyrimu buvo siekiama parinkti geriausias algoritmo NSGA-II/LSP parametrų $E_G$ ir $E_L$ vertes, atitinkamai apibrėžiančias, kas kiek tikslo funkcijų reikšmių skaičiavimų bus vykdoma lokalioji paieška, ir kiek tikslo funkcijų reikšmių skaičiavimų bus skiriama vienai lokaliajai paieškai (žr. 2.3 poskyrį). Tuo tikslu kiekvienas testo uždavinys buvo sprendžiama naudojant skirtingas parametrų $E_G \in \{500, 1000, 2000, 3000, 4000, 5000\}$ ir $E_L \in \{50, 100, 200, 300, 400, 500\}$ reikšmes, taip tiriant 36 skirtingas parametrų kombinacijas $(E_G, E_L)$. Globaliojo optimizavimo parametrai buvo parinkti pagal literatūroje [63] pateiktas rekomendacijas: $\pi_c = 0,8$ ir $\pi_m = 1/d$ (žr. 1.3.4 poskyrį).

Kiekvienam testo uždaviniui spręsti buvo skiriama po 25000 tikslo funkcijų skaičiavimų. Kiekvienas eksperimentas buvo kartojamas 100 kartų, naudojant skirtingas (atsitiktiniu būdu sugeneruotas) pradines populiacijas, ir vertinama vidutinė hipertūrio mato (žr. 1.1.4 poskyrį) vertė.

Konkrečios parametrų kombinacijos $(E_G, E_L)$ tinkamumas buvo vertinamas skaičiuojant, kiek testo uždavinių, naudojant šiuos parametrus, buvo išspręsta geriausiai. Eksperimento rezultatai, pateikti 3.4 lentelėje, rodo, kad sunku parinkti tokį parametrų rinkinį, kuris būtų tinkamiausias visiems testo uždaviniams, tačiau matome, kad parametrų $E_G \in \{500, 1000, 2000\}$ ir $E_L \in \{300, 400, 500\}$ naudojimas davė geriausius rezultatus – šie parametrai geriausiai tiko $\sim 42$ % testo uždaviniu.

Šių parametrų įtaka algoritmo efektyvumui taip pat buvo vertinama pagal procentinę išraišką skirtumo tarp didžiausios ir mažiausios hipertūrio ($HV$) vertės, gautos naudojant visas 36 $(E_G, E_L)$ kombinacijas. Kitaip tariant buvo vertinamas hipertūrio vertės procentinis nuostolis, parinkus blogiausią parametrų kombinaciją. Rezultatai parodė, kad parinkus blogiausią parametrų kombinaciją 5 testo uždaviniams hipertūrio reikšmė sumažėjo mažiau nei 5 %, 15 testo uždavinių – mažiau nei 10 % ir 20 testo uždavinių – mažiau nei 20 %. Didžiausia įtaka buvo uždaviniams ZDT4, UP8, OKA2, ZDT6 ir UP10 – hipertūrio nuostolis buvo atitinkamai 28 %, 35 %, 37 %, 56 % ir 91 %.

Antruoju eksperimentiniu tyrimu buvo siekiama palyginti siūlomų algoritmų NSGA-II/LS ir NSGA-II/LSP kokybę tarpusavyje ir su klasikiniu NSGA-II algoritmu, įgyvendintu kaip nurodyta literatūroje [63]. Visi trys algoritmai buvo naudojami 3.3 lentelėje pateiktiems testo uždaviniams spręsti. Kiekvienas uždavinys buvo sprendžiamas naudojant 25000 tikslo funkcijų skaičiavimų. Kiekvieno uždavinio sprendimas buvo kartojamas

3.4 lentelė: Kiekis testo funkcijų, kurioms konkreti parametrų kombinacija $(E_G, E_L)$ buvo geriausia

| $E_L$ | $E_G$ | | | | | |
|---|---|---|---|---|---|---|
| | 500 | 1000 | 2000 | 3000 | 4000 | 5000 |
| 50 | 1 | 1 | 0 | 0 | 0 | 0 |
| 100 | 0 | 0 | 0 | 0 | 0 | 0 |
| 200 | 1 | 1 | 0 | 2 | 1 | 3 |
| 300 | 2 | 1 | 1 | 0 | 1 | 0 |
| 400 | 1 | 3 | 0 | 0 | 0 | 0 |
| 500 | 0 | 3 | 1 | 0 | 0 | 2 |

100 kartų naudojant skirtingas pradines populiacijas. Lokalioji paieška buvo vykdoma kas 1000 funkcijų perskaičiavimų ($E_G = 1000$). Lokaliajam optimizavimui buvo parenkama 10 sprendinių, kurių kiekvieno optimizavimui buvo skiriama po 400 funkcijų perskaičiavimų ($E_L = 400$). Tikimybė $\pi_L$, naudojama kaimyninio taško generavimui lokaliojo optimizavimo algoritme NSGA-II/LSP, buvo lygi $1/d$, kaip ir mutavimo dažnis klasikiniame NSGA-II algoritme.

Algoritmai buvo lyginami pagal 5 matus (žr. 1.1.4 poskyrį):

- nedominuojamų taškų skaičių ($PS$);
- Pareto fronto persidengimą ($C$);
- hipertūrį ($HV$);
- apibendrintą atstumą ($IGD$);
- nedominuojamų taškų sklaidą ($\Delta$).

Eksperimentinio tyrimo rezultatai yra pateikti 3.5–3.9 lentelėse, kuriose pateiktos vidutinės matų $PS$, $C$, $HV$, $IGD$ ir $\Delta$ reikšmės kartu su standartiniais nuokrypiais (StDev). Geriausios vidutinės reikšmės yra paryškintos.

Kiekvieno algoritmo efektyvumas taip pat buvo vertinamas pagal skaičių testo funkcijų, kurias, išsprendė geriausiai. Algoritmai buvo vertinami pagal visus 5 matus: $PS$, $C$, $HV$, $IGD$ ir $\Delta$.

Tyrimo rezultatai yra pavaizduoti 3.3 paveiksle, kuriame vertikaliojoje ašyje pateikti geriausiai išspręstų testo uždavinių kiekiai, horizontaliojoje ašyje – vertinimo matai, o skirtingi stulpeliai reiškia skirtingą algoritmą: NSGA-II, NSGA-II/LS ir NSGA-II/LSP. Rezultatai rodo, kad daugiausiai nedominuojamų sprendinių ($PS$) NSGA-II algoritmas rado 7 testo uždaviniams, kuomet NSGA-II/LS ir NSGA-II/LSP daugiausiai nedominuojamų sprendinių rado atitinkamai 8 ir 11 testo uždavinių. Vertinant pagal Pareto frontų persidengimą ($C$), NSGA-II ir NSGA-II/LS algoritmai geriausiai išsprendė po 5, NSGA-II/LSP – 14, ir 1 testo uždaviniui visi trys algoritmai pateikė nulines mato $C$ vertes. Vertinant pagal gautas hipertūrio ($HV$) vertes, NSGA-II algoritmas geriausiai išsprendė 5 testo uždavinius, NSGA-II/LS – 3, o NSGA-II/LSP – 18 testo uždavinių. Vertinant pagal apibendrinto atstumo ($IGD$) matą, NSGA-II geriausiai išsprendė 3 testo uždavinius, kuomet algoritmai NSGA-II/LS ir NSGA-II/LSP geriausiai išsprendė po atitinkamai 8 ir 15 testo uždavinių. Vertinant pagal paskutinį kriterijų – nedominuojamų taškų sklaidą ($\Delta$), NSGA-II algoritmas geriausiai sprendė 7, NSGA-II/LS – 6, o NSGA-II/LSP – 13 testo uždavinių.

Remiantis šiais rezultatais galime teigti, kad NSGA-II/LSP geriausiai sprendė daugiausiai testo uždavinių, rezultatus vertinant pagal bet kurį iš tyrime naudotų kokybės matų.

3.5 lentelė: Pareto fronto dydžio mato ($PS$) vertės, gautos sprendžiant įvairius testo uždavinius

| Testo | NSGA-II | | NSGA-II/LS | | NSGA-II/LSP | |
|---|---|---|---|---|---|---|
| uždavinys | Vidurkis | StDev | Vidurkis | StDev | Vidurkis | StDev |
| ZDT1 | **96,7** | 15,3 | 94,4 | 2,30 | 96,1 | 2,30 |
| ZDT2 | 91,5 | 20,6 | 77,7 | 17,2 | **95,6** | 2,44 |
| ZDT3 | **94,9** | 14,1 | 86,0 | 4,60 | 92,2 | 6,43 |
| ZDT4 | **72,4** | 15,6 | 67,0 | 8,10 | 59,6 | 12,8 |
| ZDT6 | 16,1 | 3,49 | 14,1 | 2,30 | **77,2** | 7,93 |
| DTLZ1 | 62,1 | 31,1 | **87,7** | 3,70 | 84,0 | 18,9 |
| DTLZ2 | 92,9 | 3,9 | **93,4** | 2,00 | 92,5 | 2,34 |
| DTLZ3 | 50,7 | 23,9 | 82,7 | 6,20 | **86,5** | 14,5 |
| DTLZ4 | 62,6 | 30,1 | 63,4 | 35,3 | **92,5** | 3,80 |
| DTLZ7 | 86,6 | 5,23 | **93,5** | 1,90 | 86,1 | 4,45 |
| UP1 | 39,7 | 6,55 | 64,2 | 14,6 | **78,6** | 6,91 |
| UP2 | **92,4** | 6,96 | 92,0 | 3,70 | 90,6 | 3,46 |
| UP3 | 16,9 | 5,73 | 28,5 | 6,80 | **63,6** | 13,5 |
| UP4 | **77,6** | 3,39 | 73,6 | 4,20 | 51,0 | 6,75 |
| UP7 | 46,8 | 4,15 | 70,8 | 19,1 | **82,3** | 7,78 |
| UP8 | 83,8 | 7,69 | 77,4 | 11,8 | **86,2** | 5,74 |
| UP10 | 23,1 | 21,7 | 23,1 | 15,0 | **26,4** | 21,0 |
| KNO1 | 89,3 | 5,68 | **95,6** | 1,90 | 81,5 | 6,67 |
| OKA1 | 28,2 | 14,7 | 48,0 | 6,90 | **63,2** | 9,93 |
| OKA2 | 5,90 | 8,00 | 13,2 | 4,80 | **24,0** | 6,48 |
| VLMOP2 | **86,3** | 16,7 | 85,7 | 5,30 | 79,9 | 3,92 |
| Kursawe | 64,3 | 8,30 | **66,1** | 3,90 | 60,8 | 4,32 |
| DMM | 66,6 | 11,6 | **80,5** | 7,60 | 76,0 | 7,59 |
| Comet | 70,1 | 10,1 | **73,1** | 10,9 | 71,9 | 8,22 |
| VLMOP3 | 85,3 | 5,33 | **85,9** | 6,20 | 82,7 | 5,71 |
| SPH | **86,8** | 3,55 | 86,5 | 4,90 | 74,8 | 6,47 |

3.5 lentelėje matyti, kad pagal Pareto fronto dydžio matą, daugiausiai geriausių įverčių gauta NSGA-II/LSP. Nors kai kuriais atvejais NSGA/LSP ir NSGA-II gauti rezultatai yra panašūs, tačiau daugeliu šių atvejų NSGA-II gauti didesni standartiniai nuokrypiai. Didžiausi skirtumai matyti uždaviniams ZDT6, UP3, UP10, OKA1 ir OKA2 – NSGA-II/LSP rado daugiau nei du kartus didesnes Pareto frontų aproksimacijas, lyginant su NSGA-II.

3.6 lentelė: Pareto frontų persidengimo mato ($C$) vertės, gautos sprendžiant įvairius testo uždavinius

| Testo | NSGA-II | | NSGA-II/LS | | NSGA-II/LSP | |
|---|---|---|---|---|---|---|
| uždavinys | Vidurkis | StDev | Vidurkis | StDev | Vidurkis | StDev |
| ZDT1 | 0,00 | 0,00 | 1,80 | 1,50 | **88,1** | 3,56 |
| ZDT2 | 0,00 | 0,00 | 0,10 | 0,40 | **88,5** | 3,75 |
| ZDT3 | 0,90 | 1,10 | 12,4 | 4,40 | **85,4** | 6,85 |
| ZDT4 | 2,80 | 2,00 | **15,0** | 4,10 | 8,50 | 12,4 |
| ZDT6 | 0,00 | 0,00 | 0,00 | 0,00 | **21,4** | 7,87 |
| DTLZ1 | 0,00 | 0,00 | 38,9 | 15,8 | **64,0** | 19,3 |
| DTLZ2 | 51,0 | 12,0 | 25,6 | 9,50 | **70,3** | 5,98 |
| DTLZ3 | 0,00 | 0,00 | 34,2 | 15,1 | **61,1** | 15,6 |
| DTLZ4 | 0,10 | 1,30 | 32,8 | 28,5 | **84,1** | 5,18 |
| DTLZ7 | 6,60 | 4,20 | 67,2 | 10,1 | **78,4** | 5,24 |
| UP1 | 4,00 | 5,60 | 0,40 | 1,60 | **40,7** | 10,7 |
| UP2 | 0,00 | 0,20 | 0,00 | 0,10 | **14,4** | 4,50 |
| UP3 | 0,00 | 0,00 | 0,00 | 0,00 | **1,85** | 7,37 |
| UP4 | 0,00 | 0,00 | **0,80** | 0,00 | 0,00 | 0,00 |
| UP7 | 3,90 | 5,70 | 2,90 | 8,10 | **37,0** | 18,6 |
| UP8 | **58,3** | 6,20 | 0,00 | 0,00 | 12,5 | 7,01 |
| UP10 | **15,4** | 22,0 | 0,00 | 0,00 | 0,90 | 5,09 |
| KNO1 | 11,4 | 10,0 | **39,2** | 4,60 | 25,6 | 9,13 |
| OKA1 | 0,00 | 0,10 | **0,03** | 0,20 | 0,00 | 0,00 |
| OKA2 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 | 0,00 |
| VLMOP2 | 63,2 | 6,10 | **63,7** | 5,90 | 58,8 | 5,35 |
| Kursawe | 1,00 | 0,60 | 1,40 | 0,60 | **1,64** | 0,75 |
| DMM | 51,5 | 32,0 | **77,2** | 8,20 | 70,8 | 7,70 |
| Comet | **65,2** | 9,40 | 28,5 | 7,00 | 32,3 | 8,12 |
| VLMOP3 | **79,1** | 6,40 | 52,4 | 4,90 | 48,2 | 5,21 |
| SPH | **10.5** | 4,90 | 0,37 | 0,60 | 0,00 | 0,00 |

Nors pagal Pareto fronto aproksimacijos dydžio matą (žr. 3.5 lentelę) kai kuriems testo uždaviniams NSGA-II rado didžiausias Pareto fronto aproksimacijas, Pareto frontų persidengimo matas (žr. 3.6 lentelę) rodo, kad daugeliu atvejų tik keli aproksimacijos taškai priklausė tikrajam Pareto frontui (pavyzdžiui uždaviniams ZDT1, ZDT3, ZDT4, UP3 ir UP4).

3.7 lentelė: Hipertūrio mato ($HV$) vertės, gautos sprendžiant įvairius testo uždavinius

| Testo | NSGA-II | | NSGA-II/LS | | NSGA-II/LSP | |
|---|---|---|---|---|---|---|
| uždavinys | Vidurkis | StDev | Vidurkis | StDev | Vidurkis | StDev |
| ZDT1 | 0,6466 | 0,0020 | 0,6558 | 0,0011 | **0,6603** | 0,0005 |
| ZDT2 | 0,3046 | 0,0120 | 0,2452 | 0,0828 | **0,3272** | 0,0004 |
| ZDT3 | 0,5061 | 0,0071 | 0,5109 | 0,0014 | **0,5137** | 0,0020 |
| ZDT4 | **0,6103** | 0,0220 | 0,5245 | 0,0446 | 0,5372 | 0,0519 |
| ZDT6 | 0,0098 | 0,0023 | 0,0251 | 0,0065 | **0,3892** | 0,0035 |
| DTLZ1 | 0,4601 | 0,0180 | 0,4315 | 0,0468 | **0,4759** | 0,0510 |
| DTLZ2 | **0,2097** | 0,0002 | 0,2086 | 0,0004 | 0,2093 | 0,0004 |
| DTLZ3 | 0,1769 | 0,0170 | 0,1621 | 0,0265 | **0,2008** | 0,0226 |
| DTLZ4 | 0,1595 | 0,0770 | 0,1379 | 0,0877 | **0,2091** | 0,0007 |
| DTLZ7 | 0,4135 | 0,0130 | 0,4176 | 0,0019 | **0,4199** | 0,0007 |
| UP1 | 0,5053 | 0,0400 | 0,5130 | 0,0476 | **0,5505** | 0,0266 |
| UP2 | 0,5915 | 0,0070 | 0,6024 | 0,0089 | **0,6217** | 0,0106 |
| UP3 | 0,3738 | 0,0260 | 0,3742 | 0,0306 | **0,4253** | 0,0384 |
| UP4 | 0,2395 | 0,0035 | 0,2363 | 0,0043 | **0,2570** | 0,0044 |
| UP7 | 0,3026 | 0,0700 | 0,3183 | 0,0763 | **0,3554** | 0,0795 |
| UP8 | 0,5794 | 0,0140 | 0,5700 | 0,0182 | **0,6175** | 0,0175 |
| UP10 | 0,0447 | 0,0420 | **0,0465** | 0,0308 | 0,0434 | 0,0400 |
| KNO1 | 0,6641 | 0,0120 | **0,6864** | 0,0070 | 0,6716 | 0,0132 |
| OKA1 | 0,5320 | 0,0140 | 0,5749 | 0,0115 | **0,5766** | 0,0117 |
| OKA2 | 0,0928 | 0,0180 | 0,1259 | 0,0234 | **0,1681** | 0,0397 |
| VLMOP2 | **0,3086** | 0,0005 | 0,3084 | 0,0006 | 0,3068 | 0,0012 |
| Kursawe | 0,2908 | 0,0011 | 0,2914 | 0,0013 | **0,2918** | 0,0014 |
| DMM | 0,7708 | 0,0800 | **0,8199** | 0,0006 | 0,8191 | 0,0011 |
| Comet | 0,5116 | 0,0034 | 0,5136 | 0,0033 | **0,5164** | 0,0034 |
| VLMOP3 | **0,8272** | 0,0025 | 0,8253 | 0,0028 | 0,8258 | 0,0024 |
| SPH | **0,5539** | 0,0073 | 0,5462 | 0,0053 | 0,5502 | 0,0061 |

3.7 lentelėje matyti, kad pagal hipertūrio matą NSGA-II geriausiai sprendė 5 testo uždavinius, tačiau tiems uždaviniams gautos hipertūrio vertės mažai skiriasi nuo reikšmių, gautų NSGA-II/LSP. Algoritmas NSGA-II/LSP geriausiai sprendė didžiąją dalį testo uždavinių, o didžiausi skirtumai matyti uždaviniams ZDT6, DTLZ3, DTLZ4, UP1, UP3, OKA1.

3.8 lentelė: Apibendrinto atstumo mato ($IGD$) vertės, gautos sprendžiant įvairius testo uždavinius

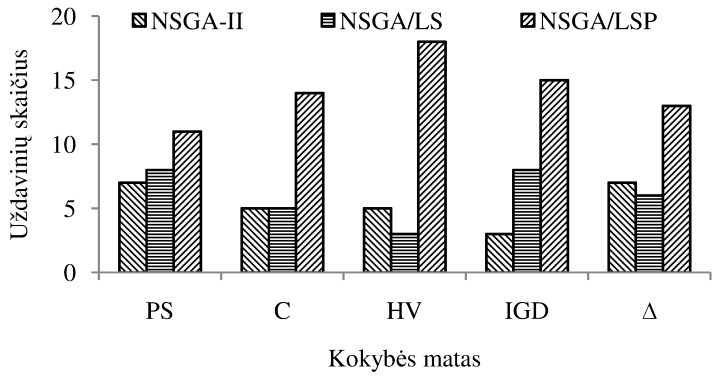
| Testo | NSGA-II | | NSGA-II/LS | | NSGA-II/LSP | |
|---|---|---|---|---|---|---|
| uždavinys | Vidurkis | StDev | Vidurkis | StDev | Vidurkis | StDev |
| ZDT1 | 0,0121 | 0,0016 | 0,0074 | 0,0017 | **0,0051** | 0,0005 |
| ZDT2 | 0,0290 | 0,0150 | 0,1094 | 0,1084 | **0,0054** | 0,0006 |
| ZDT3 | 0,0081 | 0,0062 | **0,0051** | 0,0008 | 0,0053 | 0,0018 |
| ZDT4 | **0,0581** | 0,0340 | 0,1599 | 0,0623 | 0,1068 | 0,0427 |
| ZDT6 | 0,5541 | 0,0190 | 0,4707 | 0,0265 | **0,0119** | 0,0026 |
| DTLZ1 | 0,0243 | 0,0130 | 0,0480 | 0,0331 | **0,0190** | 0,0518 |
| DTLZ2 | **0,0045** | 0,0002 | 0,0050 | 0,0005 | 0,0049 | 0,0005 |
| DTLZ3 | 0,0501 | 0,0270 | 0,0483 | 0,0275 | **0,0145** | 0,0296 |
| DTLZ4 | 0,1123 | 0,1700 | 0,1613 | 0,1956 | **0,0059** | 0,0011 |
| DTLZ7 | 0,0166 | 0,0170 | 0,0090 | 0,0095 | **0,0047** | 0,0007 |
| UP1 | 0,1280 | 0,0380 | 0,1123 | 0,0336 | **0,0904** | 0,0229 |
| UP2 | 0,0659 | 0,0150 | 0,0551 | 0,0175 | **0,0469** | 0,0166 |
| UP3 | 0,2180 | 0,0320 | 0,2112 | 0,0302 | **0,1877** | 0,0391 |
| UP4 | 0,0913 | 0,0077 | 0,0921 | 0,0074 | **0,0654** | 0,0089 |
| UP7 | 0,2326 | 0,1200 | 0,2019 | 0,1280 | **0,1708** | 0,1228 |
| UP8 | 0,1500 | 0,0300 | 0,1339 | 0,0267 | **0,1140** | 0,0351 |
| UP10 | 0,4928 | 0,1400 | **0,4654** | 0,1200 | 0,4691 | 0,1184 |
| KNO1 | 0,0846 | 0,0140 | **0,0624** | 0,0064 | 0,0751 | 0,0123 |
| OKA1 | 0,0853 | 0,0150 | **0,0603** | 0,0193 | 0,0604 | 0,0198 |
| OKA2 | 0,3668 | 0,0510 | 0,3301 | 0,0536 | **0,2377** | 0,0932 |
| VLMOP2 | **0,0059** | 0,0005 | 0,0066 | 0,0012 | 0,0079 | 0,0021 |
| Kursawe | 0,1329 | 0,0011 | **0,1326** | 0,0010 | 0,1332 | 0,0015 |
| DMM | 0,0409 | 0,0520 | **0,0089** | 0,0014 | 0,0100 | 0,0020 |
| Comet | 0,0315 | 0,0034 | 0,0295 | 0,0025 | **0,0280** | 0,0022 |
| VLMOP3 | 0,0209 | 0,0380 | **0,0118** | 0,0016 | 0,0172 | 0,0071 |
| SPH | 0,0717 | 0,0044 | **0,0709** | 0,0038 | 0,0712 | 0,0041 |

Pagal apibendrinto atstumo matą (žr. 3.8 lentelę) NSGA-II geriausiai sprendė 3 testo uždavinius, tačiau ženklus pranašumas matyti tik sprendžiant uždavinį ZDT4. Kokybės matai $C$, $HV$ ir $IGD$ aprašo panašias Pareto fronto savybės. Tai atsispindi ir eksperimentų rezultatuose – beveik visiems testo uždaviniams, kuriems buvo gautas geriausias mato $C$ įvertis, gauti geriausi ir matų $HV$ bei $IGD$ įverčiai.

3.9 lentelė: Pareto frontą aproksimuojančių taškų pasiskirstymo mato ($\Delta$) vertės, gautos sprendžiant įvairius testo uždavinius

| Testo | NSGA-II | | NSGA-II/LS | | NSGA-II/LSP | |
|---|---|---|---|---|---|---|
| uždavinys | Vidurkis | StDev | Vidurkis | StDev | Vidurkis | StDev |
| ZDT1 | **0,00009** | 0,00004 | 0,00015 | 0,00005 | 0,00011 | 0,00004 |
| ZDT2 | 0,00063 | 0,00081 | 0,00673 | 0,00773 | **0,00011** | 0,00004 |
| ZDT3 | 0,00019 | 0,00026 | **0,00011** | 0,00015 | 0,00013 | 0,00011 |
| ZDT4 | **0,00146** | 0,00120 | 0,00477 | 0,00241 | 0,00249 | 0,00144 |
| ZDT6 | 0,07628 | 0,01500 | 0,06772 | 0,01330 | **0,00032** | 0,00013 |
| DTLZ1 | 0,00031 | 0,00013 | **0,00018** | 0,00017 | 0,00557 | 0,03590 |
| DTLZ2 | 0,00017 | 0,00003 | 0,00019 | 0,00005 | **0,00016** | 0,00005 |
| DTLZ3 | 0,00270 | 0,00190 | **0,00025** | 0,00016 | 0,00337 | 0,02242 |
| DTLZ4 | 0,01586 | 0,02700 | 0,02268 | 0,03742 | **0,00014** | 0,00003 |
| DTLZ7 | 0,00069 | 0,00079 | 0,00034 | 0,00054 | **0,00009** | 0,00002 |
| UP1 | 0,00508 | 0,00630 | 0,00219 | 0,00180 | **0,00146** | 0,00101 |
| UP2 | 0,00101 | 0,00083 | 0,00064 | 0,00062 | **0,00056** | 0,00062 |
| UP3 | 0,03557 | 0,02300 | 0,01605 | 0,00864 | **0,00923** | 0,00395 |
| UP4 | **0,00146** | 0,00032 | 0,00151 | 0,00030 | 0,00176 | 0,00060 |
| UP7 | 0,01846 | 0,01900 | 0,00858 | 0,00832 | **0,00512** | 0,00464 |
| UP8 | **0,00162** | 0,00150 | 0,00303 | 0,00188 | 0,00163 | 0,00155 |
| UP10 | 0,17940 | 0,18000 | 0,12659 | 0,15271 | **0,12065** | 0,10789 |
| KNO1 | 0,00239 | 0,00045 | **0,00163** | 0,00015 | 0,00221 | 0,00041 |
| OKA1 | 0,00341 | 0,00220 | 0,00139 | 0,00129 | **0,00110** | 0,00100 |
| OKA2 | 0,11960 | 0,05100 | 0,05850 | 0,03328 | **0,02227** | 0,01488 |
| VLMOP2 | **0,00011** | 0,00002 | 0,00012 | 0,00003 | 0,00015 | 0,00007 |
| Kursawe | **0,00216** | 0,00019 | 0,00227 | 0,00035 | 0,00254 | 0,00032 |
| DMM | 0,00057 | 0,00026 | **0,00040** | 0,00010 | 0,00046 | 0,00018 |
| Comet | **0,00136** | 0,00035 | 0,00152 | 0,00040 | 0,00146 | 0,00035 |
| VLMOP3 | 0,00094 | 0,00027 | 0,00091 | 0,00022 | **0,00086** | 0,00024 |
| SPH | 0,00596 | 0,00098 | **0,00562** | 0,00086 | 0,00572 | 0,00111 |

Vertinant pagal Pareto fronto aproksimacijos taškų sklaidos matą, ženklūs skirtumai gauti 6 testo uždaviniams: ZDT6, DTLZ7, UP2, UP3 ir UP7 uždavinius geriausiai sprendė NSGA-II/LSP, o DTLZ3 uždavinį – NSGA/LS. Vertinant pagal visus 5 matus, NSGA-II/LSP geriausiai sprendė 7 uždavinius, NSGA-II/LS – 2, o klasikinis NSGA-II – nė vieno.

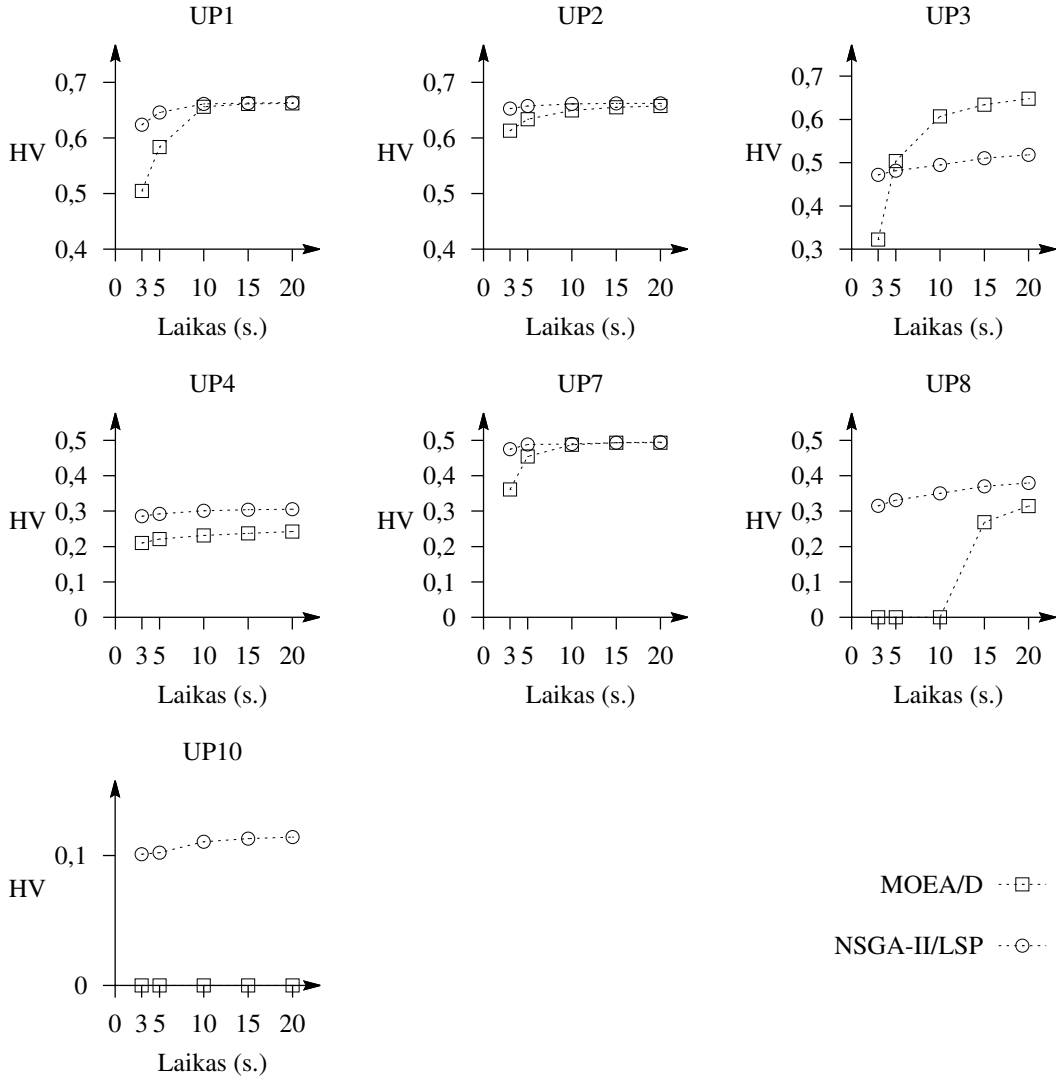



3.3 pav.: Testo uždavinių pasiskirstymas pagal geriausiai juos sprendusius algoritmus

Siūlomas algoritmas NSGA-II/LSP buvo lyginamas su algoritmu MOEA/D (angl. *Multi Objective Evolutionary Algorithm based on Decomposition*) [76]. Šis evoliucinis algoritmas yra grįstas daugiakriterio optimizavimo uždavinio dekomponavimu į kelis optimizavimo pagal vieną kriterijų uždavinius. Algortimai buvo lyginami sprendžiant UP1–UP10 testo uždavinius, kadangi 2009 metais MOEA/D buvo pripažintas geriausiai šiuos uždavinius sprendžiančiu algoritmu. Eksperimentų rezultatai pateikti

NSGA-II/LSP ir MOEA/D efektyvumas buvo vertinami pagal pasiektą hipertūrio mato $HV$ vertę per tam tikrą laiką. Kiekvieno uždavinio sprendimui buvo skiriamas 20 sekundžių. Hipertūrio vertės buvo matuojamos 3-čią, 5-tą, 10-tą, 15-tą ir 20-tą uždavinio sprendimo sekundę. Kiekvieno uždavinio sprendimas buvo kartojamas 100 kartų, naudojant skirtingas pradines populiacijas, ir vertinami vidutiniai rezultatai. Eksperimentų rezultatai pateikti 3.4 paveiksle. Čia kiekviename grafike horizontaliojoje ašyje vaizduojami laiko momentai, o horizontaliojoje – hipertūrio mato vertės. Skirtingas grafikas atitinka skirtingą testo uždavinį, o skirtingos kreivės – skirtingą optimizavimo algoritmą. Paveiksle matyti, per tą patį sprendimo laiką, algoritmu NSGA-II/LSP gaunamos didesnės hipertūrio mato vertės beveik visiems testo uždaviniams. MOEA/D sprendė geriau tik vieną testo uždavinį – UP3. Taip pat matyti, kad NSGA-II/LSP turi ženklų pranašumą sprendžiant trijų kriterijų optimizavimo uždavinius UP8 ir UP10. Sprendžiant UP10 uždavinį per 20 sekundžių MOEA/D nepateikė nė vieno optimalaus sprendinio.

Remiantis šiais rezultatais galima teigti, kad NSGA-II/LSP galima rasti geresnė Pareto fronto aproksimaciją per fiksuotą sprendimo laiką.

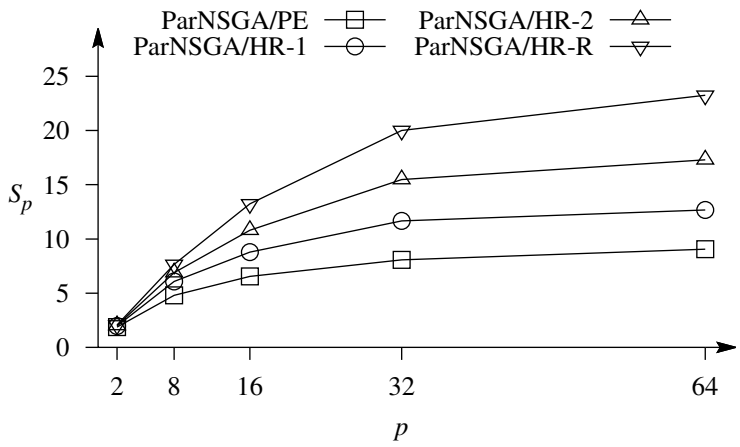



3.4 pav.: Hipertūrio mato vertės, gautos MOEA/D ir NSGA-II/LSP algoritmais sprendžiant UP1–UP10 testo uždavinius

## 3.3 Lygiagrečiųjų daugiakriterio optimizavimo algoritmų tyrimas

Šiame poskyryje aptariami eksperimentinio tyrimo, kuriuo buvo tiriami 2.5.2 poskyryje pasiūlytų lygiagrečiųjų daugiakriterio optimizavimo algoritmų ParNSGA/PE, ParNSGA/HR-1, ParNSGA/HR-2 ir ParNSGA/HR-R efektyvumas, rezultatai. Algoritmų efektyvumas buvo tiriamas sprendžiant dviejų kriterijų testo uždavinį ZDT1 (žr. 3.3 lentelę) naudojant 2, 4, 8, 16, 32 ir 64 paskirstytosios atminties procesorius.

Testo uždavinio spręsti buvo naudojama 512 individų populiacija ir atliekama 250 iteracijų. Viso buvo atliekama $512 \times 250 = 128000$ tikslo funkcijų

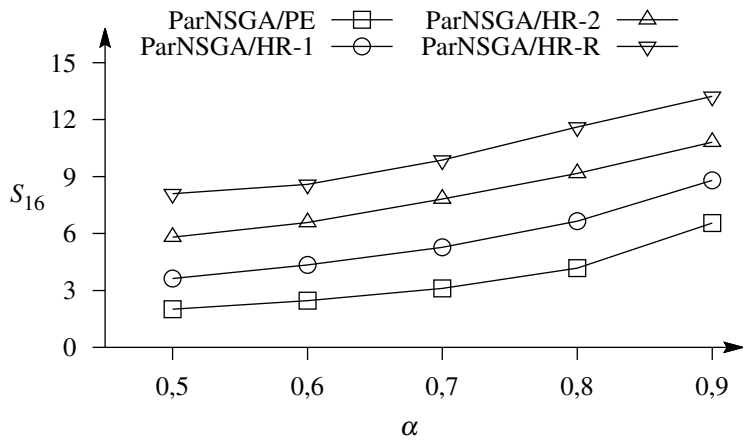



3.5 pav.: Lygiagrečiojo NSGA-II algoritmo spartinimo koeficientas esant skirtingoms lygiagretinimo strategijoms

reikšmių skaičiavimų, kurie reikalavo $\sim 90$ % nuoseklaus NSGA-II algoritmo vykdymo laiko. Testo uždavinys buvo sprendžiamas 10 kartų, naudojant skirtingas pradines populiacijas, ir vertinama vidutinė sprendimo trukmė. Algoritmų efektyvumas buvo vertinamas pagal algoritmo spartinimo koeficientą, $T_0$ verte laikant algoritmo NSGA-II, realizuoto, kaip nurodyta literatūroje [63], vykdymo trukmę.

Eksperimentų rezultatai pateikti 3.5 paveiksle. Čia horizontalioje ašyje pateikiamas skaičiavimams naudojamų procesorių skaičius, o vertikalioje – algoritmo spartinimo koeficientas. Skirtingos kreivės atitinka skirtingą lygiagretųjį algoritmą. Paveiksle matyti, kad mažiausias spartinimo koeficientas gautas algoritmu ParNSGA/PE. Tai yra natūralu, nes šis algoritmas paskirsto tik testo funkcijos reikšmių perskaičiavimus, todėl, pagal Amdalo dėsnį (žr. (1.43) formulę), didžiausias galimas algoritmo spartinimo koeficientas yra 10 (čia turima omenyje, kad funkcijų reikšmių perskaičiavimai reikalauja $\sim 90$ % nuosekliojo algoritmo vykdymo trukmės). Ženkliai geresni rezultatai gauti naudojant sprendinių rangavimo lygiagretinimo strategijas (algoritmus ParNSGA/HR-1 ir ParNSGA/HR-2) – algoritmo spartinimo koeficientas (naudojant 64 procesorius) padidėjo atitinkamai 40 % ir 90 %, lyginant su spartinimu, gautu naudojant ParNSGA/PE algoritmą. Geriausi rezultatai gauti sprendžiant uždavinį ParNSGA/HR-R algoritmu, kuris atmeta tikėtinai blogus sprendinius dar prieš pradedant duomenų mainų procesą. Gauta spartinimo koeficiento vertė $\sim 37$ % didesnė už spartinimo koeficiento vertę, gautą naudojant ParNSGA/HR-2, ir $\sim 160$ % didesnė už spartinimo vertę, gautą ParNSGA/PE algoritmu.

Šiame tyrime buvo naudojamas testo uždavinys, kurio tikslo funkcijų reikšmių skaičiavimų trukmė sudarė $\sim 90$ % viso algoritmo vykdymo truk-




3.6 pav.: Lygiagrečiojo NSGA-II algoritmo spartinimo koeficiento priklausomybė nuo $\alpha$ vertės

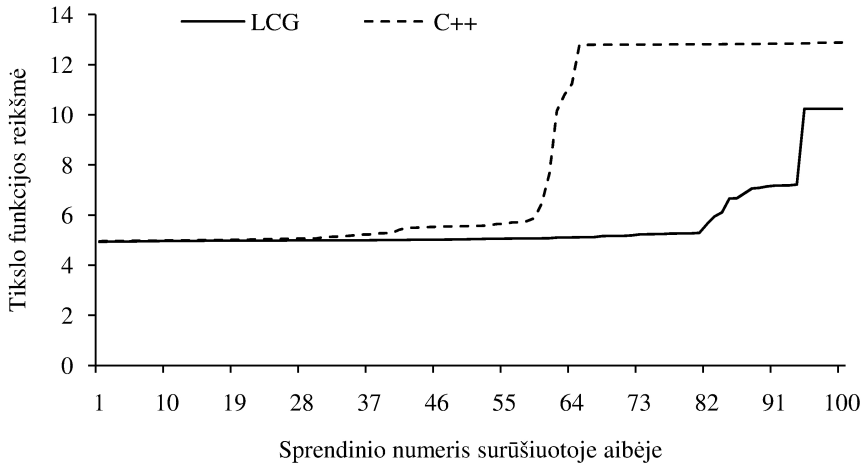
mės, arba kitaip $\alpha = 0,9$ (žr. (1.43) formulę). Natūralu, kad gautume kitokias algoritmų spartinimo koeficientų vertes, jei spręstume uždavinius, kuriems $\alpha = 0,8$ arba $\alpha = 0,5$. Siekiant ištirti $\alpha$ vertės įtaką algoritmų spartinimo koeficientui, skaičiavimai buvo atlikti sprendžiant dar 4 testo uždavinius, kuriems $\alpha = 0,8$, $\alpha = 0,7$, $\alpha = 0,6$ ir $\alpha = 0,5$. Skaičiavimai buvo atlikti naudojant 16 paskirstytos atminties procesorių. Eksperimentų rezultatai yra pavaizduoti 3.6 paveiksle. Čia horizontalioje ašyje pateikiama nuosekliojo algoritmo vykdymo trukmės dalis, užimama tikslo funkcijos reikšmių skaičiavimui ($\alpha$ vertė), o vertikalioje ašyje – lygiagrečiojo algoritmo spartinimo koeficientas. Paveiksle matyti, kad sprendžiant testo uždavinius, reikalaujančius mažiau laiko reikšmių skaičiavimui, visiems tirtiems lygiagretiesiems algoritmams gaunamas mažesnis algoritmo spartinimo koeficientas. Tačiau visoms $\alpha$ vertėms, tiriamų algoritmo lygiagretinimo strategijų naudojimas turi ženklią įtaką algoritmo spartinimo koeficientui.

## 3.4 MGA uždavinio sprendimas PSO algoritmu

Šiame poskyryje aptariamas eksperimentinio tyrimo, kurio metu buvo tiriama 2.2 poskyryje pasiūlyta PSO algoritmo modifikacija ir 2.5.1 poskyryje aprašytų lygiagretinimo strategijų efektyvumas, rezultatai.

Tyrimo metu buvo sprendžiamas erdvėlaivių skrydžių trajektorijų optimizavimo (MGA) uždavinys (žr. 1.2.1 poskyrį).

Pirmojoje tyrimo dalyje buvo tiriama atsitiktinių skaičių generatoriaus įtaka algoritmo kokybei. Tuo tikslu uždavinys buvo sprendžiamas PSO algoritmu naudojant skirtingus atsitiktinių skaičių generatorius: vidinį C++

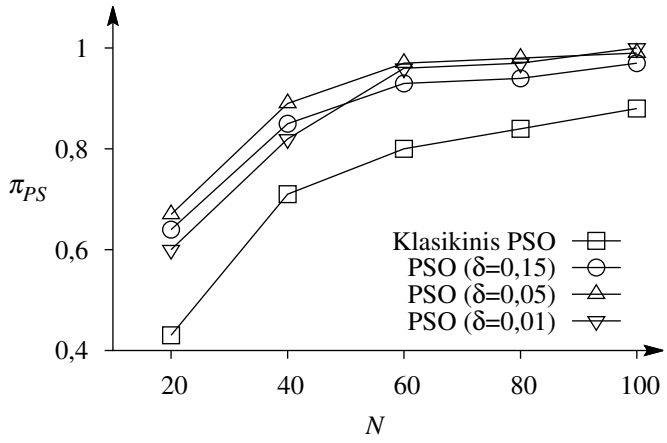



3.7 pav.: MGA uždavinio sprendiniai, gauti naudojant skirtingus PASG

generatorių, iškviečiamą komanda *rand*(), ir LCG su parametrais $a = 7777$, $c = 101$ ir $m = 32771$ (žr. 1.4 poskyrį). Buvo atliekama 100 eksperimentų. Abiems generatoriams buvo priskiriama vienoda pradinė reikšmė, tačiau kiekviename eksperimente ši reikšmė buvo keičiama.

Naudojant abu generatorius gautos sprendinių aibės surūšiuotos didėjimo tvarka ir pateiktos 3.7 paveiksle. Čia horizontaliojoje ašyje vaizduojamas sprendinio numeris surūšiuotoje aibėje, o vertikaliojoje ašyje – tikslo funkcijos reikšmė. Paveiksle matyti, kad geresni rezultatai gauti naudojant LCG generatorių – ∼80 sprendinių tikslo funkcijos reikšmės buvo mažesnės už 5.3. Naudojant C++ generatorių, tokių sprendinių buvo rasta 41. Vertinant pagal blogiausius ratus sprendinius, geresni rezultatai gaunami naudojant LCG generatorių – blogiausio sprendinio, rasto naudojant LCG, tikslo funkcijos reikšmė lygi 10,2, tuo tarpu blogiausio sprendinio, rasto naudojant C++ vidinį generatorių, tikslo funkcijos reikšmė lygi 12,9.

Antrojoje tyrimo dalyje buvo tiriama paieškos srities mažinimo įtaka PSO algoritmo efektyvumui. Tuo tikslu uždavinys buvo sprendžiamas klasikiniu PSO algoritmu bei mažinant paieškos sritį, naudojant skirtingus srities mažinimo koeficientus: 0,15, 0,05 ir 0,01. Paieškos srities mažinimas buvo vykdomas po pusės tikslo funkcijos reikšmių skaičiavimų. Visais atvejais buvo vykdoma 100 generacijų, naudojant skirtingus dalelių spiečių (populiacijų) dydžius: 20, 40, 60, 80 ir 100 dalelių. Kadangi PSO algoritmas nėra deterministinis, kiekvienas eksperimentas buvo kartojamas 100 kartų, naudojant skirtingas pradines populiacijas. Algoritmų kokybė buvo vertinama pagal tikimybę rasti priimtiną tikslo funkcijos reikšmę turintį sprendinį. Priimtinomis tikslo funkcijos reikšmėmis buvo laikomos reikšmės, mažesnės

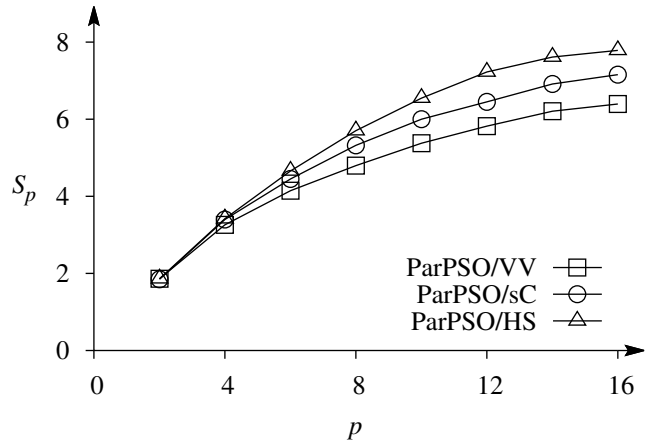



3.8 pav.: Tikimybės rasti priimtiną sprendinį priklausomybė nuo spiečiaus dydžio.

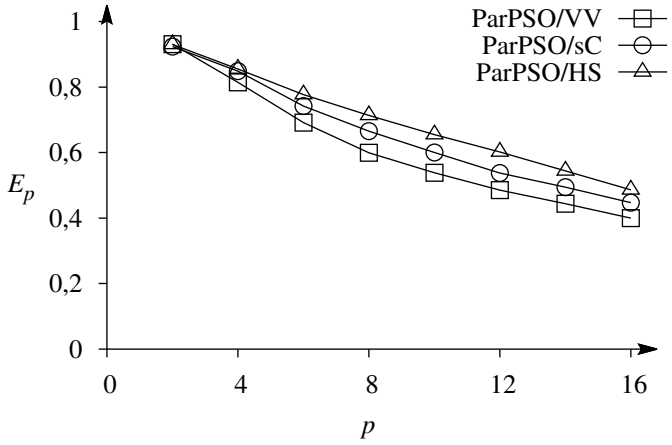
už 5,308 [77]. Tikimybė buvo vertinama eksperimentų, kuriais buvo rastas priimtinas sprendinys, skaičių dalinant iš visų vykdytų eksperimentų skaičiaus.

Eksperimentų rezultatai yra pateikti 3.8 paveiksle, kuriame pavaizduota tikimybės rasti priimtiną sprendinį priklausomybė nuo spiečiaus dydžio. Skirtingos kreivės reiškia skirtingus algoritmus: PSO ir PSO su skirtingais paieškos srities mažinimo koeficientais $\delta$: 0,15, 0,05 ir 0,01 (atitinkamai PSO($\delta = 0,15$), PSO($\delta = 0,05$) ir PSO($\delta = 0,01$)). Paveiksle matyti, kad tikimybė rasti priimtiną sprendinį priklauso nuo spiečiaus dydžio – daugiau dalelių spiečiuje – didesnė tikimybė rasti priimtiną sprendinį. Paieškos srities mažinimas ženkliai padidino priimtino sprendinio radimo tikimybę – tikimybė rasti priimtiną sprendinį naudojant 100 dalelių spiečių ir srities mažinimo koeficientą 0,01 buvo įvertinta 1, kuomet tikimybė rasti priimtiną sprendinį nemažinant paieškos srities buvo įvertinta apie 0,85. Tačiau esant mažesniam spiečiui, geriau naudoti didesnį paieškos srities mažinimo koeficientą (mažiau mažinti sritį). Tai gali būti paaiškinama tarpinio sprendinio, kuris naudojamas sudarant naują paieškos sritį, tikslumu – kuo tikslesnis tarpinis sprendinys, tuo daugiau galima mažinti paieškos sritį. Jei tarpinis sprendinys nėra pakankamai tikslus, tai per daug mažindami, gausime sritį, kuriai nepriklausys optimalus sprendinys.

Trečiuoju eksperimentu buvo tiriamas lygiagrečiųjų PSO algoritmo versijų ParPSO/sC, ParPSO/VV, ParPSO/HS ir ParPSO/aC (žr. 2.5.1 poskyrį) efektyvumas. Šio eksperimento metu taip pat buvo sprendžiamas MGA uždavinys. Lygiagretieji skaičiavimai buvo atliekami naudojant 2, 4, 8 ir 16 paskirstytosios atminties procesorių, vertinant algoritmų spartinimo




3.9 pav.: Lygiagrečiojo PSO algoritmo spartinimo koeficientas naudojant skirtingas komunikavimo strategijas

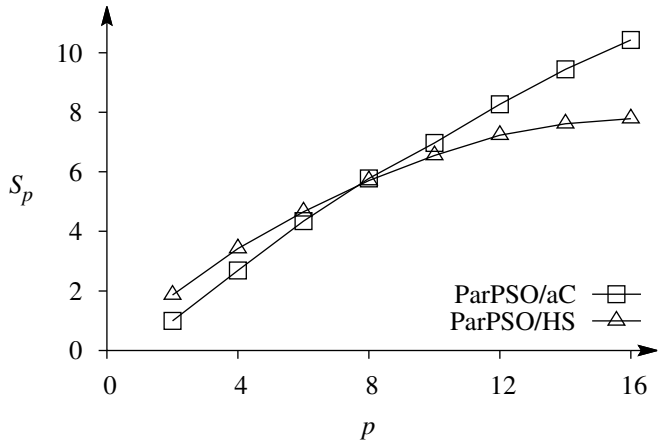



3.10 pav.: Lygiagrečiojo PSO algoritmo efektyvumo koeficientas naudojant skirtingas komunikavimo strategijas

ir efektyvumo koeficientus.

Gautos sinchroninių algoritmų ParPSO/sC, ParPSO/VV ir ParPSO/HS spartinimo ir efektyvumo koeficientų priklausomybės nuo naudojamų procesorių skaičiaus yra pateiktos atitinkamai 3.9 ir 3.10 paveiksluose. Juose matyti, kad geriausi spartinimo ir efektyvumo koeficientai gauti algoritmu ParPSO/HS, naudojančiu hierarchinės sklaidos duomenų apsikeitimo strategiją. Šiek tiek prastesni efektyvumo matai gauti algoritmu ParPSO/sC, naudojančiu sinchroninę šeimininkas-darbininkas strategiją, o prasčiausi – algoritmu ParPSO/VV.

Algoritmai realizuoti pagal sinchroninę PSO algoritmo versiją – po kiekvienos generacijos visi procesoriai apsikeičia informacija apie geriausius žinomus sprendinius ir visi procesoriai atlieka atitinkamą skaičiavimo darbų




3.11 pav.: Lygiagrečiojo PSO algoritmo spartinimo koeficientas naudojant hierarchinę ir asinchroninę šeimininko-darbininko komunikavimo strategijas

dalį.

Geriausius efektyvumo įvertinimus gavusį sinchroninį algoritmą ParPSO/HS palyginsime su asinchroniniu algoritmu ParPSO/aC. Šiame algoritme vienas procesorius yra išskiriamas kaip procesorius-šeimininkas, kuris atsakingas už duomenų mainus tarp kitų procesorių ir kuriam neskiriami skaičiavimo darbai (žr. 2.5.1 poskyrį).

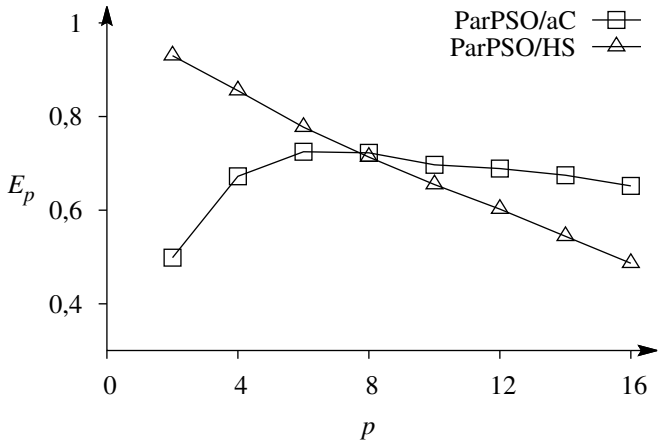
3.11 ir 3.12 paveiksluose pateiktos abiejų algoritmų spartinimo ir efektyvumo koeficientų priklausomybės nuo skaičiavimams naudojamų procesorių skaičiaus. Paveiksle matyti, kad skaičiavimams naudojant mažą procesorių skaičių (2, 4 arba 6), algoritmo ParPSO/HS spartinimo ir efektyvumo koeficientai yra geresni. Tačiau skaičiavimams naudojant daugiau procesorių, naudingiau yra naudoti algoritmą ParPSO/aC, taip paliekant vieną procesorių be skaičiavimo darbo, bet palaikantį komunikavimą tarp kitų, skaičiavimo darbus atliekančių procesorių.

## 3.5 CFL uždavinio sprendimas NSGA-II algoritmu

2.4 poskyryje aprašyti NSGA-II taikymo būdai diskrečiajam CFL uždaviniui spręsti. Taip pat pasiūlyta lokalios paieškos strategija, taikytina šiam uždaviniui spręsti.

Šiame poskyryje aptariami eksperimentinio tyrimo, kuriuo buvo siekiama ištirti 2.4 poskyryje siūlomų sprendimo būdų tinkamumą, rezultatai.

Tyrimo metu buvo sprendžiamas diskretusis CFL uždavinys (žr.

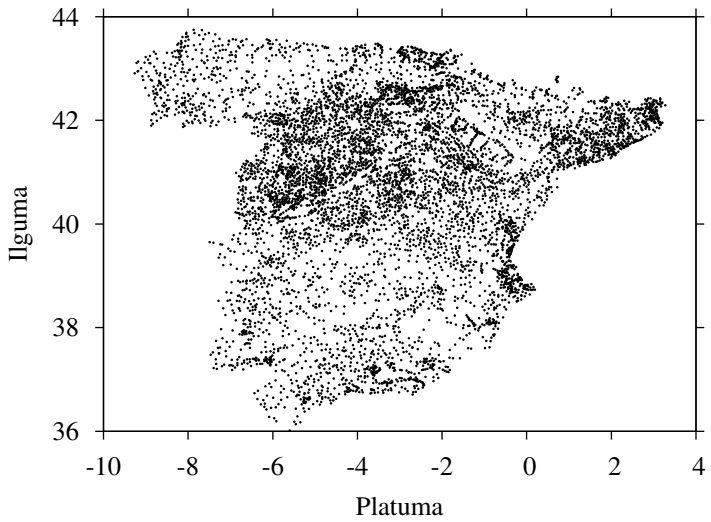



3.12 pav.: Lygiagrečiojo PSO algoritmo efektyvumo koeficientas naudojant hierarchinę ir asinchroninę šeimininko-darbininko komunikavimo strategijas

1.2.2 poskyrį), naudojant realius duomenis: 6961 Ispanijos miestų geografines koordinates (žr. 3.13 paveikslą) ir gyventojų populiacijas.

Tyrimo metu buvo laikoma, kad įmonės A ir B turi po 3 objektus ($n_A = n_B = 3$), lokalizuotus 6 daugiausiai klientų turinčiuose paklausos taškuose (Madride, Barselonoje, Valencijoje, Sevilijoje, Saragosoje ir Malagoje). Kandidatų aibę $L$ sudarė $k$ didžiausias populiacijas turinčių paklausos taškų, išskyrus pirmuosius 3 (Madridą, Barseloną ir Valenciją), kuriuose jau yra įmonei $A$ priklausantys objektai.

Buvo tiriamos 4 skirtingos parametrų $k$ ir $n_X$ kombinacijos, pateiktos




3.13 pav.: Ispanijos miestų geografinės koordinatės

3.10 lentelėje. Čia $n_X$ – skaičius naujų objektų, kuriems bus parenkamos optimalios vietos (uždavinio kintamųjų skaičius).

3.10 lentelė: Tyrimui naudotos uždavinio parametrų kombinacijos

| | I | II | III | IV |
|---|---|---|---|---|
| $n_X$ | 3 | 3 | 5 | 10 |
| $k$ | 100 | 500 | 100 | 500 |

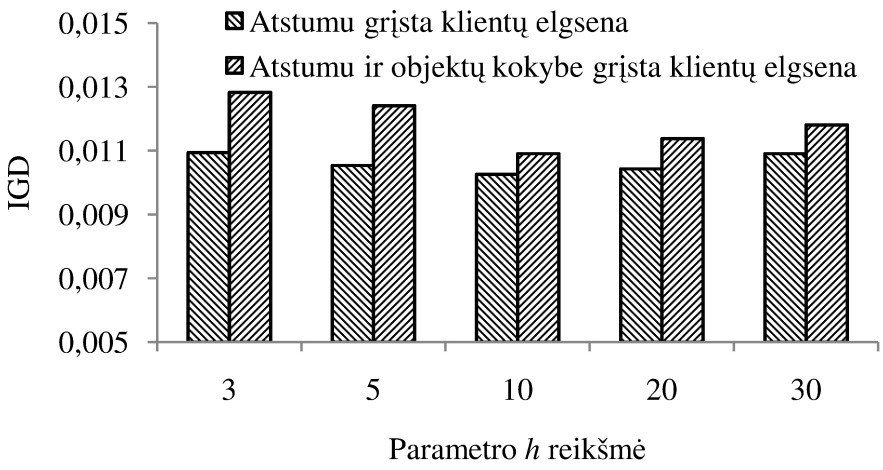
Buvo naudojamos tokios rinkoje esančių objektų kokybės parametrų $q_i$, $i = 1, 2, \ldots, 6$ vertės: 57, 60, 59, 45, 56, 36. Šios vertės parinktos atsitiktiniu būdu, atsižvelgiant į tai, kad objekto kokybės įtaka jo patrauklumui būtų konkurencinga atstumo iki paklauso taško įtakai (žr. (1.15) formulę). Visiems naujiems objektams buvo parenkamos vienodos kokybės vertės $q_n = 35$. Taip pat buvo tiriamas ir atskiras uždavinio atvejis, kai visų (naujų ir jau veikiančių) objektų kokybės vertės yra vienodos: $q_i = q_n = 1$, $i, j = 1, 2, \ldots, 6$. Šiuo atveju klientų elgsena grįsta tik atstumu iki objekto – klientai renkasi artimiausią objektą.

Algoritmų kokybė buvo vertinama apibendrinto atstumo (IGD) matu (žr. 1.1.4 poskyrį). Uždavinio su pirmaisiais trimis parametrų rinkiniais (žr. 3.10 lentelę) tikrieji Pareto frontai $P$ buvo rasti naudojant pilno perrinkimo algoritmą. Priklausomai nuo parametrų rinkinio, pilnajam perrinkimui reikėjo nuo $C_{100}^3$ iki $C_{500}^5$ tikslo funkcijų reikšmių skaičiavimų (algoritmo vykdymo trukmė buvo nuo 15 minučių iki 78 valandų, skaičiavimams naudojant i5 CPU 760@2.8GHz procesorių). Taip pat buvo ieškoma Pareto frontų aproksimacijų NSGA-II algoritmu. Buvo vykdoma 250 iteracijų, naudojant 100 individų populiaciją (viso 25000 tikslo funkcijos reikšmių skaičiavimų). Kiekvienas eksperimentas buvo kartojamas 100 kartų, naudojant skirtingas pradines populiacijas, o vertinami vidutiniai rezultatai.

Visais tirtais atvejais Pareto forntai buvo gan tiksliai aproksimuoti algoritmu NSGA-II: didžiausia apibendrinto atstumo (IGD) reikšmė buvo mažesnė už 0,004.

Sudėtingiau buvo sprendžiamas uždavinys su $n_X = 10$ ir $k = 500$ (ketvirtasis parametrų rinkinys). Šiuo atveju praktiškai neįmanoma atlikti pilnąjį perrinkimą per priimtiną laiką (tai turėtų užtrukti apie 54 dienas). Todėl Pareto frontu buvo laikoma Pareto fronto aproksimacija, sudaryta apjungiant visas (apie 1500), eksperimentų metu gautas, aproksimacijas.

Esant atstumu grįstam klientų elgsenos modeliui, Pareto frontą sudarė 937 nedominuojami sprendiniai, o atstumu ir objekto kokybe grįstam klientų elgsenos modeliui – 703 nedominuojami sprendiniai.

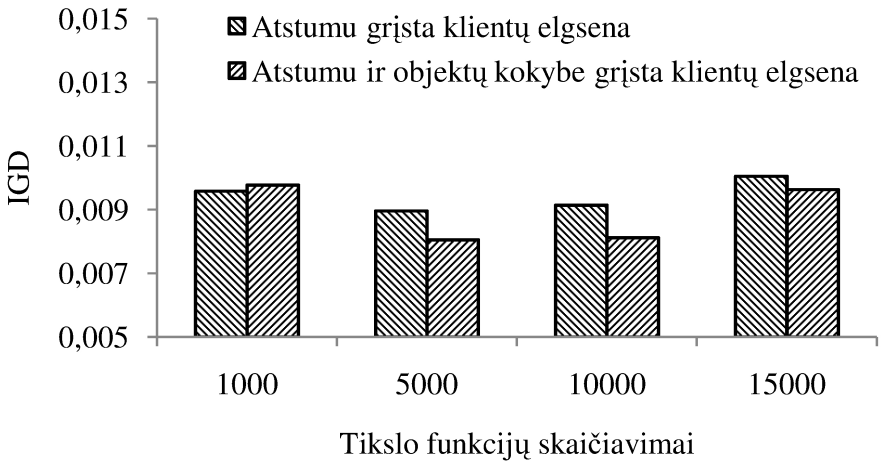



3.14 pav.: Vidutinės IGD vertės gautos sprendžiant CFL uždavinį NSGA-II algoritmu, naudojant skirtingas parametro $h$ reikšmes

Visi Pareto frontai buvo aproksimuojami NSGA-II algoritmu, naudojant 100 individų populiaciją ir atliekant 250 iteracijų. Buvo tiriamos skirtingos parametro $h$, apibrėžiančio objektų, naudojamų mutavimo metu, aibes (žr. 2.4 poskyrį), vertės: 3, 5, 10, 20, 30 ir $k-1$. Paskutiniu atveju, kai $h = k-1$, objektų aibės, naudojamos mutavimui, sutampa su aibe $L \setminus \{x_i\}$, čia $x_i$ reprezentuoja keičiamą objektą. Šiuo atveju vidutinė mato IGD vertė buvo 0,0144. Geresni rezultatai gauti naudojant mažesnes kandidatų aibes: $h \in \{3, 5, 10, 20, 30\}$. Šie rezultatai pateikti 3.14 paveiksle, kuriame skirtingi stulpeliai reiškia skirtingus klientų elgsenos modelius. Paveiksle matyti, kad parametro $h$ vertė turi įtakos aproksimacijos kokybei. Tinkamiausia parametro vertė yra $h = 10$ – atliekant mutavimą patartina naudoti kandidatų aibę, sudarytą iš 10 artimiausių objektų.

Toks pat uždavinys buvo sprendžiamas NSGA-II algoritmu su lokaliosios paieškos strategija (žr. 2.4 poskyrį). Buvo tiriamos 4 situacijos, NSGA-II algoritmui skiriant 24000, 20000, 15000 ir 10000 tikslo funkcijų skaičiavimų (atitinkamai atliekant 240, 200, 150 ir 100 iteracijų). Likę tikslo funkcijų skaičiavimai, atitinkamai 1000, 5000, 10000 ir 150000 buvo skiriami lokaliajam optimizavimui, pradiniais sprendiniais naudojant NSGA-II algoritmu gautą Pareto fronto aproksimaciją. Tiek mutavimo NSGA-II algoritme metu, tiek generuojant naujus sprendinius lokaliojoje paieškoje, panaudota parametro reikšmė $h = 10$.

Gauti rezultatai pateikti 3.15 paveiksle. Paveiksle matyti, kad lokaliajai paieškai vertinga skirti nuo 5000 iki 10000 tikslo funkcijų skaičiavimų. Lyginant 3.14 ir 3.15 paveiksluose pateiktus rezultatus matyti, kad dalies tikslo funkcijų skaičiavimų skyrimas lokaliajai paieškai gali pagerinti rezultatus iki 25 % (lokaliajai paieškai skiriant 5000–10000 tikslo funkcijų skaičiavimų).




3.15 pav.: Vidutinės IGD vertės gautos sprendžiant CFL uždavinį NSGA-II su lokaliąją paieška algoritmu, lokaliajai paieškai skiriant skirtingą tikslo funkcijų skaičiavimų kiekį

## 3.6 CFL uždavinio sprendimas lygiagrečiuoju NSGA-II

Šiame poskyryje aptariami rezultatai, gauti sprendžiant CFL uždavinį 2.5.2 poskyryje pasiūlytais lygiagrečiaisiais daugiakriterio optimizavimo algoritmais ParNSGA/DR, ParNSGA/HR, ParNSGA/HR-R ir ParNSGA/DR-R.

Sprendžiamo optimizavimo uždavinio tikslas buvo rasti optimalias vietas 5 objektams. Šiuo atveju naujų objektų vietos buvo parenkamos ne iš kandidatų aibės, bet parenkant geografines koordinates. Kadangi kiekvieno objekto vietą aprašo dvi koordinatės, tai bendras uždavinio kintamųjų skaičius buvo 10. Vėliau objektų skaičius buvo padidintas iki 25, taip gaunant uždavinį, turintį 50 kintamųjų.

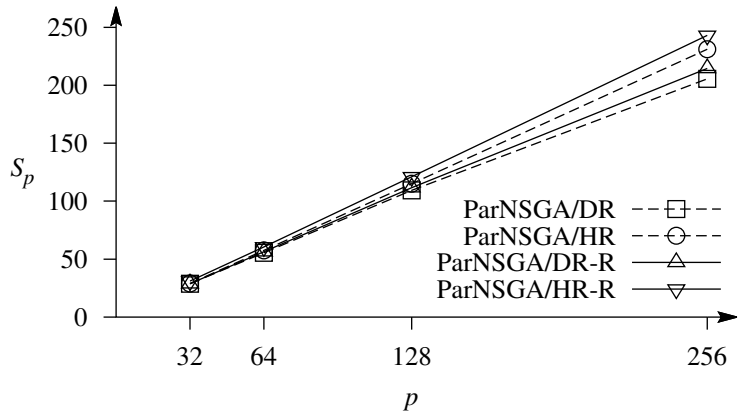
Buvo naudojami skirtingi populiacijų dydžiai: 256, 512, 1024 ir 2048 individų. Visais atvejais buvo atliekama 250 iteracijų. Kiekvienas eksperimentas buvo kartojamas 10 kartų ir vertinama vidutinė algoritmo vykdymo trukmė.

Priklausomai nuo populiacijos dydžio ir kintamųjų skaičiaus, uždavinio sprendimas nuosekliuoju algoritmu truko nuo 18 minučių (naudojant 256 individų populiaciją 10 kintamųjų uždaviniui spręsti) iki daugiau nei 12 valandų (naudojant 2048 individų populiaciją 50 kintamųjų uždaviniui spręsti).

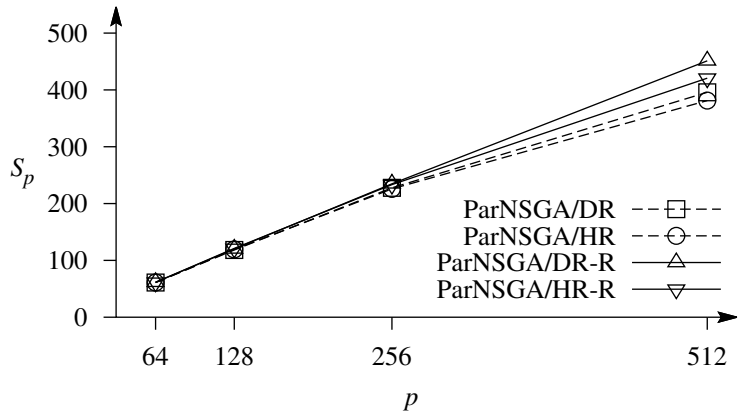
3.16 ir 3.17 paveiksluose yra pateikti lygiagrečiųjų algoritmų ParNSGA/DR, ParNSGA/HR, ParNSGA/HR-R ir ParNSGA/DR-R spartinimo

koeficientų priklausomybė nuo naudojamų procesorių skaičiaus, uždaviniui spręsti naudojant 256 (3.16 paveikslas) ir 512 (3.17 paveikslas) individų populiacijas. Atitinkamai populiacijų dydžiams buvo naudojama iki 256 ir 512 paskirstytosios atminties procesorių.

Paveiksluose matyti, kad abiem atvejais didžiausias spartinimo koeficientas buvo gautas algoritmais, naudojančiais blogų sprendinių atmetimo strategiją (ParNSGA/HR-R ir ParNSGA/DR-R algoritmai), tačiau, esant 256 individų populiacijai didesnis spartinimo koeficientas buvo gautas algoritmu ParNSGA/HR-R, naudojančiu hierarchinę sprendinių rangavimo lygiagretinimo strategiją HR (žr. 2.5.2 poskyrį), o esant 512 individų populiacijai, didesnis spartinimo koeficientas buvo gautas algoritmu ParNSGA/DR-R,




3.16 pav.: Lygiagrečiųjų algoritmų spartinimo koeficiento priklausomybė nuo procesorių skaičiaus (populiacijos dydis 256; kintamųjų skaičius 10)




3.17 pav.: Lygiagrečiųjų ParNSGA algoritmų spartinimo koeficiento priklausomybė nuo procesorių skaičiaus (populiacijos dydis 512; kintamųjų skaičius 10)

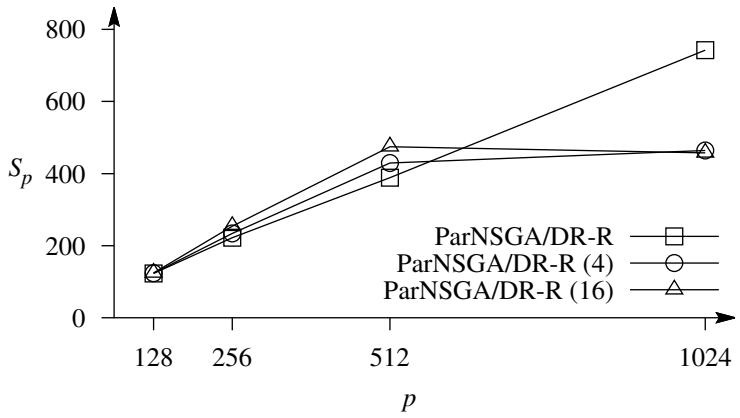
naudojančiu paskirstytąją sprendinių rangavimo lygiagretinimo strategiją DR (žr. 2.5.2 poskyrį). Toks strategijos DR pranašumas gali būti paaiškinamas sąlyginai didelėmis kompiuterių-darbininkų prastovomis, kuomet kompiuteris-šeimininkas atlieka paskutinius sprendinių rangavimo veiksmus strategijoje HR.

Tas pats optimizavimo uždavinys buvo sprendžiamas ParNSGA/DR-R algoritmu, naudojant 1024 ir 2048 individų populiacijas.

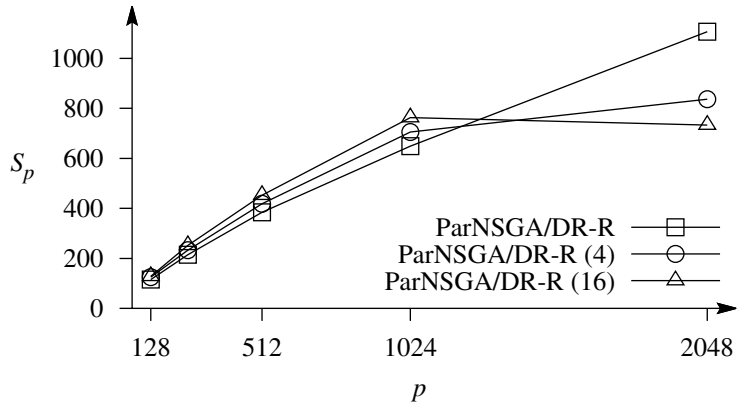
Naudojant 1024 individų populiaciją buvo atliekami 3 eksperimentai: naudojant iki 1024 paskirstytosios atminties procesorių; naudojant iki 256 grupių po 4 bendrosios atminties procesorių; naudojant iki 64 grupių po 16 bendrosios atminties procesorių. Visais 3 atvejais buvo naudojama iki 1024 procesorių.

Naudojant 2048 individų populiaciją taip pat buvo atlikti analogiški eksperimentai: naudojant iki 2048 paskirstytosios atminties procesorių; naudojant iki 512 grupių po 4 bendrosios atminties procesorių; naudojant iki 128 grupių po 16 bendrosios atminties procesorių.

Eksperimento rezultatai, gauti naudojant 1024 ir 2048 individų populiacijas, pateikti atitinkamai 3.18 ir 3.19 paveiksluose. Juose matyti, kad procesorių grupavimas į 4 ir 16 bendrosios atminties procesorių grupes (atitinkamai ParNSGA/DR-R (4) ir ParNSGA/DR-R (16)) yra vertingas skaičiavimams naudojant iki 512 procesorių, kai populiacijos dydis yra 1024 individų, ir – iki 1024 procesorių, kai populiacijos dydis yra 2048 individai. Naudojant maksimalų procesorių skaičių bendrosios atminties procesorių naudojimas nėra naudingas, bet priešingai – spartinimo koeficientas mažesnis nei algoritmo, naudojančio tik paskirstytosios atminties procesorius.



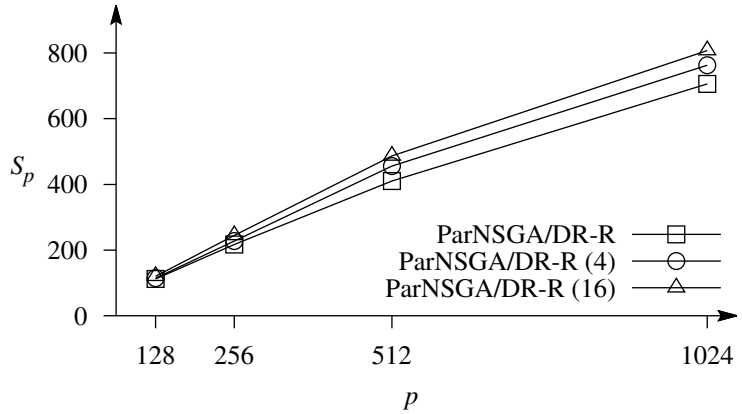

3.18 pav.: ParNSGA/DR algoritmo spartinimo koeficiento priklausomybė nuo procesorių skaičiaus (populiacijos dydis 1024; kintamųjų skaičius 10)




3.19 pav.: ParNSGA/DR algoritmo spartinimo koeficiento priklausomybė nuo procesorių skaičiaus (populiacijos dydis 2048; kintamųjų skaičius 10)

Pastebėkime, kad naudojant maksimalų kiekį procesorių, kiekvienam procesoriui siunčiama po vieną populiacijos individą, o vieno sprendinio rangavimas nėra lygiagretinamas. Todėl pradiniame sprendinių rangavimo etape bendrosios atminties procesorių naudojimas neįgauna pranašumo prieš paskirstytosios atminties procesorių naudojimą. Nors naudojant bendrosios atminties procesorius sutaupoma duomenų perdavimo kaštų, tačiau reikia papildomų laiko kaštų bendrosios atminties procesorių sinchronizacijai.

Bendrosios atminties procesorių naudojimo privalumus parodo rezultatai eksperimento, kurio metu buvo sprendžiamas 50 kintamųjų uždavinys. Šiuo atveju kelis kartus padidėjo duomenų perdavimo tarp paskirstytosios




3.20 pav.: ParNSGA/DR algoritmo spartinimo koeficiento priklausomybė nuo procesorių skaičiaus (populiacijos dydis 1024; kintamųjų skaičius 50)

atminties procesorių kaštai.

Eksperimento metu buvo naudojama 1024 individų populiacija ir iki 1024 bendrosios ir paskirstytosios atminties procesorių, kaip ir prieš tai atliktame eksperimente.

Eksperimento rezultatai pateikti 3.20 paveiksle, kuriame matyti, kad bendrosios atminties procesorių naudojimas turi pranašumą.

## 3.7 Trečiojo skyriaus apibendrinimas ir išvados

1. Įgyvendinta ir eksperimentiniu būdu ištirta dalelių spiečiaus optimizavimo algoritmo modifikacija. Tyrimo rezultatai parodė, kad sprendžiant erdvėlaivių trajektorijų optimizavimo uždavinį, tinkamas paieškos srities siaurinimas gali pagerinti algoritmo kokybę iki 25 %.

2. Lygiagrečiojo dalelių spiečiaus optimizavimo algoritmo eksperimentinis tyrimas parodė, kad geriausias spartinimo koeficientas pasiekiamas naudojant asinchroninę strategiją, vieną iš procesorių išskiriant komunikavimui valdyti.

3. Eksperimentiniu būdu ištirtas disertacijoje siūlomas daugiakriterio optimizavimo hibridinis algoritmas. Nustatyta, kad siūlomas memetinis algoritmas sprendžia geriau už klasikinį genetinį algoritmą NSGA-II nuo 53 % iki 77 % testo funkcijų, priklausomai nuo kokybės vertinimo mato.

4. Įgyvendintos ir eksperimentiniu būdu ištirtos sprendinių rangavimo daugiakriterio optimizavimo algoritmuose lygiagretinimo strategijos. Tyrimo rezultatai parodė, kad siūlomų lygiagretinimo strategijų naudojimas gali padidinti spartinimo koeficientą 1,6 karto.

# Bendrosios išvados

Išsprendus darbe suformuluotus uždavinius gautos tokios išvados:

1. Įgyvendinta dalelių spiečiaus optimizavimo algoritmo modifikacija, grįsta leistinosios srities mažinimu. Eksperimentiniu tyrimu nustatyta, kad tinkamas leistinosios srities mažinimas gali padidinti algoritmo efektyvumą iki 25 % sprendžiant erdvėlaivių skrydžių trajektorijų optimizavimo uždavinį.

2. Lygiagrečiojo dalelių spiečiaus optimizavimo algoritmo eksperimentinis tyrimas parodė, kad geriausias algoritmo spartinimo koeficientas pasiekiamas lygiagretinant asinchroninę dalelių spiečiaus optimizavimo algoritmo versiją, vieną iš procesorių išskiriant informacijos mainų užtikrinimui.

3. Pasiūlytas lokaliojo daugiakriterio optimizavimo algoritmas, modifkuojant vieno agento stochastine paiešką grįstą algoritmą. Siūlomas algoritmas įkomponuotas į daugiakriterį genetinį algoritmą, taip sudarant hibridinį globaliojo daugiakriterio optimizavimo algoritmą. Eksperimentiniu tyrimu nustatyta, kad pasiūlytas algoritmas sprendžia geriau už klasikinį daugiakriterio optimizavimo genetinį algoritmą nuo 53 % iki 77 % testo uždavinių, priklausomai nuo vertinimo kriterijaus.

4. Pasiūlytos sprendinių vertinimo daugiakriterio optimizavimo algoritmuose lygiagretinimo strategijos. Eksperimentinio tyrimo rezultatai parodė, kad siūlomos algoritmo lygiagretinimo strategijos gali padidinti lygiagrečiojo algoritmo spartinimo koeficientą 1,6 karto.

5. Pasiūlyta mutavimo genetiniame algoritme strategija, taikytina sprendžiant konkuruojančių objektų vietos parinkimo uždavinį. Pasiūlytos strategijos pagrindu įgyvendintas lokaliojo optimizavimo algoritmas. Eksperimentinis tyrimas parodė, kad tinkamas siūlomos mutacijos strategijos ir lokaliosios paieškos naudojimas gali padidinti genetinio algoritmo efektyvumą ∼43 %, sprendžiant konkuruojančių objektų vietos parinkimo uždavinį.

# Literatūros sąrašas


[1] G. Dzemyda, V. Šaltenis, V. Tiešis. *Optimizavimo metodai.* Mokslo aidai, 2007.

[2] A. Žilinskas. *Matematinis programavimas.* VDU leidykla, 2005.

[3] T. Bäck, D.B. Fogel, Z. Michalewicz (redaktoriai). *Evolutionary Computation 1: Basic Algorithms and Operators.* Institute of Physics Publishing, Bristol, 2000.

[4] T. Bäck, D.B. Fogel, Z. Michalewicz (redaktoriai). *Evolutionary Computation 2: Advanced Algorithms and Operators.* Institute of Physics Publishing, Bristol, 2000.

[5] A. Törn and A. Žilinskas. *Global Optimization*, volume 350 of *Lecture Notes in Computer Science.* Springer-Verlag, 1989.

[6] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[7] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, MA, USA, 1998.

[8] M.D. Vose. *The Simple Genetic Algorithm: Foundations and Theory.* MIT Press, Cambridge, MA, USA, 1998.

[9] D. Ashlock. *Evolutionary Computation for Modeling and Optimization.* Springer, 2004.

[10] T. Bäck, D.B. Fogel, Z. Michalewicz (redaktoriai). *Handbook of Evolutionary Computation.* IOP Publishing Ltd., Bristol, UK, 1997.

[11] J. Kennedy and R. Eberhart. Particle swarm optimization. In *IEEE International Conference on Neural Networks, 1995. Proceedings.*, volume 4, pages 1942–1948, 1995.

[12] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *1997 IEEE International Conference on Systems, Man, Cybernetics*, volume 5, pages 4104–4108, 1997.

[13] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, oct 1995. doi: 10.1109/MHS.1995.494215.

[14] M. Dorigo. *Optimization, Learning and Natural Algorithms*. Daktaro disertacija, Politecnico di Milano, 1992.

[15] A. Colorni, M. Dorigo, V. Maniezzo. Distributed optimization by ant colonies. In *European Conference on Artificial Life*, pages 134–142, 1991.

[16] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, M. Zaidi. The bees algorithm, a novel tool for complex optimisation problems. In *Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459. Elsevier, 2006.

[17] D. Šešok. *Santvarų topologijos optimizavimas genetiniais algoritmais*. Daktaro disertacija, VGTU, 2008.

[18] R. Paulavičius. *Globalusis optimizavimas su simpleksiniais posričiais*. Daktaro disertacija, MII, 2010.

[19] E. Filatovas. *Daugiakriterinių optimizavimo uždavinių sprendimas interaktyviuoju būdu*. Daktaro disertacija, MII, 2012.

[20] J.G. Lin. Multiple-objective problems: Pareto-optimal solutions by method of proper equality constraints. *IEEE Transactions on Automatic Control*, 21(5):641–650, 1976.

[21] E. Zitzler, K. Deb, L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8: 173–195, 2000.

[22] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, E. Tsang. Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. In *Proceedings of the Congress on Evolutionary Computation (CEC)*, pages 3234–3241. IEEE Press, 2006.

[23] B. Addis, A. Cassioli, M. Locatelli, F. Schoen. A global optimization method for the design of space trajectories. *Computational Optimization and Applications*, 48(3):635–652, 2011.

[24] D. Izzo, V.M. Becerra, D.R. Myatt, S.J. Nasuto, J.M. Bishop. Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. *Journal of Global Optimization*, 38(2):283–296, June 2007.

[25] T.L. Friesz, T. Miller, R.L. Tobin. Competitive network facility location models: a survey. *Papers of the Regional Science Association*, 65: 47–57, 1998.

[26] F. Plastria. Static competitive facility location: An overview of optimisation approaches. *European Journal of Operational Research*, 129 (3):461–470, 2001.

[27] C.S. ReVelle, H.A. Eiselt, M.S. Daskin. A bibliography for some fundamental problem categories in discrete location science. *European Journal of Operational Research*, 184(3):817–848, 2008.

[28] A. Ghosh and C.S. Craig. FRANSYS: a franchise distribution system location model. *Journal of Retailing*, 67(4):466–495, 1991.

[29] R.L. Francis, T.J. Lowe, A. Tamir. Demand point aggregation for location models. In Z. Drezner and H. Hamacher (redaktoriai), *Facility Location: Application and Theory*, pages 207–232. 2002.

[30] L. Hakimi. Location with spatial interactions: Competitive locations and games. In Z. Drezner (redaktorius), *Facility Location: A Survey of Applications and Methods*, pages 367–386. Springer, 1995.

[31] R. Suarez-Vega, D. Santos-Peante, P. Dorta-Gonzalez. Discretization and resolution of the $(r|X_p)$-medianoid problem involving quality criteria. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 12(1):111–133, 2004.

[32] R. Suarez-Vega, D.R. Santos-Penate, D.P. Gonzalez. The follower location problem with attraction thresholds. *Papers in Regional Science*, 86(1):123–137, 2007.

[33] P.H. Peeters and F. Plastria. Discretization results for the Huff and Pareto-Huff competitive location models on networks. *TOP*, 6:247–260, 1998.

[34] D. Serra and R. Colome. Consumer choice and optimal locations models: Formulations and heuristics. *Papers in Regional Science*, 80(4): 439–464, 2001.

[35] T. Drezner and Z. Drezner. Finding the optimal solution to the Huff based competitive location model. *Computational Management Science*, 1(2):193–208, 2004.

[36] J.G. Villegas, F. Palacios, A.L. Medaglia. Solution methods for the bi-objective (cost-coverage) unconstrained facility location problem with an illustrative example. *Annals of Operations Research*, 147:109–141, 2006.

[37] L. Huapu and W. Jifeng. Study on the location of distribution centers: A bi-level multi-objective approach. In *Logistics*, pages 3038–3043. American Society of Civil Engineers, 2009.

[38] S. Liao and C. Hsieh. A capacitated inventory-location model: Formulation, solution approach and preliminary computational results. In B.C. Chien, T.P. Hong, S.M. Chen, M. Ali (redaktoriai), *Next-Generation Applied Intelligence*, volume 5579 of *Lecture Notes in Computer Science*, pages 323–332. Springer Berlin Heidelberg, 2009.

[39] F.J. Solis and B. Wets. Minimization by random search techniques. *Mathematics of Operations Research*, 6(1):19–30, 1981.

[40] S. Das, A. Abraham, A. Konar. Particle swarm optimization and differential evolution algorithms: Technical analysis, applications and hybridization perspectives. *Studies in Computational Intelligence*, 116: 1–38, 2008.

[41] C. Darwin. *On the origin of species*. John Murray, 1859.

[42] M. Srinivas and L.M. Patnaik. Genetic algorithms: a survey. *Computer*, 27(6):17–26, 1994. doi: 10.1109/2.294849.

[43] N. Srinivas and K. Deb. Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995.

[44] K. Mitra, K. Deb, S.K. Gupta. Multiobjective dynamic optimization of an industrial nylon 6 semibatch reactor using genetic algorithms. *Journal of Applied Polymer Science*, 69(1):69–87, 1998.

[45] D.S. Weile, E. Michielssen, D.E. Goldberg. Genetic algorithm design of Pareto optimal broadband microwave absorbers. *IEEE Transactions on Electromagnetic Compatibility*, 38(3):518–525, 1996.

[46] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858. Springer, 2000.

[47] D.E. Knuth. *The art of computer programming, volume 3: (2nd ed.) sorting and searching*. Addison Wesley Longman Publishing Co., Inc., 1998.

[48] D.P. Kroese, T. Taimre, Z.I. Botev. *Handbook of Monte Carlo Methods (Wiley Series in Probability and Statistics)*. Wiley, 1 edition, 2011.

[49] D.H. Lehmer. Mathematical methods in large-scale computing units. In *2nd Symposium on Large-Scale Digital Calculating Machinery*, pages 141–146. Harvard University Press, 1951.

[50] M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8 (1):3–30, 1998.

[51] G. Marsaglia. On the randomness of pi and other decimal expansions. *InterStat*, pages 1–17, 2005.

[52] F.J. Massey. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[53] R. Čiegis. *Lygiagretieji algoritmai*. Technika, 2002.

[54] M. Baravykaitė. *Lygiagrečių algoritmų šablonų tyrimas ir kūrimas*. Daktaro disertacija, VGTU, 2006.

[55] S. Ivanikovas. *Lygiagrečių skaičiavimų taikymo daugiamačiams duomenims vizualizuoti problemos*. Daktaro disertacija, MII, 2010.

[56] A. Jakušev. *Diferencialinių lygčių ir jų sistemų skaitinio sprendimo algoritmų lygiagretinimo technologijos kūrimas, analizė ir taikymai*. Daktaro disertacija, VGTU, 2007.

[57] T. Petkus. *Kompiuterių tinklo panaudojimas interaktyviame optimizavime*. Daktaro disertacija, VPU, 2001.

[58] R. Šablinskas. *Lygiagrečiųjų algoritmų tyrimas paskirstytos atminties lygiagretiems kompiuteriams*. Daktaro disertacija, VDU, 1999.

[59] J. Žilinskas. *Black Box Global Optimization: Covering methods and their parallelization*. Daktaro disertacija, KTU, 2002.

[60] A. Igumenov. *Lygiagretieji ir paskirstytieji elektros energiją tausojantys skaičiavimai*. Daktaro disertacija, MII, 2012.

[61] V. Starikovičius. *Parallel numerical algorithms for multiphase flow models*. Daktaro disertacija, Vilniaus universietas, 2002.

[62] G.M. Amdahl. Validity of the single-processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485, 1967.

[63] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2002.

[64] J. F Schutte, J.A Reinbolt, B.J Fregly, R. T Haftka, A.D. George. Parallel global optimization with the particle swarm algorithm. *Journal of Numerical Methods in Engineering*, 61:2296–2315, 2003.

[65] J.L. Redondo, J. Fernández, I. García, P.M. Ortigosa. Parallel algorithms for continuous competitive location problems. *Optimization Methods and Software*, 23(5):779–791, 2008.

[66] J.L. Redondo, I. Garcia, P.M. Ortigosa. Parallel evolutionary algorithms based on shared memory programming approaches. *The Journal of Supercomputing*, 58:270–279, 2011.

[67] K. Deb, P. Zope, A. Jain. Distributed computing of pareto-optimal solutions with evolutionary algorithms. In C.M. Fonseca, P.J. Fleming,

E. Zitzler, L. Thiele, K. Deb, editors, *Evolutionary Multi-Criterion Optimization*, volume 2632 of *Lecture Notes in Computer Science*, pages 534–549. Springer Berlin Heidelberg, 2003.

[68] F. Streichert, H. Ulmer, A. Zell. Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In C.C. Coello, A.A. Hernández, E. Zitzler (redaktoriai), *Evolutionary Multi-Criterion Optimization*, volume 3410 of *Lecture Notes in Computer Science*, pages 92–107. Springer Berlin / Heidelberg, 2005.

[69] J.J. Durillo, A.J. Nebro, F. Luna, E. Alba. A study of master-slave approaches to parallelize NSGA-II. In *IEEE International Symposium on Parallel and Distributed Processing, 2008 (IPDPS 2008)*, pages 1–8, 2008.

[70] C.A.C. Coello, G.B. Lamont, D.A.V. Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems (2nd Edition)*. Springer-Verlag Inc., New York, NJ, USA, 2007.

[71] Q. Zhang, A. Zhou, S. Zhao, P.N. Suganthan, W. Liu, S. Tiwari. Multiobjective optimization test instances for the cec 2009 special session and competition. Technical report, University of Essex, School of Computer Science and Electrical Engineering, 04 2008.

[72] C. Grosan and A. Abraham. Approximating Pareto frontier using a hybrid line search approach. *Information Sciences*, 180(14):2674–2695, 2010.

[73] F. Kursawe. A variant of evolution strategies for vector optimization. In *Parallel Problem Solving from Nature. 1st Workshop, PPSN I, volume 496 of Lecture Notes in Computer Science*, pages 193–197. Springer-Verlag, 1991.

[74] K. Deb. Multi-objective genetic algorithms: Problem difficulties and construction of test problems. *Evolutionary Computation*, 7:205–230, 1999.

[75] E. Zitzler, M. Laumanns, L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. C. Giannakoglou, D.T. Tsahalis, J. Périaux, K.D. Papailiou, and T. Fogarty (redaktoriai), *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100. International Center for Numerical Methods in Engineering, 2001.

[76] Q. Zhang and H. Li. MOEA/D: A multi-objective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation*, 11(6):712–731, 2007.

[77] T. Vinko, D. Izzo, C. Bombardelli. Benchmarking different global optimisation techniques for preliminary space trajectory design. In *58th International Astronautical Congress*, 2007.

# Autoriaus publikacijų disertacijos tema sąrašas


[A1] **A. Lančinskas**, J. Žilinskas. Methods for generation of random numbers in parallel stochastic algorithms for global optimization. *Journal of Young Scientists*, 2(27):118–122, 2010. ISSN 1648-8776.

[A2] **A. Lančinskas**, J. Žilinskas, P.M. Ortigosa. Investigation of parallel particle swarm optimization algorithm with reduction of the search area. In *Cluster Computing Workshops and Posters (CLUSTER WORKSHOPS), 2010 IEEE International Conference on*, p. 1–5, Sept. 2010. ISBN 978-1-4244-8395-2.

[A3] **A. Lančinskas**, J. Žilinskas, P.M. Ortigosa. Local optimization in global multi-objective optimization algorithms. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, p. 323–328, 2011. ISBN 978-1-4577-1122-0.

[A4] **A. Lančinskas**, J. Žilinskas. Approaches to parallelize pareto ranking in NSGA-II algorithm. In *Parallel Processing and Applied Mathematics, Vol. 7204 of Lecture Notes in Computer Science*, p. 371–380, 2012. ISSN 0302-9743.

[A5] **A. Lančinskas**, J. Žilinskas. Solution of multi-objective competitive facility location problems using parallel NSGA-II on large scale computing systems. In *Applied Parallel and Scientific Computing, Vol. 7782 of Lecture Notes in Computer Science*, p. 422–433, 2013. ISSN 0302-9743.

[A6] **A. Lančinskas**, P.M. Ortigosa, J. Žilinskas. Multi-Objective Single Agent Stochastic Search in Non-dominated Sorting Genetic Algorithm. *Nonlinear Analysis: Modelling and Control*. [Priimtas publikavimui]

Algirdas Lančinskas

# ATSITIKTINĖS PAIEŠKOS GLOBALIOJO OPTIMIZAVIMO ALGORITMŲ LYGIAGRETINIMAS

Daktaro disertacija

Fiziniai mokslai (P 000)
Informatika (09 P)
Informatika, sistemų teorija (P 175)

Algirdas Lančinskas

# PARALLELIZATION OF RANDOM SEARCH GLOBAL OPTIMIZATION ALGORITHMS

Doctoral dissertation

Physical Sciences (P 000)
Informatics (09 P)
Informatics, Systems Theory (P 175)