VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Atsitiktinės paieškos optimizavimo algoritmų vertinimas

Evaluation of Random Search Optimization Algorithms

Magistro baigiamasis darbas

Atliko:	Rimantas Norvilas	(parašas)
Darbo vadovas:	dr. Algirdas Lančinskas	(parašas)
Recenzentas:	prof. dr. Romas Baronas	(parašas)

Vilnius – 2017

SANTRAUKA

Sudėtinga įvertinti atsitiktinės paieškos algoritmus, nes nepriklausomai nuo pasirinkto efektyvumo vertinimo metodo, norint gauti patikimus statistinius įverčius reikia aibės eksperimentų atlikimo ir detalios gautų rezultatų analizės. Darbe nagrinėjami atsitiktinės paieškos globaliojo optimizavimo algoritmų efektyvumo vertinimo metodai. Darbe siūlomas naujas algoritmų vertinimo metodas paremtas gauto sprendinio tikslumo ir tikimybės jį gauti tarpusavio priklausomybe. Siūlomas metodas buvo validuojamas vertinant gerai žinomų atsitiktinės paieškos optimizavimo algoritmų efektyvumą sprendžiant įvairius globaliojo optimizavimo testo uždavinius. Gauti eksperimentinio tyrimo rezultatai parodė, kad siūlomu metodu gautuose vertinimuose atsispindi įvairios algoritmų elgesio charakteristikos, tokios kaip gebėjimas rasti globalųjį sprendinį ar lokaliųjų sprendinių trauka. Metodas taip pat yra tinkamas efektyviam skirtingų algoritmų efektyvumo palyginimui. Taigi šiame darbe pristatytas naujas algoritmų vertinimo metodas, kuris naudoja pasiūlytą hipertūrio skaičiavimo principą ir dominuojamumo sąryšį. Pasiūlytas metodas suteikia papildomos informacijos apie atsitiktinės paieškos algoritmus.

Raktiniai žodžiai: globalusis optimizavimas, atsitiktinė paieška, algoritmų vertinimo metodai, hipertūris, dominuojamumo sąryšis.

SUMMARY

It is difficult to evaluate a random search algorithms, because regardless of a chosen method of efficiency evaluation, in order to obtain reliable statistical estimates set of experiments and a detailed analysis of the obtained results is required. The work deals with an efficiency evaluation methods of random search algorithms for global optimization. A new algorithm assessment method based on the accuracy of the resulting solution, and the probability of it getting interdependence is proposed in this work. The proposed method has been validated by assessment of well-known random search optimization algorithms efficiency on a set of global optimization test problems. The experimental results showed that the proposed method to obtain estimates show the different algorithms behavioral characteristics such as the ability to find a global solution or attraction of local solutions. The method is also suitable for the efficient comparison of different algorithms efficiency. Thus, this paper presents a new algorithm assessment method, which uses the proposed hypervolume calculation principle and dominance relation. The proposed method provides additional information about the random search algorithms.

Keywords: global optimization, random search, benchmarking, algorithms evaluation methods, hypervolume, dominance relation.

TURINYS

ĮVADAS 4

1. GLOBALUSIS OPTIMIZAVIMAS 6

1.1. Globaliojo optimizavimo uždaviniai 6

1.2. Optimizavimo metodų klasifikacija 8

1.3. Optimizavimo uždavinių klasifikacija 9

1.4. Optimizavimas naudojant atsitiktinę paiešką 10

1.5. Atsitiktinės paieškos optimizavimo algoritmų vertinimas 13

2. ATSITIKTINĖS PAIEŠKOS ALGORITMŲ VERTINIMAS 22

2.1. Algoritmų palyginimas hipertūriu 23

2.2. Dominuojamumo vertinimas 31

2.3. Hipertūris be dominuojančių taškų 33

2.4. Algoritmų tarpusavio dominuojamumas 34

3. EKSPERIMENTINIS TYRIMAS 37

3.1. Algoritmų vertinimo įrankis 37

3.2. Testiniai uždaviniai 40

3.3. Atsitiktinės paieškos algoritmai 42

3.4. Eksperimento rezultatai 45

4. REZULTATAI IR IŠVADOS 56

LITERATŪRA 57

ĮVADAS

Jau nuo neatmenamų laikų žmonės sprendžia optimizavimo uždavinius – ieško optimalių, tam tikrais aspektais geriausių alternatyvų. Šiomis dienomis optimizavimo uždaviniai yra sutinkami įvairiose mokslo ir pramonės srityse, o dėl spartaus technologijų vystymosi formuluojami vis sudėtingesni, daug skaičiavimo resursų reikalaujantys uždaviniai [GLM12; HPV00]. Dėl šios priežasties efektyvių algoritmų sudėtingiems optimizavimo uždaviniams spręsti kūrimas ir vystymas yra aktuali šių dienų mokslo sritis.

Paprastai efektyviausiai yra sprendžiami optimizavimo uždaviniai, turintys analitines išraiškas, tenkinančias tam tikras matematines savybes, tokias kaip tolydumas, iškilumas, diferencijuojamumas ir pan. Tačiau ne visi praktikoje sutinkami optimizavimo uždaviniai tenkina šias savybes arba netgi nėra žinoma adekvati matematinė išraiška. Dėl nežinomos matematinės išraiškos yra ribojamas klasikinių matematinio optimizavimo algoritmų taikymas optimizavimo uždaviniams spręsti [Lan13].

Kita vertus, sprendžiant praktinius optimizavimo uždavinius ne visada būtina rasti tikslų uždavinio sprendinį, bet pakanka apytikslio sprendinio – tikslaus sprendinio aproksimacijos. Tam yra plačiai taikomi atsitiktinės paieškos optimizavimo metodai, kurie gali būti taikomi praktiškai bet kurio optimizavimo uždavinio sprendimui [Ash06]. Atsitiktinės paieškos metodai turi potencialą išspręsti plataus spektro problemas, kurių negalima išspręsti deterministiniais algoritmais. Kitas atsitiktinės paieškos metodų privalumas yra santykinai lengvas pritaikymas sudėtingiems uždaviniams. Kadangi paprastai šie metodai remiasi funkcijų reikšmių įvertinimu, jie gali būti lengvai pritaikomi įvairiems optimizavimo uždaviniams spręsti, bet šių metodų trūkumas yra tai, kad jie yra pritaikomi kiekvienam specifiniam uždaviniui „bandymų ir klaidų“ metodu [Zab09].

Pats paprasčiausias atsitiktinės paieškos metodas yra Monte-Karlo metodas, kuris remiasi atsitiktinių uždavinio sprendinių generavimu. Sudėtingesni atsitiktinės paieškos metodai, tokie kaip valdoma atsitiktinė paieška (angl. *Controlled Random Search*, CRS), genetiniai algoritmai (angl. *Genetic Algorithms*, GA) ar dalelių spiečiaus optimizacija (angl. *Particle Swarm Optimization*, PSO) yra vadinami adaptyviaisiais atsitiktinės paieškos metodais, kurie naują sprendinį apskaičiuoja atsižvelgdami į jau anksčiau atliktų skaičiavimų rezultatus. Kadangi visi šie algoritmai yra grįsti atsitiktinių skaičių generavimu, kiekvieną kartą gaunami skirtingi rezultatai net ir nekei-

čiant pradinių duomenų. Tai apsunkina algoritmo efektyvumo vertinimą, sprendžiant vieną ar kitą optimizavimo uždavinį.

Paprastai atsitiktinės paieškos optimizavimo algoritmų efektyvumo vertinimui pasitelkiami statistiniai metodai ir vertinami tokiais kriterijais kaip vidutinis gauto sprendinio tikslumas, gautų sprendinių išsibarstymas (dispersija), tikimybė gauti tam tikro tikslumo sprendinį po numatyto sprendimo laiko arba numatyto tikslo funkcijų perskaičiavimų skaičiaus ir pan. Literatūroje yra siūloma daug ir įvairių atsitiktinės paieškos optimizavimo algoritmų vertinimo metodų, o konkretaus metodo(-ų) pasirinkimą lemia aktualios vertinamo algoritmo savybės [CS00; HL15; ROŽ12]. Nepriklausomai nuo pasirinkto vertinimo metodo, norint gauti patikimus statistinius įverčius, atsitiktinės paieškos optimizavimo algoritmo vertinimas reikalauja aibės eksperimentų atlikimo ir detalios gautų rezultatų analizės įvertinant tiek atskirų eksperimentų rezultatus, tiek paskaičiuojant visų atliktų eksperimentų statistinius įverčius [AKZ05; HL15].

Atsižvelgiant į poreikį apdoroti ir didelius duomenų kiekius ir vertinti įvairias jų savybes, efektyviam algoritmų vertinimui yra svarbus įrankis, atliekantis, algoritmo vertinimui, reikiamus skaičiavimus. Bus sudaromas įrankio patogiam atsitiktinės paieškos optimizavimo algoritmų vertinimui ir lyginimui veikiančio įrankio prototipas, apjungiantis aibę skirtingų vertinimo metodų, aktualių šių dienų atsitiktinės paieškos algoritmams vertinti.

Šio darbo **tyrimo objektas** yra atsitiktinės paieškos globaliojo optimizavimo algoritmų vertinimo metodai.

Darbo tikslas yra pasiūlyti metodą, skirtą atsitiktinės paieškos optimizavimo algoritmų efektyvumo vertinimui, atsižvelgiant į algoritmo dominuojamumą ir sukurti jį realizuojančią taikomąją programą. Norint pasiekti šį tikslą išsikelti šie **uždaviniai:**

1. Atrinkti ir išanalizuoti aktualius atsitiktinės paieškos algoritmų vertinimo metodus.
2. Pasiūlyti atsitiktinės paieškos algoritmų, vertinimo metodą, vertinantį algoritmų tarpusavio dominuojamumą.
3. Realizuoti pasiūlytą metodą pasirinkta programavimo kalbą.
4. Eksperimentiniu būdu ištirti pasiūlytą, atsitiktinės paieškos algoritmų vertinimo metodą.

1. GLOBALUSIS OPTIMIZAVIMAS

Žmonės ieško optimalių, geriausių sprendinių jau nuo neatmenamų laikų. Optimalumo terminas buvo įvestas Leibnico tik 1710 metais, tačiau senieji graikai Euklidas ir Aleksandrijos Heronas jau sprendė optimizavimo uždavinį, mėgindami įrodyti, kad optinėse sistemose šviesa renkasi trumpiausią kelią [DŠT07; Lan13].

Nuo neatmenų laikų žmonės siekia tobulumo daugelyje sričių. Žmonės visada norėjo pasiekti didžiausią laimės lygį pasitelkiant kuo mažiau pastangų. Pavyzdžiui, ekonomikoje pelnas ir pardavimai turi būti kuo didesni, o išlaidos – kuo mažesnės. Taigi, optimizavimas yra vienas seniausių mokslų, kuris netgi persikelia į kasdienį gyvenimą. Žmonija vystosi, auga jos poreikiai, dėl to ir reikia spręsti vis sudėtingesnius optimizavimo uždavinius. [MBK11; Neu06].

1.1. Globaliojo optimizavimo uždaviniai

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:

$$f_{min} = \min_{\mathbf{x}\in\mathbb{D}} f(\mathbf{x}) \tag{1}$$

Čia funkcija $f : \mathbb{D} \to \mathbb{R}^d$ yra vadinama tikslo funkcija,

$$\mathbf{x} = (x_1, x_2, ..., x_d) \tag{2}$$

tikslo funkcijos (2) *kintamųjų vektorius*, *d* – kintamųjų skaičius, o $\mathbb{D} \subset \mathbb{R}^d$ – parametrų reikšmių *leistinoji sritis*.

Analogiškai apibrėžiamas maksimalios tikslo funkcijos reikšmės paieškos uždavinys:

$$f_{max} = \max_{\mathbf{x}\in\mathbb{D}} f(\mathbf{x}) \tag{3}$$

Nemažindami bendrumo, toliau nagrinėsime minimalios tikslo funkcijos reikšmės paieškos atvejį. Parametrų vektorius $\mathbf{x}^* \in \mathbb{D}$, toks, kad

$$f(\mathbf{x}^*) = f_{min} \tag{4}$$

yra vadinamas tikslo uždavinio *optimaliu sprendiniu* (optimumu). Beje, minimizavimo uždavinys yra ekvivalentus maksimizavimo uždaviniui su ta pačia tikslo funkcija, tik padauginta iš minus vieno. Sprendinys $\mathbf{x}^*$, kuris yra optimalus savo aplinkoje (poaibio $\mathbb{D}' \subset \mathbb{D}$ optimalus sprendinys), yra vadinamas *lokaliuoju optimaliu sprendiniu*, o sprendinys, kuris yra optimalus visoje leistinojoje srityje $\mathbb{D}$, yra vadinamas *globaliuoju optimaliu sprendiniu*. Lokaliojo arba globaliojo optimalaus sprendinio paieška yra vadinama atitinkamai *lokaliuoju optimizavimu* arba *globaliuoju optimizavimu* [Lan13].

Globaliojo optimizavimo algoritmas „užkerta“ kelią konvergavimui į lokalų optimalų sprendinį, pasitelkiant tam tikras priemones, taip padidindamas tikimybę rasti globalų optimalų sprendinį [TŽ89].

Labai svarbu, kai globaliojo optimizavimo metodą realizuojantis algoritmas yra pritaikomas konkrečiai uždavinių klasei, algoritme turi būti analizuojama visa žinoma informacija apie uždavinį, nes kuo daugiau žinoma informacijos apie uždavinį – tuo geresnių sprendinių galima tikėtis [Šeš08].

Sprendžiant praktikoje pasitaikančius optimizavimo uždavinius dažnai tenka atsižvelgti į daugiau nei vieną kriterijų. Tokie optimizavimo uždaviniai, turintys dvi ar daugiau tikslo funkcijų, yra vadinami daugiakriteriais optimizavimo uždaviniais. Natūralu, kad kriterijai gali būti prieštaringi vienas kitam, ir dažnai neįmanoma rasti vieno sprendinio, kuris būtų geriausias pagal visus uždavinio kriterijus. Todėl daugiakriterio optimizavimo uždavinio sprendimo tikslas – rasti aibę kompromisinių sprendinių – kitaip vadinamą Pareto optimaliais sprendiniais, kurie visi yra optimalūs tam tikra prasme [LOŽ13; Ste86].

Nors globaliojo optimizavimo uždaviniai dažnai yra sudėtingi ir reikalaujantys daug skaičiavimo resursų, jie yra sutinkami įvairiose inžinerijos srityse, tokiose kaip inžineriniame projektavime, mechanikoje, civilinėje ir cheminėje inžinerijose, struktūriniame optimizavime molekulinėje biologijoje ir molekulinėje architektūroje, grafiniam apdorojimui ir pan. [ZSM+93]. Inžinerijos

funkcijos įtrauktos į globaliojo optimizavimo uždavinius yra dažnai apibūdinamos kaip juodos dėžės (angl. *Black Box*) tipo funkcijos, kurios gali būti traktuojamos kaip paprogramės, kurios grąžina funkcijų reikšmes [Zab98]. Šio tipo inžinerinis uždavinio pavyzdys gali būti sumažinti struktūros svorį, sumažinant apkrovą iki tam tikro lygio [Zab98]. Kitas pavyzdys, kai optimizavimas pritaikomas spręsti konkuruojančių objektų vietos parinkimo uždavinius, kuriuose tikslo funkcijos dažnai traktuojamos, kaip juodos dėžės tipo funkcijos [LOŽ14; LŽ12; LŽ14].

Optimalaus sprendinio radimas dažnai yra sudėtingas ir daug skaičiavimo resursų reikalaujantis uždavinys, kurio išsprendimas per priimtiną laiką kartais yra neįmanomas. Kita vertus, sprendžiant praktinius uždavinius nėra būtinybės rasti tikslų sprendinį, bet pakanka nurodyti priimtino tikslumo aproksimaciją. Tam yra naudojamos įvairios optimalaus sprendinio aproksimavimo strategijos, tokios kaip, Genetiniai [Mit98; Vos99; Wal99] bei Modeliuojamojo atkaitinimo [KV$^+$83; Pup09], Dalelių spiečiaus [EK$^+$95; KE95; KE97] arba Skruzdžių kolonijos [CDM$^+$91; Dor92] optimizacijos algoritmai ir pan.

1.2. Optimizavimo metodų klasifikacija

Paprastai optimizavimo metodas yra pasirenkamas, kad atitiktų sprendžiamo uždavinio struktūrą. Pavyzdžiui, tiesinio ir netiesinio programavimo uždaviniams spręsti parenkami atitinkami algoritmai, kurie naudoja atitinkamus uždavinio savybes. Pavyzdžiui, netiesinis uždavinys bus geriau sprendžiamas algoritmais, kurie naudoja Hessian matricas, tačiau sunku pasirinkti geriausią algoritmą [Mur85]. Dėl tos priežasties yra eksperimentuojama su keliais algoritmais. Globaliojo optimizavimo srityje yra tyrimui atviras klausimas, koks yra geriausias algoritmas tam tikrai problemai. Visa tai sukuria poreikį išvystyti geresnį algoritmų elgesio supratimą įvairiems uždavinių tipams [Zab13].

Moksliniuose šaltiniuose [BFM00a; BFM00b; DŠT07; Žil05] yra aprašomi įvairūs globaliojo optimizavimo metodai. Monografijoje [TŽ89] pateikiama tokia globaliojo optimizavimo metodų klasifikacija.

1. Tiesioginiai metodai:

   (a) atsitiktinės paieškos metodai,

(b) apibendrinti nusileidimo metodai,

(c) grupavimo metodai.

2. Netiesioginiai metodai:

   (a) metodai, aproksimuojantys lygio aibes,

   (b) metodai, aproksimuojantys tikslo funkciją.

3. Metodai, užtikrinantys sprendinio tikslumą:

   (a) padengimo metodai.

1.3. Optimizavimo uždavinių klasifikacija

Optimizavimo uždaviniams gali būti formuluojamos įvairios tikslo funkcijų ir apribojimų išraiškos, todėl optimizavimo uždaviniai gali būti labai skirtingi. Visi optimizavimo uždaviniai, atsižvelgiant į tikslo funkcijas ir apribojimus yra priskiriami tam tikroms grupėms [Kal07].

1. Pagal tikslo funkcijos iškilumą:

   (a) neiškilojo programavimo uždaviniai dažniausiai turi ne vieną lokalų optimalų sprendinį ir vienas iš jų yra globalus;

   (b) iškilojo programavimo uždaviniai yra kai iškilos yra leistinųjų sprendinių sritis ir tikslo funkcija bei egzistuoja vienas optimalus sprendinys.

2. Pagal tiesiškumą, tiesinio programavimo uždavinyje visi apribojimai ir tikslo funkcija yra išreikšti tiesinėmis funkcijomis, o netiesinio programavimo uždaviniuose bent vienas iš apribojimų arba (ir) tikslo funkcija yra netiesinė. Netiesinių uždavinių gali būti labai skirtingų, nes apribojimai ir tikslo funkcijos gali būti įvairios, išskiriamos pagrindiniai jų tipai:

   (a) kvadratinio programavimo uždaviniai, kuriuose tikslo funkcija kvadratinės formos, o apribojimai aprašomi tiesėmis;

   (b) trupmeninio programavimo uždaviniai, kuriuose tikslo funkcija yra trupmeninė;

   (c) geometrinio programavimo uždaviniai, kuriuose tikslo funkcija ir apribojimai išreikšti polinomais.

3. Pagal kintamųjų pobūdį išskiriami:

   (a) tolydūs programavimo uždaviniai, kuriuose leistinąją sprendinių sritį sudaro apribojimus atitinkantys vektoriai ir galimi leistinieji sprendiniai;

   (b) diskrečiojo programavimo uždaviniai, kuriuose leistinųjų sprendinių sritis susideda tik iš tam tikrų diskrečiųjų elementų rinkinio;

   (c) sveikaskaitinio programavimo uždaviniai, kuriuose leistinasis ir optimalus sprendinys gali priklausyti tik sveikųjų skaičių aibei;

   (d) bulinio programavimo uždaviniai, kuriuose visi nežinomieji įgyja logines reikšmes: 0 arba 1.

4. Pagal kitus požymius taip pat išskiriami:

   (a) parametrinio programavimo uždaviniai, kuriuose nuo tam tikrų parametrų priklauso apribojimų koeficienai ir (arba) laisvieji nariai ir tikslo funkcija;

   (b) dinaminio programavimo uždaviniai, kurių sprendimas atliekamas pasitelkiant dinaminio programavimo metodus;

   (c) stochastiniai uždaviniai, kuriuose dalis parametrų (gali būti ir visi) yra neapibrėžti (gali būti ir dalinai neapibrėžti) arba atsitiktiniai dydžiai.

Ne visada optimizavimo uždavinį galima išspręsti analitiškai, juo labiau, ne visada pavyksta uždavinį išreikšti matematine išraiška. Gali būti sudėtinga netgi nustatyti ar nagrinėjamame uždavinyje rastas sprendinys yra minimumo taškas, ar įrodyti, kad potencialių sprendinių aibė yra baigtinė.

1.4. Optimizavimas naudojant atsitiktinę paiešką

Atsitiktinės paieškos algoritmai naudoja atsitiktinius skaičius apskaičiuojant tikslo funkcijos reikšmes [Zab09]. Nustatyta, kad atsitiktinės paieškos algoritmai gali efektyviai spręsti sudėtingus optimizavimo uždavinius, kurių sprendimas deterministiniais algoritmais reikalauja daug skaičiavimo resursų arba iš viso negali būti išsprendžiami per priimtiną laiką [Zab09].

Atsitiktinės paieškos algoritmai yra sąlyginai lengvai sudaromi ir tinka įvairiems optimizavimo uždaviniams spręsti – tereikia užtikrinti galimybę apskaičiuoti tikslo funkcijos reikšmes leistinosios srities taškuose [Lan13].

Atsitiktinės paieškos algoritmai, dažniausiai sudaromi „bandymų ir klaidų“ metodu, t. y. daug kartų atliekant bandymus ir modifikuojant įvairius algoritmus. Tokiu būdu algoritmai pritaikomi konkrečiam optimizavimo uždaviniui spręsti, taip pasiekiant geriausią uždavinio sprendimo rezultatą. Kita vertus, toks specifiniam uždaviniui parenkamas algoritmas nebūtinai bus tinkamas, efektyviam kito uždavinio sprendimui [Zab09].

Vienas iš paprasčiausių atsitiktinės paieškos algoritmų yra Monte-Karlo algoritmas, kurio galimų sprendinių koordinatės generuojamos kaip nepriklausomi atsitiktiniai dydžiai [DŠT07].

Monte-Karlo (angl. *Monte-Carlo*) algoritmas – kompiuterinis skaičiavimo algoritmas, kuriame uždaviniui spręsti naudojamas atsitiktinių skaičių generatorius [Gen06]. Šis algoritmas atsirado 1949 metais, kartu su pirmaisiais elektroniniais kompiuteriais, kur buvo pritaikyta tikimybių teorija sudėtingiems procesams atominiuose reaktoriuose modeliuoti. Monte-Karlo algoritmas – skaičiavimo metodas, pagrįstas statistiniu modeliavimu ir gautų rezultatų apdorojimu statistiniais metodais. Šis algoritmas leidžia brangiai kainuojančius bandymus pakeisti modeliavimu kompiuteriais ir labai sumažina tyrimų trukmę. Monte-Karlo tipo algoritmai dažniausiai naudojami fizikinių ir matematinių sistemų modeliavimui, kai neįmanoma gauti tikslių rezultatų naudojant deterministinį algoritmą [Pup09].

Bendra algoritmo schema: parenkami bandymų taškai $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$ leistinojoje srityje $\mathbb{D}$ ir šiuose taškuose apskaičiuojamos tikslo funkcijos reikšmės

$$f(\mathbf{x}_i), i = 1, N \tag{5}$$

ir parenkama mažiausia tikslo funkcijos reikšmė:

$$\min_{1 \leq i \leq N} f(\mathbf{x}_i), \tag{6}$$

ir atitinkamas minimumo taškas:

$$\mathbf{x}^* = \arg\min_{1 \leq i \leq N} f(\mathbf{x}_i). \tag{7}$$

Tokie algoritmai yra paprasti, tačiau neefektyvūs – tikimybė rasti globalųjį optimalų sprendinį artėja į vienetą, kai bandymų skaičius artėja į begalybę. Beje, funkcijos reikšmė artėja į minimumą, kai eksperimentų skaičius artėja į begalybę:

$$\min_{1\leq i\leq N} f(\mathbf{x}_i) \xrightarrow{N\to\infty} \min_{x\in\mathbb{D}} f(\mathbf{x}). \tag{8}$$

Kita vertus, šie algoritmai yra paprastai realizuojami, todėl neretai yra naudojami optimizavimo uždavinių savybių nustatymui: globaliųjų ir lokaliųjų optimalių sprendinių skaičiui nustatyti, leistinosios srities apibrėžimui ir pan. [Lan13]. Atsitiktinai sugeneruoti galimi sprendiniai gali būti naudojami kaip pradiniai taškai lokaliojo optimizavimo algoritmams.

Nors Monte-Karlo algoritmas yra labai paprastas, jis nėra efektyvus, kadangi neatsižvelgia į ankstesnių bandymų rezultatus. Todėl sudėtingesniems uždaviniams yra naudojami sudėtingesni algoritmai, kurie numato informacijos, gautos ankstesnių bandymų (sėkmingų ir nesėkmingų) panaudojimą generuojant naujus sprendinius. Atsižvelgiant į tai ar atliekant skaičiavimus yra atsižvelgiama į ankstesnių bandymų rezultatus atsitiktinės paieškos algoritmai yra skirstomi į [Lan13]:

- neadaptyvius – neatsižvelgiantys į ankstesnių bandymų rezultatus;
- adaptyvius – atsižvelgiantys į anksčiau atliktų bandymų rezultatus.

Adaptyvieji atsitiktinės paieškos metodai yra skirstomi į šias grupes.

- Modeliuojamojo atkaitinimo metodus [KV⁺83; Pup09].
- Evoliucinius skaičiavimus:
  - genetinius algoritmus [Mit98; Vos99; Wal99],
  - evoliucinius algoritmus [Ash06; BFM97].
- Spiečiaus sumanumo strategijas:
  - dalelių spiečiaus optimizavimą [EK⁺95; KE95; KE97],
  - skruzdžių kolonijos optimizavimą [CDM⁺91; Dor92],
  - bičių algoritmus [PGK⁺11].

1.5. Atsitiktinės paieškos optimizavimo algoritmų vertinimas

Nėra vieno universalaus algoritmo, kurio efektyvumas būtų vienodas, sprendžiant visus optimizavimo uždavinius, kaip tai teigia „nemokamų pietų nebuvimo“ teorema [WM97]. Teorema teigia, kad algoritmo gaunama nauda vienoje uždavinių klasėje būtinai atsveria kitų uždavinių sprendimų efektyvumą. Todėl neretai konkrečiam uždaviniui spręsti būna skirtas tam tikras algoritmas. Taip pat visi algoritmai turi stipriąsias ir silpnąsias puses, kurios daro įtaką algoritmo efektyvumui. Jei optimizavimo uždavinys kokiu nors metodu išsprendžiamas baigtiniu žingsnių skaičiumi, algoritmo efektyvumas gali būti vertinamas algoritmų sudėtingumo teorijos kriterijais [Žil05].

Kadangi tik retais atvejais būna žinomi teoriškai įrodyti teiginiai apie algoritmo efektyvumą, dėl to dažniausiai algoritmus tenka lyginti eksperimentiškai, sprendžiant specialiai parinktus testinius uždavinius [DŠT07]. Labai sudėtinga daryti vienareikšmiškas išvadas apie algoritmų efektyvumą tam tikriems uždaviniams spręsti, nes teisingų testinių uždavinių parinkimas ir eksperimentavimas su skirtingais algoritmais gali duoti ne būtinai visiškai korektiškas išvadas. Eksperimentų rezultatai priklauso nuo:

- algoritmo sudarymo;
- specifinių parametrų reikšmių;
- sustojimo sąlygų;
- kitų aplinkybių.

Tam tikriems algoritmams sudėtinga pasinaudoti vienu iš matematinės statistikos analizės priemone – konvergavimo greičiu. Dėl šios priežasties mokslininkai remiasi skaitiniais bandymais su testinių uždavinių rinkiniais, kurių pirmasis buvo aprašytas Dixon ir Szegö 1975 metais [DS78; HL15]. Taigi pasitelkiant tam tikrus algoritmų vertinimo kriterijus galima palyginti skirtingus algoritmus.

Norint įvertinti atsitiktinės paieškos algoritmo efektyvumą, reikia atlikti daug eksperimentų, kadangi atliekant eksperimentus gaunamos skirtingos reikšmės. Pavyzdžiui, vadovėlyje „Optimizavimo Metodai“ [DŠT07] išskiriami trys pagrindiniai algoritmų lyginimo kriterijai:

1. tam tikros klasės uždavinių optimalaus sprendinio radimo sėkmė;
2. skaičiavimo trukmė;
3. tikslo funkcijos reikšmių perskaičiavimų skaičius.

Sprendžiant sudėtingus uždavinius algoritmas gali „įstrigti“ ir nepateikti optimalaus sprendinio, o pateikti tik sprendinius su tam tikra paklaida [DŠT07].

Algoritmo efektyvumas yra susijęs su algoritmo vykdymui susijusių resursų kiekiu. Pavyzdžiui, algoritmo efektyvumas dažnai yra vertinamas atsižvelgiant į centrinio procesoriaus (angl. *Central Processing Unit, CPU*) sugaištą laiką, tačiau šis laikas dažnai negali būti laikomas tinkamu dėl įvairių priežasčių, tokių, kaip skirtingo programavimo stiliaus, skirtingų kompiuterių ir pan. Tačiau dažnai funkcijų perskaičiavimų skaičius yra naudojamas, kaip alternatyva centrinio procesoriaus laikui. Beje, funkcijų perskaičiavimų skaičius yra nepriklausomas nuo techninės įrangos ir programuotojo įgūdžių ir tai yra labiau informatyvu, nei centrinio procesoriaus laikas [Ran10]. Šiame darbe algoritmo efektyvumas vertinamas atsižvelgiant į gauto sprendinio tikslumą esant fiksuotam funkcijų perskaičiavimų skaičiui. Efektyvesniu algoritmu bus laikomas algoritmas, kuris randa geresnes reikšmes po tam tikro skaičiaus tikslo funkcijos perskaičiavimų.

Algoritmo efektyvumo vertinimas

Globaliojo optimizavimo algoritmai, kurių efektyvumas matuojamas, atsižvelgiant į funkcijos perskaičiavimų skaičių ir tai aprašoma literatūroje, pavyzdžiui: Hendrix ir Tóth [HB$^+$10] mini tokį algoritmo vertinimo būdą.

Dažnai keliamas klausimas, kokio tipo algoritmai geriau sprendžia tam tikros klasės uždavinius. Toks klausimas reikalauja nagrinėti algoritmo apskaičiuotas reikšmes po N tikslo funkcijų perskaičiavimų. Taip pat analizuojamos gautos ir tarpinės reikšmės, pavyzdžiui po $N/2$ funkcijos perskaičiavimų. Galiausiai analizuojamos kokios gautos geriausios ir blogiausios reikšmės. Tokios įžvalgos yra reikalingos galutinam sprendimui, kuris algoritmas yra tinkamiausias spręsti tam tikros klasės arba tam tikrą uždavinį [Hol75].

Literatūroje vyrauja tendencija analizuoti algoritmo visumą, pavyzdžiui, koks yra algoritmo visų apskaičiuotų reikšmių vidurkis, o ne tirti sisteminiu būdu įvairias algoritmo apskaičiuotas reikšmes. Toliau bus aptariami tyrimai, kurie pristato naujus algoritmus ir yra cituojami literatūroje, kur neatsižvelgiama į algoritmo apskaičiuotas tam tikras reikšmes. Chelouah ir Siarry aptaria genetinį algoritmą ir atlieka skaitinius testus, matuojančius tik funkcijų perskaičiavimų vidurkiais [CS00]. Jelasity ir kt. iliustruoja efektyvumą su algoritmo apskaičiuotomis vidurkio

reikšmėmis tam tikram tikslo funkcijos perskaičiavimų skaičiui [JOG01]. Recchioni ir Scoccia pristato euristiką apribotai optimizacijai ir įvertina gradientinių įvertinimų vidurkius ir eksperimentų vidurkius [RS00]. İlker Birbil ir Fang lygina funkcijų perskaičiavimo vidurkius prieš pasiektų funkcijų vidurkines reikšmes, naujam elektromagnetizmo mechanizmui [BF03]. Redondo ir kt. pristatė evoliucinį algoritmą skirtą daugiamatėms skalėms ir iliustruoja jų efektyvumą pasitelkiant vidutinį skaičiavimo laiką [ROŽ12]. 2002 metais Dolan ir Moré pristatė našumo profilius (angl. *Performance Profiles*), kurie skirti deterministiniams algoritmams kuomet lyginami deterministiniai globaliojo optimizavimo algoritmai, kaip neseniai t. y. 2014 metais atliktame tyrime Misener ir Floudas su taikomąja programa „MINLP Solver“ [DM02; MF14]. Samprata buvo išplėsta stochastinių algoritmų analizavimui, tai sukėlė daugiau dėmesio vidurkinėms reikšmėms, nors šiame tyrime taip pat buvo diskutuojama dėl sudėtingesnių įvertinimų [AKZ05]. Našumo profilių pavyzdys yra pateikiamas tyrime apie dalelių spiečių [VV07]. Nagrinėjami sudėtingesnis efektyvumo nustatymas, kai sudedami stochastika paremta algoritmų samprata su deterministine euristika. Pavyzdžiui, Müller ir Schoemaker straipsnyje, kur atsitiktinė atranka yra sujungiama su radialine euristikos funkcija [MS14]. Deterministinių euristikų palyginimą su stochastinėmis atliko Rios ir Sahinidis, sujungdami daug algoritmo efektyvumą, apibūdinančių įvertinimų [RS13]. Stochastiniai algoritmai parodo reikalingus pokyčius į kuriuos turėtų būti atsižvelgiama atliekant algoritmų lyginamąją analizę [HL15].

Svarbu įvertinti algoritmų kokybę individualiems testams. Atsitiktinės paieškos algoritmams efektyvumas gali būti vertinamas lyginamąja analize norint įvertinti kiek geriau (ar blogiau) veikia algoritmas [HL15].

Optimalaus sprendinio radimo vertinimui yra naudojama keletas vertinimo metodų.

Vidurkis – tai taškas, kuris visiems statistinės eilutės elementams yra vidutiniškai artimiausias. Galima apskaičiuoti tik vidurkį kiekybiniams duomenims. Vidurkis $\overline{x}$ yra apskaičiuojamas formule:

$$\overline{x} = \sum_{j=1}^{n} x_j \tag{9}$$

sudedant visas kiekybinio kintamojo reikšmes x ir suma padalijama iš skaičiaus n – elementų skaičius [ČM01]. Atsitiktinės paieškos optimizavimo algoritmuose vidurkis parodo kokia vidutinė reikšmė yra gaunama.

Nupjautasis vidurkis – atmetant tam tikras duomenų aibės didžiausias ir mažiausias reikšmes ir tik tada apskaičiuojamas vidurkis. Nupjautasis vidurkis taikomas kai: ekstremalios reikšmės yra nepatikimos, kai duomenų aibės reikšmių pasiskirstymas asimetriškas. Nupjautasis vidurkis, kurio reikšmė nulis procentų yra lygus aritmetiniam vidurkiui [ČM01]. Atsitiktinės paieškos optimizavimo algoritmuose atsitiktinis vidurkis naudojamas, kai norima palyginti reikšmes, atmetant vidurkį iškreipiančius sprendinius, pavyzdžiui algoritmas gali apskaičiuoti tik vieną kartą labai blogą sprendinį, kuris labai iškreipia vidurkį, kadangi gautas blogas sprendinys tik vieną kartą jis yra atmetamas ir laikomas nereikšmingu.

Dispersija – sklaidos charakteristika, kuri parodo duomenų sklaidą apie vidurkį. Ši charakteristika atsižvelgia į visus duomenis ir pateikia atstumą nuo vidurkio kvadratiniais vienetais. Dispersija apskaičiuojama formule:

$$s^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j - \overline{x})^2, \tag{10}$$

kur s^2 yra dispersija, n – elementų skaičius duomenų aibėje [ČM01]. Atsitiktinės paieškos optimizavimo algoritmuose dispersija parodo gautų sprendinių sklaidą aplink vidurkį.

Standartinis nuokrypis – vienas iš dažniausiai naudojamų sklaidos matų, kuris kaip ir dispersija parodo vidutinę duomenų sklaidą apie vidurkį, tačiau yra matuojamas tokiais pačiais vienetais, kaip ir aibėje esantys duomenys. Taigi standartinį nuokrypį su duomenimis lengviau interpretuoti ir lyginti. Standartinis nuokrypis s apskaičiuojamas:

$$s = \sqrt{s^2}, \tag{11}$$

ištraukiant šaknį iš dispersijos s^2 [ČM01]. Atsitiktinės paieškos optimizavimo algoritmuose standartinis nuokrypis naudojamas palyginti gautus sprendinius ir šių sprendinių sklaidą aplink vidurkį.

Pasikliautinasis intervalas – parametrų įverčiai yra atsitiktiniai dydžiai ir išsibarsčiusios realizacijos apie tikrąją parametro reikšmę. Tarkime, kad yra stebimas atsitiktinis dydis, kuris turi skirstinį P_θ su nežinomu $\theta \in \Theta$, pasirenkamas skaičius $0 < Q < 1$. Tarkime, kad turime dvi tokias statistikas $\widehat{\theta_1}, \widehat{\theta_2}$, kad

$$P(\widehat{\theta_1} \leq \theta \leq \widehat{\theta_2}) = Q. \tag{12}$$

Intervalas $[\widehat{\theta_1}, \widehat{\theta_2}]$ yra vadinamas θ parametro pasikliautinuoju intervalu. Pasikliovimo lygmeniu vadinamas skaičius Q. Dažniausiai naudojami pasikliovimo lygmenys $Q = 0{,}9;\ 0{,}95;\ 0{,}99$. Statistikos (atsitiktiniai dydžiai) yra pasikliautinojo intervalo rėžiai $\widehat{\theta_1}, \widehat{\theta_2}$. Pasikliautinasis intervalas tai atsitiktinis dydis su skirstiniu, kurio tikimybė jog θ priklausys šiam intervalui yra didelė (lygi Q) [ČM01]. Atsitiktinės paieškos optimizavimo algoritmuose pasikliautinasis intervalas naudojamas įvertinti kokia yra tikimybė gauti tam tikrą sprendinį.

Optimalaus sprendinio radimo sėkmė – artimai susijusi ir su sprendinio tikslumu. Tačiau tikslumo sąlygos gali būti įvairios. Sprendžiant tam tikros klasės uždavinius tikslumas gali būti tikslus, o sprendžiant kitos klasės uždavinius – tikslumas gali būti nepriimtinas. Pavyzdžiui, turime tenkintis apytiksliais sprendiniais, sprendžiant globaliojo optimizavimo uždavinius su komplikuota tikslo funkcija ir su daugeliu kintamųjų. Tačiau kitiems uždaviniams keliami aukšti tikslumo reikalavimai, pavyzdžiui, lokaliems algoritmams, kuriais sprendžiami vienaekstremiai uždaviniai su diferencijuojamomis tikslo funkcijomis. Tikslumas matuojamas surasto geriausio sprendinio $\mathbf{x}_k$ atstumu nuo optimalaus sprendinio $\mathbf{x}^*$

$$\Delta\mathbf{x}_k = || \mathbf{x}_k - \mathbf{x}^* || \tag{13}$$

arba tikslo funkcijos reikšmių skirtumu

$$\Delta f = f(\mathbf{x}_k) - f(\mathbf{x}^*). \tag{14}$$

Remiantis tuo, koks yra tikslo funkcijos tipas yra parenkamas tikslumo matas. Tikslinga naudoti matą $\Delta\mathbf{x}$, jeigu tikslo funkcija optimalaus sprendinio aplinkoje nedaug kinta. Jis yra dažnas sprendžiant uždavinius, kai pagal $\Delta\mathbf{x}$ surastas lokalusis optimalus sprendinys gali būti labai toli nuo globaliojo optimalaus sprendinio, tačiau atitinkamos tikslo funkcijos reikšmės gali nedaug skirtis. Beje, tikslumo matas dažniau naudojamas aproksimacijos uždaviniams, kai tikslo funkcija nevisapusiškai atskleidžia nuostolius dėl aproksimacijos paklaidų. Pavyzdžiui, jeigu uždavinys sprendžiamas naudojantis nusileidimo metodais, kurie naudoja ir tikslo funkcijos išvestinių skaičiavimus. Jeigu išvestinių reikšmės randamos skaičiavimo metodais, tai galutinis algoritmo vertinimo kriterijus yra skaičiuojamų funkcijos reikšmių skaičius, bet jeigu išvestinės nerandamos skaičiavi-

mo metodais, tada reikia atsižvelgti į kitus algoritmo efektyvumo kriterijus [DŠT07].

Tikslo funkcijos reikšmių perskaičiavimų skaičius – šis metodas įprastai naudojamas kai uždavinys sprendžiamas norimu tikslumu. Šį matą ypač aktualu naudoti, kai reikia daug laiko ir yra brangu apskaičiuoti duotajame taške tikslo funkcijos reikšmę [DŠT07]. Pavyzdžiui, vertinant algoritmų našumą vertinamas ir funkcijų perskaičiavimų skaičius [HAR+10; HJD+].

Uždavinio sprendimo trukmė – šis algoritmų lyginimo kriterijus naudojamas jeigu optimizavimo algoritmai yra skirti optimizuoti greitai skaičiuojamas funkcijas ar jais sprendžiami uždaviniai, su vienos išraiškos funkcijomis – kvadratinio arba tiesinio programavimo uždaviniai [DŠT07].

Pavyzdžiui, vertinant optimizavimo algoritmų tyrimo rezultatus, kurie gauti naudojant skirtingus kompiuterius su skirtingomis charakteristikomis privaloma į tai atsižvelgti, taip pat taikant sudėtingus optimizavimo algoritmus pats algoritmas lemia skaičiavimo laiką, taigi skaičiavimo laikas gerai atspindi tikrąją algoritmo vertę. Nereikia pamiršti, kad skaičiavimo laikas priklauso ir nuo programavimo stiliaus ir programuotojo įgūdžių. Algoritmuose būna daug iš anksto nustatomų parametrų, kuriuos parenkant gaunamas skirtingas efektyvumas. Taigi ir parametrų parinkimas tampa lyg optimizavimo uždavinys – kurio tikslas parinkti kuo geresnius parametrus. Taip pat iš anksto parenkami ne tik parametrai, bet ir pasirenkami testiniai uždaviniai, kuriais yra mėginama įvertinti algoritmo efektyvumą, taigi nėra lengva objektyviai įvertinti kuris algoritmas yra efektyvesnis, kai tarpusavyje lyginama keletas algoritmų [DŠT07]. Tačiau, naudojantis vienu testiniu uždavinio sprendimu sudėtinga daryti tikslias išvadas, dėl tos priežasties yra sudaromos testinių uždavinių klasės [KS74].

Tikimybės rasti norimo tikslumo sprendinį vertinimas

Praktikoje ne visada būtina rasti tikslų sprendinį. Dažnai pakanka rasti sprendinio priimtino tikslumo aproksimaciją. Daug ir įvairių optimizavimo algoritmų yra aprašyta, kurie remiasi tikslo funkcijų perskaičiavimo skaičiu, kad būtų nustatomas sustojimo kriterijus ir remiantis vidutine tikslo funkcijos reikšme nustatomas algoritmo efektyvumas [HAR+10; HJD+; HL15]. Straipsnyje [HL15] nagrinėjamos tam tikro tikslumo aproksimacijos ir tikimybės ją rasti priklausomybė.

Tarkime, kad algoritmui A skiriama N tikslo funkcijų perskaičiavimų. Po k eksperimentų

gaunamos tokios aproksimacijos:

$$\mathbf{y}_1^*,\ \mathbf{y}_2^*,\ \mathbf{y}_3^*, ..., \mathbf{y}_k^*. \tag{15}$$

Kiekvieną gautą aproksimaciją galime palyginti su optmaliu sprendiniu $\mathbf{y}^*$ ir įvertinti paklaidą nuo optimalaus sprendinio:

$$\|\mathbf{y}^* - \mathbf{y}_i^*\| \tag{16}$$

Taigi kiekvienam gautam sprendiniui galime apskaičiuoti paklaidas ir apskaičiuoti kiek kartų su kokia paklaida yra gautas sprendinys. Tam algoritmo A efektyvumui įvertinti naudojant N tikslo funkcijos perskaičiavimų ir k eksperimentų yra apibrėžiama tikimybių pasiskirstymo funkcija (angl. *Cumulative Distribution Function*, CDF):

$$CDFR_N^{[A]}(y) = t\left(\mathbf{Y}_N \leq y\right), \tag{17}$$

taigi apskaičiuojama t – tikimybė gauti tam tikro tikslumo sprendinį po tam tikro N funkcijų perskaičiavimų. Visa tai galime išreikšti sumos formule

$$CDFR_N^{[A]}(y) = \frac{1}{k}\sum_{i=1}^{k} I_{y_i}, \tag{18}$$

čia

$$I_{y_i} = \begin{cases} 1, \text{ jei } y_i > y; \\ 0, \text{ jei } y_i \leq y. \end{cases} \tag{19}$$

Taigi, matome, kad tikimybių pasiskirstymo funkcija skaičiuoja reikšmes, kurios yra mažesnės už y_i-ajį elemetą, didėjant i-ąjam elementui didėja ir tikimybių pasiskirstymo funkcijos reikšmė, tagi tikimybių pasiskirstymo funkcija yra kaupiančioji. Kadangi matome, kad formulėje (žr. (18) formulę) vienetas dalinamas iš k ir sumuojamos reikšmės yra mažiau už vienetą ir daugiau nei nulis, visa tai užtikrina, kad visada apskaičiuota tikimybių pasiskirstymo funkcijos reikšmė yra tarp nulio ir vieneto – $0 \leq CDFR \leq 1$.

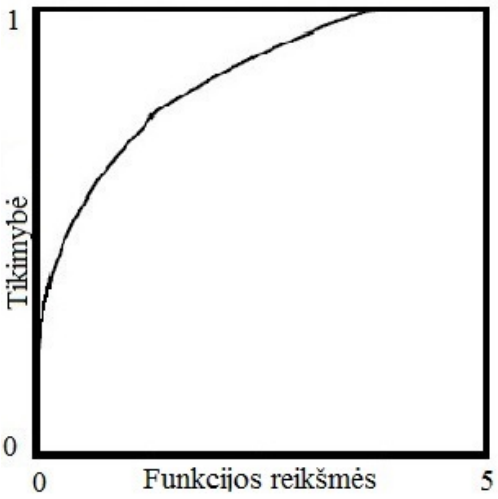
Pavyzdžiui, naudojant algoritmą – Tikroji atsitiktinė paieška (angl. *Pure Random Search, PRS*) išspręskime du skirtingus uždavinius: Six-hump camel-back funkcija:

$$f(x) = 4x_1^2 - 2{,}1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4 \tag{20}$$

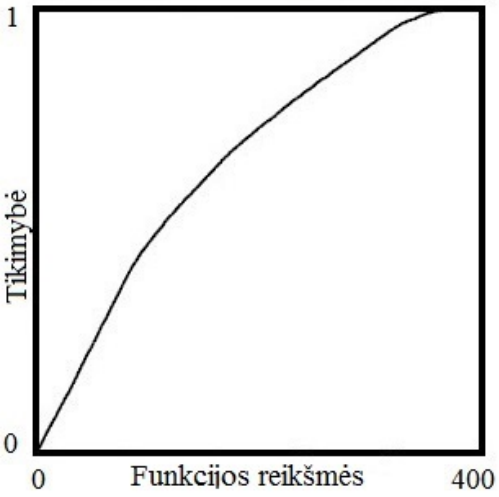
kurio $X = [-3, 2]^2$ ir Bi-spherical funkciją:

$$f(x) = min\{(x_1 - 1)^2,(x_1 + 1)^2 + 0{,}01\} + \sum_{2}^{n} x_i^2 \tag{21}$$

su $X = [-2, 2]^2$. Tikimybių pasiskirstymo funkcijos rezultatai, gauti po 10000 funkcijos perskaičiavimų, pateikiami 1 ir 2 paveiksluose.



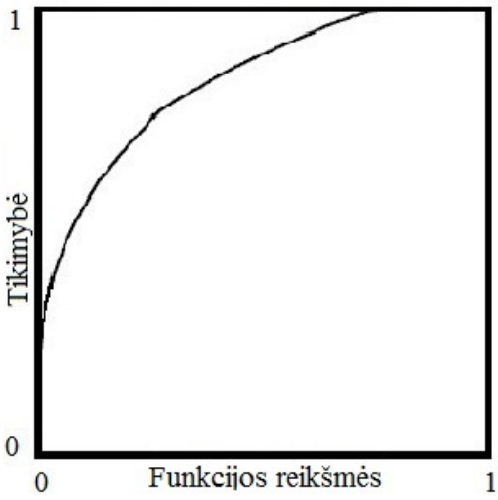
1 pav. Six-hump camel-back funkcija [HL15]



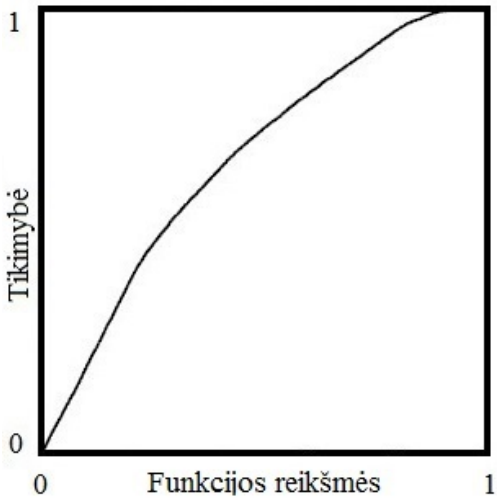
2 pav. Bi-spherical funkcija [HL15]

Sprendžiant Six-hump camel-back funkciją gauti rezultatai pavaizduoti tikimybių pasiskirstymo funkcija (žr. 1 ir 2 pav.). Six-hump camel-back funkcijos reikšmes gauti su paklaida 50 ir mažesne tikimybė yra $12{,}5\%$, o sprendžiant Bi-spherical funkciją panašus rezultatas pasiekiamas tik su 40% tikimybe gauti panašias reikšmes.

Iš gautų rezultatų (žr. 1 ir 2 pav.) matome, kad sprendžiant Six-hump camel-back uždavinį su 50 paklaida tikimybė gauti sprendinius (kurių paklaida bus nuo 0 iki 50) yra $12{,}5\%$, o sprendžiant Bi-spherical uždavinį panašus rezultatas pasiekiamas tik su 40% tikimybe gauti panašias reikšmes. Taip pat 1 ir 2 pav. matome, kad apskaičiuotos tikslo funkcijos reikšmės gali būti išreiškiamos reikšmėmis nuo nulio iki vieneto t. y. $[0;\ 1]$. Tokiu būdu yra daug lengviau palyginti funkcijos apskaičiuotų reikšmių charakteristikas su skirtingais testais. Tikimybių pasiskirstymo funkcijos grafikus galime pavaizduoti vienetiniame kvadrate (žr. 3 ir 4) pav.



3 pav. Six-hump camel-back funkcija normuota [HL15]



4 pav. Bi-spherical funkcija normuota [HL15]

Tikimybių ir funkcijų reikšmės pasiskirsto vienetiniame kvadrate (žr. 3 ir 3 pav.), nes tikimybių pasiskirstymo funkcijos grafiko:

- ašis pažymėta – funkcijos reikšmės yra funkcijos apskaičiuotos reikšmės, kurios normuotos nuo nulio iki vieneto;
- ašis pažymėta – tikimybė, kuri išreikša nuo nulio iki vieneto. Tikimybė parodo kokia tikimybe galima gauti tam tikrą (ir visas mažesnes) reikšmę.

Taigi normavus tikimybių pasiskirstymo funkcijos grafikus juos lengviau palyginami.

2. ATSITIKTINĖS PAIEŠKOS ALGORITMŲ VERTINIMAS

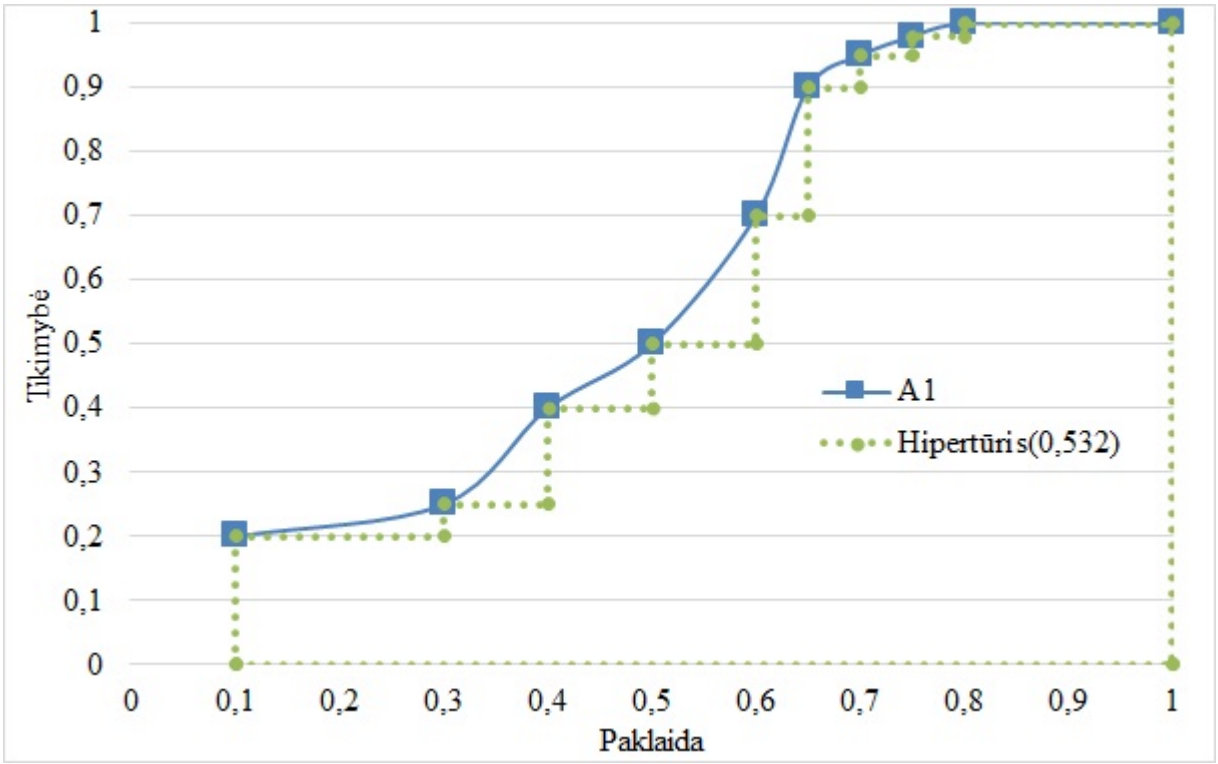
Šiame skyriuje aprašomas siūlomas algoritmų palyginimo metodas vertinantis dviejų algoritmų tarpusavio dominuojamumą. Šis metodas skirtas analizuoti algoritmais apskaičiuotus sprendinius, kurie gaunami sprendžiant vienakriterius globaliojo optimizavimo uždavinius. Šio metodo tikslas palyginti skirtingus algoritmus arba tą patį algoritmą, naudojant skirtingus jo parametrus. Pirmiausia buvo sukurtas metodas, kuris lygino algoritmus atsižvelgiant į tikimybių pasiskirstymo funkcijos grafiko hipertūrius, o vėliau šis metodas buvo tobulinamas ir taip sukurtas algoritmų tarpusavio dominuojamumo vertinimo metodas. Trumpas šio metodo aprašas:

- Turime 2 algoritmus $A1$ ir $A2$. Norime įvertinti, kaip $A1$ algoritmas yra dominuojamas $A2$ algoritmo.
- Atliekami eksperimentai. Algoritmui $A1$ skiriama tam tikras skaičius tikslo funkcijų perskaičiavimų. Po tam tikro skaičiaus eksperimentų gaunamos reikšmės, kurios normuojamos intervale nuo 0 iki 1. Apskaičiuojamos tikimybių pasiskirstymo funkcijos grafiko reikšmės (žr. 18 puslapį).
- Pritaikomas hipertūrio skaičiavimo principas. Apskaičiuojamas $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūris. Hipertūrio skaičiavimo principas bus aprašytas 2.1 poskyryje.
- Pritaikomas dominuojamumo principas, norint įvertinti, kaip algoritmas $A1$ yra dominuojamas $A2$ algoritmo. Pašaliname taškus iš $A1$ tikimybių pasiskirstymo funkcijos grafiko, kurie yra dominuojami $A2$ algoritmo tikimybių pasiskirstymo funkcijos grafiko taškų, dominuojamumas bus aprašytas 2.2 poskyryje.
- Apskaičiuojamas hipertūris $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko, be taškų dominuojamų $A2$ algoritmo tikimybių pasiskirstymo funkcijos grafiko ir tai bus aprašyta 2.3 poskyryje.
- Pritaikomas Pareto frontų persidengimo principas ir apskaičiuojamas dominuojamumo koeficientas, kurio formulė bus aprašyta 2.4 poskyryje.

Toliau išsamiai aprašomas siūlomas atsitiktinės paieškos algoritmų tarpusavio dominuojamumo vertinimo metodas.

2.1. Algoritmų palyginimas hipertūriu

Toliau aprašomas siūlomas hipertūrio principo taikymas tikimybių pasiskirstymo funkcijos grafikų hipertūrio skaičiavimui. Siūlomas metodas yra grįstas tikimybių pasiskirstymo funkcija (žr. 18 puslapį), kuriai pritaikomas hipertūrio skaičiavimo principas. Hipertūris (angl. *Hypervolume*) yra naudojamas daugiakriteriame optimizavime, kuris apibrėžia plotą, ribojamą tikrojo Pareto fronto aproksimuojančių taškų ir nurodyto atskaitos taško (angl. *Reference Point*). Šis matas atspindi Pareto fronto aproksimacijos tikslumą [Lan13].



5 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris

Panagrinėkime 5 pav. pateiktą tikimybių pasiskirstymo funkcijos grafiką, kuris sudarytas iš N taškų:

$$(t_1;\, p_1), (t_2;\, p_2), ..., (t_N;\, p_N),$$

kur t – tikimybė, p – paklaida.

Pasinaudojant šiais taškais galime apskaičiuoti pasiskirstymo funkcijos hipertūrį. Per šiuos taškus nubrėžiame laiptuotą figūrą taip kaip parodyta 5 pav. (pažymėta taškine linija). Kadangi, tikimybių pasiskirstymo funkcija visada tenkina šias savybes:

- tai yra didėjanti (kaupiančioji) funkcija;
- $0 \leq t \leq 1$ ir $0 \leq p \leq 1$;

tai, t ir p reikšmės gali būti tik mažesnės arba lygios vienetui ir nemažesnės už nulį – $0 \leq p_N, p_N \leq 1$, o N yra taškų skaičius. Norint išreikšti algoritmo tikimybių pasiskirstymo funkciją skaitine reikšme, skaičiuojame hipertūrį, kuris gaunamas apskaičiuojant sudarytos laiptuotos figūros plotą, kaip tai daroma skaičiuojant Pareto fronto hipertūrį [Lan13].
Laiptuota figūra gaunama:

- apačią apribojant tiese $y = 0$;
- kairę pusę apribojant tiese $y = p_1$;
- dešinę pusę apribojant tiese $x = 1$;
- viršų apribojant atkarpomis tarp gretimų taškų: $(p_i; t_i)$ ir $(p_{i+1}; t_i)$.

Matematiškai šios figūros hipertūrį galime išreikšti formule:

$$S = \sum_{i=2}^{N} \left((p_i - p_{i-1}) * t_{i-1} \right) + (1 - p_N) * t_N \tag{22}$$

Kadangi, hipertūris skaičiuojamas apribojant tiese $x = 1$, tai pavyzdžiui, jeigu grafiko x-ašies galutinis taškas nėra lygus vienetui ($p_N \neq 1$), tada pabaigoje pridedamas dar vienas taškas – $(1;\ p_{N-1})$, kurio tikimybė yra kaip ir paskutinio taško, o paklaidos reikšmė lygi vienetui. Tokiu būdu užtikrinama, kad hipertūris visada būtų skaičiuojamas iki $x = 1$. Formulėje (22) už tai yra atsakinga ši dalis: $(1 - p_N) * t_N$, kuri apskaičiuoja hipertūrį, kurį gautume sujungę taškus:

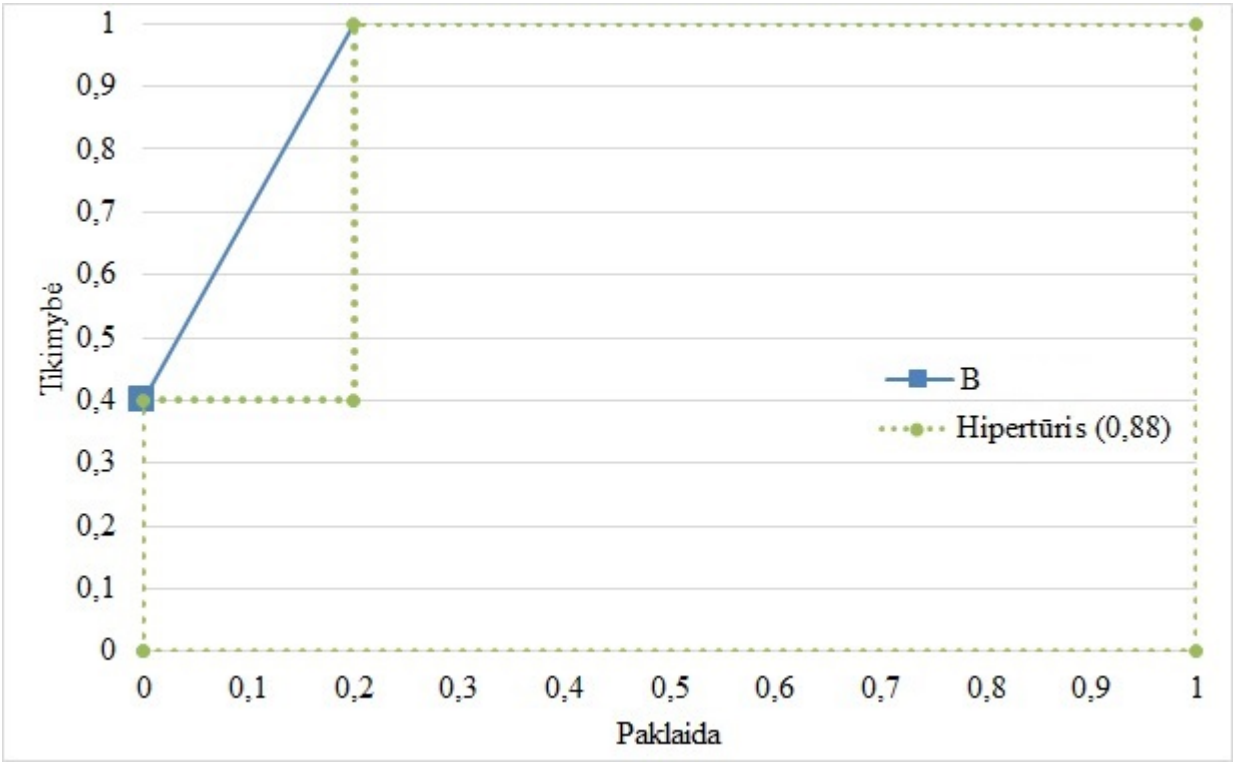
$$(p_N;\ t_N), (p_N;\ 0), (1, t_N)\ ir\ (1, 0).$$

Hipertūris visada skaičiuojamas apribojant tiesėmis $x = p_1$, pavyzdžiui jeigu pirmojo taško reikšmė yra $(0{,}05;\ 0{,}1)$, t. y. paklaida lygi $p_1 = 0{,}05$ ir tikimybė $t_1 = 0{,}1$, tai hipertūris skaičiuojamas apribojant tiese $x = 0{,}05$.

Visada gautas hipertūris yra daugiau arba lygu už 0 ir mažiau arba lygus 1 ($0 \leq S \leq 1$), nes y-ašies (tikimybių) visos reikšmės yra tarp nulio ir vieneto t. y. $(0 \leq t \leq 1)$ ir x-ašies (paklaidų) reikšmės taip pat yra tarp nulio ir vieneto t. y. $(0 \leq p \leq 1)$. Taigi, visos galimos hipertūrio reikšmės pasiskirsto vienetiniame kvadrate. Kai $S = 1$, tai reiškia, kad algoritmas apskaičiavo tik vieną – patį geriausią rezultatą, o kai $S = 0$ algoritmas apskaičiavo tik vieną – pačią blogiausią reikšmę arba nėra gauta rezultatų aktualiame intervale. Pavyzdžiui, jeigu turime algoritmus A_1 ir A_2, o jų plotai yra $S_{A_1} = 0{,}79$ ir $S_{A_2} = 0{,}75$. Taigi, atsižvelgiant į hipertūrį, algoritmas – S_{A_1} yra geresnis, nes $S_{A_1} > S_{A_2}$ t. t. $0{,}79 > 0{,}75$. Šiame pavyzdyje matome, kad algoritmo – A_1 hipertūris lyginant su algoritmu A_2 yra didesnis $0{,}04$. Taigi, kuo didesnis hipertūris – tuo efektyvesnis algoritmas.

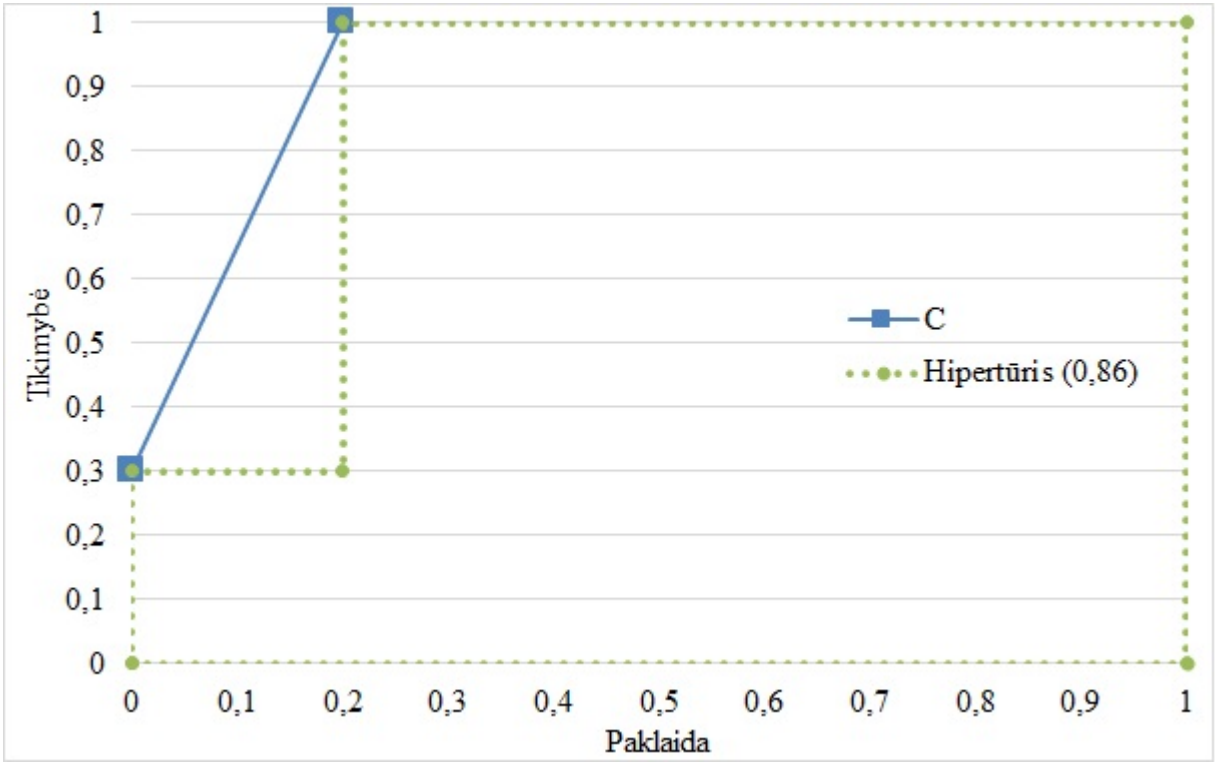
Hipertūrio skaičiavimo principas, atsižvelgia į algoritmo savybes rasti reikšmes su tam tikra paklaida ir tam tikra tikimybe. Tarkime, kad turime skirtingus algoritmus: A, B, C, D, E, F. Su kiekvienu algoritmu atliekami eksperimentai naudojant 100 tikslo funkcijos perskaičiavimų. Panagrinėkime gautus eksperimentų rezultatus ir šių algoritmų savybių įtaką hipertūrio reikšmei, kaip tikimybių pasiskirstymo funkcijos reikšmių pasiskirstymas daro įtaką hipertūriui.

Turime A algoritmą, kurio visos gautos reikšmės yra optimalios – algoritmo apskaičiuoti sprendiniai po šimto ($N = 100$) perskaičiavimų gauti su nuline paklaida. Gauname A algoritmo tikimybių pasiskirstymo funkcijos grafiką su vieninteliu tašku – $(0;\ 1)$. Pritaikius hipertūrio skaičiavimo formulę, gauname, kad A algoritmo hipertūris yra lygus vienam – $S_A = 1$. Taigi, gauta maksimali hipertūrio reikšmė, kai algoritmas apskaičiuoja tik pačias geriausias reikšmes, o jeigu algoritmas nerastų reikšmių nagrinėjamu tikslumu (arba reikšmės būtų su maksimalia paklaida), gautume hipertūrį lygų nuliui.



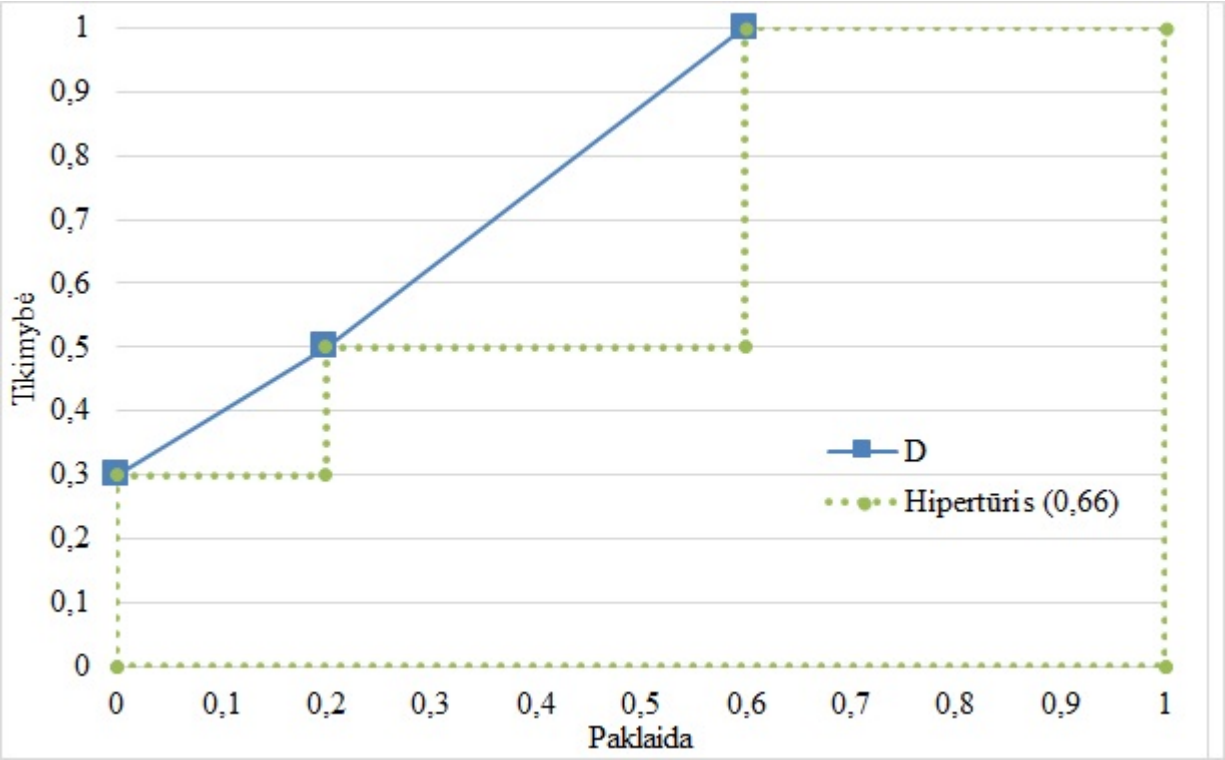
6 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris B algoritmo

Turime B algoritmą, kurio gautos reikšmės yra dvejopos ir jau žinome, kad B algoritmas yra blogesnis, nei A algoritmas, nes visos gautos reikšmės A algoritmo yra tik optimalios. Optimalių sprendinių B algoritmas apskaičiavo – 40%, o visos kitos likusios – 60% reikšmių yra gautos su $0{,}2$ paklaida. Sudaromas B algoritmo tikimybių pasiskirstymo grafikas (žr. 6 pav.), kuris turi du taškus: $(0;\ 0{,}4)$ ir $(0{,}2;\ 1)$. Akivaizdu, kad B algoritmo (žr. 6 pav.) grafiko pasiskirstymo funkcijos hipertūris yra mažesnis, nei A algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūris. Pritaikius hipertūrio formulę, gauname B algoritmo tikimybių pasiskirstymo funkcijos hipertūrį lygų $0{,}88$ ($S_B = 0{,}88$). Kadangi B algoritmo tik 40% reikšmių yra optimalių, o A algoritmo 100% reikšmių yra optimalių, tai B algoritmas yra mažiau efektyvus, tai ir hipertūrio reikšmė yra mažesnė $S_B < S_A$. Taigi, hipertūrio matas reaguoja į tai kiek optimalių sprendinių yra gaunama – kuo mažiau optimalių sprendinių tuo mažesnis ir hipertūris.



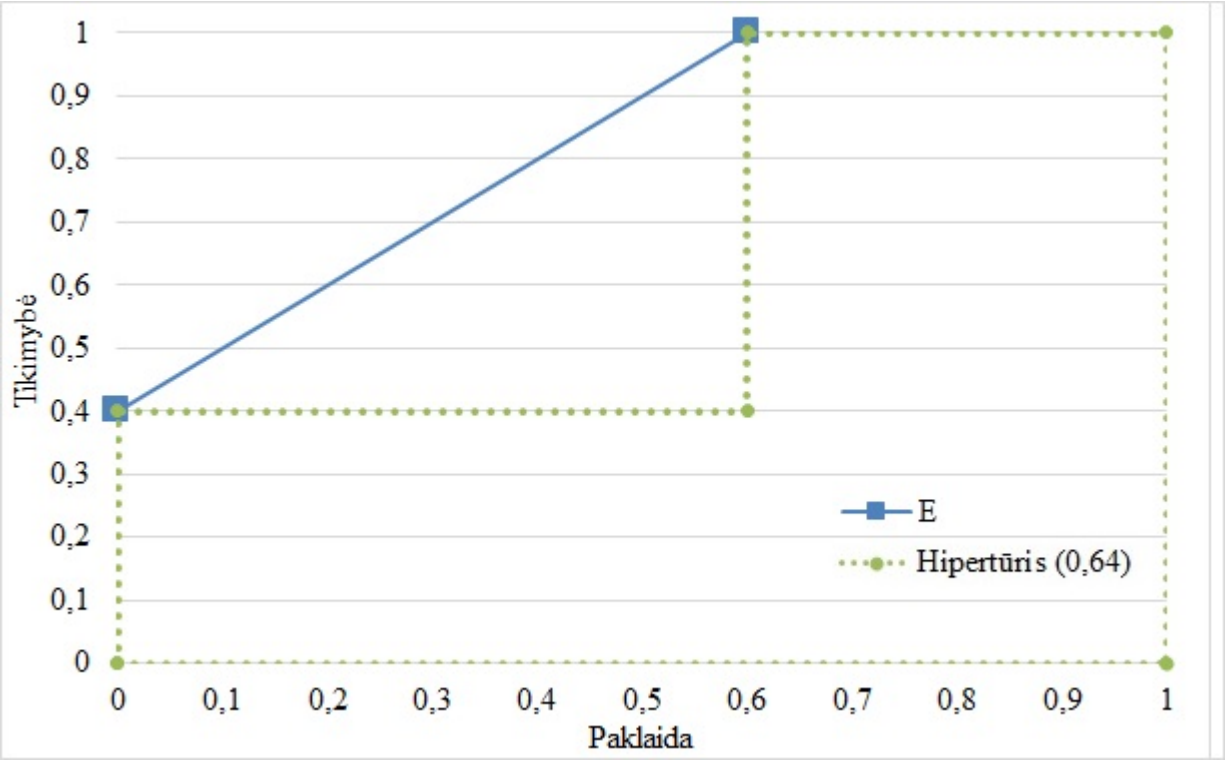
7 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris C algoritmo

Turime C algoritmą, kuris panašus į B algoritmą, tačiau skiriasi tik tai, kad C algoritmas apskaičiavo mažiau optimalių sprendinių – 30%, o likusios reikšmės gautos su $0{,}2$ paklaida, taip, kaip ir B algoritmas, bet jų yra 10% daugiau t. y. 70% gautų reikšmių yra su $0{,}2$ paklaida. Kadangi, C algoritmas apskaičiavo 10% mažiau optimalių sprendinių, nei B algoritmas, tai B algoritmas yra geresnis. Ir tai akivaizdžiai matosi grafikuose, kad algoritmo B tikimybių pasiskirstymo funkcijos hipertūris grafike (žr. 6 pav.) yra didesnis, nei C algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūris, kuris matosi 7 pav. Pritaikius (22) formulę, gauname, kad $S_C < S_B$. Taigi, parodo hipertūrio jautrumą optimalių sprendinių skaičiui.



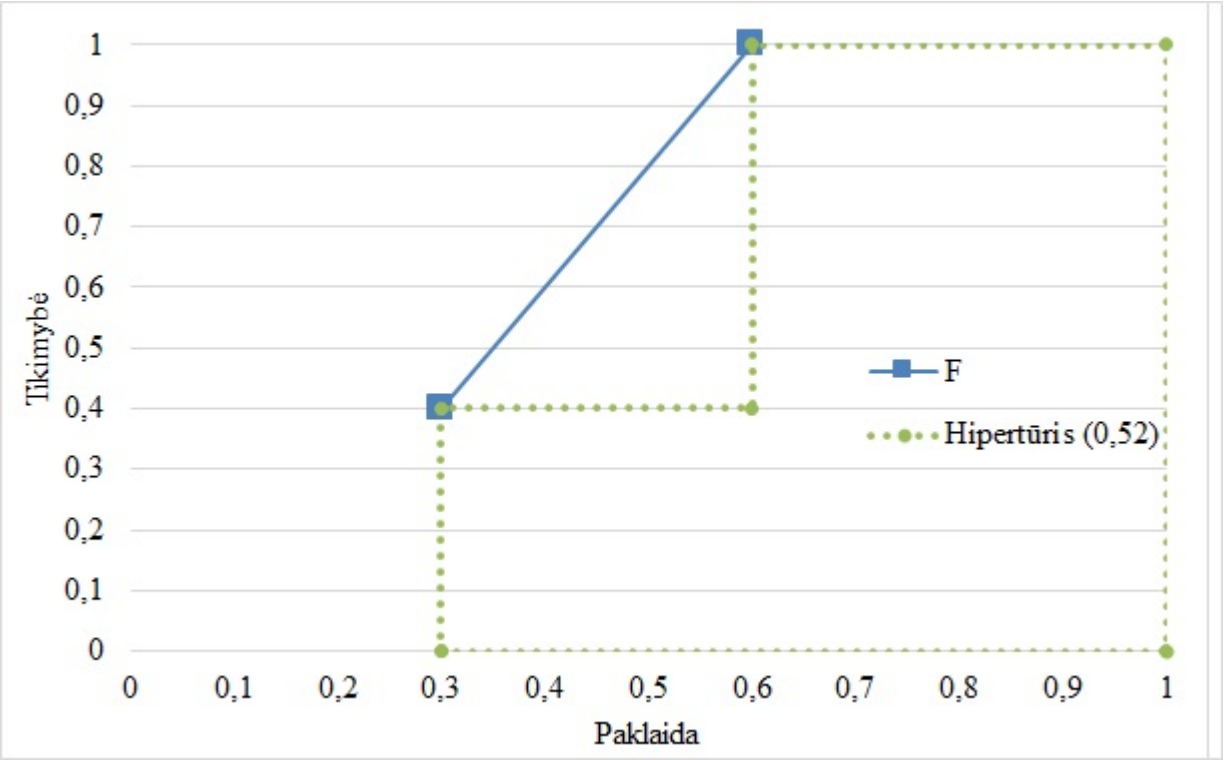
8 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris D algoritmo

Turime D algoritmą, kuris panašus į C algoritmą, kuris taip pat apskaičiavo 30% optimalių sprendinių ir gauta reikšmių su $0{,}2$ paklaida, kurių C algoritmas gavo 70%, o D algoritmas tik 20%. Tačiau D algoritmas likusias 50% apskaičiavo su dar didesne paklaida t. y. $0{,}6$ (žr. 8 pav.). Apskaičiavus D algoritmo tikimybių pasiskirstymo funkcijos hipertūrį gauname, kad D algoritmo hipertūris yra $0{,}66$ t. y. $S_C = 0{,}66$. Taigi, D algoritmo hipertūris yra mažesnis, nei C algoritmo t. y. $S_D < S_C$. Kadangi D algoritmo pusė sprendinių yra su didesne paklaida, nei C algoritmo, tai atitinkamai D algoritmo hipertūris yra $0{,}2$ mažesnis už C algoritmo hipertūrį. Taigi, kuo didesne paklaida algoritmo apskaičiuotos reikšmės, tuo mažesnis hipertūris yra gaunamas.



9 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris E algoritmo

Turime E algoritmą, kuris panašus į D algoritmą, bet E algoritmas neapskaičiuoja reikšmių su $0{,}2$ paklaida, bet apskaičiuoja reikšmes su $0{,}6$ paklaida, tačiau E algoritmas apskaičiuoja 10% daugiau optimalių sprendinių, nei D algoritmas, tai matome 9 pav. Optimalių sprendinių E algoritmas apskaičiavo 40%, o likusios reikšmės t. y. 60% gauta su $0{,}6$ paklaida. Apskaičiavus hipertūrį gauname, kad E algoritmo hipertūris lygus $0{,}64$ t. y. $S_E = 0{,}64$, taigi E algoritmo hipertūrio reikšmė mažesnė tik $0{,}02$ už D algoritmo tikimybių pasiskirstymo hipertūrį t. y. $S_E < S_D$. Taigi, E algoritmas turi 10% daugiau optimalių sprendinių, tačiau D algoritmas apskaičiavo 20% reikšmių su daug mažesne paklaida t. y. $0{,}2$, kadangi E algoritmo 60% reikšmių yra su didele $(0{,}6)$ paklaida, taigi algoritmas D yra geresnis, nei E algoritmas ir tai parodo hipertūrio matas. Matome, kad hipertūrio matas atsižvelgia ne tik į tai, kuris algoritmas apskaičiavo daugiau optimalių sprendinių, bet ir į tai su kokiomis paklaidomis yra gautos reikšmės ir koks jų santykis tarp visų apskaičiuotų reikšmių. Taigi, parodomas kaip hipertūris reaguoja į lokalių minimumų pasiskirstymą.



10 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris F algoritmo

Turime F algoritmą, kuris yra panašus į E algoritmą, nes E algoritmo 40% surastų reikšmių yra optimalūs sprendiniai, o F algoritmo 40% surastų reikšmių yra su $0{,}3$ paklaida, tai matome 10 pav. Taigi, F algoritmas nė karto nerado optimalaus sprendinio. Apskaičiavus hipertūrį gauname, kad jo reikšmė lygi $0{,}52$. Matome, kad F algoritmo hipertūris yra mažesnis, nei E algoritmo t. y. $S_F < S_E$ ir atitinkamai $0{,}52 < 0{,}64$. Taigi, kadangi F algoritmo gautos geriausios reikšmės yra su $0{,}3$ paklaida, tai atitinkamai gaunamas mažesnis hipertūris. Lyginant su E algoritmu, gaunamas $0{,}12$ mažesnis hipertūris. Hipertūris reaguoja į lokalių minimumų pasiskirstymą ir į optimalių sprendinių nebuvimą.

Taigi, pasirinktas hipertūrio skaičiavimo principas atsižvelgia į algoritmo savybes su kuo mažesniu tikslo funkcijų perskaičiavimų skaičiumi apskaičiuoti optimalų sprendinį ir optimalaus sprendinio aproksimaciją su kuo mažesne paklaida. Taigi matome, kad A algoritmas yra pats geriausias, o kiti algoritmai abėcėles tvarka yra blogesni, pats blogiausias iš pateiktų yra F algoritmas, tai parodo ir apskaičiuoti hipertūriai: $S_A = 1$; $S_B = 0{,}88$; $S_C = 0{,}86$; $S_D = 0{,}66$; $S_E = 0{,}64$; $S_F = 0{,}52$. Beje, pasirinktas hipertūrio skaičiavimo metodas reaguoja į tokias algoritmo savybes: optimalų sprendinių skaičių, lokalių minimumų pasiskirstymą.

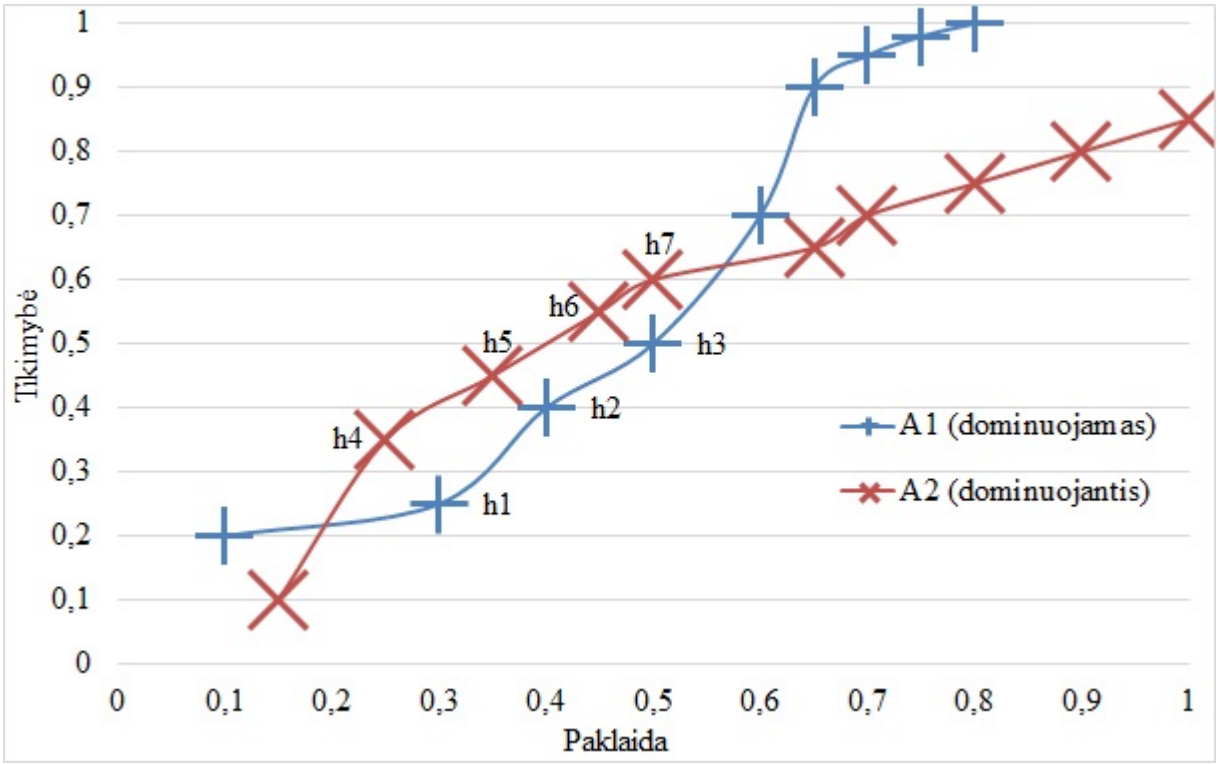
Apskaičiuojant tikimybių pasiskirstymo funkcijos hipertūrį gaunamas matas, kuri parodo al-

goritmo savybes tam tikrame rėžyje. Ši hipertūrio metrika gali būti panaudojama lyginant algoritmus. Lyginti tikimybių pasiskirstymo funkcijas, remiantis tik hipertūriu – dažnai gali būti sudėtinga, nes hipertūris skirtingoms tikimybių pasiskirstymo funkcijomis gali būti vienodas arba labai artimas.

2.2. Dominuojamumo vertinimas

Ankstesiame poskyryje pasiūlytas ir aprašytas algoritmų palyginimo metodas, kai algoritmai lyginami atsižvelgiant į algoritmų tikimybių pasiskirstymo funkcijos grafiko hipertūrį. Tačiau, galime atsižvelgti į du kriterijus: gautų reikšmių paklaidą ir tų reikšmių tikimybę.

Tam pritaikome dominuojamumo sąryšį, kuris panašiai taikomas ir daugiakriteriame optimizavime [Žil05] – lyginant gautų sprendinių dominuojamumą. Šie du kriterijai: paklaida ir tikimybė prieštarauja vienas kitam. Geresnis rezultatas gaunamas, kai didesnė tikimybė p ir mažesnė paklaida p.



11 pav. Tikimybių pasiskirstymo funkcijos dominavimas

Dominuojanti reikšmė yra ta, kuri geresnė bent pagal vieną kriterijų, o kiti kriterijai nėra blogesni. Taigi, pritaikome dominuojamumo principą tarp dviejų tikimybių pasiskirstymo funkci-

jų reikšmių. Panagrinėkime grafiką su dvejomis tikimybių pasiskirstymo funkcijomis – žr. 11 paveikslą. Turime dviejų algoritmų $A1$ ir $A2$ tikimybių pasiskirstymo funkcijos grafikus žr. 11 pav. Kiekvieną grafiką sudaro taškai

$$(h_1, h_2, h_3, ..., h_N),$$

kur h – taškas, kurį sudaro dvi reikšmės t – tikimybė ir p – paklaida, t. y. $h = (t;\, p)$. Gali būti sudėtinga vien tik iš algoritmų tikimybių pasiskirstymo funkcijų grafikų įvertinti, kuris algoritmas yra geresnis. Pavyzdžiui, iš grafiko matome, kad: $A1$ algoritmas yra geresnis kai apskaičiuoja reikšmes su paklaidomis apytiksliai nuo $0{,}1$ iki $0{,}2$ ir kai paklaidos būna apytiksliai nuo $0{,}2$ iki $0{,}58$, o $A2$ algoritmas yra geresnis, kai apskaičiuoja reikšmes su paklaidomis apytiksliai nuo $0{,}2$ iki $0{,}58$. Pritaikome hipertūripo skaičiavimo formulę ir gauname:

$$S_{A1} = 0{,}532, S_{A2} = 0{,}465.$$

Taigi $A1$ algoritmo hipertūris yra $0{,}067$ didesnis už $A2$ algoritmo hipertūrį. Pavaizduoti taškai grafike (žr. 11 pav.): $h1, h2, h3$, kurie priklauso $A1$ algoritmo tikimybių pasiskirstymo funkcijai, o taškai: $h4, h5, h6, h7$ priklauso $A2$ algoritmo tikimybių pasiskirstymo funkcijai. Iš grafiko matome, kad taškai: $h1, h2, h3$ yra dominuojami $A2$ algoritmo taškais: $h4, h5, h6, h7$, nes:

- taškas $h1$ dominuojamas taško $h4$;
- taškas $h2$ dominuojamas taško $h5$;
- taškas $h3$ dominuojamas taškų $h6$ ir $h7$.

Pavyzdžiui, taškas $h1$ geresnis už tašką $h4$, taškas $h2$ geresnis už tašką $h5$ ir $h3$ geresnis už tašką $h6$ pagal du kriterijus: paklaidą ir tikimybę, o taškas $h3$ yra geresnis už $h7$ tik pagal vieną kriterijų – tikimybę. Taigi dominuojanti reikšmė, analizuojant tikimybių pasiskirstymo funkcijų reikšmes, yra kai, reikšmė yra geresnė pagal du kriterijus, arba geresnė pagal vieną kriterijų, o kitas kriterijus yra vienodas. Tarkime, turime tikimybių pasiskirstymo funkcijos grafiko taškus: $t(p_a;\, t_a)$ ir $t(p_b;\, t_b)$, kur p – paklaida, o t – tikimybė. Taškas $r(p_a;\, t_a)$ yra dominuojamas taško $r(p_b;\, t_b)$, kai tenkinama bent viena iš šių sąlygų:

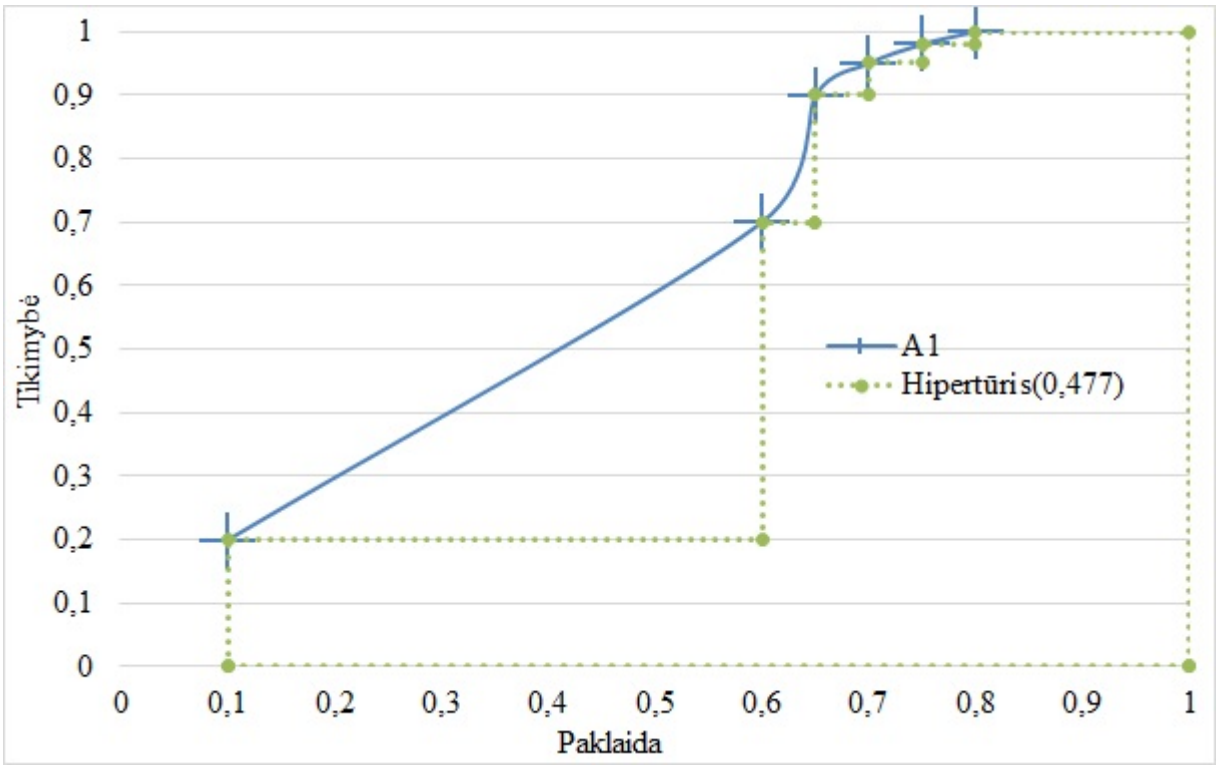
- $p_a < p_b$ ir $t_a < t_b$;

- $p_a \leq p_b$ ir $t_a < t_b$;
- $p_a < p_b$ ir $t_a \leq t_b$;

Pavyzdžiui, turime du skirtingus $A3$ ir $A4$ tikimybių pasiskirstymo funkcijos grafikus ir palyginame skirtingų algoritmų taškus: $A3$ algoritmo tikimybių pasiskirstymo funkcijos grafiko taškas – $h_{A3} = (0{,}3;\ 0{,}4)$ ir A_{A4} algoritmo tikimybių pasiskirstymo funkcijos grafiko taškas – $h_{A4} = (0{,}3;\ 0{,}5)$. Iš taškų h_{A3} ir h_{A4} matome, kad paklaidos yra vienodos, tačiau algoritmo $A4$ tikimybių pasiskirstymo grafiko taško h_{A4} matome, kad tikimybė yra $0{,}5$, o $A3$ tikimybių pasiskirstymo funkcijos grafiko algoritmo taško h_{A3} tikimybė yra $0{,}3$, taigi taškas h_{A4} yra dominuojantis prieš h_{A3} tašką, nes reikšmių paklaidos yra lygios, o tikimybė yra didesnė.

2.3. Hipertūris be dominuojančių taškų

Pritaikę dominuojamumo principą, pašaliname taškus $h1, h2, h3$ (žr. 11) iš A algoritmo tikimybių pasiskirstymo funkcijos ir gauname grafiką (žr. 12) be $h1, h2, h3$ taškų.



12 pav. Tikimybių pasiskirstymo funkcijos grafiko hipertūris

Taigi, grafike (žr. 12 pav.) $A1$ funkcijai pritaikytas dominuojamumo principas ir $A1$ funkcijai palikti taškai, kurie yra dominuojantys ir taškai, kurių negalima palyginti su funkcijos $A2$

(žr. 11 pav.) taškų reikšmėmis t. y. paliekame taškus $A1$ funkcijos be taškų, dominuojamų $A2$ (žr. 11 pav.) funkcijos taškų. Pritaikome aukščiau aprašytą hipertūrio skaičiavimo principą. Hipertūrio skaičiavimo principą galime išreikšti ir grafiškai – sudaroma laiptuota figūra, kuri pažymėta taškine linija (žr. 12 pav.) ir apskaičiuojamas šios figūros plotas. Apskaičiuojamas $A1$ algoritmo tikimybių pasiskirstymo funkcijos be taškų, dominuojamų $A2$ algoritmo tikimybių pasiskirstymo funkcijos – hipertūris, kuris yra mažesnis, nei anksčiau apskaičiuotas (žr. 11 pav.).

2.4. Algoritmų tarpusavio dominuojamumas

Pritaikius dominuojamumą gauname naują funkciją, kurią galime palyginti su funkcija (grafikas be pašalintų taškų), kurios dominuojamumą norime sužinoti. Tam pritaikome straipsnyje [ZDT00] aprašytą persidengimo metriką (angl. *Coverage Metric, C*), kuri panašiai naudojama ir daugiakrititeriame optimizavime. Tarkime, kad lyginame du algoritmus: $A1$ su $A2$, kurie yra:

- $A1$ – dominuojamas;
- $A2$ – dominuojantis.

Pateikiama formulė, kuria lyginame algoritmų dominuojamumą:

$$C(A1, A2) = \frac{S_{A1} - S'_{A1}}{S_{A1}}; \tag{23}$$

Čia S_{A1} – $A1$ algoritmo hipertūris. S'_{A1} – $A1$ algoritmo be taškų, dominuojamų $A2$ algoritmo hipertūris.

Apskaičiuojame $A1$ dominuojamumą su $A2$. Turime reikšmes: $S_{A1} = 0{,}532$; $S'_{A1} = 0{,}477$. Taigi galime apskaičiuoti:

$$C(A1, A2) = \frac{0{,}532 - 0{,}477}{0{,}532} = 0{,}103$$

apskaičiavus gauname, kad $A1$ algoritmas yra dominuojamas $A2$ algoritmo $0{,}103$ dominuojamumo koeficientu.

Dominuojamumas išreiškiamas pagal formulę, kuria gaunamas dominuojamumo koeficien-

tas, kuris tarpusavyje palygina dviejų algoritmų dominuojamumą ir nusako kiek algoritmas yra dominuojamas kito algoritmo. Šia metrika gaunamos skaitinės reikšmės, kurias sprendimų priėmėjui lengva palyginti, kaip lengva palyginti ir lyginamus algoritmus, kuris yra geresnis (tam tikrame rėžyje).

Formule gautas matas visada tenkina sąlygą: $1 \leq C(A1, A2) \geq 0$ ir kai:

- $C = 0$, tai $A1$ nėra dominuojamas algoritmo $A2$;
- $C = 1$, tai $A1$ pilnai (visi taškai) dominuojamas algoritmo A_2.

Beje, reikia atsižvelgti, kad formulės paramaterai negali būti sumaišyti, nes formulė nėra komutatyvi:

$$C(A1, A2) \neq C(A2, A1) \tag{24}$$

taigi, svarbu lyginant algoritmus, kurį laikome dominuojamu, o kurį dominuojančiu, nes nuo to priklauso gauta reikšmė. Tai galima įsitikinti apskaičiavus $A2$ algoritmo dominuojamumą su $A1$ algoritmu t. y. $C(A2, A1)$. Randame A2 algoritmo be taškų, dominuojamų $A1$ algoritmo, hipertūrį. Skaičiuojame hipertūrį – algoritmo $A2$ tikimybių pasiskirstymo funkcijos, kurią sudaro taškai: $h4, h5, h6, h7$. Gauname, kad $S'_{A2} = 0{,}408$. Turime $A2$ algoritmo hipertūrį ($S_{A2} = 0{,}465$), taigi apskaičiuojame $A2$ algoritmo dominuojamumą su $A1$ algoritmu:

$$C(A2, A1) = \frac{0{,}465 - 0{,}408}{0{,}465} = 0{,}123$$

gauname, kad $A2$ algoritmas yra dominuojamas $A1$ algoritmo $0{,}123$ dominuojamumo koeficientu.

Taigi norint apskaičiuoti, kaip $A1$ algoritmas yra dominuojmas $A2$ algoritmo reikia:

- iš algoritmų $A1$ ir $A2$ apskaičiuotų reikšmių apskaičiuoti tikimybių pasiskirstymo funkcijos grafiko reikšmes, pagal aukščiau pateiktą aprašą (žr. 2.1 poskyrį);
- apskaičiuoti $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūrį pagal aukščiau pateiktą (žr. 22) formulę.
- iš $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko pašalinti taškus, kurie yra dominuojami $A2$ algoritmo tikimybių pasiskirstymo funkcijos grafiko reikšmių, pagal aukščiau aprašytą principą (žr. 2.2 poskyrį).

- apskaičiuoti $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko be taškų dominuojamų $A2$ algoritmo tikimybių pasiskirstymo funkcijos grafiko taškų, hipertūrį pagal aukščiau pateiktą (žr. 22) formulę.
- apskaičuoti dominuojamumo koeficientą: iš apskaičiuoto $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūrio atimti apskačiuoto hipertūrio – $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko be taškų dominuojamų $A2$ algoritmo ir gautą skirtumą padalinti iš apskaičiuoto $A1$ algoritmo tikimybių pasiskirstymo funkcijos grafiko hipertūrio.

3. EKSPERIMENTINIS TYRIMAS

Šiame skyriuje pateikiamas atliktas eksperimentas ir eksperimento rezultatai. Testiniai uždaviniai spręsti su *MATLAB* programine įranga, kurioje buvo realizuoti visi šiame skyriuje pateikti algoritmai. Atliekant eksperimentus naudojami išspręstų testinių uždavinių rezultatai. Atliekant eksperimentą naudojausi savo sukurta taikomąja programa, kurios trumpas aprašymas pateikiamas sekančiame poskyryje.

3.1. Algoritmų vertinimo įrankis

Taikomoji programa realizuota Java programine kalba, o grafinė vartotojo sąsaja sukurta, naudojantis viena didžiausių platformos J2SE (angl. *Java 2 Standard Edition*) bibliotekos rinkinių – JFC (angl. *Java Foundation Classes*), su kuria galima kurti sudėtingas grafines vartotojo sąsajas. Pasirinkta naudoti vieną iš JFC bibliotekų – Swing, kurioje yra visi grafiniai komponentai, iš kurių gali būti sudaroma grafinė vartotojo sąsaja, bei pagalbinės klasės ir instrumentai, skirti darbui su šiais komponentais. Grafikams atvaizduoti, naudojama nemokama Java biblioteka – JFreeChart.

1 lentelė. Įvedamų duomenų pavyzdys

Funkcijos perskaičiavimai	Reikšmė
500	9266947.83
1000	9367074.33
2000	9479952.17
500	9266957.32
1000	9367014.33
2000	9399962.17
500	9266987.50
1000	9367044.03
2000	9479942.77
500	9266937.53
1000	9367054.23
2000	9479960.17
500	9266938.03
1000	9367053.01
2000	9479961.97

Įvedamų duomenų analizė. Duomenų apdorojimui taikomoji programa turi gauti algoritmo apskaičiuotą reikšmę ir turi būti žinoma su kokiu funkcijų perskaičiavimų biudžetu gauta ši reikšmė. Taigi duomenų įvedimui pasirinkta duomenų struktūra, lyg duomenys būtų saugomi lentele: pirma-

me stulpelyje: po kiek funkcijos perskaičiavimų gauta reikšmė, o antrame stulpelyje – algoritmo apskaičiuota reikšmė. Pavyzdžiui, įvedamų duomenų failo domenys gali būti, kaip pavaizduota 1 lentelėje. Iš duomenų failo matome, kad atliktų eksperimentų yra keturi ir algoritmas naudojo tikslo funkcijų perskaičiavimo skaičių $N = 2000$, tačiau duomenų faile yra ir algoritmo gautos tarpinės reikšmės t. y. po 500 ir 1000 tikslo funkcijos perskaičiavimų. Šiame duomenų faile duomenys atskirti tarpu, tačiau dažnai duomenys yra atskiriami kokiu nors ženklu – vienas iš tokių dažnai naudojamų formatų yra *kableliais atskirtos reikšmės* (CSV pagal angl. *Comma-Separated Values*), kurį galima naudoti šioje taikomojoje programoje. Programos nustatymuose yra galimybė įvesti simbolį, kurį privalės turėti įvedamų duomenų failas t. y. stulpeliai privalės būti atskiriami šiuo iš anksto nustatytu simboliu. Taip pat, taikomoji programa geba automatiškai atpažinti kokiu formatu yra pateikiami duomenys ir apie tai informuoti taikomosios programos naudotoją. Taikomoji programa automatiškai suranda kokiu simboliu yra atskirti įvedamų duomenų stulpeliai. Automatiniam įvedimo duomenų formato nustatymui yra naudojamos reguliarios išraiškos (angl. *Regular expressions*). Reguliari išraiška – iš anksto aprašytų simbolių seka, kuria nusakome kurioje pozicijoje ar pozicijose ir kiek simbolių tikimės – tai yra lyg paieškos šablonas. Naudojant automatini įvedamų duomenų formato nustatymą, įvedimo duomenų faile reikia pasirūpinti vienintele sąlyga, kad kiekvienoje eilutėje būtų du skaičiai. Pirmasis skaičius būtų funkcijos perskaičiavimų skaičius (N), o kitas skaičius būtų algoritmo apskaičiuota reikšmė. Naudojant automatinį įvedamų duomenų formato atpažinimą automatiškai atpažįstama:

- kokiais simboliais bus atskiriami įvesties duomenys,
- kokiu simboliu yra atskiriami, algoritmo apskaičiuotos reikšmės, šimtosios dalys – kableliu ar tašku,
- ar duomenų failas turi tuščių eilučių, antraštę ir pan.

Jeigu programa nesugeba automatiškai aptikti įvedamų duomenų formato – vartotojas apie tai yra informuojamas. Pavaizduotame duomenų faile (žr. 1 lentelę), kuriame matome, kad duomenų faile pirma eilutė yra antraštė. Kadangi, kai kurios programos duomenų failą eksportuojant į *.csv tipo failus automatiškai prideda antraštę. Tai taikomoji programa automatiškai atpažįsta ar duomenų failas turi pirmą eilutę su antrašte. Taigi, taikomoji programa automatiškai geba atpažinti kokiu formatu yra pateikiami duomenys, tačiau geba ir automatiškai atpažinti ar algoritmas sprendžia minimizavimo ar maksimizavimo uždavinį. Analizuojant įvestus duomenis, taikomoji programa

atpažįsta sprendžiamo uždavinio tipą. Jeigu didėjant funkcijų perskaičiavimų biudžetui, didėja ir visos algoritmo apskaičiuotos reikšmės, tada tai maksimizavimo uždavinys, o jeigu su didesniais funkcijų perskaičiavimų biudžetais, mažėja reikšmės, tai – minimizavimo uždavinys. *Tikslo funkcijos perskaičiavimų skaičiaus parinkimas*. Norint analizuoti duomenis po tam tikro tikslo funkcijos perskaičiavimų skaičiaus reikia nustatyti po kiek tikslo funkcijų perskaičiavimų bus atliekama algoritmo analizė. Jeigu taikomosios programos naudotojas nepasirenka tikslo funkcijos perskaičiavimų skaičiaus, tada taikomoji programa automatiškai parenka mažiausią rastą tikslo funkcijų perskaičiavimų skaičių. Pavyzdžiui, jeigu duomenų faile yra duomenys, kurie pavaizduoti 1 lentelėje, tai tikslo funkcijų perskaičiavimų skaičius pagal nutylėjimą bus parenkamas 500. *Nagrinėjamo intervalo parinkimas*. Dažnai norima algoritmo apskaičiuotas reikšmes analizuoti tik tam tikrame intervale. Taikomoji programa turi galimybę įvesti intervalą (mažiausią ir didžiausią reikšmę), kuriame bus analizuojama algoritmo tikimybių pasiskirstymo funkcija. Jeigu taikomosios programos naudotojas neįvedė šio intervalo, tai duomenys yra parenkami automatiškai. Pagal parinktą tikslo funkcijų perskaičiavimų skaičių, įvedimo duomenų faile parenkama mažiausia ir didžiausia reikšmė. Pavyzdžiui, jeigu duomenų failas yra 1 lentelė ir pasirinktas tikslo funkcijų perskaičiavimų skaičius yra 1000, tai pagal nutylėjimą bus intervalas parenkamas nuo $9367044{,}03$ iki $9367074{,}33$. Bet kuriuo metu programos naudotojas turi galimybę šį intervalą pakeisti.

Algoritmų tarpusavio dominuojamumo palyginimas. Taikomoji programa apskaičiuoja algoritmų tarpusavio dominuojamumą. Turime z algoritmų: $A1, A2, ...Az$. Taikomoji programa visų algoritmų tarpusavio dominuojamumą pateikia lentelėje, kaip pateikta bendru atveju(žr. 2 lentelę).

2 lentelė. z algoritmų dominuojamumo palyginimas bendru atveju

	A1	**A2**	...	**Az**
A1	0	C(A1, A2)	...	C(A1, Az)
A2	C(A2, A1)	0	...	C(A2, Az)
⋮	⋮	⋮	⋮	⋮
Az	C(Az, A1)	C(Az, A2)	...	0

Pasiūlyto metodo formulė, kuri apskaičiuoja algoritmo dominuojamumą su kitu algoritmu nėra komutatyvi, taigi labai svarbu kokia tvarka yra lyginami algoritmai. Taigi, kaip ir parodyta 2 lentelėje algoritmų duominuojamumo rezultatai pateikti, laikant, kad pirmame stulpelyje esantys

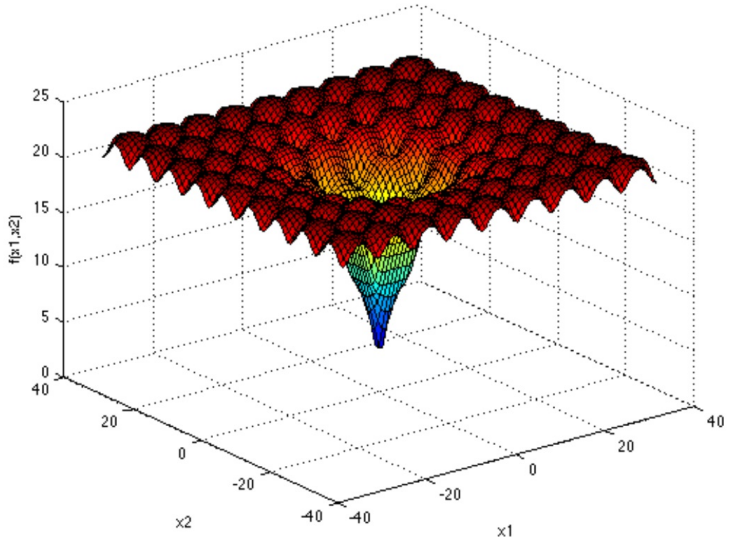
algoritmai yra dominuojami, o pirmoje eilutėje pateikti algoritmai yra dominuojantys, o kitose lentelės celėse pateikiami algoritmų dominuojamumo rezultatai. Pavyzdžiui, 2 lentelės antros eilutės ir trečio stulpelio celėje pateikiamas $C(A1, A2)$ dominuojamumo koeficientas, kai $A1$ yra dominuojamas, o $A2$ – dominuojantis. Trečios eilutės ir antro stulpelio celėje yra lyginamas algoritmų dominuojamumas, kai $A1$ yra dominuojantis, o $A2$ – dominuojamas. Pasiūlytu, algoritmų dominuojamumo vertinimu, lyginami du algoritmai, tačiau taikomoji programa pateikia vienoje lentelėje z algoritmų dominuojamumo vertinimą su visais algoritmais. Taigi taikomosios programos naudotojas gali vienoje lentelėje matyti, kuris algoritmas yra labiausiai dominuojamas ar dominuojantis. Realizuota taikomoji programa, kuri gali:

- apdoroti įvedamus duomenis ir atpažinti įvedamų duomenų struktūrą;
- nustatyti tai minimizavimo ar maksimizavimo uždavinys;
- apskaičiuoti tikimybių pasiskirstymo funkcijos grafiko reikšmes;
- apskaičiuoti tikimybių pasiskirstymo funkcijos grafiko hipertūrį;
- apskaičiuoti algoritmų tarpusavio dominuojamumo koeficientą.

3.2. Testiniai uždaviniai

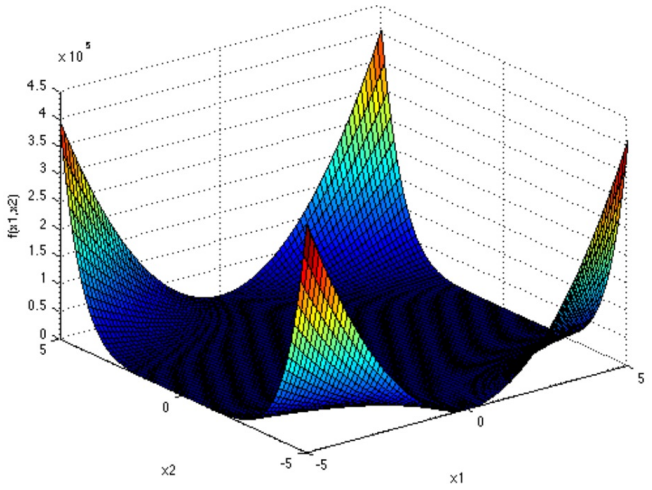
Atliekant eksperimentus naudojami šiame poskyryje aprašyti testiniai uždaviniai. Visuose testiniuose uždaviniuose kintamųjų skaičius yra du. Testiniai uždaviniai yra šie:

- Ackley – funkcija turi daug lokalių minimumo taškų (žr. 13 pav.).



13 pav. Ackley funkcija [SB15]

- Beale – funkcija neturi daug lokalių minimumo taškų (žr. 14 pav.).

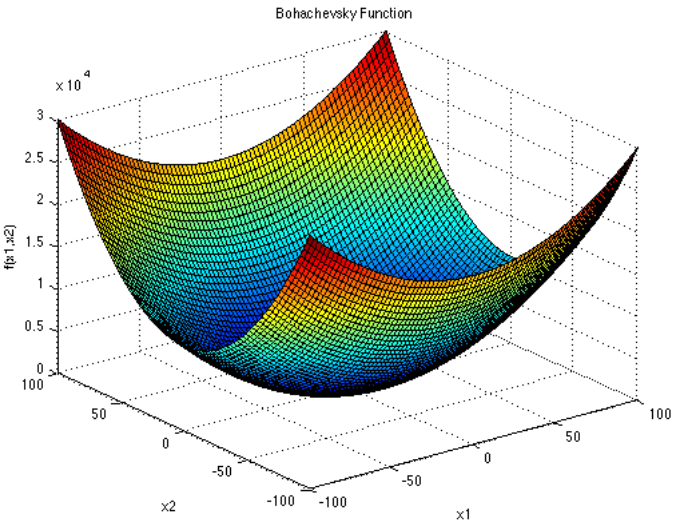


14 pav. Beale funkcija [SB15]

- Bi-spherical funkcija, kurios matematinė išraiška [HL15] yra

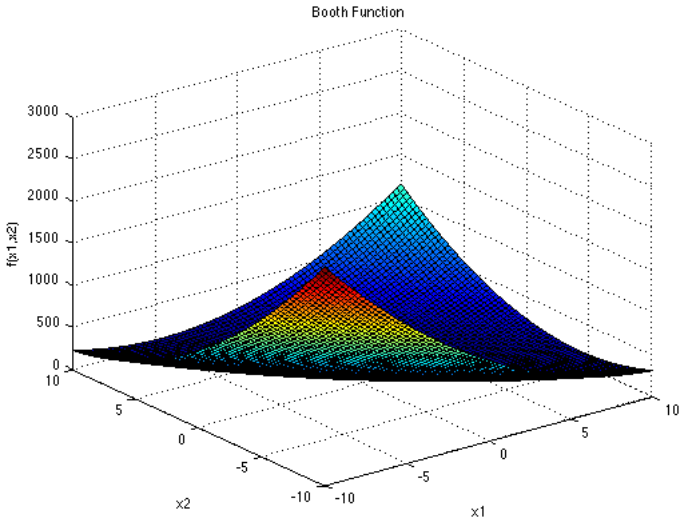
$$f(x) = min\{(x_1 - 1)^2, (x_1 + 1)^2 + 0{,}01\} + \sum_{2}^{n} x_i^2 \tag{25}$$

- Bohachevsky – funkcija, kuri neturi daug lokalių minimumo taškų (žr. 15 pav.).



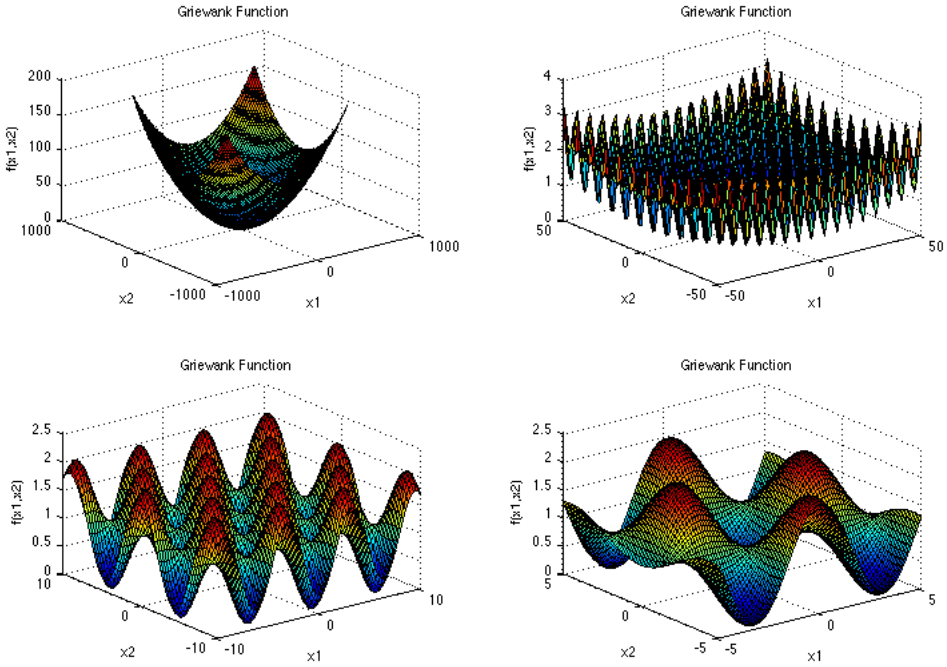
15 pav. Bohachevsky funkcija [SB15]

- Booth – funkcija, kuri neturi lokalių minimumo taškų (žr. 16 pav.).



16 pav. Bohachevsky funkcija [SB15]

- Griewank – funkcija, kuri turi labai daug lokalių minimumo taškų. Šio uždavinio sudėtingumą galime pamatyti (žr. 17 pav.) grafikuose.



17 pav. Griewank funkcija [SB15]

3.3. Atsitiktinės paieškos algoritmai

Uždaviniai sprendžiami su žemiau pateiktais atsitiktinės paieškos algoritmais.

Valdomos atsitiktinės paieškos algoritmas

Valdoma atsitiktinė paieška (angl. *controlled random search*, CRS), žemiau pateikiamas šio algoritmo pseudokodas [HL15]:

Algoritmas 1 CRS

procedure CRS(X, f, N, M)
  Generuoti ir įvertinti rinkinį $\boldsymbol{P}$ iš M atsitiktinių taškų tolygiai per X;
  $k \leftarrow M$;
  while $k < N$ **do**
    Pasirinkti atsitiktiniame pogrupyje $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{n+1}\} \subset \boldsymbol{P}$;
    $\boldsymbol{x}_k \leftarrow \frac{2}{n} \sum_{i=1}^{n} \boldsymbol{p}_i - \boldsymbol{p}_{n+1}$;
    if $\boldsymbol{x}_k \in X$ **and** $f(\boldsymbol{x}_k) < \max_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$ **then** replace $argmax_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$ by $\boldsymbol{x}_k$;
    $k \leftarrow k + 1$;
  end while
  return $argmin_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$ and $\min_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$;
end procedure

Genetinis algoritmas

Genetinis algoritmas (angl. *Genetic Algorithm*, GA), žemiau pateikiamas šio algoritmo pseudokodas [HL15]:

Algoritmas 2 GA

procedure GA(X, f, N, M)
  Generuoti ir įvertinti rinkinį $\boldsymbol{P}$ iš M atsitiktinių taškų tolygiai per X;
  $k \leftarrow M$;
  while $k < N$ **do**
    $\boldsymbol{Q} = \emptyset$;
    for $i = 1$ **to** M **do**
      Pasirinkti atsitiktiniuose $\{\boldsymbol{p}_1, \boldsymbol{p}_2\} \subset \boldsymbol{P}$;
      Generuoti $\boldsymbol{q}$ kertančius $\boldsymbol{p}_1$ ir $\boldsymbol{p}_2$.
      Mutuoti $\boldsymbol{q}$ pakeičiant kai kurias jo koordinates;
      $\boldsymbol{Q} \leftarrow \boldsymbol{Q} \cup \{\boldsymbol{q}\}$;
    end for
    Įvertinti generuojamų taškų tinkamumą $\boldsymbol{q} \in \boldsymbol{Q}$;
    $k \leftarrow k + M$;
    $\boldsymbol{P} \leftarrow M$ geriausis taškai iš $\boldsymbol{P} \cup \boldsymbol{Q}$;
  end while
  return $argmin_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$;
end procedure

Daugelio pradinių taškų algoritmas

Daugelio pradinių taškų algoritmas (angl. *Multi-Start*, MS), žemiau pateikiamas šio algoritmo pseudokodas [HL15]:

Algoritmas 3 MS

procedure MS(X, f, N, LS)
  $F \leftarrow \infty$; $k \leftarrow 1$;
  while $N > 0$ **do**
    Generuoja $\boldsymbol{s}$ tolygiai per X;
    $[\boldsymbol{x}, N_{\text{LS}}] \leftarrow \text{LS}(\boldsymbol{s}, N)$;
    if $f(\boldsymbol{x}) < F$ **then** $F \leftarrow \boldsymbol{x}_k$;
    $N \leftarrow N - N_{\text{LS}}$; $k \leftarrow k + 1$;
  end while;
  return F and $argmin_k f(\boldsymbol{x}_k)$;
end procedure

Dalelių spiečiaus optimizavimo algoritmas

Dalelių spiečiaus optimizavimas (angl. *Particle Swarm Optimization*, PSO), žemiau pateikiamas šio algoritmo pseudokodas [HL15]:

Algoritmas 4 PSO

procedure PSO(X, f, N, M, ω, c_1, c_2)
  Generuoti ir įvertinti rinkinį $\boldsymbol{P}$ iš M atsitiktinių taškų tolygiai per X;
  $\boldsymbol{x} \leftarrow argmin_{\boldsymbol{p} \in \boldsymbol{P}} f(\boldsymbol{p})$; $\boldsymbol{b}_i \leftarrow \boldsymbol{p}_i$ ir $\boldsymbol{v}_i \leftarrow 0$, $i = 1,2,\ldots,M$;
  $k \leftarrow M$;
  while $k < N$ **do**
    for $i = 1$ **to** M **do**
      for $j = 1$ **to** n **do**
        Generuoti r tolygiai per $[0,1]^2$;
        $v_{ij} \leftarrow \omega v_{ij} + 2c_1 r_1 (b_{ij} - p_{ij}) + 2c_2 r_2 (x_{bj} - p_{ij})$;
      end for
      $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \boldsymbol{v}_i$;
      if $f(\boldsymbol{p}_i) < f(\boldsymbol{b}_i)$ **then** $\boldsymbol{b}_i \leftarrow \boldsymbol{p}_i$;
      if $f(\boldsymbol{p}_i) < f(\boldsymbol{x})$ **then** $\boldsymbol{x} \leftarrow \boldsymbol{p}_i$;
    end for
    $k \leftarrow k + M$;
  end while
  return $f(\boldsymbol{x})$ and $\boldsymbol{x}$;
end procedure

Paprastosios atsitiktinės paieškos algoritmas

Paprastoji atsitiktinė paieška (angl. *Pure Random Search*, PRS), žemiau pateikiamas šio algoritmo pseudokodas [HL15]:

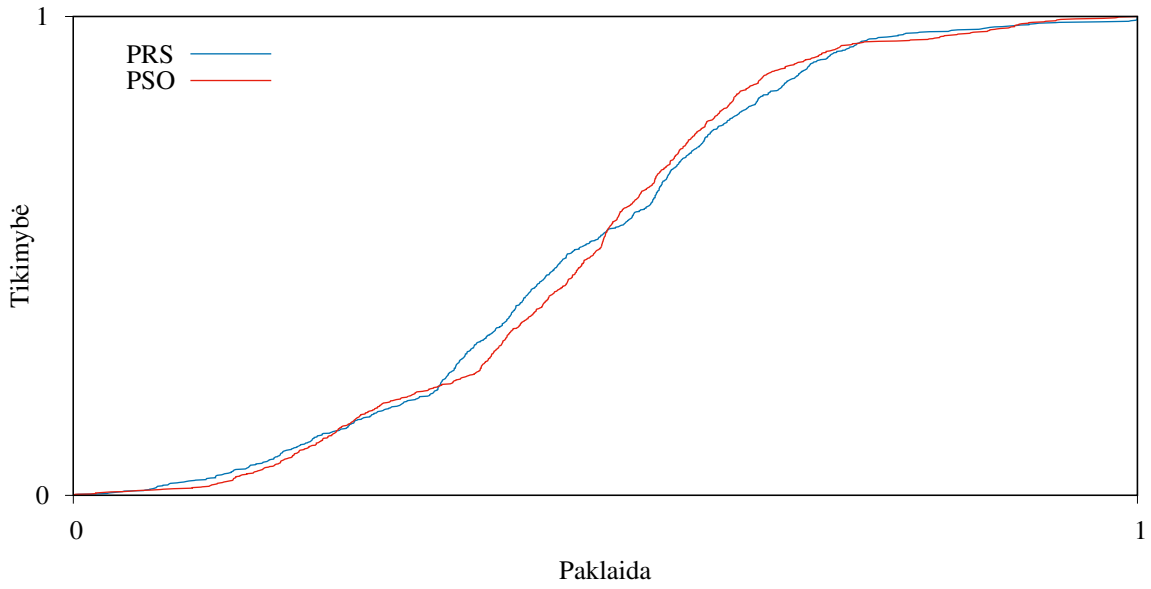
Algoritmas 5 PRS

procedure PRS(X, f, N)
    $F \leftarrow \infty$;
    for $k = 1$ **to** N **do**
        Generuoja $\boldsymbol{x}_k$ tolygiai per X;
        if $f(\boldsymbol{x}_k) < F$ **then** $F \leftarrow f(\boldsymbol{x}_k)$
    end for
    return F ir $argmin_k f(\boldsymbol{x}_k)$;
end procedure

3.4. Eksperimento rezultatai

Ackley uždavinys

Algoritmais PSO ir PRS sprendžiamas testo uždavinys – Ackley su 200 funkcijų perskaičiavimų.



18 pav. Ackley uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Matome, iš grafiko (žr. 18 pav.), kad sudėtinga pasakyti, kuris algoritmas yra efektyvesnis, nes

PRS ir PSO algoritmų apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijų grafikai susikerta daugiau nei penkis kartus. Ir dėl to pažvelgus vien į tikimybių pasiskirstymo funkcijos grafikus nėra aišku kuris algoritmas efektyvesnis.

3 lentelė. Ackley uždavinio hipertūriai.

Algoritmas	PRS	PSO
Hipertūris	0,530	0,528

Apskaičiavus algoritmų tikimybių pasiskirstymo funkcijos hipertūrius (žr. 3 lentelę) matome, kad jie skiriasi nežymiai tik 0,002.

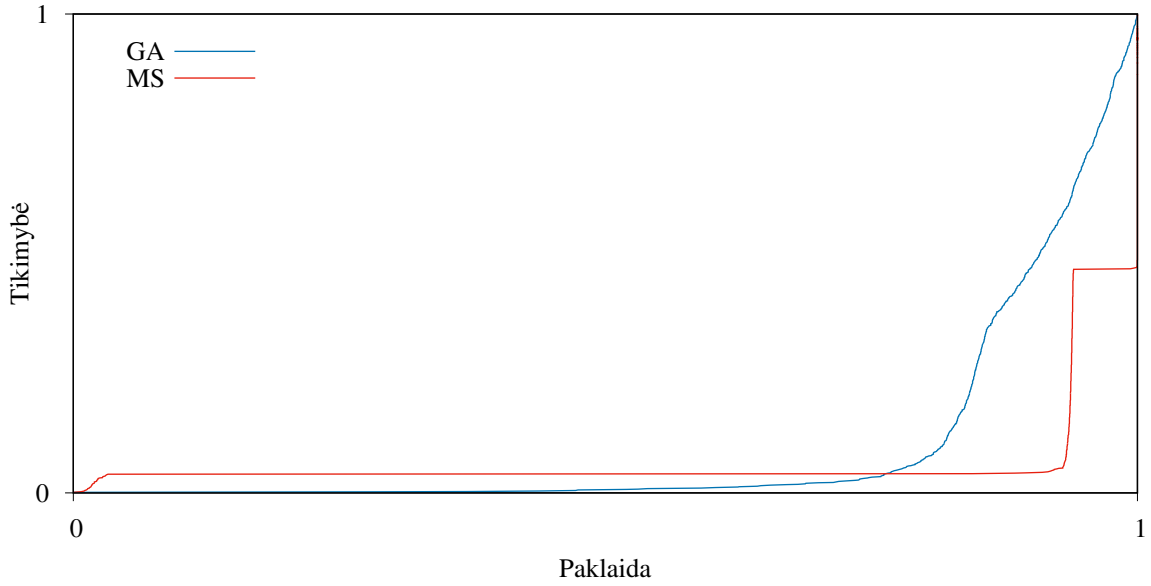
4 lentelė. Ackley uždavinį spręstų algoritmų tarpusavio dominuojamumo koeficientai.

A1 \ A2	PRS	PSO
PRS	0,000	0,104
PSO	0,057	0,000

Tačiau pritaikius algoritmų tarpusavio dominuojamumą ir apskaičiavus reikšmes (žr. 4 lentelę), matome, kad PRS algoritmu apskaičiuotos reikšmės yra dominuojamos PSO algoritmo 0,057 dominuojamumo koeficientu, o PSO yra dominuojamas algoritmo PRS tik 0,104 dominuojamumo koeficiento. Matome, kad abiejų algoritmų apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafikų hipertūriai yra labai artimi, bet dominuojamumo koeficientų reikšmės skiriasi beveik dvigubai. Taigi, atsižvelgiant į algoritmų tarpusavio dominuojamumo metriką, galime teigti, kad PSO algoritmas yra efektyvesnis sprendžiant Ackley uždavinį, nes jis yra mažiau dominuojamas PRS algoritmo.

Beale uždavinys

Uždavinį Beale (d=2) išspręstas su 200 funkcijų perskaičiavimų su algoritmais GA ir MS.



19 pav. Beale uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Iš tikimybių pasiskirstymo funkcijos grafiko (žr. 19 pav.) matome, kad MS algoritmas randa reikšmes su mažesne paklaida lyginant su algoritmo GA rastomis reikšmėmis.

5 lentelė. Beale uždavinio hipertūriai.

Algoritmas	GA	MS
Hipertūris	0,104	0,065

Tačiau MS algoritmo apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafiko hipertūris yra beveik dvigubai mažesnis (žr. 5 lentelę) lyginant su GA – tai akivaizdžiai matosi ir grafike (žr. 19 pav.).

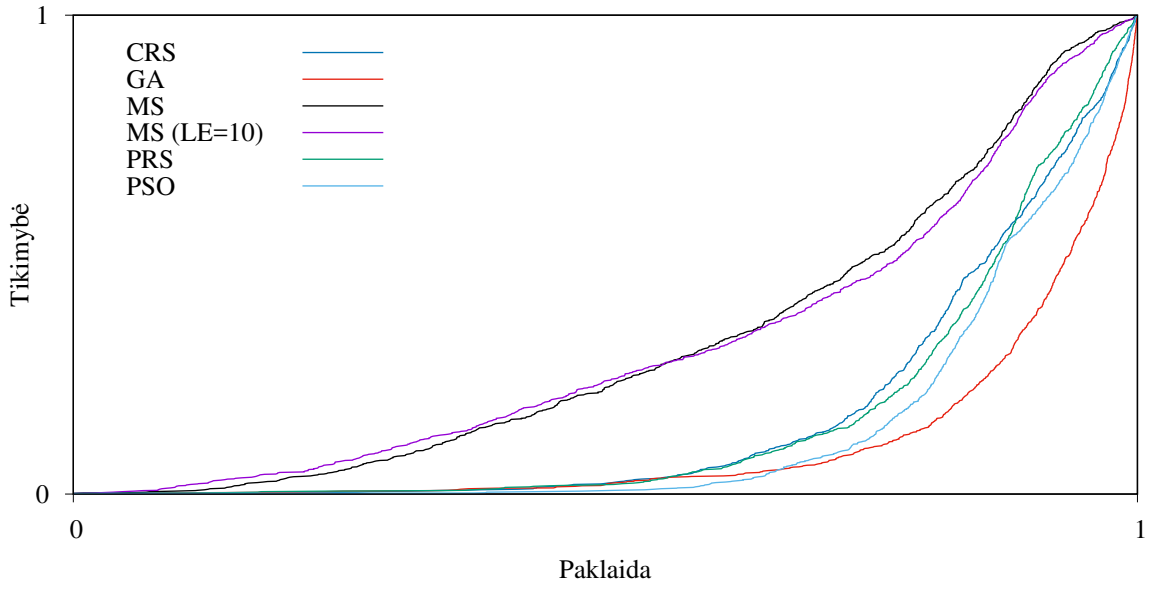
6 lentelė. Beale uždavinį spręstų algoritmų tarpusavio dominuojamumo koeficientai.

A1 \ A2	GA	MS
GA	0,000	0,416
MS	0,040	0,000

Pritaikius algoritmų tarpusavio dominuojamumą ir apskaičiavus reikšmes (žr. 6 lentelę) gauname, kad algoritmas MS yra dominuojamas GA algoritmo 0,416 dominuojamumo koeficientu t. y. beveik pusė MS apskaičiuotų reikšmių yra dominuojama GA algoritmo. Tačiau MS algoritmas yra labai mažai dominuojamas GA algoritmo. Taigi hipertūris ir dominuojamumo koeficientas parodo, kad GA algoritmas yra efektyvesnis, nei MS algoritmas.

Bi-spherical uždavinys

Uždavinys Bi-spherical išspręstas su algoritmais GA, CRS, MS(LE=10), PSO, MS, PRS su 200 funkcijos perskaičiavimų.



20 pav. Bi-spherical uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafike (žr. 20 pav.), matome, kad šis uždavinys efektyviausiai sprendžiamas su MS tipo algoritmais t. y. MS ir MS(LE=10).

7 lentelė. Bi-spherical uždavinio hipertūriai.

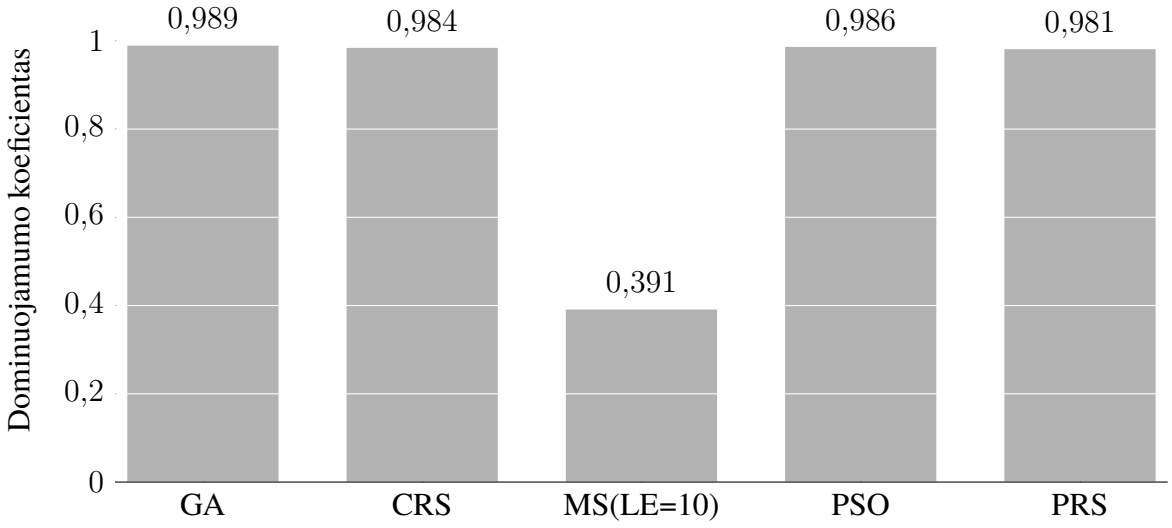
Algoritmas	GA	CRS	MS(LE=10)	PSO	PRS	MS
GA	0,104	0,161	0,310	0,134	0,159	0,312

Algoritmų MS ir MS(LE=10) apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos hipertūriai yra labai artimi – skiriasi tik 0,002 reikšme (žr. 7 lentelę).

8 lentelė. Bi-spherical uždavinį spręstų algoritmų tarpusavio dominuojamumo koeficientai.

A1 \ A2	GA	CRS	MS(LE=10)	PSO	PRS	MS
GA	0,000	0,808	0,989	0,743	0,806	0,989
CRS	0,001	0,000	0,990	0,001	0,136	0,984
MS(LE=10)	0,000	0,000	0,000	0,000	0,000	0,391
PSO	0,025	0,804	0,986	0,000	0,986	0,986
PRS	0,002	0,306	0,981	0,000	0,000	0,981
MS	0,000	0,000	0,153	0,000	0,000	0,000

Palyginus algoritmų MS ir MS(LE=10) tarpusavio dominuojamumą (žr. 8 lentelę), matome, kad algoritmas MS yra daug labiau dominuojamas algoritmo MS(LE=10), negu MS(LE=10) dominuojamas MS algoritmo.

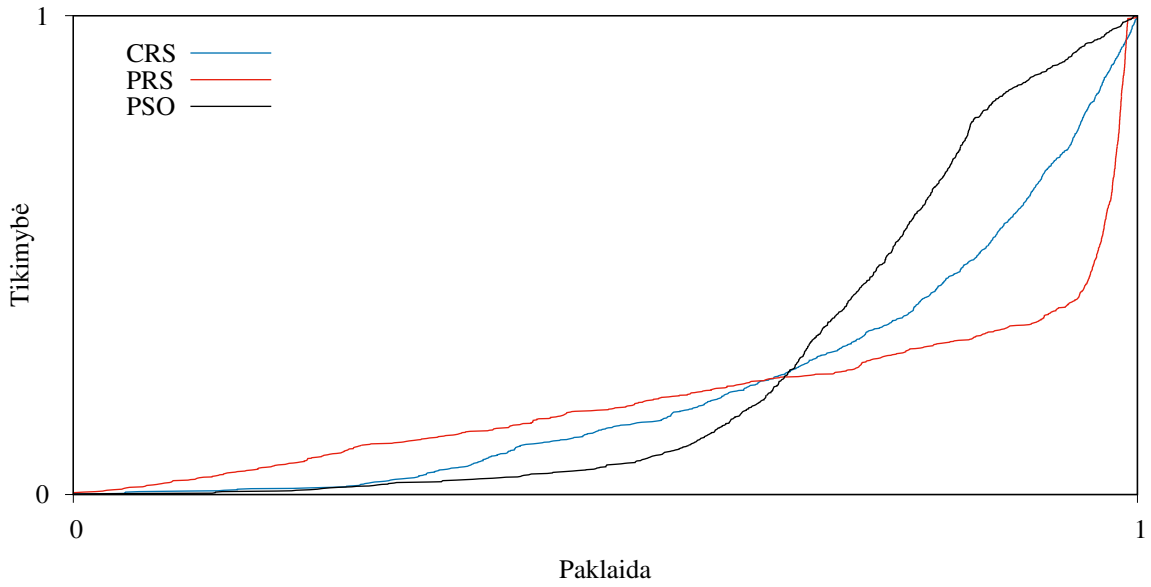


21 pav. MS dominuojamumas prieš kitus algoritmus. MS dominavimas.

Iš grafiko (žr. 21) matome, kad MS beveik visiškai dominuoja algoritmus: GA, CRS, PSO ir PRS. Lnetelėje (žr. 8) matome, kad MS yra dominuojamas tik vienintelio MS(LE=10) algoritmo. Tačiau MS(LE=10) daugiau, nei dvigubai yra dominuojamas MS algoritmo. Taigi atsižvelgiant į algoritmų tarpusavio dominuojamumą, algoritmas MS yra efektyvesnis. Taip pat sprendžiant Bi-spherical testinį uždavinį yra efektyviau naudoti MS(LE=10) algoritmą su ribota lokalia paieška.

Bohachevsky uždavinys

Bohachevsky testinis uždavinys išspręstas su algoritmais CRS, PRS ir PSO. Funkcijų perskaičiavimų skaičius – 1000.



22 pav. Bohachevsky uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafike (žr. 22 pav.), matome, kad šis uždavinys efektyviausiai sprendžiamas su PRS algoritmu, kai apskaičiuota reikšmė neviršija apytiksliai 0,68 paklaidos. Tačiau, kai paklaida apytiksliai nuo 0,68 tikimybė rasti norimo tikslumo sprendinį geresnės reikšmės yra gaunamas su CRS algoritmu. Matome iš grafiko (žr. 22 pav.), kad visi 3 algoritmai (CRS, PRS ir PSO) apskaičiuoja reikšmes skirtingai ir vien tik iš grafiko sudėtinga pasakyti, kuris algoritmas yra efektyviausias.

9 lentelė. Bohachevsky uždavinio hipertūriai.

Algoritmas	CRS	PRS	PSO
Hipertūris	0,201	0,224	0,255

Apskaičiavus hipertūrius (žr. 9 lentelę) matome, kad hipertūriai skiriasi nežymiai, bet PSO algoritmo apskaičiuotų reikšmių tikimybių pasiskirstymo grafiko hipertūris yra didesnis, nei PRS ir CRS.

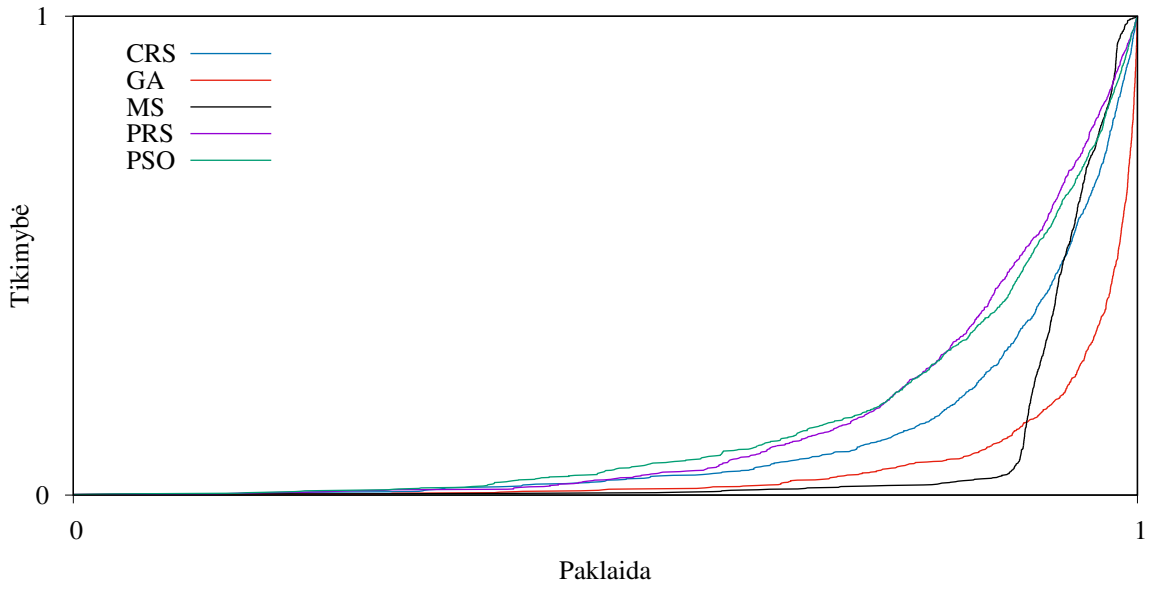
10 lentelė. Bohachevsky uždavinį sprėstų algoritmų tarpusavio dominuojamumo koeficientai.

A1 \ A2	PRS	PSO	CRS
PRS	0,000	0,171	0,170
PSO	0,116	0,000	0,096
CRS	0,200	0,386	0,000

Apskaičiavus algoritmų tarpusavio dominuojamumą (žr. 10 lentelę) matome, kad kiekvienas algoritmas yra dominuojamas kitų dviejų algoritmų. Tačiau PSO algoritmas yra mažiausiai dominuojamas kitų algoritmų, lyginant su PRS ir CRS algoritmais. Taigi galime teigti, kad PSO algoritmas yra efektyvesnis, nei PRS ir CRS, beje nors ir nežymiai tai parodė ir hipertūris.

Booth uždavinys

Booth testo uždavinys išspręstas su algoritmais PRS, MS, GA, PSO ir CRS. Funkcijų perskaičiavimų skaičius – 500.



23 pav. Booth uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafike (žr. 23 pav.), matome, kad šis uždavinys efektyviausiai sprendžiamas su PRS ir PSO algoritmais. Akivaizdžiai matosi iš gra-

fiko, kad algoritmai: MS, GA ir CRS sprendžia šį uždavinį ne taip efektyviai, kaip PRS ir PSO algoritmai.

11 lentelė. Booth uždavinio hipertūriai.

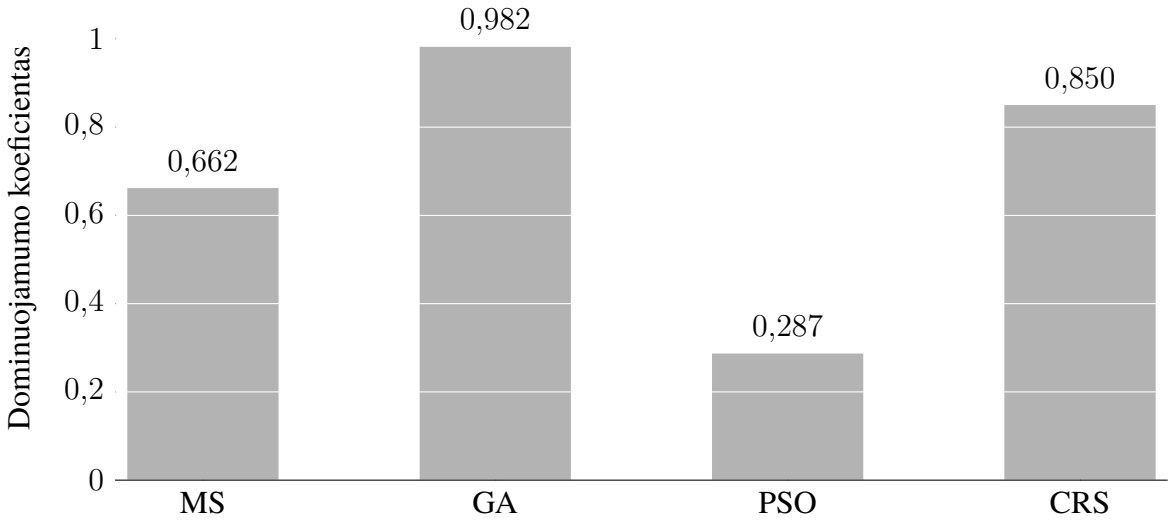
Algoritmas	PRS	MS	GA	PSO	CRS
Hipertūris	0,148	0,073	0,055	0,150	0.112

Apskaičiavus hipertūrius (žr. 11 lentelę) matome, kad algoritmų: PRS ir PSO hipertūriai skiriasi nežymiai. Iš likusių algoritmų matome, kad CRS algoritmo hipertūris yra didžiausias, taigi šis algoritmas yra efektyviausias lyginant su MS ir GA algoritmais. Tačiau, kadangi PRS ir PSO hipertūriai skiriasi nežymiai nėra aišku, kuris algoritmas efektyvesnis.

12 lentelė. Booth uždavinį spręstų algoritmų tarpusavio dominuojamumo koeficientai.

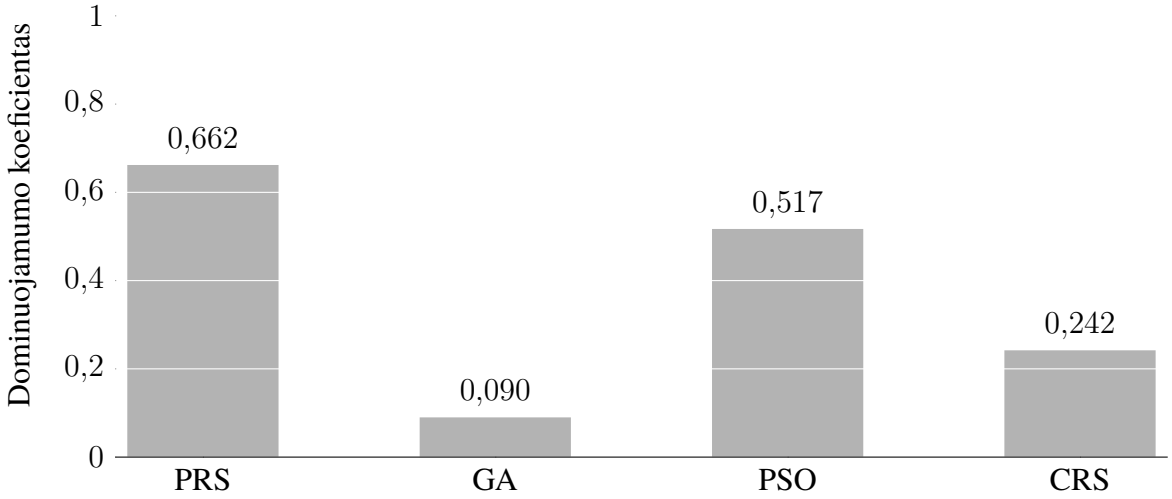
A2 / A1	PRS	MS	GA	PSO	CRS
PRS	0,000	0,010	0,000	0,144	0,002
MS	0,662	0,000	0,090	0,517	0,242
GA	0,982	0,361	0,000	0,982	0,982
PSO	0,287	0,031	0,000	0,000	0,000
CRS	0,850	0,130	0,000	0,000	0,000

Apskaičiavus algoritmų tarpusavio dominuojamumą (žr. 12 lentelę) matome, kad PRS algoritmas yra dominuojamas 3 algoritmų, tačiau dviejų algoritmų yra dominuojamas nežymiai tik $0{,}01$ ir $0{,}002$ dominuojamumo koeficientų. Likusio algoritmo dominuojamas $0{,}144$ dominuojamumo koeficiento. Pagal hipertūrio reikšmę ir iš lentelės (žr. 12) matome, kad kitas efektyviausias algoritmas yra PSO. Beje, PSO algoritmas yra dominuojamas dviejų algoritmų, kaip ir CRS, tačiau PSO algoritmas yra mažiau dominuojamas PRS ir MS algoritmų, nei CRS algoritmas dominuojamas PRS ir MS algoritmų. Taigi, efektyviausi algoritmai yra PRS ir PSO algoritmai, sprendžiant Booth uždavinį.



24 pav. PRS dominuojamumas prieš kitus algoritmus. PRS dominavimas.

Iš grafiko (žr. 24) matome, kaip PRS algoritmas dominuoja kitus algoritmus, akivaizdžiai matosi, kad PRS algoritmas labiausiai dominuoja GA algoritmą. Taip pat iš grafiko matome, kad PRS algoritmas mažiausiai dominuoja PSO algoritmą, kuris yra vienas iš dviejų efektyviausių algoritmų, kurie sprendžia Booth uždavinį. Kadangi PSO algoritmas dominuojamas PRS algoritmo $0{,}287$, o PRS algoritmas dominuojamas PSO algoritmo $0{,}144$ dominuojamumo koeficientų, tai galime teigti, kad PRS algoritmas yra efektyviausias, atsižvelgiant į algoritmų tarpusavio dominuojamumą. Beje PRS algoritmo hipertūrio reikšmė yra didžiausia.



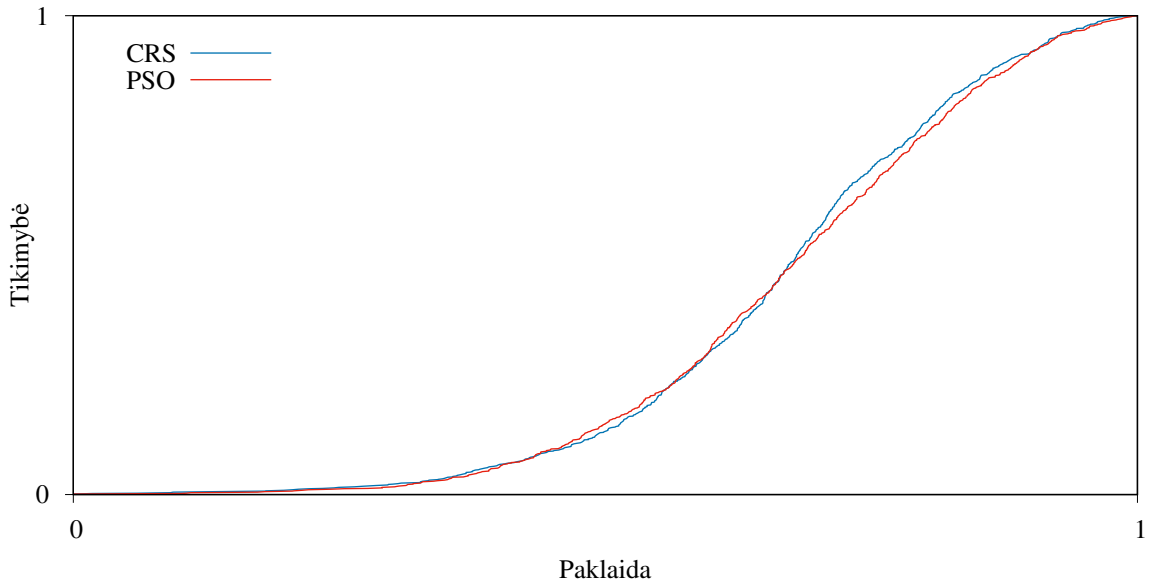
25 pav. MS dominuojamas kitų algoritmų. MS dominuojamas.

Grafike (žr. 25) pavaizduota, kaip MS algoritmas yra dominuojamas kitų algoritmų. Grafike (žr. 23 pav.) matome, kad MS algoritmas tam tikrose atkarpose yra tikrai mažiau efektyvus lyginant

su likusiais algoritmais. Visa tai pavaizduota ir grafike (žr. 25) matome, kad visos reikšmės daugiau nei nulis.

Griewank uždavinys

Griewank testo uždavinys išspręstas su algoritmais PRS ir PSO. Funkcijų perskaičiavimų skaičius – 200.



26 pav. Griewank uždavinio sprendinių tikimybių pasiskirstymo funkcijos grafikas

Apskaičiuotų reikšmių tikimybių pasiskirstymo funkcijos grafike (žr. 26 pav.), matome, kad šis uždavinys sprendžiamas su CRS ir PSO algoritmais labai panašiai. Grafikai yra šalia vienas kito ir susikerta daugiau nei penkis kartus. Žiūrint vien tik į šiuos grafikus sudėtinga pasakyti, kuris algoritmas yra efektyvesnis.

13 lentelė. Griewank uždavinio hipertūriai.

Algoritmas	PSO	CRS
Hipertūris	0,324	0,329

Apskaičiavus hipertūrius (žr. 13 lentelę) matome, kad algoritmų: PSO ir CRS hipertūriai skiriasi labai nežymiai, nes algoritmų apskaičiuotų reikšmių skaičius su panašia paklaida yra beveik vienodas. Taigi matome, kad remiantis vien tik hipertūrio matu sudėtinga įvertinti kuris algoritmas efektyvesnis.

14 lentelė. Griewank uždavinį spręstų algoritmų tarpusavio dominuojamumo koeficientai.

A1 \ A2	PSO	CRS
PSO	0,000	0,185
CRS	0,027	0,000

Apskaičiavus algoritmų tarpusavio dominuojamumą (žr. 13 lentelę) matome, kad PSO algoritmas dominuojamas CRS algoritmo daugiau nei šešis kartus, lyginant kaip CRS algoritmas dominuojamas PSO algoritmo. Atsižvelgiant į hipertūrio matą CRS algoritmas yra efektyvesnis, tačiau hipertūris yra tik nežymiai didesnis. Tačiau atsižvelgiant į dominuojamumo koeficientus aiškiai matosi, kad CRS algoritmas yra efektyvesnis, nei PSO algoritmas sprendžiant Griewank uždavinį.

4. REZULTATAI IR IŠVADOS

Rezultatai

1. Išnagrinėnti aktualūs atsitiktinės paieškos globaliojo optimizavimo algoritmų efektyvumo vertinimo metodai.
2. Pasiūlytas hipertūrio skaičiavimo principas tikimybių pasiskirstymo funkcijos grafikams.
3. Pasiūlytas naujas algoritmų vertinimo metodas.
4. Sukurta taikomoji programa, realizuojanti siūlomą algoritmų vertinimo metodą, hipertūrio skaičiavimą ir kt.
5. Atliktas siūlomo hipertūrio principo taikymo algoritmų efektyvumo vertinimui validavimas.
6. Atliktas siūlomo metodo validavimas vertinant gerai žinomų atsitiktinės paieškos optimizavimo algoritmų efektyvumą sprendžiant įvairius globaliojo optimizavimo testo uždavinius.

Išvados

1. Pritaikius hipertūrio skaičiavimo principą tikimybių pasiskirstymo funkcijos grafikai gali būti išreiškiami skaitine reikšme ir tokiu būdu leidžia lengviau palyginti algoritmų efektyvumą.
2. Eksperimentinio tyrimo rezultatai parodė, kad pasiūlytas hipertūrio skaičiavimo principas atsižvelgia į algoritmo savybes rasti reikšmes su tam tikra paklaida ir tam tikra tikimybe.
3. Darbe siūlomas atsitiktinės paieškos algoritmų tarpusavio dominuojamumo vertinimo metodas, grįstas tikimybės rasti pageidaujamo tikslumo sprendinį vertinimu, padeda išskirti algoritmus dominuojančius pagal efektyvumą sprendžiant aktualų optimizavimo uždavinį.
4. Gauti eksperimentinio tyrimo rezultatai parodė, kad siūlomu metodu gautuose vertinimuose atsispindi įvairios algoritmų elgesio charakteristikos, tokios kaip gebėjimas rasti globalųjį sprendinį ar lokaliųjų sprendinių trauka.
5. Eksperimentinio tyrimo rezultatai parodė, kad siūlomas metodas tinkamas efektyviam skirtingų algoritmų efektyvumo palyginimui.

Literatūra

[AKZ05] M. M. Ali, C. Khompatraporn ir Z. B. Zabinsky. A Numerical Evaluation of Several Stochastic Algorithms on Selected Continuous Global Optimization Test Problems. *Journal of Global Optimization*, 31(4):635–672, 2005.

[Ash06] D. Ashlock. *Evolutionary Computation for Modeling and Optimization*. Springer Science & Business Media, 2006.

[BF03] Ş. İ. Birbil ir S. Fang. An Electromagnetism-like Mechanism for Global Optimization. *Journal of Global Ooptimization*, 25(3):263–282, 2003.

[BFM00a] T. Bäck, D. B. Fogel ir Z. Michalewicz. *Evolutionary Computation 1: Basic Algorithms and Operators*. Tom. 1. CRC Press, 2000.

[BFM00b] T. Bäck, D. B. Fogel ir Z. Michalewicz. Evolutionary computation 2-advanced algorithms and operations. *Taylor Francis*, 1, 2000.

[BFM97] T. Bäck, D. Fogel ir Z. Michalewicz. Handbook of Evolutionary Computation. *Release*, 97(1):B1, 1997.

[CDM+91] A. Colorni, M. Dorigo, V. Maniezzo ir k.t. Distributed Optimization by ant Colonies. *Proceedings of the First European Conference on Artificial Life*. Tom. 142. Paris, France, 1991, p. 134–142.

[CS00] R. Chelouah ir P. Siarry. A Continuous Genetic Algorithm Designed for the Global Optimization of Multimodal Functions. *Journal of Heuristics*, 6(2):191–213, 2000.

[ČM01] V. Čekanavičius ir G. Murauskas. Statistika ir jos taikymai. *Vilnius, TEV*, 239, 2001.

[DM02] E. D. Dolan ir J. J. Moré. Benchmarking Optimization Software With Performance Profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[Dor92] M. Dorigo. Optimization, learning and natural algorithms. *Ph. d. thesis, politecnico di milano, italy*, 1992.

[DS78] L. C. W. Dixon ir G. P. Szegö. *Towards Global Optimisation 2*. North-Holland Amsterdam, 1978.

[DŠT07] G. Dzemyda, V. Šaltenis ir V. Tiešis. Optimizavimo metodai. *Vilnius: mokslo aidai*, 2007.

[EK+95] R. C. Eberhart, J. Kennedy ir k.t. A New Optimizer Using Particle Swarm Theory. *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Tom. 1. New York, NY, 1995, p. 39–43.

[Gen06] J. E. Gentle. *Random Number Generation and Monte Carlo Methods*. Springer Science & Business Media, 2006.

[GLM12] M. Gaviano, D. Lera ir E. Mereu. A Parallel Algorithm for Global Optimization Problems in a Distribuited Computing Environment. *Applied Mathematics*, 3(10):1380, 2012.

[HAR+10] N. Hansen, A. Auger, R. Ros, S. Finck ir P. Pošík. Comparing Results of 31 Algorithms from the Black-box Optimization Benchmarking BBOB-2009. *Proceedings of the 12th annual conference companion on genetic and evolutionary computation*. GECCO '10. ACM, P, 2010, p. 1689–1696. ISBN: 978-1-4503-0073-5. DOI: 10.1145/1830761.1830790.

[HB+10] E. M. T. Hendrix, G. Boglárka ir k.t. *Introduction to Nonlinear and Global Optimization*. Springer New York, 2010.

[HJD+] R. Hill, B. Johansson, A. Dunkin, R. Ingalls, J. Hu ir P. Hu. On the Performance of the Cross-entropy Method.

[HL15] E. M. T. Hendrix ir A. Lančinskas. On Benchmarking Stochastic Global Optimization Algorithms. *Informatica*, 26(4):649–662, 2015.

[Hol75] J. H. Holland. *Adaptation in Natural and Artificial Systems: an Introductory Analysis With Applications To Biology, Control, and Artificial Intelligence.* U Michigan Press, 1975.

[HPV00] R. Horst, P. M. Pardalos ir N. Van Thoai. *Introduction to Global Optimization*. Springer Science & Business Media, 2000.

[JOG01] M. Jelasity, P. M. Ortigosa ir I. García. UEGO, an Abstract Clustering Technique for Multimodal Global Optimization. *Journal of Heuristics*, 7(3):215–233, 2001.

[Kal07] S Kalanta. Taikomosios optimizacijos pagrindai. vilnius: technika. 480 p. Tech. atask. ISBN 978-9955-28-160-3. doi: 10.3846/924-S, 2007.

[KE95] J. Kennedy ir R. Eberhart. Particle Swarm Optimization. *Neural Networks, 1995. Proceedings., IEEE International Conference on*. Tom. 4, 1995-11, 1942–1948 vol.4. DOI: 10.1109/ICNN.1995.488968.

[KE97] J. Kennedy ir R. C Eberhart. A Discrete Binary Version of the Particle Swarm Algorithm. *1997 ieee international conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*. Tom. 5. IEEE, 1997, p. 4104–4108.

[KS74] B. Kvedaras ir M. Sapagovas. Skaičiavimo metodai. *Vilnius:" Mintis*, 1974.

[KV+83] S. Kirkpatrick, M. P. Vecchi ir k.t. Optimization by Simmulated Annealing. *Science*, 220(4598):671–680, 1983.

[Lan13] A. Lančinskas. Atsitiktinės paieškos globaliojo optimizavimo algoritmų lygiagretinimas. Disertacija. Vilniaus universitetas, 2013.

[LOŽ13] A. Lančinskas, P. Ortigosa ir J. Žilinskas. Multi-objective Single Agent Stochastic Search in Non-dominated Sorting Genetic Algorithm. *Nonlinear Analysis: Modelling and Control*, 18(3):293–313, 2013.

[LOŽ14] A. Lančinskas, P. M.z Ortigosa ir J. Žilinskas. Parallel shared-memory multi-objective stochastic search for competitive facility location. *European Conference on Parallel Processing*. Springer, 2014, p. 71–82.

[LŽ12] A. Lančinskas ir J. Žilinskas. Solution of multi-objective competitive facility location problems using parallel nsga-ii on large scale computing systems. *International Workshop on Applied Parallel Computing*. Springer, 2012, p. 422–433.

[LŽ14] A. Lančinskas ir J. Žilinskas. *Parallel Multi-objective Memetic Algorithm for Competitive Facility Location. Parallel Processing and Applied Mathematics: 10th International Conference, PPAM 2013, Warsaw, Poland, September 8-11, 2013, Revised Selected Papers, Part ii*. R. Wyrzykowski, J. Dongarra, K. Karczewski ir J. Waśniewski, redaktoriai. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, p. 354–363. ISBN: 978-3-642-55195-6. DOI: 10.1007/978-3-642-55195-6_33.

[MBK11] D. Mačiūnas, R. Belevičius ir J. Kaunas. Daugiakriteris sijynų optimizavimas genetiniais algoritmais. *Science: future of lithuania*, 3(6), 2011.

[MF14] R. Misener ir C. A. Floudas. ANTIGONE: Algorithms for Continuous/Integer Global Optimization of Nonlinear Equations. *Journal of Global Optimization*, 59(2-3):503–526, 2014.

[Mit98] M. Mitchell. *An ntroduction to Genetic Algorithms*. MIT press, 1998.

[MS14] J. Müller ir C. A. Shoemaker. Influence of Ensemble Surrogate Mmodels and Sampling Strategy on the Solution Quality of Algorithms for Computationally Expensive Black-box Global Optimization Problems. *Journal of Global Optimization*, 60(2):123–144, 2014.

[Mur85] K. G. Murty. Linear programming, by. 1985.

[Neu06] A. Neumaier. Global Optimization and Constraint Satisfaction. *Proceedings of GICOLAG Workshop (of the Research Project Global Optimization, Integrating Convexity, Optimization, Logic Programming and Computational Algebraic Geometry)*, 2006.

[PGK+11] D. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim ir M. Zaidi. The Bees Algorithm–a Novel Tool for Complex Optimisation. *Intelligent Production Machines and Systems-2nd I* PROMS Virtual International Conference (3-14 July 2006)*, 2011.

[Pup09] L. Pupeikienė. Optimizavimo metodų tyrimas ir taikymas profiliuotų mokyklų tvarkaraščių sudarymo uždaviniuose. Disertacija. Vilniau Gedimino technikos universitetas, 2009.

[Ran10] G. P. Rangaiah. *Stochastic Global Ooptimization: Techniques and Applications in Chemical Engineering*. World Scientific, 2010.

[ROŽ12] J. L. Redondo, Pilar M. Ortigosa ir J. Žilinskas. Multimodal Evolutionary Algorithm for Multidimensional Scaling With City-block Distances. *Informatica*, 23(4):601–620, 2012.

[RS00] M. Recchioni ir A. Scoccia. A Stochastic Algorithm for Constrained Global Optimization. *Journal of Global Optimization*, 16(3):257–270, 2000.

[RS13] L. M. Rios ir N. V. Sahinidis. Derivative-free Optimization: a Review of Algorithms and Comparison of Software Implementations. *Journal of Global Optimization*, 56(3):1247–1293, 2013.

[SB15] S. Surjanovic ir D. Bingham. Virtual Library of Simulation Experiments: Test Function and Datasets, Optimization Test Problems. 2015. URL: `https://www.sfu.ca/~ssurjano/optimization.html` (žiūrėta 2017-05-01).

[Ste86] R. E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Aapplications*. Wiley, 1986.

[Šeš08] D. Šešok. Santvarų topologijos optimizavimas genetiniais algoritmais. Disertacija. Vilniaus Gedimino technikos universitetas, 2008.

[TŽ89] A. Torn ir A. Žilinskas. *Global Optimization*. Springer-Verlag New York, Inc., New York, NY, USA, 1989. ISBN: 0-387-50871-6.

[Vos99] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*. Tom. 12. MIT press, 1999.

[VV07] A. I. F. Vaz ir L. N. Vicente. A Particle Swarm Pattern Search Method for Bound Constrained Global Optimization. *Journal of Global Optimization*, 39(2):197–219, 2007.

[Wal99] J. P. Walser. *Integer Optimization by Local Search: a Domain-independent Approach*. Springer-Verlag, 1999.

[WM97] D. H. Wolpert ir W. G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Eevolutionary Computation*, 1(1):67–82, 1997.

[Zab09] Z. B. Zabinsky. Random Search Algorithms. *Wiley Encyclopedia of Operations Research and Management Science*, 2009.

[Zab13] Z. B. Zabinsky. *Stochastic Adaptive Search for Global Optimization*. Tom. 72. Springer Science & Business Media, 2013.

[Zab98] Z. B. Zabinsky. Stochastic methods for practical global optimization. *Journal of Global Optimization*, 13(4):433–444, 1998.

[ZDT00] E. Zitzler, K. Deb ir L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

[ZSM+93] Z. B. Zabinsky, R. L. Smith, J. F. McDonald, H. E. Romeijn ir D. E. Kaufman. Improving hit-and-run for global optimization. *Journal of global optimization*, 3(2):171–192, 1993.

[Žil05] A. Žilinskas. Matematinis programavimas. *Kaunas, vdu leidykla*, 2005.