VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

TEORINĖS MECHANIKOS KATEDRA

Valentina Gerfolveden

GA STRATEGIJŲ MECHANINIŲ STRUKTŪRŲ MASEI OPTIMIZUOTI TYRIMAS

INVESTIGATION OF GA STRATEGIES IN MASS OPTIMIZATION OF MECHANICAL STRUCTURES

Baigiamasis magistro darbas

Technomatematikos studijų programa, valstybinis kodas 621G16001

Matematikos studijų kryptis

Vilnius, 2013

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
TEORINĖS MECHANIKOS KATEDRA

TVIRTINU
Katedros vedėjas

_____
(Parašas)

_____
(Vardas, pavardė)

_____
(Data)

Valentina Gerfolveden

GA STRATEGIJŲ MECHANINIŲ STRUKTŪRŲ MASEI OPTIMIZUOTI TYRIMAS

INVESTIGATION OF GA STRATEGIES IN MASS OPTIMIZATION OF MECHANICAL STRUCTURES

Baigiamasis magistro darbas

Technomatematikos studijų programa, valstybinis kodas 621G16001
Matematikos studijų kryptis

Vadovas _____ _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Konsultantas _____ _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Konsultantas _____ _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Lietuvių kalbos konsultantas _____ _____ _____
(Moksl. laipsnis/pedag. vardas, vardas, pavardė) (Parašas) (Data)

Vilnius, 2013

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS
FUNDAMENTINIŲ MOKSLŲ FAKULTETAS
TEORINĖS MECHANIKOS KATEDRA

Fizinių mokslų sritis

Matematikos mokslo kryptis

Matematikos studijų kryptis

Technomatematikos studijų programa, valstybinis kodas 62401P109

TVIRTINU
Katedros vedėjas
(Parašas)
Dmitrij Šešok
(Vardas, pavardė)
2012 09 14
(Data)

BAIGIAMOJO MAGISTRO DARBO UŽDUOTIS
2012 Nr. 1
Vilnius

Studentei **Valentinai Gerfolveden**
(Vardas, pavardė)

Baigiamojo darbo tema:

Ga strategijų mechaninių struktūrų masei optimizuoti tyrimas

patvirtinta 2012 m. spalio 22 d. dekano potvarkiu Nr. 360 fm

Baigiamojo darbo užbaigimo terminas 2013 m. birželio 10 d.

BAIGIAMOJO DARBO UŽDUOTIS:

Apžvelgti šiuolaikinius optimizavimo algoritmus, parodyti genetinių algoritmų vietą jų tarpe. Apžvelgti mechaninių struktūrų skaičiavimo metodus. Parašyti programas pasirinkta programavimo kalba tiesioginiam uždaviniui – plokščiojo kūno, plokščiojo įtempimų būvio statikos uždaviniui – spręsti, bei genetiniam algoritmui realizuoti. Pritaikyti genetinį algoritmą kūno masės minimizavimo uždaviniui.Verifikuoti programas. Ištirti kelias genetinio algoritmo strategijas (elitinę atranką, kryžminimo būdus, mutacijos būdus) siekiant nustatyti palankiausias duotajam uždaviniui spręsti.

Baigiamojo darbo rengimo konsultantai:
Doc. D. Šešok
(Moksl. laipsnis, vardas, pavardė)

Vadovas ..... habil. dr. Rimantas Belevičius
(Parašas) (Moksl. laipsnis, vardas, pavardė)

Užduotį gavau
.....
(Parašas)
Valentina Gerfolveden
(Vardas, pavardė)
2012 09 14
(Data)

Vilniaus Gedimino technikos universiteto egzaminų, sesijų ir baigiamųjų darbų rengimo bei gynimo organizavimo tvarkos aprašo 2011-2012 m. m.
1 priedas

(Baigiamojo darbo sąžiningumo deklaracijos forma)

VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

Valentina Gerfolveden, 20073492
(Studento vardas ir pavardė, studento pažymėjimo Nr.)

Fundamentinių mokslų fakultetas
(Fakultetas)

Technomatematika, TMfm-11
(Studijų programa, akademinė grupė)

BAIGIAMOJO DARBO (PROJEKTO) SĄŽININGUMO DEKLARACIJA

2013 m. birželio 6 d.

Patvirtinu, kad mano baigiamasis darbas tema „GA strategijų mechaninių struktūrų masei optimizuoti tyrimas" patvirtintas 2012 m. spalio 22 d. dekano potvarkiu Nr. 360fm, yra savarankiškai parašytas. Šiame darbe pateikta medžiaga nėra plagijuota. Tiesiogiai ar netiesiogiai panaudotos kitų šaltinių citatos pažymėtos literatūros nuorodose.

Parenkant ir įvertinant medžiagą bei rengiant baigiamąjį darbą, mane konsultavo mokslininkai ir specialistai: doc. dr. Dmitrij Šešok. Mano darbo vadovas prof. habil. dr. Rimantas Belevičius.

Kitų asmenų indėlio į parengtą baigiamąjį darbą nėra. Jokių įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs (-usi).

(Parašas)

Valentina Gerfolveden
(Vardas ir pavardė)

Vilniaus Gedimino technikos universitetas
Fundamentinių mokslų fakultetas
Teorinės mechanikos katedra

ISBN ISSN
Egz. sk. .....
Data .....-.....-.....

Antrosios pakopos studijų **Technomatematikos** programos baigiamasis darbas

Pavadinimas **GA strategijų mechaninių struktūrų masei optimizuoti tyrimas**

Autorius **Valentina Gerfolveden**

Vadovas **prof. habil. dr. Rimantas Belevičius**

Kalba: lietuvių

Anotacija

Šio baigiamojo magistro darbo tikslas yra nustatyti genetinių algoritmų (GA) strategiją, kuri leistų efektyviausiai atlikti mechaninės struktūros masės minimizavimą. Minimizuojama tam tikros formos plokščiojo deformuojamo kūno masė, esant žinomai apkrovai ir žinomoms kraštinėms sąlygoms. Uždaviniui teikiami stiprumo ir geometriniai apribojimai. Aptariami kontinuumo diskretizavimo metodai bei globaliojo optimizavimo metodai. Tiesioginiam uždaviniui spręsti pasirinktas baigtinių elementų metodas, optimizavimo – genetiniai algoritmai. Baigiamajame darbe tiriama kaip optimizavimo rezultatai priklauso nuo GA atrankos metodų, kryžminimo metodų ir konkrečių GA parametrų: populiacijos dydžio, mutacijos tikimybės, generacijų skaičiaus. Pateikiami išanalizuoti ir apibendrinti eksperimentų rezultatai. Darbo užduotis realizuota C++ kalba. Darbą sudaro 7 dalys: įvadas, diskretizavimo ir optimizavimo metodų apžvalga, uždavinio formulavimas, programos algoritmo schema, tyrimai ir eksperimentai, išvados, literatūros sąrašas.

Darbo apimtis – 69 p. teksto be priedų, 47 iliustr., 22 lent., 20 bibliografinių šaltinių.

Atskirai pridedami darbo priedai.

Prasminiai žodžiai: Genetiniai algoritmai, genetinių algoritmų strategijos, masės minimizavimas, baigtinių elementų metodas.

Vilnius Gediminas Technical University
Faculty of Fundamental Sciences
Department of Engineering Mechanics

ISBN ISSN
Copies No. .....
Date .....-.....-.....

Master Degree Studies **Technomathematics** study programme Master Thesis

Title **Investigation of GA Strategies in Mass Optimization of Mechanical Structures**

Author **Valentina Gerfolveden**

Academic supervisor **Prof Dr Habil Rimantas Belevičius**

Thesis language: Lithuanian

Annotation

The goal of master thesis is to define the effective Genetic Algorithm (GA) strategy for mass minimization of mechanical structures. The mass of plane body of given shape, loading, and boundary conditions is minimized under the strength and geometrical constraints. The methods of global optimization and discretization of continual structures are discussed. For the solution of direct problem the finite element method is chosen, while for minimization problem the genetic algorithms are employed. In this thesis, we investigate the dependence of optimization results on the methods of GA selection, GA crossover, and specific GA parameters: population size, mutation probability, and the number of generations. The summarized results of numerical experiments are presented. Coding is in the syntax of C++ language.

Thesis consists of 7 chapters: introduction, overview of the optimization and discretization methods used, problem formulation, algorithm of program, investigations and experiments, conclusion, and literature, totally of 69 pages of plain text without appendixes, 47 pictures, 22 tables, 20 bibliographical entries.

Appendixes are included.

Keywords: Genetic algorithms, genetic algorithms strategies, mass minimization, finite element method.

Turinys

PAVEIKSLŲ SĄRAŠAS 9
LENTELIŲ SĄRAŠAS 11
ĮVADAS 12
1. DISKRETIZAVIMO IR OPTIMIZAVIMO METODŲ APŽVALGA 13
1.1. Kontinuumo diskretizavimo metodai 13
1.1.1. Baigtinių skirtumų metodas 14
1.1.2. Kraštinių elementų metodas 14
1.1.3. Baigtinių elementų metodas 15
1.2. Globaliojo optimizavimo metodai 16
1.2.1. Genetiniai algoritmai 16
1.2.1.1. Chromosomos kodavimas 17
1.2.1.2. Tinkamumo funkcija 19
1.2.1.3. Atrankos metodai 20
1.2.1.4. Kryžminimo metodai 24
1.2.1.5. Mutacija 27
1.3. Pirmojo skyriaus išvados 28
2. UŽDAVINIO FORMULAVIMAS 28
2.1. Uždavinio sprendimas baigtinių elementų metodu 29
2.1.1. Baigtinių elementų metodas. Elemento formos funkcijos 29
2.1.2. Baigtinio elemento CST priklausomybės 31
2.2. Uždavinio sprendimas genetiniais algoritmais 37
2.2.1. Chromosomos kodavimas 37
2.2.2. Atrankos metodai 39
2.2.3. Kryžminimo metodai 39
2.2.4. Mutacija 40
2.2.5. Algoritmo nutraukimas 40
3. PROGRAMOS ALGORITMO SCHEMA 41
4. TYRIMAI IR EKSPERIMENTAI 42
4.1. GA populiacijos dydžio tyrimas 43
4.2. GA elitinės strategijos tyrimas 47
4.3. GA kryžminimo strategijų tyrimas 57
4.4. GA iteracijų skaičiaus tyrimas 58
4.5. GA mutacijos strategijų tyrimas 61

4.6. Eksperimentų apibendrinimas..... 66
IŠVADOS..... 68
LITERATŪROS SĄRAŠAS..... 69
PRIEDAI..... 71
A priedas. Programos kodas..... 71

Paveikslų sąrašas

1.1 pav. Dvejetainis chromosomos kodavimas.....18
1.2 pav. Perstatymus koduojanti chromosoma..... 19
1.3 pav. Tiesioginio kodavimo chromosomos..... 19
1.4 pav. Ruletės atrankos metodas.....21
1.5 pav. Rangų atrankos metodas..... 22
1.6 pav. Turnyrinės atrankos metodas..... 23
1.7 pav. Vieno taško kryžminimas..... 25
1.8 pav. Dviejų taškų kryžminimas..... 26
1.9 pav. Kelių taškų kryžminimas..... 26
1.10 pav. Pastovus kryžminimas..... 26
1.11 pav. Pjaustymas..... 27
1.12 pav. Vieno bito mutacija..... 27
2.1 pav. Strypo skaičiuojamasis modelis..... 29
2.2 pav. Trikampis elementas..... 31
2.3 pav. Strypas padalintas į 20 CST elementų..... 32
2.4 pav. Trikampė plokštelė..... 34
2.5 pav. Ansamblio matricos surinkimas..... 35
2.6 pav. Baigtinių elementų mazgų apribojimas..... 36
2.7 pav. Elipsės pusašės..... 36
2.8 pav. GA schema..... 37
2.9 pav. Sprendžiamo uždavinio chromosoma..... 38
2.10 pav. Pradinių koordinačių chromosoma..... 38
3.1 pav. Sukurtos programos veikimo algoritmas..... 41
4.1 pav. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio..... 45
4.2 pav. Standartinio nuokrypio priklausomybė nuo populiacijos dydžio..... 45
4.3 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio..... 47
4.4 pav. Vidurkių priklausomybė nuo populiacijos dydžio..... 49
4.5 pav. Minimumų priklausomybė nuo populiacijos dydžio..... 49
4.6 pav. Maksimumų priklausomybė nuo populiacijos dydžio..... 50
4.7 pav. Vidurkio standartinių nuokrypių priklausomybė nuo populiacijos dydžio..... 50
4.8 pav. Minimumo standartinių nuokrypių priklausomybė nuo populiacijos dydžio..... 51
4.9 pav. Maksimumo standartinių nuokrypių priklausomybė nuo populiacijos dydžio..... 51
4.10 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio..... 53

4.11 pav. Vidurkių ir minimumų priklausomybė nuo elitinių individų skaičiaus..... 54
4.12 pav. Maksimumų priklausomybė nuo elitinių individų skaičiaus..... 55
4.13 pav. Standartinio nuokrypio priklausomybė nuo elitinių individų skaičiaus..... 55
4.14 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo elitinių individų skaičiaus..... 56
4.15 pav. Ruletės atranka. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus.... 59
4.16 pav. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus..... 60
4.17 pav. Elito atranka. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus..... 60
4.18 pav. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus..... 61
4.19 pav. Tikslo funkcijos reikšmių priklausomybė nuo mutacijos tikimybės..... 63
4.20 pav. Standartinio nuokrypio priklausomybė nuo mutacijos tikimybės..... 63
4.21 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo mutacijos tikimybės..... 64
4.22 pav. Standartinio nuokrypio priklausomybė nuo δ..... 66
4.23 pav. Mažiausios masės strypas..... 67
4.24 pav. Artimiausias vidurkiui sprendinys..... 67

Lentelių sąrašas

2.1 lentelė. Pradinės mazgų koordinatės..... 38
4.1 lentelė. Pirmo tyrimo GA parametrai..... 43
4.2 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio..... 44
4.3 lentelė. Pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio..... 46
4.4 lentelė. Antro tyrimo GA parametrai..... 48
4.5 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio..... 48
4.6 lentelė. Pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio..... 52
4.7 lentelė. Trečio tyrimo GA parametrai..... 53
4.8 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo elitinių individų skaičiaus..... 53
4.9 lentelė. Pasikliautinųjų intervalų priklausomybė nuo elitinių individų skaičiaus..... 56
4.10 lentelė. Ketvirto tyrimo GA parametrai..... 57
4.11 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo kryžminimo taškų skaičiaus..... 58
4.12 lentelė. Standartinio nuokrypio priklausomybė nuo kryžminimo taškų skaičiaus..... 58
4.13 lentelė. Šešto tyrimo GA parametrai..... 62
4.14 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo mutacijos tikimybės..... 62
4.15 lentelė. Standartinio nuokrypio priklausomybė nuo mutacijos tikimybės..... 63
4.16 lentelė. Pasikliautinųjų intervalų priklausomybė nuo mutacijos tikimybės..... 64
4.17 lentelė. Septinto tyrimo GA parametrai..... 65
4.18 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo δ..... 65
4.19 lentelė. Standartinio nuokrypio priklausomybė nuo δ..... 65
4.20 lentelė. Pasikliautinųjų intervalų priklausomybė nuo δ..... 66
4.21 lentelė. Optimalūs GA parametrai..... 67

Įvadas

Optimizavimo uždaviniai aktualūs įvairiose žmonių veiklos srityse. Medikai stengiasi nustatyti mažiausias vaistų dozes, kuriomis pasiektų didžiausią gydomąjį poveikį. Telekomunikacijų specialistai norėtų užtikrinti saugų signalo siuntimą kanalais į pačius tolimiausius pasaulio kampelius su kuo mažesne klaidos tikimybe. Įmonių vadovai siekia organizuoti gamybą, kad mažiausiomis sąnaudomis būtų gaunamas didžiausias pelnas. Visuotinio transporto tinklo strategai nori žinoti, kaip sudaryti autobusų, traukinių tvarkaraščius, kad keleivių vežimas būtų kuo efektyvesnis. Kasdienėje veikloje dažnai optimizuojama intuityviai nesinaudojant jokia teorija. Tačiau svarbiais ir sudėtingais atvejais stengiamasi uždavinį suformuluoti tiksliai ir jį spręsti tam skirtais metodais.

Optimizavimas taip pat yra neatskiriama inžinerinės veiklos dalis. Inžinieriams dažnai tenka priimti įvairius sprendimus: kaip suprojektuoti statybinę konstrukciją, kad ji būtų stipri, standi ir kiek galima pigesnė; kaip racionaliai sukirpti medžiagą, kad būtų kuo mažiau atliekų; kaip pervežti produkciją, kad išlaidos būtų minimalios ir kt. Optimizavimo reikšmė technikoje ypač išaugo pradėjus vartoti kompiuterines projektavimo sistemas. Kadangi optimizavimo uždaviniai yra svarbūs ir teoriniu, ir taikomuoju požiūriais yra sukurta daugybė optimizavimo metodų bei juos realizuojančios programinės įrangos.

Tyrimo tikslas

Tyrimo tikslas yra nustatyti genetinių algoritmų (toliau – GA) strategiją, kuri leistų efektyviausiai atlikti mechaninės struktūros masės optimizavimą.

Darbo uždaviniai

Tyrimo tikslui pasiekti reikia išspręsti šiuos uždavinius:

1. Susipažinti su GA strategijomis, kurios leistų atlikti mechaninės struktūros masės optimizavimą.
2. Susipažinti su mechaninių struktūrų diskretizavimo metodais.
3. Sukurti pasiūlytų technologijų įgyvendinimo algoritmus ir eksperimentiškai patikrinti jų galimybes.
4. Išbandyti apytikslį mechaninių struktūrų analizės metodą tikslo funkcijai ir apribojimams skaičiuoti.
5. Ištirti, kaip optimizavimo rezultatas priklauso nuo GA atrankos metodų.

6. Ištirti, kaip optimizavimo rezultatas priklauso nuo GA kryžminimo metodų.
7. Ištirti, kaip optimizavimo rezultatas priklauso nuo konkrečių GA parametrų: populiacijos dydžio, mutacijos tikimybės, generacijų skaičiaus.

Šiame darbe minimizuosime tam tikros formos plokščiojo deformuojamo kūno masę, esant žinomai apkrovai ir žinomoms kraštinėms sąlygoms. Iš literatūros žinome, kad tokie formos optimizavimo uždaviniai yra daugiaekstremiai, todėl globaliojo tikslo funkcijos ekstremumo paieškai taikysime globaliojo optimizavimo metodus. Be to, uždavinyje, kurį spręsime, projektavimo kintamųjų daugiau nei dešimt, todėl daugiaekstremį uždavinį turėsime spręsti stochastiniais (tikimybiniais) optimizavimo metodais. Deterministiniai algoritmai tokios apimties uždaviniui reikalaus nerealių kompiuterio resursų.

Tikslo funkcija yra kūno masė, kurią siekiame minimizuoti, nepažeizdami žinomo maksimalaus įtempio ir pusiausvyros apribojimų. Tikslo funkcijos reikšmei nustatyti ir apribojimams tikrinti pasirenkame baigtinių elementų metodą (toliau – BEM). Šis metodas yra plačiai naudojamas pasaulinėje praktikoje.

1. Diskretizavimo ir optimizavimo metodų apžvalga

1.1. Kontinuumo diskretizavimo metodai

Tamprumo teorijos uždavinių matematiniai modeliai yra aprašomi diferencialinėmis lygtimis ir formuluojami kaip kraštiniai uždaviniai. Analizinis tokių uždavinių sprendimas yra sudėtingas, o tikslus sprendinys yra gaunamas tiesiogiai integruojant lygtis. Tikslus sprendinys yra universalus, tačiau jis gaunamas, paprastai, tik taisyklingos formos sritims ir esant paprastoms kraštinėms sąlygoms.

Kompiuterinės technologijos bei skaitiniai matematikos metodai orientuoti į apytikslį mechanikos uždavinių sprendimą. Siekiant uždavinį išspręsti skaitiniais metodais, funkcinis matematinis modelis turi būti transformuotas į algebrinį uždavinį.

Kraštinio uždavinio transformacija į algebrinį pavidalą yra tapatinama su kontinualaus modelio diskretizavimu. Kontinuumu plačiąja prasme yra vadinama materiali sistema, nenutrūkstamai pasiskirsčiusi erdvėje. Būdvardis diskretusis šiuo atveju apibūdina struktūras (objektus ar reiškinius), kurios aprašomos baigtiniu skaičiumi kintamųjų ar kitų rodiklių. Tokios struktūros yra sudarytos iš įvairių smulkesnių sudėtinių detalių – elementų. Tokių elementų sudarymas, aprašymas ir panaudojimas sudėtingesnėms struktūroms modeliuoti yra natūralus būdas, kuriuo ilgus metus naudojosi ir naudojasi daugelio sričių inžinieriai. Kontinuumo sutapatinimas su diskrečia struktūra yra matematinės abstrakcijos rezultatas [5].

Diskretizavimo metodas yra būdas, kuriuo pradinė struktūra paverčiama diskrečiąja struktūra. Diskretizavimo būdai naudoja įvairias prielaidas ir hipotezes, kuriomis identifikuojami kontinualiosios struktūros būseną aprašantys kintamieji ir kiti rodikliai. Vieni iš metodų remiasi žmogaus intuicija, kiti pagrįsti griežtais matematiniais teiginiais. Aptarkime kelis diskretizavimo metodus.

1.1.1. Baigtinių skirtumų metodas

Baigtinių skirtumų metodas (finite difference method) sukurtas 1920 m. netiesinėms hidrodinamikos lygtims spręsti, tuomet buvo vadinamas kvadratų metodu. Nuo tada šis metodas taikomas įvairių sričių problemoms spręsti.

Baigtinių skirtumų metodas (toliau – BSM) pagristas aproksimacija, t.y. dalinių išvestinių lygčių pakeitimu dalinių skirtumų lygtimis [16]. Ši BSM aproksimacija yra algebrinės formos ir ieškomo kintamojo vertė sprendimo srityje nustatoma iš greta esančių mazgų verčių. Sprendžiant uždavinius baigtinių skirtumų metodu, atliekami šie etapai:

1. Nagrinėjama sritis padengiama tinkleliu, dažniausiai reguliarios formos.
2. Diferencialinė lygtis išreiškiama ekvivalenčia baigtinių skirtumų lygtimi, kurioje nagrinėjamo mazgo vertė priklauso nuo gretimų mazgų verčių skirtumų.
3. Sudarytų skirtumų lygčių sistema sprendžiama atsižvelgus į tam tikras analizuojamos sistemos ribines ir (arba) pradines sąlygas.

Tai palyginti nesudėtingas ir be ypatingų problemų programiškai įgyvendinamas metodas. Tačiau BSM išvestines skaičiuoja skaitiniais metodais, todėl jis yra jautrus dideliems gradientams ir trūkioms funkcijoms. Be to, jį sunkiau pritaikyti netaisyklingos formos sritims.

1.1.2. Kraštinių elementų metodas

Kontinualūs objektai gali būti aprašyti ne tik diferencialinėmis lygtimis, bet ir integralinėmis lygtimis. Jei uždavinį galima aprašyti integraline kontūrine lygtimi, tai tokį uždavinį galima spręsti kraštinių elementų metodu (toliau – KEM). Sudalijus kontūrą į kraštinius elementus ir aprašius funkcijų pasiskirstymą išilgai kontūro per formos funkcijos integralines lygtis galima užrašyti algebrinių lygčių sistemą. Funkcijų reikšmės srities viduje randamos analizuojant. Vienas iš principinių trūkumų – metodas reikalauja gilių matematinių žinių. Uždavinio sprendimas KEM pasunkėja, kai terpė nevienalytė. Tokiu atveju reikia aprašyti ir terpių sandūrą. Gerai tinka sritims, kur nedaug kontūrų. Taip pat nesunkiai integruojamas su BEM.

1.1.3. Baigtinių elementų metodas

Tarp šiuo metu naudojamų skaitinių diskretizavimo metodų ryškiai dominuoja baigtinių elementų metodas. Šis metodas sujungia daugelį savybių, būdingų matematiškai pagrįstiems skaitiniams diferencialinių lygčių sprendimo metodams ir žmogaus intuicija grindžiamiems inžineriniams metodams.

Baigtinių elementų metodo raidos istorija prasideda šeštajame dešimtmetyje. Vieną iš pirmų straipsnių 1956 m. su bendraautoriais parašė Turneris, o besibaigiant septintajam dešimtmečiui pasirodė pirmosios knygos, skirtos BEM. Didelę įtaką baigtinių elementų metodo raidai turėjo O. C. Zienkiewicziaus darbai [18].

Sprendžiant BEM, kūnas modeliuojamas baigtinių elementų rinkiniu. Kiekviename elemente, dažniausiai jo išorėje, pažymimi tam tikri taškai, vadinami mazgais. Mazgai yra skirti elemento geometrijai, būsenos kintamiesiems ar kitiems rodikliams apibrėžti. Daroma prielaida, kad baigtiniai elementai tarpusavyje jungiami tik elemento mazgais. Inžineriniu požiūriu baigtinis elementas suprantamas kaip realiai egzistuojantis tam tikros formos ir baigtinių matmenų elementas (geometrinė figūra arba idealizuotas realios konstrukcijos elementas). Matematiniu požiūriu tas pats elementas suvokiamas kaip matematinės abstrakcijos rezultatas. Rinkinio sudarymo procedūros nepriklauso nuo požiūrio ir nuo konkretaus elemento sudarymo būdo.

Metodo esmė: perėjimas nuo begalinio mazgų laisvumo laipsnių kiekio prie baigtinio, padalinant sritis (uždavinio bei kraštinių sąlygų apibrėžimo sritis) į baigtinius elementus ir parenkant kiekvienam iš jų a priori nežinomos funkcijos kitimo dėsnį priklausomai nuo kol kas dar nežinomų tos funkcijos mazginių verčių.

Skaičiavimai BEM taikomi šiais etapais:

- bet kuri sudėtingos geometrinės formos sritis pavaizduojama baigtiniu nesudėtingos formos elementų skaičiumi;
- kiekvieno elemento srityje diferencialinė lygtis dalinėmis išvestinėmis apytiksliai pakeičiama algebrinių lygčių sistema;
- elementų algebrinės lygtys jungiamos į bendrą vienareikšmiškai išspendžiamą algebrinių lygčių sistemą [1].

Baigtinių elementų metodo pagrindu yra sukurta daug universalių bei specializuotų kompiuterių programų. Šiame darbe autorė naudosis jos originalia, baigtinių elementų metodą CST (Constant Strain Triangle) elementams realizuojančia programa, kuri buvo sukurta programavimo kalba C++.

1.2. Globaliojo optimizavimo metodai

Šiame darbe tikslo funkcija imama kūno masė ir sprendžiamas minimizavimo uždavinys. Lokaliojo minimumo taškas apibrėžiamas kaip taškas, kuriame tikslo funkcijos reikšmė ne didesnė už reikšmes bet kuriame gretimame taške. Jei toks taškas leistinojoje srityje tik vienas, optimizavimo uždavinys palyginti paprastai sprendžiamas lokaliaisiais metodais.

Praktikoje susiduriama su globaliojo optimizavimo uždaviniais, kai lokaliųjų minimumų yra ne vienas, o jų skaičius gali būti didžiulis – visus juos peržiūrėti praktiškai neįmanoma. Tačiau atsiradusią daugelio ekstremumų problemą turėsime išspręsti, teks ieškoti mažiausio iš daugelio lokaliųjų minimumų.

Globalieji optimizavimo metodai dažniausiai būna apytiksliai. Literatūroje gausu įvairiausių globaliojo optimizavimo metodų, idėjų, tyrimų. Daugelis siūlomų metodų grindžiami autorių intuicija ir išradingumu, taip pat uždavinių sprendimo patirtimi. Paprastai pasiūlomas intuicija pagrįstas algoritmas ir jo efektyvumas palyginamas su kitų algoritmų efektyvumu. Tokie metodai vadinami euristiniais [5].

Atsižvelgiant į sprendimo schemą, algoritmai būna deterministiniai arba stochastiniai. Įvairiems optimizavimo uždaviniams spręsti plačiai taikomi stochastiniai algoritmai. Jų pritaikomumą nulemia paprastumas ir universalumas. Veiksmų seka, kurią reikia atlikti norint pasiekti rezultatą, tokių algoritmų yra tikimybinė, jų eiliškumas priklauso nuo atsitiktinumo. Kiekvieną kartą, spręsdami uždavinį su tais pačiais parametrais, gauname skirtingus (atsitiktinius) rezultatus, todėl būtina daug kartų leisti uždavinį.

Stochastinių algoritmų klasė yra pakankamai plati, apimanti daug žinomų atsitiktinės paieškos algoritmų, aptarkime plačiai taikomus GA, kurie bus naudojami šiame darbe.

1.2.1. Genetiniai algoritmai

Genetiniai algoritmai ir jų sudarymo principas buvo pasiūlyti XX-ojo amžiaus 70-aisiais metais Johno Hollando (John Holland). Pradedant Hollando pirmuoju darbu, GA buvo sėkmingai išbandyti, sprendžiant įvairius optimizavimo uždavinius.

Šie algoritmai yra kryptingos atsitiktinės paieškos metodai. Esminis genetinių algoritmų skirtumas nuo kitų yra tas, kad čia operuojama ne su vienu sprendiniu, o su sprendinių populiacija. GA mėgdžioja biologinę evoliuciją, natūraliąją atranką ir principą „išlieka tik labiausiai prisitaikantys“. Pagrindinės sąvokos, kurios naudojamos modeliuojant biologinės evoliucijos procesus, yra individas (kitaip dar vadinamas chromosoma) ir populiacija. Individas yra tam tikras elementarus, daugiau neskaidomas vienetas, objektas. Didesnė ar mažesnė

individų grupė sudaro populiaciją. Nauja populiacija iš turimos gaunama pritaikius genetines operacijas – kryžminimą, mutaciją, atranką. Taip gaunamos naujos sprendinių populiacijos su geresnėmis savybėmis, kurios turi įtakos laipsniškam artėjimui globaliojo optimumo link. GA paprastai būdingas labai greitas konvergavimas pradiniame sprendinio sprendimo etape ir lėtas nemonotoninis konvergavimas į lokalųjį sprendinį po tam tikro generacijų skaičiaus [6; 8].

Yra didžiulė šių metodų įvairovė, naudojamos įvairios kryžminimo, mutacijos ir kandidatų atmetimo taisyklės. Pavyzdžiui, mes galime operuoti su viena didele sprendinių populiacija, tačiau galime turėti ir kelias mažesnes (lygiagrečias) populiacijas. Galime pasirinkti „tėvus“ (prieš tai buvusios populiacijos individus), atsižvelgdami į jų vertingumą (tinkamumą), t.y. absoliučią tikslo funkcijos reikšmę – arba į kokius nors kitus įvertinius, charakteristikas, kriterijus. Galima labai įvairiai realizuoti kryžminimo bei mutavimo procedūras. Dar galime pakeisti visus be išimties einamosios populiacijos narius (sprendinius) naujai gautais individais po kiekvieno algoritmo ciklo iteracijos įvykdymo. Kitu atveju galime pakeisti tik jų dalį paliekant individus su kuo mažesne (ar didesne) – nelygu, koks optimizavimo uždavinys (minimizavimo ar maksimizavimo) sprendžiamas – tikslo funkcijos reikšme. Dar reikia pastebėti, kad genetiniai algoritmai yra stochastiniai algoritmai, todėl algoritmą su tais pačiais parametrais būtina vykdyti keletą kartų. Aptarkime plačiau kelias GA strategijas.

1.2.1.1. Chromosomos kodavimas

Genetikai organizmus apibūdina dviem terminais – genotipu ir fenotipu. Genotipas - tai organizmo genetinės informacijos (genų) visuma. Fenotipas – visų organizmo požymių visuma, t.y. kuo objektas yra realiame pasaulyje. Fenotipas yra realizuota, įgyvendinta genetinė informacija. Tokiu būdu kiekvienas genas, t. y. genotipo informacijos elementas, turi savo atvaizdą fenotipe [2]. Ta pati terminologija yra naudojama genetiniuose algoritmuose. GA populiacija sudaroma iš aibės chromosomų, kurios dar vadinamos individais arba populiacijos nariais. Chromosomos sudarytos iš elementų, vadinamų genais. Tariame, kad potencialus uždavinio sprendinys gali buti atstovaujamas parametrų rinkinio. Šie parametrai, vadinami genais, yra sujungiami kartu ir tokiu būdu gaunama tam tikra reikšmių seka arba kitaip chromosoma, genotipas. Kiekvieną fenotipo požymį atitinka vienas genas objekto genotipe (pavyzdį žr. 2.2.1 poskyryje). Fenotipo pavertimas į genotipą yra vadinamas užkodavimu, o genotipo atvertimas į fenotipą vadinamas atkodavimu.

Prieš pradedant spręsti uždavinį GA, pirmiausia iškyla klausimas, kaip reprezentuoti sprendinius chromosomomis ir kokiu būdu jas užkoduoti. Yra daug chromosomų kodavimo būdų ir visi jie parenkami pagal tam tikrus veiksnius. Pagrindinis jų – algoritmo taikymo sritis.

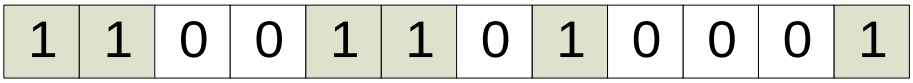
Pasirinkta duomenų struktūra turėtų gerai apibūdinti uždavinio sprendinį. Taip pat geresnis kodavimas yra toks, kur mažesni chromosomos pokyčiai reiškia mažesnį tinkamumo funkcijos pokytį ir atvirkščiai.

Koduoti galima tiesioginiu arba netiesioginiu būdu. Tiesioginis būdas, tai toks būdas, kai individo genotipas ir fenotipas sutampa. Netiesioginis būdas, tai toks būdas, kai yra naudojamos kokios nors užkodavimo ar atkodavimo funkcijos genotipui ir fenotipui tarpusavyje keisti.

Apžvelkime kelis kodavimo metodus.

Dvejetainis kodavimas

Tradiciškai genetinių algoritmų individams aprašyti naudojamos dvejetainės išraiškos, sudarytos iš 0 ir 1. Dvejetainis kodavimas dažniausiai naudojamas ir yra populiarus dėl savo paprastumo ir galimybės užkoduoti praktiškai bet ką. Taip pat reikalauja mažai atminties ir yra nesudėtinga apibrėžti bendrus kryžminimo ir mutacijos operatorius. Be to, jis buvo naudotas ir pačiuose pirmuose genetinių algoritmų bandymuose. Šis kodavimo būdas pavaizduotas 1.1 paveiksle.



1.1 pav. Dvejetainis chromosomos kodavimas

Tačiau toks kodavimo būdas turi trūkumų. Pagrindinis dvejetainio kodavimo trūkumas – mažam pokyčiui fenotipe yra reikalingas sąlyginai žymus pokytis genotipe. Be to, tokia duomenų struktūra nėra natūrali daugelio uždavinių sprendiniams koduoti ir neretai reikalaujanti pataisymų [12; 15; 19].

Grėjaus (Gray) kodavimas

Grėjaus kodavimas dažnai naudojamas norint sveikuosius skaičius išreikšti bitų eilute. Šis kodavimo būdas pasižymi specifiniu dvejetainio kodo išdėstymu. Grėjaus kodavimas padeda išvengti aukščiau aprašytos, mažų pokyčių problemos, nes gretimi skaičiai skiriasi mažesniu pozicijų skaičiumi – vieno bito reikšme.

Perstatymų kodavimas

Perstatymų kodavimas gali būti naudojamas rikiavimo uždaviniuose. Pavyzdžiui, keliaujančio pirklio uždavinio sprendiniams koduoti. Tokiame kodavime chromosoma yra numerių eilutė (1.2 pav.), kuri nusako poziciją sekoje. Kaip ir dvejetainiame kodavime taip ir čia kartais reikia pataisymų po kryžminimo ar mutacijos operacijų, kad nauji sprendiniai liktų prasmingi [13].

8	2	3	0	7	5	1	9	11	4	6	10

1.2 pav. Perstatymus koduojanti chromosoma

Tiesioginis kodavimas

Kintamųjų reikšmės saugomos genetiniame vektoriuje gali būti tiesiog realieji skaičiai, t.y. kiekvienas genas atspindi tikrąją reikšmę. Taip pat chromosomos eilutė gali būti sudaryta iš bet kokių reikšmių (1.3 pav.), nuo skaičių iki įvairių objektų [12; 13].

2,5	0,5	-1,1	0,3	2,1	0,1

3u	9a	7b	1s	4a	3c	1f	9k

kairėn	dešinėn	žemyn

1.3 pav. Tiesioginio kodavimo chromosomos

Be šių kodavimų egzistuoja ir sudėtingesnių, pavyzdžiui, medžio metodas. Jis dažniausiai naudojamas koduoti programos ar algoritmo struktūrai. Tokiame metode chromosoma gali reprezentuoti medį sudarytą iš programavimo kalbos funkcijų ar komandų [12], [13].

1.2.1.2. Tinkamumo funkcija

Prieš pradedant spręsti uždavinį taip pat reikalinga tinkamumo (tikslo) funkcija, kuria galime įvertinti tam tikrų individų (sprendinių) gerumą. Be to, galime palyginti du individus tarpusavyje. Individo tinkamumo funkcijos reikšmė priklauso nuo jo genų rinkinio, t.y. tikslo funkcija yra apskaičiuojama pagal chromosomos genų požymius. Pagal apskaičiuotas tikslo

funkcijos reikšmes generuojamos sekančios kartos, t.y. atrenkami individai palikuonių generavimui. Priklausomai nuo uždavinio, tinkamumo funkcija gali būti realizuota kaip matematinė funkcija ar eksperimentas su individu.

1.2.1.3. Atrankos metodai

Atrankos (angl. selection) metodo tikslas yra pagal tam tikrus kriterijus atrinkti reikiamą skaičių individų (sprendinių) iš dabartinės populiacijos ir sudaryti naują individų aibę tolimesniems algoritmo žingsniams. Atrankos kriterijų aprašo tikslo funkcija, kuri ir lemia atranką, t.y. vieno individo pranašumas lyginant su kitu individu nustatomas pagal jų tikslo funkcijos vertes. Tinkamiausieji individai turi daugiausiai galimybių išlikti ir susilaukti palikuonių. Tokiu būdu bandoma atkartoti stipriausių ir labiausiai prisitaikiusių individų išlikimą gamtoje.

Dauguma atrankos funkcijų yra tikimybinės ir sukurtos taip, kad atrinktų tinkamiausius ir nedidelę dalį sprendinių, kurie yra mažiau tinkami. Tokiu būdu užtikrinama, kad išliktų įvairovė ir išvengiama nukrypimo į netinkamus sprendinius. Paprastai būtent atrankos metodai įtakoja populiacijos kokybę ir konvergavimo greitį.

Dažniausi ir daugiausiai išstudijuoti atrankos metodai yra ruletės ir turnyrinė atranka. Taip pat dažnai naudojama ir elito atranka, kurios metu keliems tinkamiausiems individams leidžiama iš karto pereiti į naująją populiaciją. Žemiau aprašysime šias populiarias bei mažiau naudojamas atrankos strategijas.

Ruletės atranka

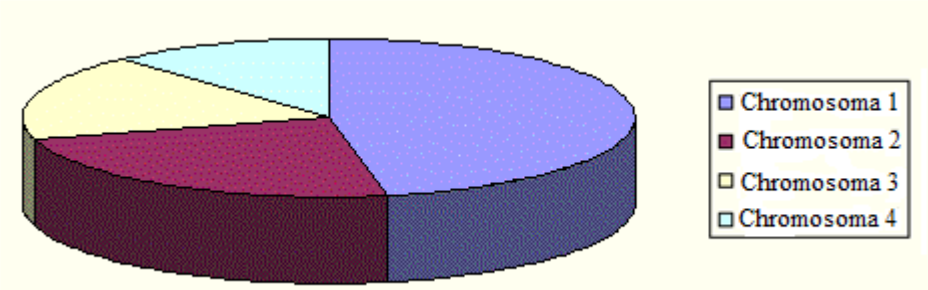
Ruletės atranka (angl. roulette wheel selection) – istoriškai pirmasis aprašytas atrankos metodas genetiniams algoritmams, kurį pasiūlė J. Hollandas kartu su pirmuoju genetiniu algoritmu [12].

Ruletės metode kiekvienas individas yra renkamas pagal jo tinkamumo reikšmę – kuo didesnis tinkamumas, tuo didesnė tikimybė. Tikimybė, kad individas bus atrinktas į kitą populiaciją, yra proporcinga tą individą atitinkančios tikslo funkcijos reikšmės ir suminės visų populiacijos individų tikslo funkcijų reikšmės santykiui [20].

$$P(i) = \frac{f(i)}{\sum_{i=1}^{N} f(i)},$$

čia $f(i)$ – i-tojo individo tikslo funkcijos reikšmė.

Paprastai tariant, šis metodas gali būti atvaizduotas kaip ruletė – kiekvienas individas gauna ruletės rato dalį. Kuo tinkamesnis individas, tuo didesnę dalį gauna (1.4 pav.).



1.4 pav. Ruletės atrankos metodas

Ruletė yra „išsukama“ N kartų, kur N – individų skaičius populiacijoje. Pozicija ties kuria ruletė sustoja atitiks vieną individą. Toks algoritmas galėtų būti realizuotas taip:

1. Susumuojame visų individų tinkamumo reikšmes, gauname – S,

$$S = \sum_{i=1}^{N} f(i).$$

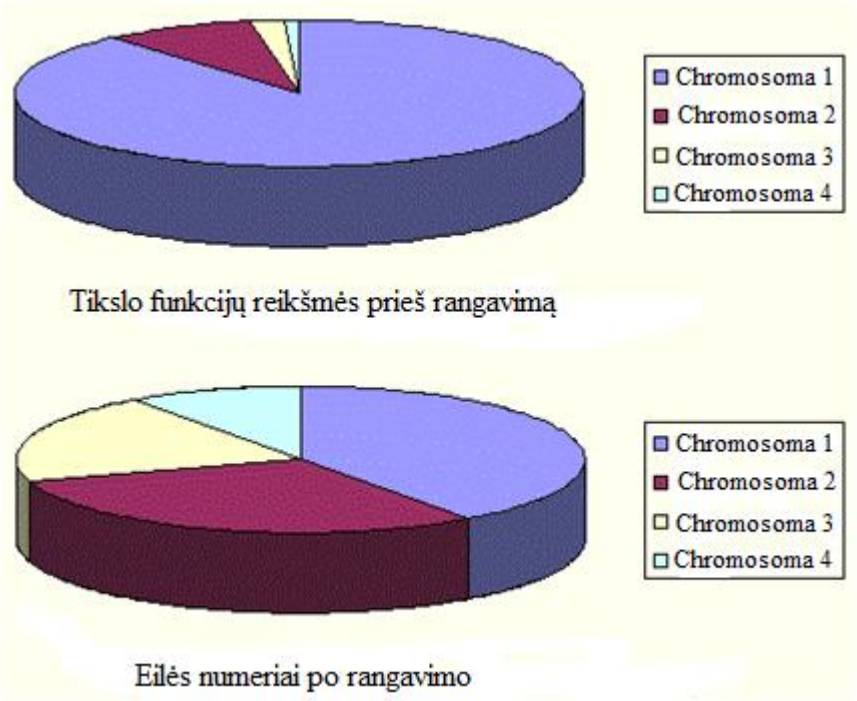
2. Sugeneruojame atsitiktinį skaičių – r, iš intervalo (0; S).
3. Pereiname per populiaciją sumuodami tikslo funkcijų reikšmes f_i. Jei S > r, grąžiname paskutinį pridėtą individą.
4. Grįžtame į 2-ą žingsnį. Kartojame N kartų [12].

Deja toks algoritmas turės problemų kai vieno ar kelių individų tinkamumo reikšmės sudarys didelę dalį visos ruletės. Tokiu atveju mažo tinkamumo sprendiniai turės labai mažai šansų būti pasirinktais. Kita problema – artėjant link algoritmo pabaigos individai supanašėja. Tokiu atveju visos ruletės dalis bus beveik vienodos. Yra aprašyta keletas būdų spręsti šioms problemoms. Aptarkime pirmos problemos sprendimui dažniausiai naudojamą rangų metodą.

Rangų atranka

Ruletės atranka nebus efektyvi jei, pavyzdžiui, vieno individo tikslo funkcijos reikšmė sudarys 90 % visos ruletės. Šiam reiškiniui ištaisyti yra naudojamas rangų metodas (angl. rank selection). Jo metu kiekvienam individui yra priskiriamas eilės numeris nuo 1 iki N, pradedant

nuo blogiausią tinkamumą turinčių sprendinių ir užbaigiant geriausiu, t.y. blogiausio individo numeris – 1, o geriausio – N. Tačiau kadangi tikslo funkcijos reikšmės čia neturi tokio didelio skirtumo, tai gali atsiliepti konvergavimo greičiui [12; 13]. Rangų atrankos metodas pavaizduotas 1.5 paveiksle.

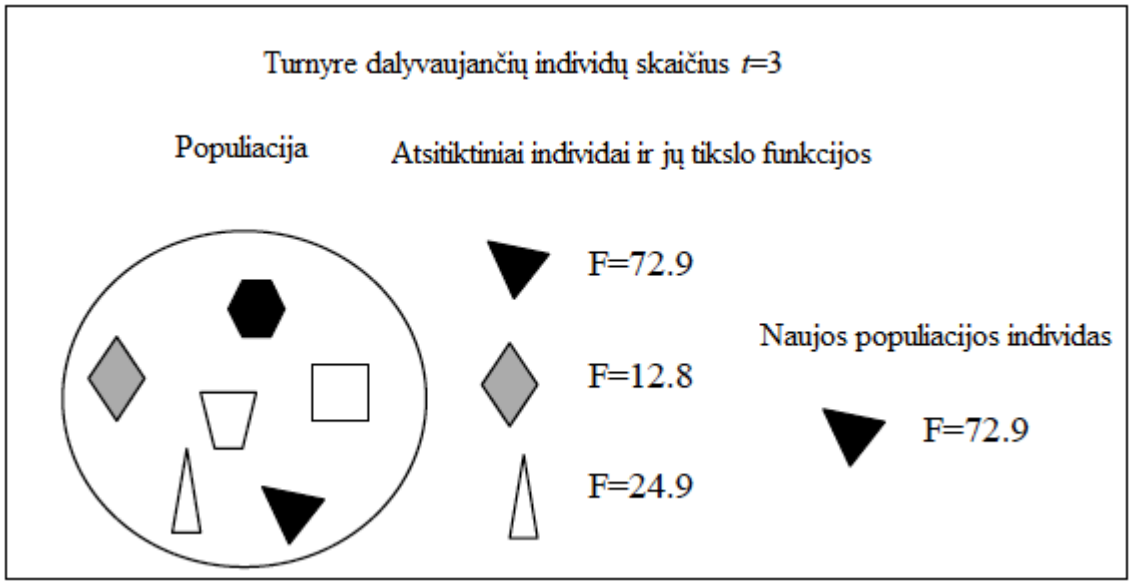


1.5 pav. Rangų atrankos metodas

Turnyrinė atranka

Turnyrinė atranka (angl. tournament selection) išsprendžia antrą ruletės metodo problemą. Artėjant link algoritmo pabaigos, mažėja skirtumai tarp individų tikslo funkcijų, t.y. dauguma individų tampa pakankamai geri. Tokiu atveju ruletės atrankos efektyvumas itin sumažėja, nes ji nejautriai reaguoja į smulkius skirtumus, metodas panašėja į atsitiktinę atranką. Turnyrinio atrankos metodo idėja yra paimti kažkiek atsitiktinių individų iš populiacijos ir atrinkti tą, kurio tinkamumas yra geriausias šiame pogrupyje. Tai garantuoja, kad geresni individai tėvais taps dažniau nei kiti. Tačiau yra tikimybė, kad bus prarasti tokie individai kurie turėdami gerą tinkamumo funkcijos reikšmę bus atmetami, kadangi tik vienas individas iš pogrupio išrenkamas tolesniam dauginimuisi, o visi kiti bus nugalėti geriausio individo [9; 15; 20].

Turnyrinė atranka dažniausiai yra naudojama lyginant du individus. Paveiksle 1.6 pavaizduota turnyrinė atranka iš 3-jų individų.



1.6 pav. Turnyrinės atrankos metodas

Elito atranka

Visuose prieš tai aprašytuose atrankos metoduose atrinktų į naują populiaciją individų kiekis atitinka pradinių individų kiekį. Vėliau pritaikius kitas GA operacijas nauji suformuoti individai pakeičia visą prieš tai buvusią populiaciją. Šie metodai neturi galimybės išsaugoti geriausių individų ir tokiu atveju yra tikimybė, kad bus prarasti geri sprendiniai. Elito atranka (angl. elitist selection) yra vienas iš šios problemos galimų sprendimų.

Elito atrankos metodo idėja yra perkelti vieną ar kelis sėkmingiausius individus (sprendinius) į kitą populiaciją be jokių pakeitimų. Kita labai panaši metodika išlaikyti seną populiaciją yra tiesiog pakeičiant blogesnes tikslo funkcijas turinčius individus naujai sukurtais, tokiu būdu išsaugant geriausius sprendinius. Tokios atrankos privalumas prieš paprastą tikimybinę atranką yra tai, kad geriausias populiacijos individas monotoniškai mažėja kiekvienos iteracijos metu. Tačiau šis atrankos metodas kaip ir visi algoritmai turi ir trūkumų.

Elito atrankos metode susiduriama su dviem pagrindinėmis problemomis. Pirmoji problema – konvergavimas į lokalųjį minimumą. Geriausias individas parenkamas labai dažnai, taigi visa populiacija linkusi konverguoti į šią chromosomą. Dėl šios priežasties stabdoma genetinio algoritmo pažanga. Antroji problema – individų supanašėjimas. Progresuojant GA sprendinių tinkamumo reikšmių skirtumai sumažėja, taigi tikimybė pasirinkti geriausią tampa

panaši į kitų populiacijos narių pasirinkimo tikimybę [7; 13]. Šiai problemai išspręsti naudojamas aukščiau aprašytas turnyrinės atrankos metodas.

Pastovios būsenos atranka

Visuose prieš tai aptartuose atrankos algoritmuose visi arba beveik visi individai yra keičiami naujai suformuotais kiekvienoje generacijoje. Pastovios būsenos atrankos metode (angl. steady-state selection) kiekvienoje kartoje pakeičiama tik maža populiacijos dalis. Paprastai pakeičiami tik keli individai turintis blogiausias tikslo funkcijos reikšmes. Pastovios būsenos atranka dažniausiai naudojama mokinančių klasifikatorių sistemose (angl. learning classifier systems). Sprendžiant tokius uždavinius svarbus ne pavieniai individai o visos populiacijos tinkamumas [11; 12].

1.2.1.4. Kryžminimo metodai

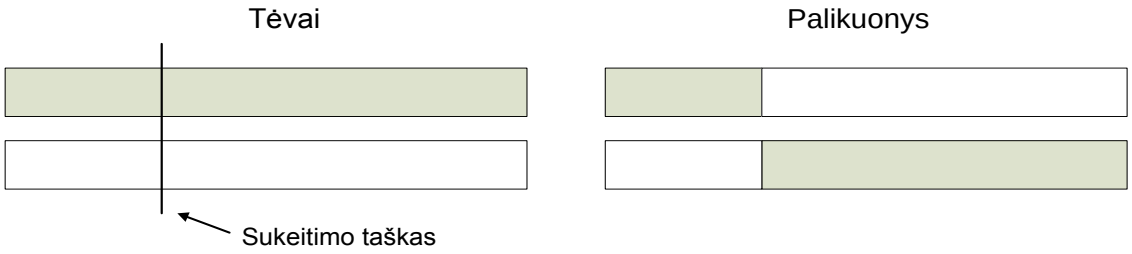
Pagal evoliucijos teoriją, būdai, kuriais tėvų požymiai perduodami palikuonims, atlieka labai svarbų vaidmenį populiacijos raidoje. GA už požymių perdavimą atsakingas metodas vadinamas kryžminimu (angl. crossover). Šis metodas apibrėžia, kaip chromosomų tėvų sandara perduodama palikuonims. Jungiant genus iš dviejų individų tikimasi, kad rezultatas bus tik geresnis individas. Žinoma, taip įvyksta ne visada, bet bendru atveju šiuo metodu gaunamos sėkmingos individų kartos [19; 20].

Kiekvieno naujo sprendinio sukūrimui iš populiacijos imama pora individų tėvų, kurie sukryžminami ir gaunamas naujas individas, vaikas. Tėvai paprastai imami iš individų atrinktų atrankos metodu. Toliau imama kita pora ir vėl sukuriamas vaikas, ir taip tęsiama kol pasiekiamas tam tikras naujos kartos individų skaičius, sukuriama pilna nauja sprendinių populiacija. Kryžminimas įvyksta su iš anksto nustatyta tikimybe, t.y. kai kurios individų poros gali būti nekryžminamos. Galiausiai po kryžminimo naują populiaciją sudarys individai, kurių chromosomų sandara yra pasikeitusi. Bendras populiacijos tikslo funkcijų reikšmių vidurkis taip pat pakis [4].

Genetinio algoritmo kryžminimo metodo pasirinkimas ir realizavimas priklauso nuo sprendžiamo uždavinio bei naudojamo chromosomų kodavimo būdo. Fiksuoto ilgio vektoriumi koduojamų chromosomų kryžminimui dažnai naudojami vieno taško, kelių taškų ir pastovus kryžminimas. Aptarkime keletas kryžminimo variantų.

Vieno taško kryžminimas

Vienas paprasčiausių kryžminimo metodų yra vieno taško kryžminimas (angl. single point crossover). Jo metu atsitiktinai nustatomas keitimo taškas, tada naujai chromosomai sudaryti dalis iki to taško yra paimama iš vieno tėvo, o likusi dalis iš kito (1.7 pav.) [12].



1.7 pav. Vieno taško kryžminimas

Dviejų taškų kryžminimas

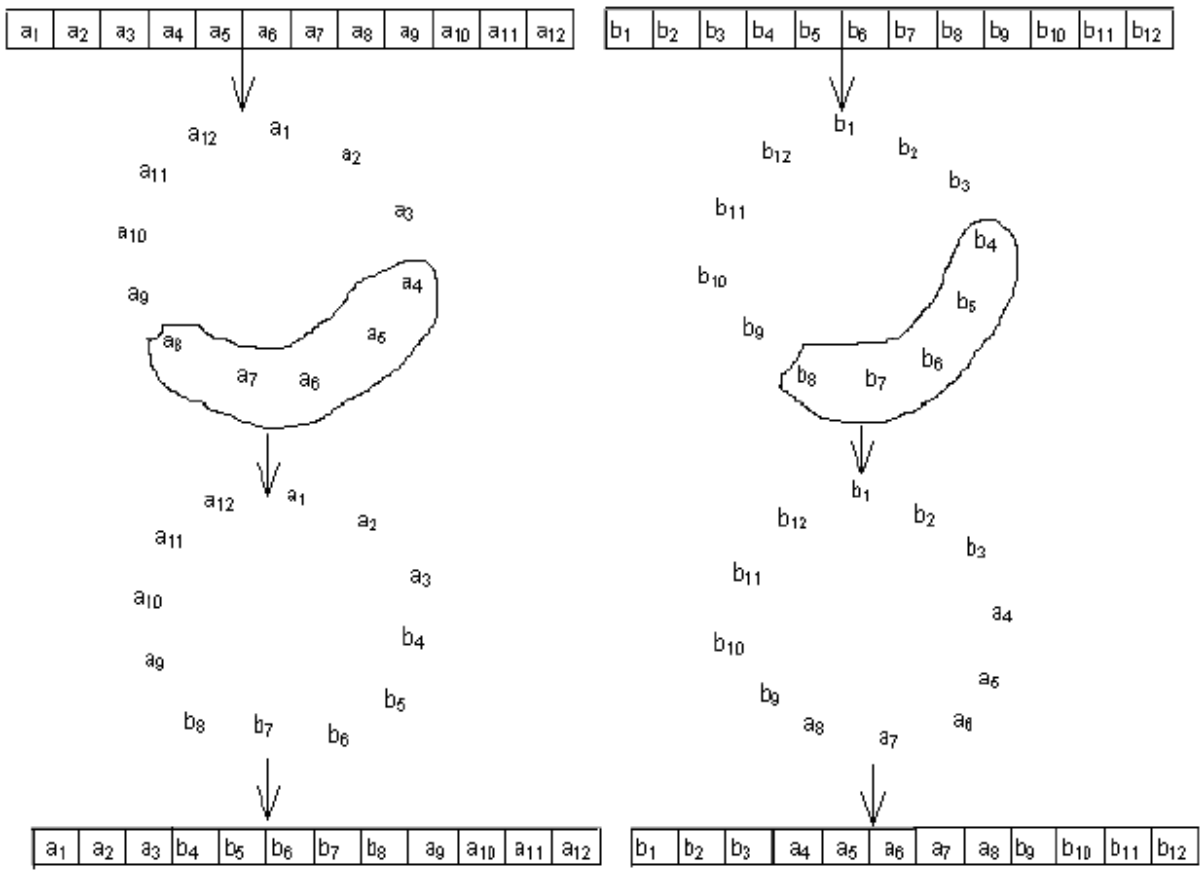
Dviejų taškų kryžminimo metodas (angl. two-point crossover) veikia panašiai kaip ir vieno taško. Čia tėvinėse chromosomose parenkami du atsitiktiniai taškai ir chromosoma sudalinama į tris dalis. Duomenys, esantys tarp šių atsitiktinai pasirinktų taškų sukeičiami vietomis. Rezultatas – palikuonys, kurie turi dalį vieno ir kito tėvo duomenų [3; 20]. Dviejų taškų kryžminimas pavaizduotas 1.8 paveiksle.

Kelių taškų kryžminimas

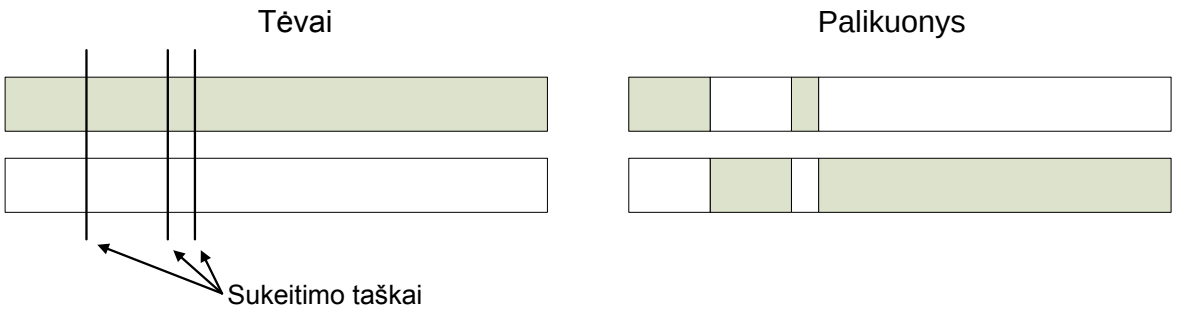
Kelių taškų kryžminimas (angl. multi-point crossover) veikia analogiškai kaip ir dviejų taškų kryžminimas, tik atsitiktinai parenkama daugiau kryžminimo taškų. Atsitiktinai sugeneruoti taškai negali kartotis [20]. Kelių (trijų) taškų kryžminimas pavaizduotas 1.9 paveiksle.

Pastovus kryžminimas

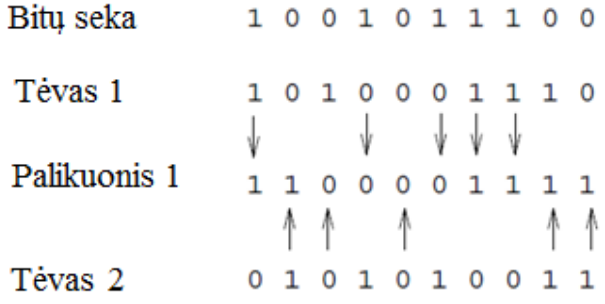
Pastovus kryžminimas (angl. uniform crossover) primena kelių taškų kryžminimą, tik čia paprastai keičiamos ne chromosomų dalis, bet atskiri genai. Šiame metode pirmiausia atsitiktinai sugeneruojamos vienetų ir nulių sekos (angl. crossover mask), t.y. sekos iš 1 ir 0, kurių ilgis atitinka chromosomų ilgį. Toliau pagal šias sekas generuojami palikuonys. 1 reiškia, kad genas bus imamas iš pirmo tėvo, 0 – iš antro [3; 15]. Žemiau esančiame 1.10 paveiksle pavaizduotas pastovaus kryžminimo metodas.



1.8 pav. Dviejų taškų kryžminimas



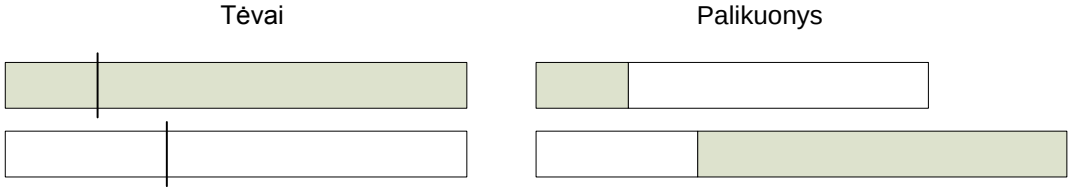
1.9 pav. Kelių taškų kryžminimas



1.10 pav. Pastovus kryžminimas

Pjaustymas

Pjaustymas (angl. cut and splice crossover) – kryžminimo technika, kai keičiamas palikuonių genų skaičius. Skirtumas nuo ankščiau minėtų technikų yra tas, kad čia, tėvų duomenyse, pasirenkami taškai skirtinguose vietose [6]. Šis kryžminimas pavaizduotas 1.11 paveiksle.



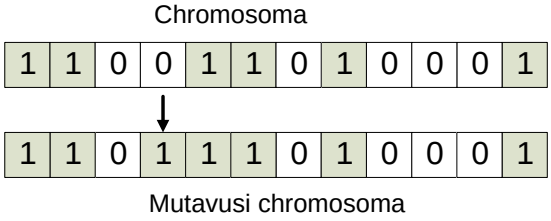
1.11 pav. Pjaustymas

1.2.1.5. Mutacija

Mutacijos (angl. mutation) paskirtis genetiniuose algoritmuose yra išlaikyti genetinę įvairovę, t.y. išvengti populiacijos chromosomų supanašėjimo, apsaugoti sprendinių populiaciją nuo lokalaus konvergavimo, nes tai vestų į evoliucijos sulėtėjimą ar net visišką stagnaciją. Šiam tikslui pasiekti daromi smulkūs, atsitiktiniai pakeitimai individuose tikintis, kad kai kurie pakeitimai turės teigiamos įtakos individo tinkamumui ir tas individas įneš naujovių į populiaciją [10; 20]. Mutacijos veikimas, kaip ir kryžminimo atveju, priklauso nuo chromosomos kodavimo būdo.

Klasikinės mutacijos veikimą sudaro atsitiktinai generuojami skaičiai, kurie nurodo įvyks ar neįvyks mutacija tam tikrai duomenų daliai. Mutacija dažniausiai vykdoma su nedidele tikimybe. Tokiu atveju mutuoti gali kiekvienas chromosomos genas. Jeigu mutacijos tikimybė didesnė, tada mutuoja, paprastai, tik vienas chromosomos genas.

Bitų vektoriumi koduojamoje chromosomoje kiekvienas bitas su iš anksto pasirinkta tikimybe yra keičiamas į priešingą, t.y. iš 1 į 0 arba iš 0 į 1. Bitų mutacija pavaizduota žemiau esančiame 1.12 paveiksle.



1.12 pav. Vieno bito mutacija

Tiesioginiame kodavimo metode mutacija priklauso nuo naudojamų reikšmių. Naudojant, pavyzdžiui, realius skaičius mutacija gali būti tiesiog nedidelio, atsitiktinio, realaus skaičiaus pridėjimas ar atėmimas. Perstatymų kodavimo metode gali būti tiesiog sukeičiami du genai vietomis.

Pradinėje genetinio algoritmo idėjoje kryžminimas yra laikomas pagrindine paieškos operacija, o mutacija nustumiama į šalį, kaip pagalbinis veiksnys. Tačiau kiek vėliau susidomėjimas mutacija atsirado. Šiuo metu yra dvi vyraujančios nuomonės dėl to kas svarbiau mutacija ar kryžminimas. Vieni teigia, kad kryžminimas svarbiausias, o mutacija tik užtikrinanti, kad nebūtų prarastas sprendimo potencialas. Kiti teigia, kad kryžminimas reikalingas tik tam, kad paskleistų naujoves sukurtas mutacijų. Taip pat manoma, kad kryžminimas tai didelės tikimybės mutacija [3; 15].

1.3. Pirmojo skyriaus išvados

Paprastai globalaus optimizavimo uždaviniai reikalauja didžiulių kompiuterio išteklių, todėl perspektyviai atrodo stochastiniai (tikimybiniai) globalaus optimizavimo metodai [17]. Galima daryti prielaidą, kad tarp jų genetiniai algoritmai, modeliuojantys gamtos evoliucijos dėsnius, gali būti pranašesnį už kitus algoritmus. Aišku, GA nėra idealūs algoritmai visiems optimizavimo uždaviniams ir turi savo trūkumų. Praktika rodo, kad GA negali garantuoti globalaus sprendinio radimo per fiksuotą iteracijų skaičių. Be to, dažniausiai GA teikia tik apytikslį uždavinio sprendinį. Tačiau šis trūkumas nėra labai svarbus praktiniams uždaviniams, kuriuose teorinis globalusis sprendinys dažniausiai nėra būtinas. Taigi iš globalaus optimizavimo algoritmų pagrindinis dėmesys bus skiriamas GA strategijoms. Pirmiausia tirsime tikslo funkcijos reikšmių priklausomybę nuo populiacijos dydžio. Tikslo funkcijos reikšmei apskaičiuoti bei uždavinio apribojimams patikrinti taikysime BEM. Tai labiausiai paplitęs apytikslis skaičiavimo metodas. Išbandysime dvi atrankos strategijas: ruletės ir elito. Turėdami optimaliausią populiacijos dydį tirsime tikslo funkcijos reikšmių priklausomybę nuo elitinių individų skaičiaus. Taip pat ištirsime kelias kryžminimo ir mutacijos strategijas. Parodysime, kad uždavinys konverguoja.

2. Uždavinio formulavimas

Šio darbo tikslas yra nustatyti GA strategiją, kuri leistų efektyviausiai atlikti mechaninės struktūros masės optimizavimą. Šiam tikslui pasiekti spręsime paprastą, tipinį uždavinį.

Minimizuosime tam tikros formos plokščiojo deformuojamo kūno masę, esant žinomai apkrovai ir žinomoms kraštinėms sąlygoms (apribojimus žr. 2.1.2 poskyryje *apribojimai* punkte).

$$\min_{X \in A} M(X); \tag{1}$$

čia $M(X)$ – strypo masė kilogramais;

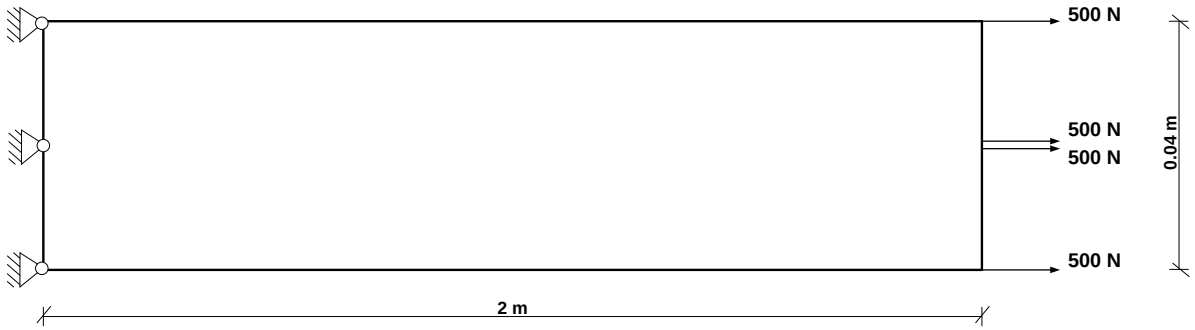
X – projektavimo kintamieji (mazgų koordinatės *x* ir *y* metrais);

A – galimos mazgų koordinatės *x* ir *y* metrais.

Kaip tipinį uždavinį pasirinkome strypą. Jo optimalią formą galima numatyti intuityviai, todėl uždavinys gali būti naudingas kaip siūlomų algoritmų bei pagal juos parašytų programų validavimo priemonė.

Minimizuosime, keičiant kai kurių mazgų koordinates, plieninio strypo masę. Jo ilgis 2 m, aukštis 0,04 m ir storis 0,02 m. Medžiagos tankis yra $7800\ \mathrm{kg/m^3}$, tamprumo modulis – 210 GPa, Puasono koeficientas – 0,3. Strypas yra įtvirtintas nepaslankiais šarnyrais trijuose mazguose. Išskirstyta apkrova yra pridėta trijuose mazguose ir sudaro 2 kN. Maksimalūs įtempimai elementuose neturi viršyti $3\ \mathrm{N/mm^2}$. Aprašytas strypas pavaizduotas 2.1 paveiksle.

Prieš sprendžiant strypo optimizavimo uždavinį reikia mokėti apskaičiuoti pradines to strypo charakteristikas, surasti tikslo funkcijos reikšmę ir patikrinti optimizavimo uždavinio apribojimus. Kaip jau minėjome, šiam tikslui pasirinkome baigtinių elementų metodą. Šiuo metodu galėsime rasti strypo įtempimus, mazgų poslinkius bei apskaičiuoti tikslo funkcijos reikšmes.



2.1 pav. Strypo skaičiuojamasis modelis

2.1. Uždavinio sprendimas baigtinių elementų metodu

2.1.1. Baigtinių elementų metodas. Elemento formos funkcijos

Baigtinių elementų metodas įkūnija matematikų gerai ištirtų pusiau analizinių metodų idėją. Apytiksliam kraštinio uždavinio sprendimui pasirenkama ieškomosios funkcijos $f(x)$

išraiška taikant vieną iš matematikoje jau žinomų funkcijų, dažniausiai daugianaris, turinti baigtinį narių skaičių:

$$f(x) \approx \alpha_0 + \alpha_1 f_1(x) + \alpha_2 f_2(x) + \cdots + \alpha_i f_i(x) + \cdots + \alpha_n f_n(x); \quad (2)$$

čia α_i - nežinomieji koeficientai;

$f_i(x)$ - pasirinktos funkcijos.

Ši išraiška pasirenkama taip, kad visada tapatingai tenkintų kraštines sąlygas. Taigi funkcijos $f\ (x)$ radimas pakeičiamas koeficientų $\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_i, \dots, \alpha_n$, kuriems apskaičiuoti visada gali būti sudaryta *n+1* skaičius algebrinių lygčių, nustatymu. O tai reiškia, kad diferencialinės lygties sprendimas pakeičiamas tiesinių lygčių sistemos sprendimu.

Esminis skirtumas tarp klasikinių pusiau analizinių metodų ir BEM glūdi funkcijos apibrėžimo srityje – tolydi išraiška (2) keičiama analogiška išraiška, galiojančia tik vieno elemento ribose:

$$f(x) \approx u^e(x) = \alpha_0^e + \alpha_1^e {f_1}^e(x) + \alpha_2^e {f_2}^e(x) + \cdots + \alpha_i^e {f_i}^e(x) + \cdots + \alpha_n^e {f_n}^e(x). \quad (3)$$

Taikant BEM, kraštinio uždavinio sprendinys užsiduodamas (aproksimuojamas) diskretiškai tolydžiomis funkcijomis. Tolydumas elemente apsprendžiamas funkcijų ${f_i}^e(x)$ savybėmis, o trūkius tarp elementų tenka nagrinėti papildomai, laikantis tam tikrų reikalavimų.

Išraiška (3) nėra patogi baigtiniams elementams sudaryti. Baigtinių elementų metodo praktikoje įprasta ne apskaičiuoti nežinomus koeficientus, o nustatyti interpoliacines išraiškas, į kurias įeina funkcijų ar jų išvestinių mazguose reikšmės. Atmetus išvestines, pasirenkama tokia apytikslė nagrinėjamos funkcijos išraiška:

$$u^e(x) = N_1^e(x)U_1^e + N_2^e(x)U_2^e + \cdots + N_i^e(x)U_i^e + \cdots + N_n^e(x)U_n^e. \quad (4)$$

Funkcijų $N_1^e(x)$ šeima vadinama elemento formos funkcijomis. Formos funkcijos užima išskirtinę vietą BEM teorijoje ir praktikoje. Jos turi užtikrinti išraiškos (4) logiką elemento mazguose, t.y. $u^e(x_i) = U_i^e$. Tai reškia, kad formos funkcijos pasižymi specialiomis savybėmis:

$$N_i^e(x_i) = 1; \quad (5\text{ a})$$

$$N_i^e(x_j) = 0. \quad (5\text{ b})$$

Savybės (5) yra panaudojamos formos funkcijų $N_i^e(x)$ šeimai sudaryti, kai $i = 1, 2, \dots, n$.

Apibendrindami visas išraiškas baigtiniam elementui *e*, užrašysime jas matricine forma:

$$u^e(x) = [N(x)]^e U^e; \quad (6)$$

čia $[N(x)]^e$ - elemento formos funkcijų matrica;

U^e - elemento mazginių nežinomųjų vektorius.

Užsidavus (6) sudaromas elemento *e* modelis:

$$[K]^e U^e = F^e. \tag{7}$$

Standartine surinkimo procedūra gaunamas modelis:

$$[K] \cdot \{U\} = \{F\}; \tag{8}$$

čia K – koeficientų matrica;

U – nežinomųjų vektorius;

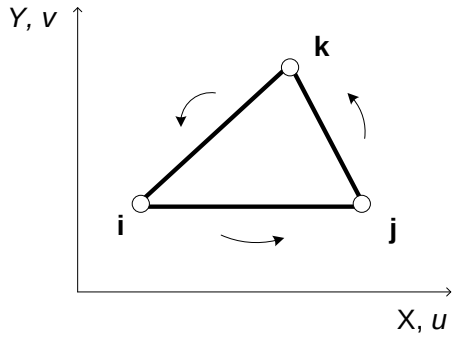
F – laisvųjų narių mazguose pridėtų poveikių vektorius.

Tokiu būdu kuriant baigtinio elemento modelį visos dimensijos sutelkiamos į vieną elementą, kurio geometrija, fizinės savybės ir išraiška (6) nulemia jo galutinį modelį [1].

Mūsų modelis (8) bus apytikslis, o jam sudaryti taikysime kontinualiųjų struktūrų diskretizavimo techniką, elementų formos funkcijas ir su jomis susijusias priklausomybes (2-7).

2.1.2. Baigtinio elemento CST priklausomybės

Suformuluotam uždaviniui spręsti BEM naudosime trikampį baigtinį elementą (kitaip CST) (2.2 pav.).

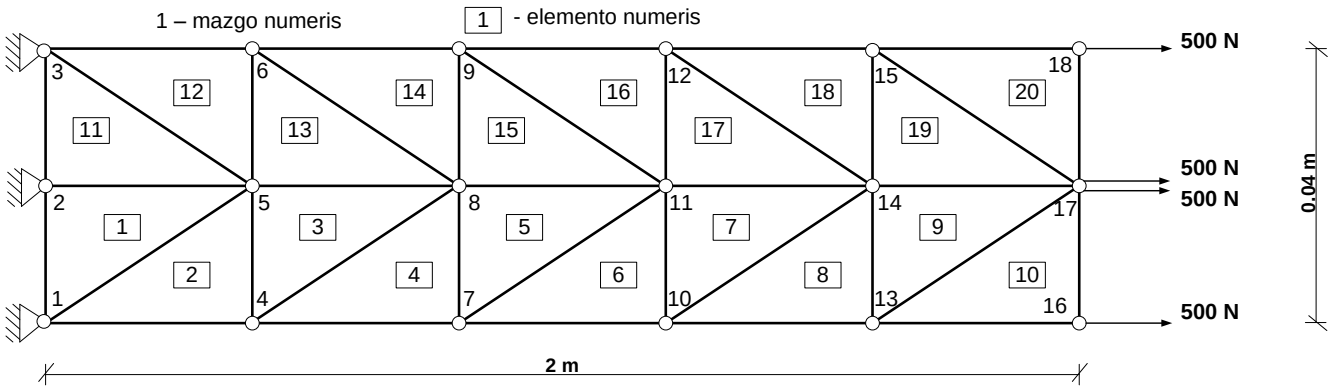


2.2 pav. Trikampis elementas

Poslinkių laukas tokiu atveju aprašomas dviem poslinkiais *u* ir *v* kiekviename mazge:

$$\{u_m\}^i = \begin{Bmatrix} u_i \\ v_i \end{Bmatrix}; \qquad \{u_m\}^e = \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{Bmatrix}.$$

Strypą (2.1 pav.) diskretizuosime baigtiniais CST (constant strain triangle) elementais kaip parodyta 2.3 paveiksle. Baigtinio elemento priklausomybes formuluosime globaliojoje koordinačių sistemoje.



2.3 pav. Strypas padalintas į 20 CST elementų

Formos funkcijos. Turėsime po tris unikalias funkcijas poslinkiams *u* ir *v*:

$$\{u\} = \sum_{i=1}^{6} N_i u_i; \qquad \{u\} = \begin{Bmatrix} u \\ v \end{Bmatrix}.$$

Jų gavimui parenkame pačius paprasčiausius – tiesinius interpoliacinius polinomus:

$$u = \alpha_1 + \alpha_2 x + \alpha_3 y; \tag{9 a}$$

$$v = \alpha_4 + \alpha_5 x + \alpha_6 y. \tag{9 b}$$

Tokiose funkcijose turime šešis nežinomus koeficientus α - tiek pat, kiek ir mazginių poslinkių. Tik išpildžius tokią sąlygą įmanoma formos funkcijos nežinomus koeficientus išreikšti per mazginius poslinkius ir mazgų koordinates. Tam į (9 a) įstačius paeiliui mazgų koordinates – turėsime gauti mazginius poslinkius:

$$\alpha_1 + \alpha_2 x_i + \alpha_3 y_i = u_i;$$
$$\alpha_1 + \alpha_2 x_j + \alpha_3 y_j = u_j;$$
$$\alpha_1 + \alpha_2 x_k + \alpha_3 y_k = u_k.$$

Analogiškai gaunamos trys lygtys poslinkiams *v*, įstatome į (9 b).

Išsprendus lygtis ir α_1, α_2, α_3 išraiškas įstačius į (9 a) gauname:

$$u = \frac{1}{2\Delta}\Big((a_i + b_i x + c_i y)u_i + \big(a_j + b_j x + c_j y\big)u_j + (a_k + b_k x + c_k y)u_k\Big); \tag{10}$$

$$a_i = x_j y_k - x_k y_j;$$
$$b_i = y_j - y_k = y_{jk};$$
$$c_i = x_k - x_j = x_{kj};$$

kiti koeficientai gaunami ciklišku indeksų keitimu: $i \to j \to k \to i$.

$$2\Delta = \begin{vmatrix} 1 & x_i & y_i \\ 1 & x_j & y_j \\ 1 & x_k & y_k \end{vmatrix} = 2 \cdot \langle trikampio\ ijk\ plotas \rangle.$$

Analogiškai:

$$v = \frac{1}{2\Delta}\Big((a_i + b_i x + c_i y)v_i + \big(a_j + b_j x + c_j y\big)v_j + (a_k + b_k x + c_k y)v_k\Big). \qquad (11)$$

Turint omenyje formos funkcijų apibrėžimus:

$$u = \sum_i^{i,j,k} N_i u_i; \qquad v = \sum_i^{i,j,k} N_i v_i,$$

(10) ir (11) galime perrašyti:

$$u = N_i u_i + N_j u_j + N_k u_k;$$
$$v = N_i v_i + N_j v_j + N_k v_k;$$
$$N_i = \frac{a_i + b_i x + c_i y}{2\Delta}; \qquad (12)$$
$$\dots$$

Deformacijos.

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_{xy} \end{Bmatrix} = \begin{Bmatrix} \partial u/\partial x \\ \partial v/\partial y \\ \partial u/\partial y + \partial v/\partial x \end{Bmatrix}.$$

BEM-e deformacijos užrašomos:

$$\{\varepsilon\} = [B]\{u_m\}^e; \qquad \{u_m\}^e = \{u_i \quad v_i \quad u_j \quad v_j \quad u_k \quad v_k\}^T;$$

$$[B] = \frac{1}{2\Delta}\begin{bmatrix} b_i & 0 & b_j & 0 & b_k & 0 \\ 0 & c_i & 0 & c_j & 0 & c_k \\ c_i & b_i & c_j & b_j & c_k & b_k \end{bmatrix}.$$

Taigi, deformacijos baigtinio elemento viduje nepriklauso nuo koordinačių – deformacijos elemento plote pastovios. Todėl elementas vadinamas CST (Constant Strain Triangle).

Įtempimai.

$$\{\sigma\} = [D]\{\varepsilon\}.$$

Plokščiajam įtempimui:

$$[D] = \frac{E}{1 - \vartheta^2}\begin{bmatrix} 1 & \vartheta & 0 \\ \vartheta & 1 & 0 \\ 0 & 0 & (1 - \vartheta)/2 \end{bmatrix},$$

čia E - tamprumo modulis (plieno 210 GPa);

ϑ - Puasono koeficientas (plieno 0,3).

Standumo matrica.

$$[K]^e = \int_V [B]^T[D][B]dV = \langle jei\ elemento\ storis\ pastovus\rangle = t\int_S [B]^T[D][B]dS =$$

$$= \langle jei\ medžiagos\ savybės\ elemente\ pastovios\rangle = t\Delta[B]^T[D][B],$$

čia V – elemento tūris;

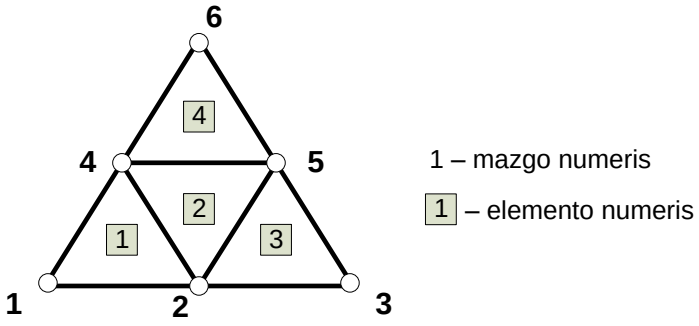
S – elemento plotas.

$[K]^e$ išraiška vienam unikaliam blokui plokščiojo įtempio atveju:

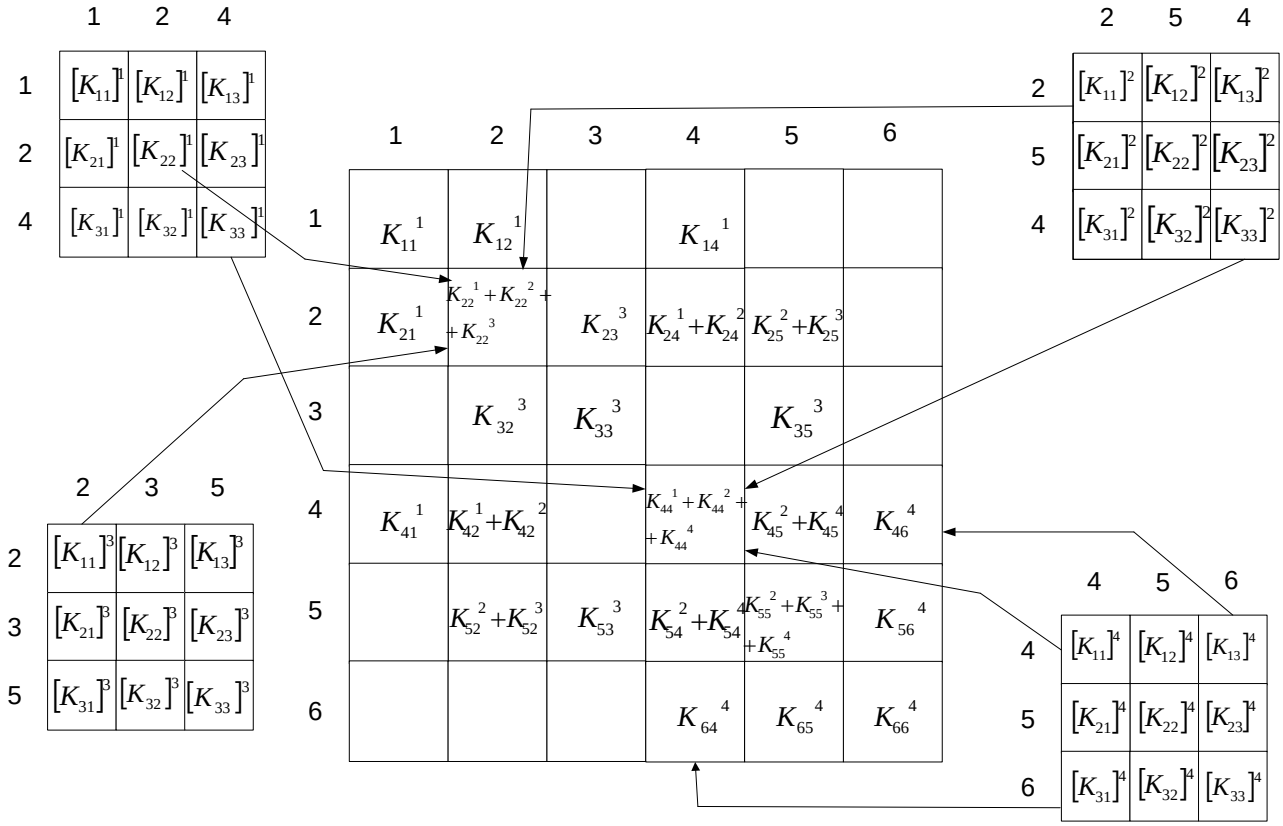
$$[K_{ij}] = \frac{Et}{4\Delta(1-\vartheta^2)}\begin{bmatrix} b_ib_j + \frac{1-\vartheta}{2}c_ic_j & \vartheta b_ic_j + \frac{1-\vartheta}{2}c_ib_j \\ \vartheta c_ib_j + \frac{1-\vartheta}{2}b_ic_j & c_ic_j + \frac{1-\vartheta}{2}b_ib_j \end{bmatrix};$$

$$[K]^e = \begin{bmatrix} [K_{11}] & [K_{12}] & [K_{13}] \\ [K_{21}] & [K_{22}] & [K_{23}] \\ [K_{31}] & [K_{32}] & [K_{33}] \end{bmatrix}.$$

Ansamblio matrica. Pasinaudosime paprastu pavyzdžiu, kad paaiškintume ansamblio matricos surinkimo eigą. Tegul turime trikampės formos plokštelę, kurią padalijome į keturis trikampius elementus. Pirmąjį elementą aprašo mazgai 1, 2 ir 4, antrąjį – 2, 5 ir 4, trečiąjį – 2, 3 ir 5, o ketvirtąjį – 4, 5 ir 6 (2.4 pav.). Kaip surenkama tokios plokštelės ansamblio matrica, grafiškai pavaizduota 2.5 paveiksle.



2.4 pav. Trikampė plokštelė



2.5 pav. Ansamblio matricos surinkimas

Poslinkių ir įtempimų skaičiavimas. Po ansamblio sudarymo, kraštinių sąlygų įvedimo, lygties (8) sprendimo: kiekvienam baigtiniam elementui atrenkami jam priklausantys poslinkiai $\{u_m\}$ ir įtempimai $\{\sigma\} = [D]\{\varepsilon\} = [D][B]\{u_m\}$. Žinant poslinkius galime apskaičiuoti tikslo funkcijos reikšmes.

$$M(X) = \sum_{i=1}^{n} S_i\, t_i \rho_i; \tag{13}$$

čia S_i – i-tojo elemento plotas kvadratiniais metrais;

t_i – i-tojo elemento storis metrais;

ρ_i – i-tojo elemento tankis kilogramais kubiniam metrui.

Kiekvieno baigtinio elemento įtempimai yra reikalingi maksimalių įtempimų apribojimui patikrinti. Kaip jau minėjome, įtempimai CST elemente yra pastovūs. Paprastai jie adresuojami elemento svorio centrui. Parašytoje programoje įtempimą kiekvienam baigtiniam elementui skaičiuojame pagal (15) formulę.

Apribojimai:

- Maksimalių įtempių. i-tojo elemento įtempiai neturi viršyti maksimalaus leistino įtempio;

$$\sigma_i \leq \sigma_{max}; \tag{14}$$

σ_i skaičiuojami pagal von Miseso (Von Mises) formulę:

$$\sigma_i = \sqrt{{\sigma_1}^2 + {\sigma_2}^2 - \sigma_1\sigma_2 + 3{\sigma_{12}}^2}, \quad (15)$$

čia σ_1, σ_2 – normaliniai įtempimai;

σ_{12} tangentinis įtempimas.

- Pusiausvyros. Struktūra turi būti pusiausvira;

$$[K] \cdot \{U\} = \{F\}; \quad (16)$$

- pasirinkti strypo elementų mazgai gali keisti savo koordinates tik elipsės plote (2.6 pav.);

Šis apribojimas bus tikrinamas pagal elipsės lygtį:

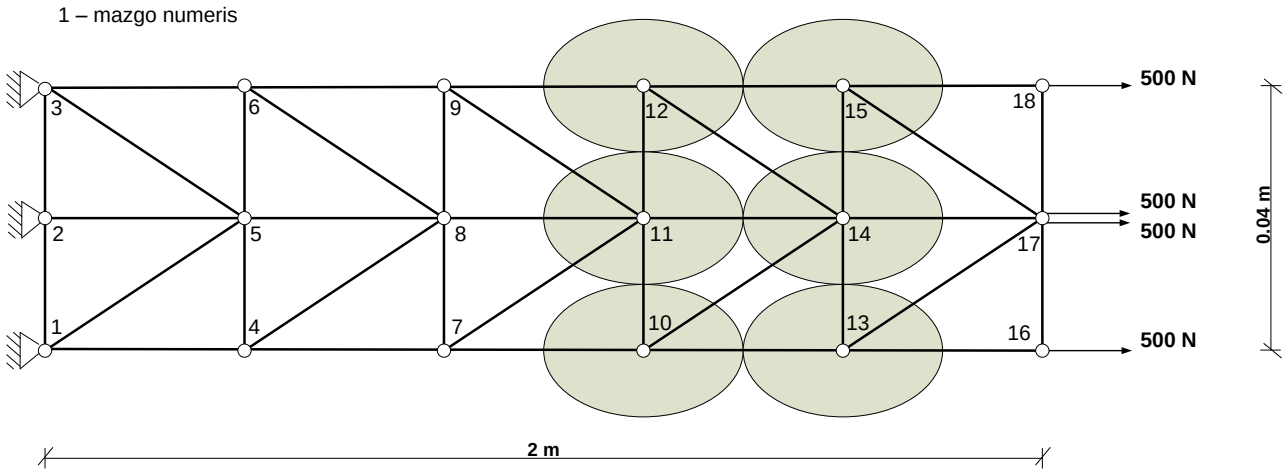
$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1, \quad (17)$$

čia *x*, *y* – mazgo koordinatės metrais;

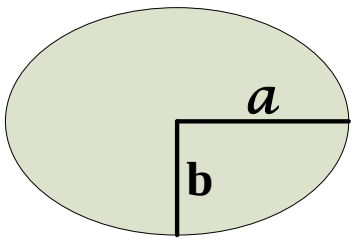
a – elipsės ilgosios ašies pusašis metrais (2.7 pav.);

b – elipsės trumposios ašies pusašis metrais (2.7 pav.).

- elipsės pusašis *a* neturi viršyti pusės atstumo tarp gretimų strypo mazgų horizontaliai (2.6 pav.);
- elipsės pusašis *b* neturi viršyti pusės atstumo tarp gretimų strypo mazgų vertikaliai.



2.6 pav. Baigtinių elementų mazgų apribojimas

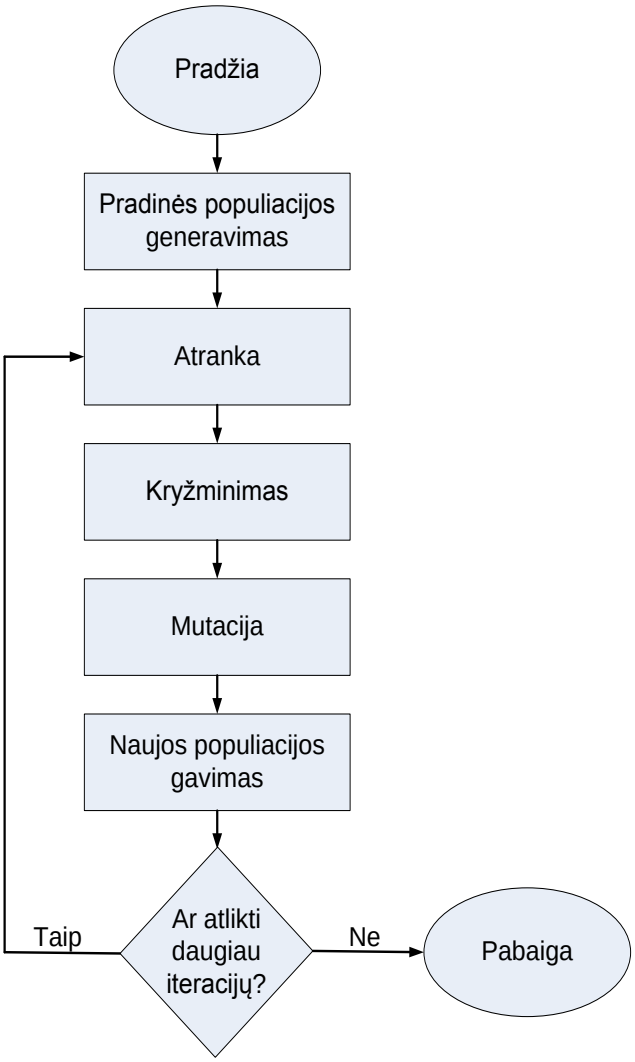


2.7 pav. Elipsės pusašės

Suformuluotą matematinį optimizavimo uždavinį spręsime genetiniais algoritmais.

2.2. Uždavinio sprendimas genetiniais algoritmais

Schematiškai GA programos veikimo algoritmas pavaizduotas 2.8 paveiksle. Aptarkime jo žingsnius.



2.8 pav. GA schema

2.2.1. Chromosomos kodavimas

Kaip jau minėjome, prieš pradedant spręsti uždavinį GA, pirmiausia iškyla klausimas kaip reprezentuoti sprendinius chromosomomis ir kokiu būdu jas užkoduoti. Šiame darbe naudosime tiesioginį kodavimo būdą, tai toks būdas, kai individo genotipas ir fenotipas sutampa.

Strypo masę minimizuosime keisdami kai kurių mazgų koordinates. Aukščiau aprašytame uždavinyje leidžiame judėti 10-15 mazgams kaip pavaizduota 2.6 paveiksle. Tokiu atveju turėsime dvylika projektavimo kintamųjų, šeši mazgai galės keisti savo koordinates.

Grįžtant prie GA terminologijos, mūsų uždavinio atveju, mazgų koordinačių *x* ir *y* eilutė yra chromosoma, o kiekviena *x* ir *y* koordinatė yra tos chromosomos genas (2.9 pav.). Dėl šios priežasties individai yra koduojami eilute kintamųjų, kurie atvaizduojami realiaisiais skaičiais, t.y. kiekviena chromosoma yra sudaryta iš realiųjų skaičių.

x_{10}	y_{10}	x_{11}	y_{11}	x_{12}	y_{12}	x_{13}	y_{13}	x_{14}	y_{14}	x_{15}	y_{15}

2.9 pav. Sprendžiamo uždavinio chromosoma

Pradinės, optimizuojamos strypo mazgų koordinatės metrais pateiktos 2.1 lentelėje. Pradinių koordinačių chromosoma pavaizduota 2.10 paveiksle.

2.1 lentelė. Pradinės mazgų koordinatės

Matmenys metrais

Mazgo numeris	**Koordinatė *x***	**Koordinatė *y***
10	1.2	0
11	1.2	0.02
12	1.2	0.04
13	1.6	0
14	1.6	0.02
15	1.6	0.04

1.2	0	1.2	0.02	1.2	0.04	1.6	0	1.6	0.02	1.6	0.04

2.10 pav. Pradinių koordinačių chromosoma

Pradinės populiacijos generavimo žingsnyje prie pradinių koordinačių yra pridedamas atsitiktinai sugeneruotas skaičius. Prie *x* ir *y* koordinatės pridedame realų skaičių iš intervalo $[-a; a]$ ir $[-b; b]$ atitinkamai. Mazgas, kaip jau minėjome, turi būti elipsės plote, todėl pridedant atsitiktinį skaičių prie *y* koordinatės šis apribojimas yra tikrinamas pagal elipsės lygtį. Taip yra sugeneruojamas mums reikalingas pradinių individų skaičius, t.y. tam tikras chromosomų skaičius, kurios turi atsitiktinius genus.

2.2.2. Atrankos metodai

Šiame darbe naudosime du atrankos metodus: ruletės ir elito. Ruletės atrankos metodas ko gero pats paprasčiausias ir vienas iš dažniausiai naudojamų metodų. Mūsų programoje jis suformuluotas taip: tikimybė, kad individas bus atrinktas į kitą populiaciją, yra proporcinga tą individą atitinkančios tikslo funkcijos reikšmės ir suminės visų populiacijos individų tikslo funkcijų reikšmės santykiui. Paprastai tariant, didžiausią tikslo funkcijos reikšmę turintys individai bus atrenkami su didele tikimybe. Šiame darbe kiekvieno individo tikslo funkcijos reikšmė bus apskaičiuojama pagal (13) formulę.

Kita strategija, kurią naudosime yra elito atranka. Elito metode atranka vykdoma analogiškai kaip ir ruletės atrankos atveju, tik čia konstruojant naują populiaciją leidžiama keliems tinkamiausiems individams iš karto pereiti į naująją populiaciją. Šiame darbe tam tikras individų, turinčių geriausią tikslo funkcijos reikšmę, skaičius pereina į kitą iteraciją be pakeitimų, o visi kiti populiacijos individai yra atrenkami ruletės metodu. Individai, kurie tenkina elito atrankos kriterijų gali dalyvauti ir ruletės atrankoje, t.y. kiekvienas individas gali būti atrinktas daugiau nei vieną kartą.

2.2.3. Kryžminimo metodai

Kryžminimas ir mutacija yra pagrindiniai genetinio algoritmo operatoriai lemiantys uždavinio sprendinių raidą. Kaip jau minėjome, jų tipų pasirinkimas ir realizavimas priklauso nuo uždavinio bei naudojamo chromosomų kodavimo būdo. Mūsų darbe bus naudojami trijų tipų standartiniai kryžminimo metodai: vieno taško, dviejų taškų ir kelių taškų. Šiame darbe kelių taškų metode naudosime trys kryžminimo taškus.

Kryžminimo žingsnyje iš dviejų atrinktų chromosomų kursime naujas chromosomas, naudojant esamų chromosomų dalis. Dviem vaikams suformuoti bus naudojami du tėvai. Elito atrankos atveju keli geriausi individai pateks į kitą populiaciją ir jiems nebus pritaikytas kryžminimo operatorius.

Parašytoje programoje vieno taško kryžminimo funkcija veikia taip kaip parodyta 1.7 paveiksle. Populiacijos individai sugrupuojami po du ir kiekvienai porai nustatoma kryžminimo pozicija. Kryžminimo pozicija yra atsitiktinis skaičius k, priklausantis sveikųjų skaičių intervalui:

$$k \in [1; n-1]; \tag{18}$$

čia n – individo genų skaičius.

Kadangi chromosomos genai yra priklausomi vienas nuo kito (y koordinatė priklauso nuo x koordinatės) programoje įdėjome patikrinimą, kuris užtikrina, kad suformuotas vaikas tenkins koordinačių apribojimus. Dviejų ir kelių taškų kryžminimo funkcijos veikia analogiškai tik atsitiktinai sugeneruojami du ir trys kryžminimo taškai atitinkamai.

Sprendžiamame uždavinyje kryžminimo tikimybė yra fiksuota ir lygi $0{,}8$. Dėl šios priežasties programoje su tikimybe $0{,}8$ bus generuojama pozicija iš intervalo $[1; n-1]$, o su tikimybe $0{,}2$ kryžminimo funkcija nebus iškviečiama visai ir pora individų nepasikeis.

2.2.4. Mutacija

Mutacijos žingsnyje atsitiktinai išrinktas genas yra pakeičiamas atsitiktine reikšme tam, kad įneštų naujos informacijos į populiaciją. Kaip jau minėjome, tai vienas iš pagrindinių genetinio algoritmo operatorių lemiančių uždavinio sprendinių raidą. Autorės pradinėje programoje mutacija yra vykdoma su tikimybe $0{,}01$. Prie atsitiktinai išrinkto geno pridedama arba atimama reikšmė δ, kuri yra apskaičiuojama pagal (19) formulę.

$$\delta = \frac{d}{10}; \tag{19}$$

čia $d = a$, kai atsitiktinis genas atitinka x koordinatę;

$d = b$, kai atsitiktinis genas atitinka y koordinatę.

Kokia operacija bus atlikta, sudėtis ar atimtis, nulemia atsitiktinai sugeneruotas skaičius. Kiekvienos operacijos tikimybė $0{,}5$.

Tyrimo metu patikrinsime kaip rezultatai (tikslo funkcija) priklauso nuo δ reikšmės. Atliksime eksperimentus su didesnėmis ir mažesnėmis δ reikšmėmis. Taip pat ištirsime kaip optimizavimo rezultatas priklauso nuo mutacijos tikimybės.

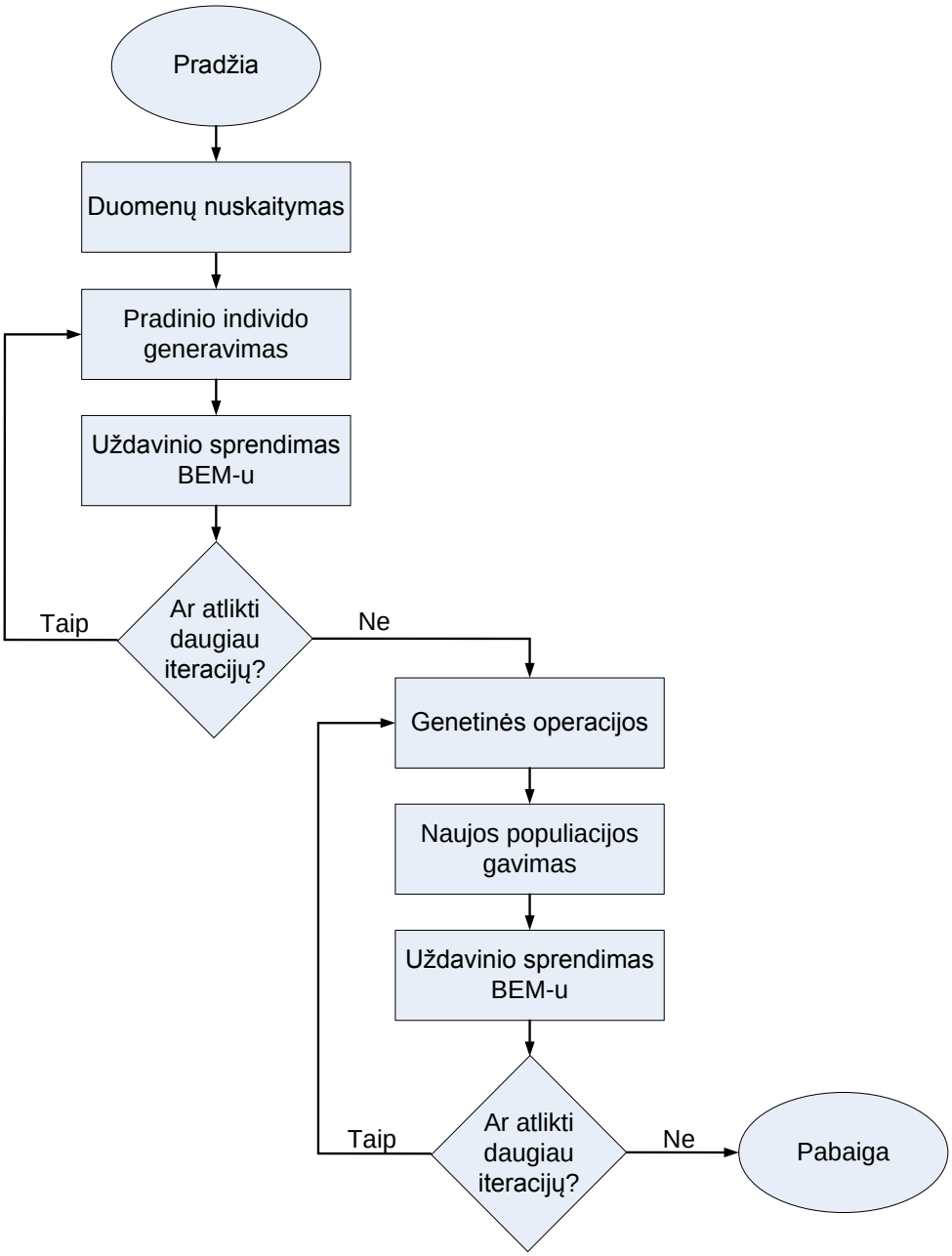
2.2.5. Algoritmo nutraukimas

Pasirinkus pradinę populiaciją, GA evoliucija prasideda nuo visiškai atsitiktinių kitimų. Atsitiktinai sugeneruota pradinė populiacija pasikeičia po pritaikytų genetinio algoritmo operatorių: atrankos, kryžminimo ir mutacijos. Pasikeitė individai, todėl pasikeičia ir juos atitinkančios tikslo funkcijos reikšmės. Kai kurie individai galėjo gautis geresni, kai kurie – blogesni. Dabar gautai populiacijai vėl galima taikyti aprašytus GA operatorius ir taip generuoti naują populiaciją. Bendras sprendinių evoliucijos procesas yra ciklišKai kartojamas kol pasiekiama nutraukimo sąlyga. GA evoliucijai stabdyti mūsų parašytoje programoje yra naudojamas dažniausias kriterijus algoritmui nutraukti – iteracijų skaičius. Šiame darbe

vykdomas 250 iteracijų ciklas. Visuose testuose uždavinys konvergavo prie mažesnio iteracijų skaičiaus, todėl 250 iteracijų pakaks su atsarga. Kiekviename ciklo žingsnyje patikriname ar gautos tikslo funkcijos reikšmės yra mažesnės už prieš tai buvusias. Jei taip, tai ši reikšmė ir iteracijos numeris yra išsaugomos. Gauti rezultatai bus naudojami eksperimentų aprašymuose ir lyginami tarpusavyje.

3. Programos algoritmo schema

Schematiškai visos programos veikimo algoritmas pavaizduotas 3.1 paveiksle. Trumpai apžvelkime jo žingsnius.



3.1 pav. Sukurtos programos veikimo algoritmas

- ***Duomenų nuskaitymas***. Šiame žingsnyje iš tekstinio failo „Duomenys.txt“ yra nuskaitomi pradiniai duomenys. Suteikiamos reikšmės CST elementų skaičiui *ne*, mazgų skaičiui *nm*, įtvirtinimų skaičiui *ni*, pridėtų jėgų skaičiui *nf* ir maksimaliam leistinam įtempimui *max_itemp*. Nuskaitomi parametrai aprašantys strypą – storis *t*, Puasono koeficientas *pk*, tamprumo modulis *E*. Suformuojami mazgų koordinačių *koordinates*, elementų ryšių *rysiai*, įtvirtinimų *itvirtinimai* ir pridėtų jėgų *jegos* masyvai. Atmintis masyvams skiriama dinamiškai. Taip pat nuskaitomi duomenys reikalingi genetinėms operacijoms: mazgų, galinčių keisti savo koordinates, skaičius *nmin*, populiacijos individų skaičius *nind*, mutacijos tikimybė *mtik* ir kryžminimo tikimybė *ktik*. Elitinės atrankos atveju papildomai nuskaitomas elitinių individų skaičius *elitn*.
- ***Pradinio individo generavimas***. Šiame žingsnyje yra generuojama atsitiktinė pradinės populiacijos chromosoma (žr. 2.2.1 poskyrių). Šis algoritmo žingsnis vykdomas cikle, kuriame sugeneruojama *nind* individų, kurie tenkina visus uždavinio apribojimus (žr. 2.1.2 poskyrio *apribojimai* punktą). Tokiu atveju pradinėje populiacijoje nėra netinkamų individų, visi jie pretenduoja užimti vietą naujoje populiacijoje su tikimybe priklausančia nuo tikslo funkcijos reikšmės. Kaip jau minėjome, kiekvieno individo tikslo funkcijos reikšmė bus apskaičiuojama pagal (13) formulę.
- ***Uždavinio sprendimas BEM-u.*** Šiame žingsnyje pirmiausiai skaičiuojama tikslo funkcijos reikšmė. Toliau surenkama standumo matrica ir įvertinamos kraštinės sąlygos. Gautoji (8) lygčių sistema, iš kurios randamas poslinkių vektorius $\{U\}$, sprendžiama Gauso metodu. Spręsdami šią sistemą tenkiname pusiausvyros ribojimą. Taip pat apskaičiuojamos elementų įtempimų reikšmės ir patikrinama ar jos neviršija *max_itemp* reikšmės.
- ***Genetinės operacijos.*** Šiame žingsnyje vykdomos visos genetinio algoritmo stadijos: atranka, kryžminimas ir mutacija (plačiau žr. 2.2 poskyryje).

Uždavinys sprendžiamas kol nepatenkintas algoritmo stabdymo kriterijus. Mūsų atveju tai iteracijų skaičius. Kadangi GA naudoja atsitiktinius dydžius, kaip jau minėjome, uždavinį su tais pačiais parametrais geriau leistį keletą kartų. Parašytoje programoje algoritmas vykdomas 250 kartų, t.y. kiekvienam GA parametrų rinkiniui yra generuojama 250 populiacijų. Be to, su kiekvienu parametrų rinkiniu atliksime po 50 nepriklausomų bandymų gautų rezultatų patikimumui užtikrinti.

4. Tyrimai ir eksperimentai

Šio darbo tikslas yra surasti optimalius tiriamo algoritmo parametrus bei nustatyti GA strategiją, kuri leistų efektyviausiai atlikti mechaninės struktūros masės optimizavimą. Šiam tikslui pasiekti reikia ištestuoti visas galimas strategijas – atrankos, kryžminimo ir mutacijos metodus. Be to, genetiniuose algoritmuose taip pat naudojami ir kiti parametrai: populiacijos individų skaičius, kryžminimo tikimybė, mutacijos tikimybė, iteracijų skaičius. Visi šie parametrai dar padidina strategijų skaičių. Tad, savaime suprantama, visų galimų variantų ištirti neįmanoma.

Šiame darbe eksperimentai buvo atliekami tokia tvarka: uždavinys sprendžiamas su įvairaus dydžio populiacijomis, elito atrankos taikymas, elitinių individų skaičiaus tyrimas, kryžminimo strategijų taikymas ir mutacijos strategijų taikymas. GA parametrų reikšmės buvo pasirenkamos prisilaikant teorinių rekomendacijų [8; 13]. Tiriant GA strategijas ir parametrų reikšmes iš viso buvo atlikta 4600 nepriklausomų eksperimentų. Genetinio algoritmo strategijos buvo vertinamos lyginant tikslo funkcijos reikšmes: geriausią (mažiausią), reikšmių vidurkį ir blogiausią (didžiausią). Apibendrinti bandymų rezultatai pateikti sekančiuose šio skyriaus poskyriuose.

4.1. GA populiacijos dydžio tyrimas

Populiacijos dydis yra labai svarbus parametras genetiniuose algoritmuose. Parinkus per mažą populiaciją galimas ankstyvas konvergavimas į lokalų sprendinį, kuris gali buti toli nuo tikslaus sprendinio, o parinkus didelę populiaciją sprendinio ieškojimas ilgai užtruks. Šiame poskyryje tirsime tikslo funkcijos reikšmių priklausomybę nuo populiacijos dydžio. Atliksime po 50 nepriklausomų bandymų su kiekvienu individų skaičiumi. 4.1 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai, naudojami pirmame tyrime.

4.1 lentelė. Pirmo tyrimo GA parametrai

Mutacijos tikimybė	0,01
Kryžminimo tikimybė	0,8
Iteracijų skaičius	250
Atrankos metodas	ruletės
Kryžminimo metodas	vieno taško

Kadangi GA nėra tikslūs algoritmai, tai rasto sprendinio gerumą vertinsime pagal vidutinį jo nuokrypį nuo vidurkio. Skaičiuosime standartinį nuokrypį, kuris parodo vidutinę duomenų sklaidą apie vidurkį. Papildomai pateiksime vidurkio pasikliautinuosius intervalus, kai

pasikliovimo lygmuo yra 95 %. Pasikliautinasis intervalas nusako kokiam intervalui priklausys tikroji parametro reikšmė. Pasikliovimo lygmuo tik atskleidžia mūsų pasitikėjimą (pasikliovimą) intervalo sudarymo taisyklėmis. Taigi, mūsų atveju, galime būti tikri 95 %, kad sekančios populiacijos tikslo funkcijos reikšmių vidurkis priklausys apskaičiuotam intervalui.

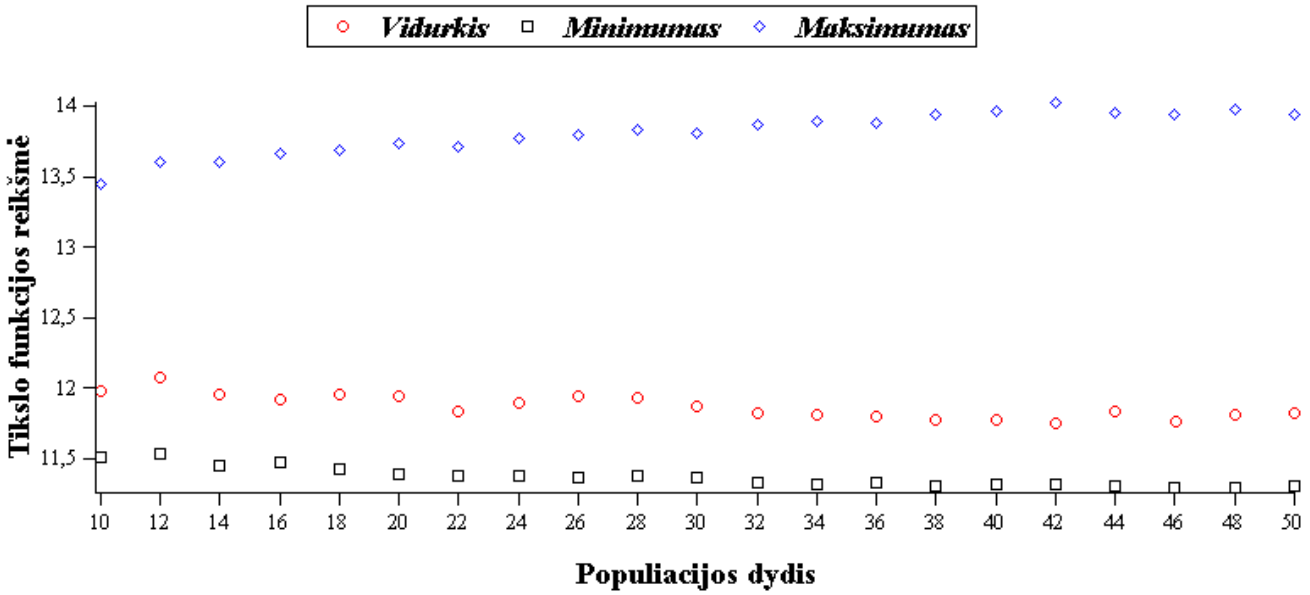
4.2 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio

Populiacijos dydis	**Vidurkis**	**Minimumas**	**Maksimumas**
10	11,982	11,509	13,448
12	12,071	11,539	13,609
14	11,959	11,450	13,600
16	11,919	11,479	13,668
18	11,953	11,424	13,684
20	11,941	11,393	13,740
22	11,836	11,375	13,711
24	11,891	11,374	13,766
26	11,945	11,366	13,793
28	11,928	11,375	13,828
30	11,872	11,364	13,810
32	11,826	11,332	13,863
34	11,808	11,314	13,890
36	11,802	11,330	13,879
38	11,780	11,313	13,937
40	11,771	11,314	13,960
42	11,752	11,315	14,029
44	11,835	11,304	13,947
46	11,763	11,295	13,945
48	11,807	11,301	13,976
50	11,824	11,305	13,939

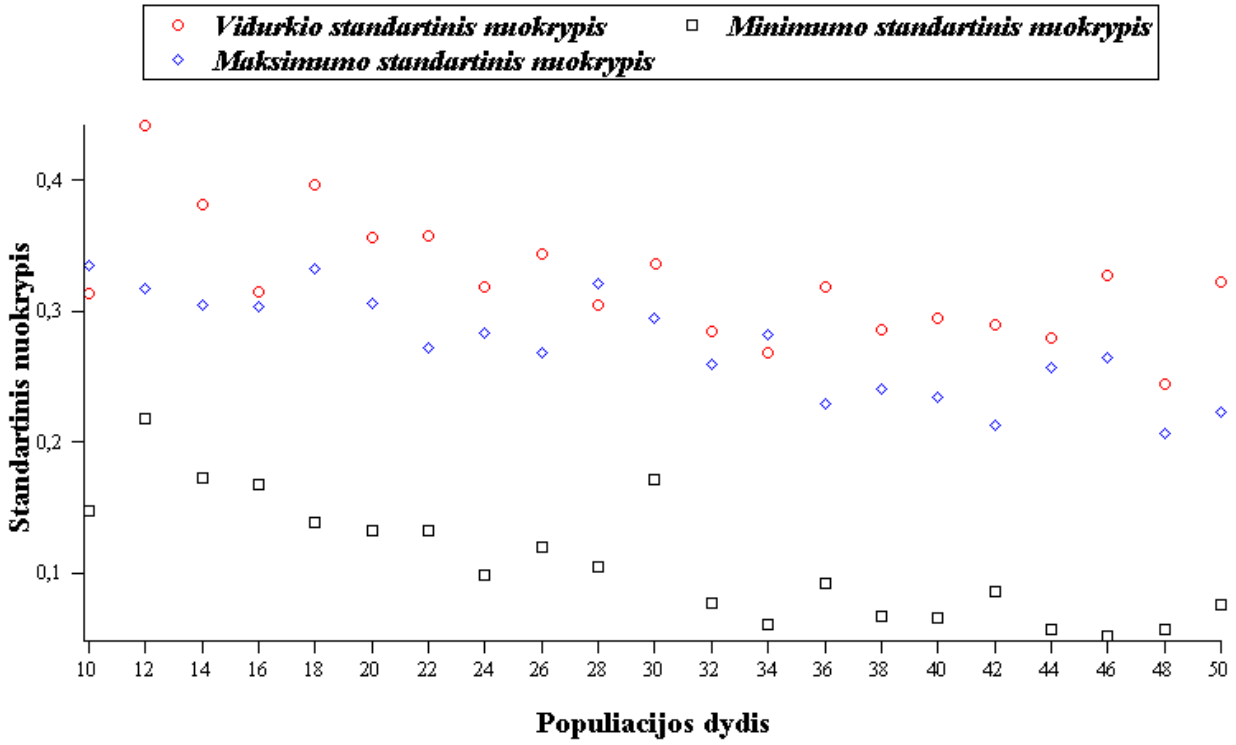
Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais, t.y. kiekvienam laukeliui apskaičiuoti atlikome 50 eksperimentų ir apskaičiavome vidurkį. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, t.y. mažiausias gautas populiacijos masių vidurkis kiekviename eksperimente, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės. Grafiškai 4.2 lentelės rezultatai pavaizduoti 4.1 paveiksle. Apatinė kreivė vaizduoja antrą 4.2 lentelės stulpelį, vidurinė – pirmą 4.2 lentelės stulpelį, viršutinė – trečią 4.2 lentelės stulpelį.

Vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių standartiniai nuokrypiai pavaizduoti 4.2 paveiksle. Apatinė kreivė vaizduoja minimalių tikslo funkcijos reikšmių sklaidą

apie vidurkį, vidurinė – maksimalių reikšmių sklaidą apie vidurkį, viršutinė – vidutinių reikšmių sklaidą apie vidurkį.



4.1 pav. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio



4.2 pav. Standartinio nuokrypio priklausomybė nuo populiacijos dydžio

4.3 lentelėje pateikta vidurkio pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio. Duomenys privalo būti pasiskirstę pagal normalųjį skirstinį, kad galėtume apskaičiuoti pasikliautinuosius intervalus. Šį reikalavimą tikrinome SPSS programa. Iš testo gavome, kad

bandymų rezultatai nėra pasiskirstę pagal normalųjį skirstinį, kai populiaciją sudaro 46 individai. Dėl šios priežasties pasikliautinasis intervalas negali būti apskaičiuotas, kai populiacijos dydis 46 individai.

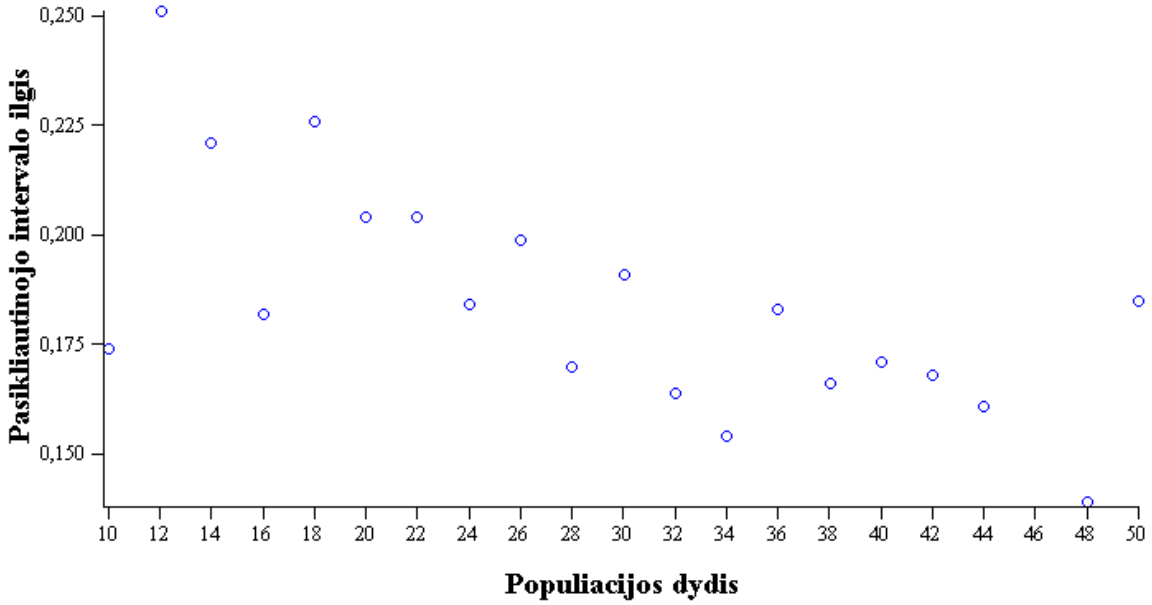
4.3 lentelė. Pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio

Populiacijos dydis	**Vidurkio pasikliautinasis intervalas**
10	[11,890; 12,065]
12	[11,958; 12,209]
14	[11,853; 12,073]
16	[11,826; 12,009]
18	[11,829; 12,055]
20	[11,831; 12,036]
22	[11,742; 11,946]
24	[11,798; 11,983]
26	[11,846; 12,046]
28	[11,854; 12,025]
30	[11,767; 11,957]
32	[11,742; 11,906]
34	[11,725; 11,880]
36	[11,717; 11,900]
38	[11,696; 11,862]
40	[11,688; 11,859]
42	[11,667; 11,836]
44	[11,758; 11,920]
46	-
48	[11,730; 11,869]
50	[11,740; 11,925]

Savaime suprantama, kuo pasikliautinasis intervalas siauresnis, tuo geriau. Pasikliautinasis intervalas priklauso nuo trijų dydžių:

1. didesnė standartinio nuokrypio reikšmė paplatina pasikliautinąjį intervalą;
2. didesnis bandymų skaičius susiaurina pasikliautinąjį intervalą;
3. pakeitę pasikliovimo lygmenį, pakeisime pasikliautinąjį intervalą. Pavyzdžiui, 99 % pasikliautinasis intervalas yra platesnis negu 95 % pasikliautinasis intervalas [14].

Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio pavaizduota 4.3 paveiksle.



4.3 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio

Iš gautų eksperimentų grafikų (4.1, 4.2 ir 4.3) galime teigti, kad optimalus populiacijos dydis yra apie 32-48 individų. Geriausią minimalią tikslo funkcijos reikšmę gavome, kai populiaciją sudarė 46 individai, o geriausią vidutinę – 42 individai. Mažiausią minimumo standartinį nuokrypį, kaip ir geriausią minimalią tikslo funkcijos reikšmę, gavome su 46 individais. Vidurkio standartiniai nuokrypiai yra didžiausi – 0,245-0,442, kas sudaro 2-4 % tikslo funkcijos reikšmių. Vidurkio pasikliautinųjų intervalų ilgiai yra pakankamai siauri, tad galime teigti, kad gauti rezultatai yra patikimi.

4.2. GA elitinės strategijos tyrimas

Kaip jau minėjome, šiame darbe naudosime du atrankos metodus: ruletės ir elito. 4.1 poskyryje pateikėme rezultatus, kuriuos gavome naudodami ruletės atranką. Šiame poskyryje taikydami elito atranką tirsime tikslo funkcijos reikšmių priklausomybę nuo populiacijos dydžio. Atliksime po 50 nepriklausomų bandymų su kiekvienu individų skaičiumi. Vėliau pasirinkę optimaliausią populiacijos dydį, tirsime tikslo funkcijos reikšmių priklausomybę nuo elitinių individų skaičiaus. 4.4 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai naudojami antrame tyrime, o 4.5 lentelėje pateikta tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio.

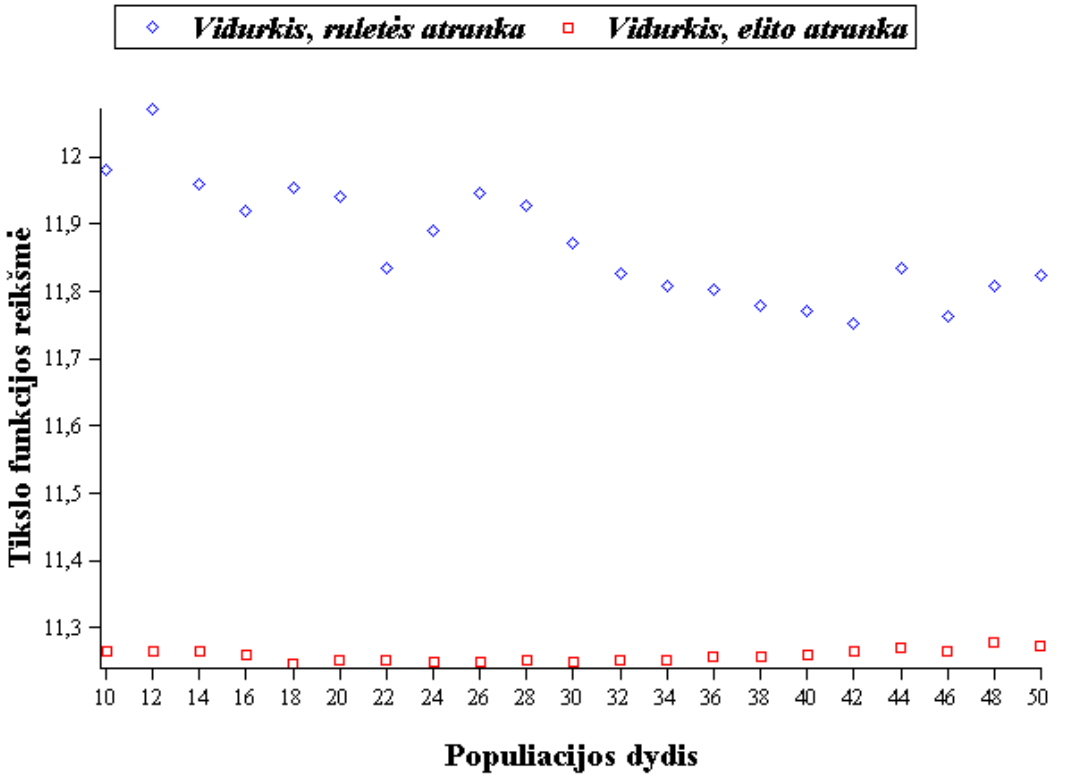
4.4 lentelė. Antro tyrimo GA parametrai

Mutacijos tikimybė	0,01
Kryžminimo tikimybė	0,8
Iteracijų skaičius	250
Atrankos metodas	elito
Elitinių individų skaičius	2
Kryžminimo metodas	vieno taško

4.5 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo populiacijos dydžio

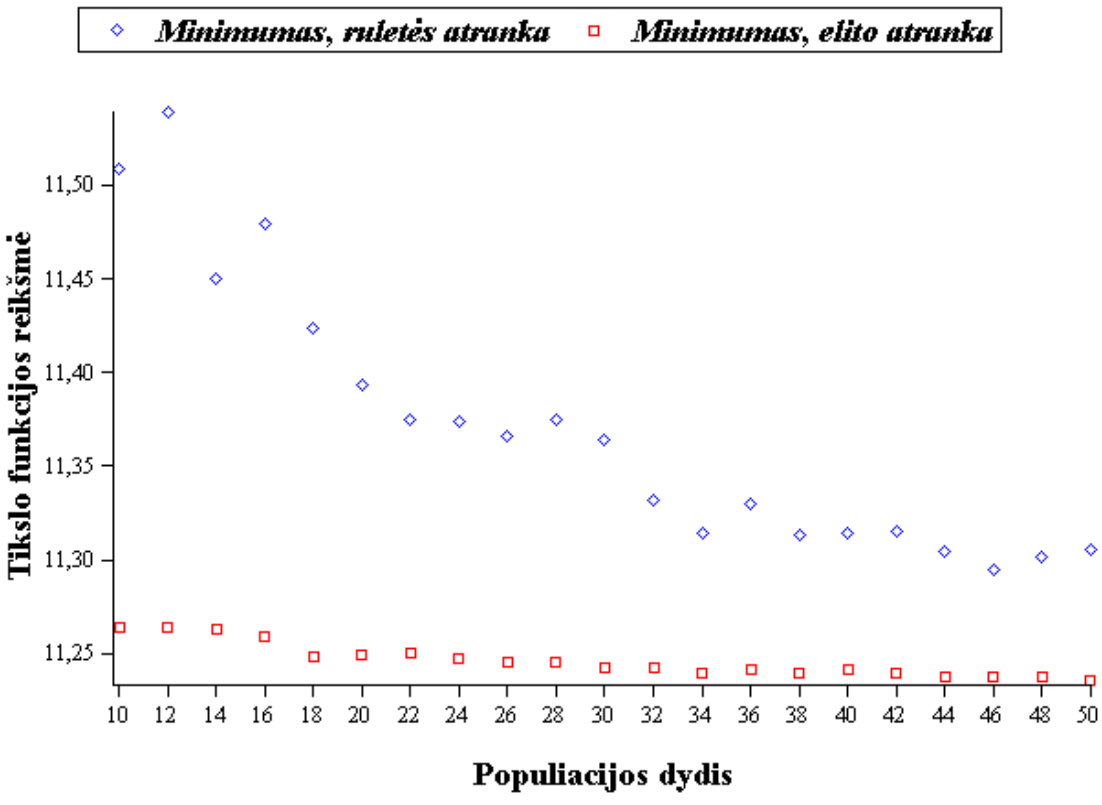
Populiacijos dydis	**Vidurkis**	**Minimumas**	**Maksimumas**
10	11,264	11,264	13,292
12	11,264	11,264	13,393
14	11,264	11,263	13,513
16	11,260	11,259	13,520
18	11,248	11,248	13,576
20	11,251	11,249	13,715
22	11,252	11,250	13,623
24	11,250	11,247	13,658
26	11,250	11,245	13,698
28	11,253	11,245	13,709
30	11,249	11,242	13,815
32	11,253	11,242	13,738
34	11,253	11,239	13,839
36	11,257	11,241	13,857
38	11,257	11,239	13,851
40	11,260	11,241	13,856
42	11,265	11,239	13,827
44	11,270	11,237	13,863
46	11,265	11,237	13,942
48	11,279	11,237	13,898
50	11,273	11,236	13,860

Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės. Tikslo funkcijos reikšmių skirtumai pavaizduoti 4.4, 4.5 ir 4.6 paveiksluose. 4.4 paveiksle apatinė kreivė vaizduoja pirmą 4.5 lentelės stulpelį, o viršutinė – pirmą 4.2 lentelės stulpelį.



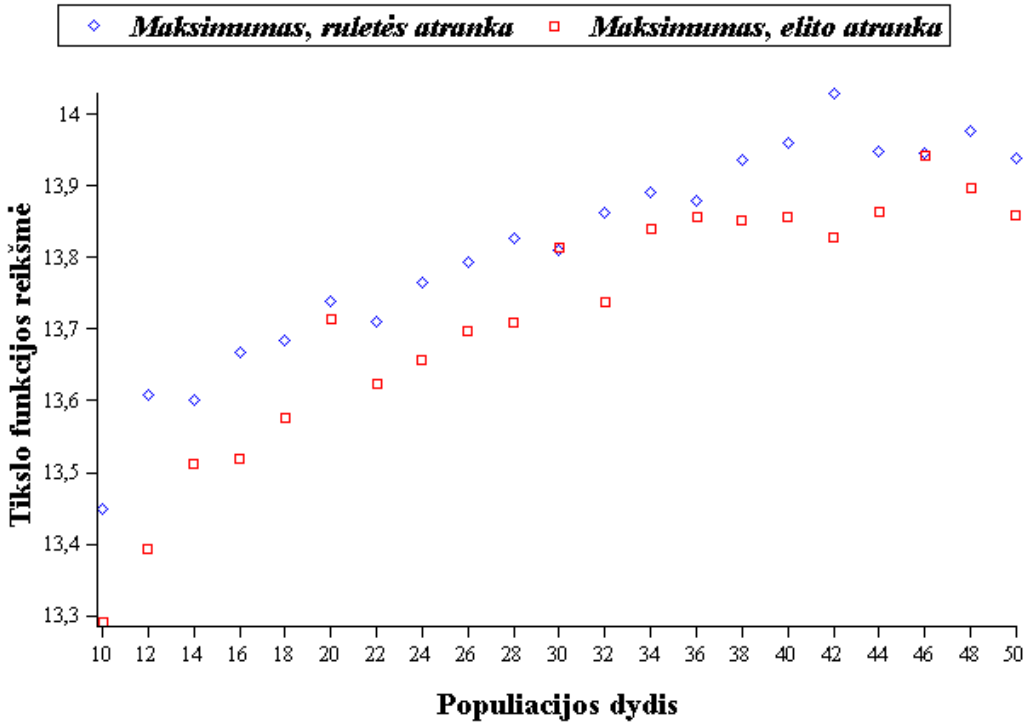
4.4 pav. Vidurkių priklausomybė nuo populiacijos dydžio

4.5 paveiksle apatinė kreivė vaizduoja antrą 4.5 lentelės stulpelį, o viršutinė – antrą 4.2 lentelės stulpelį.



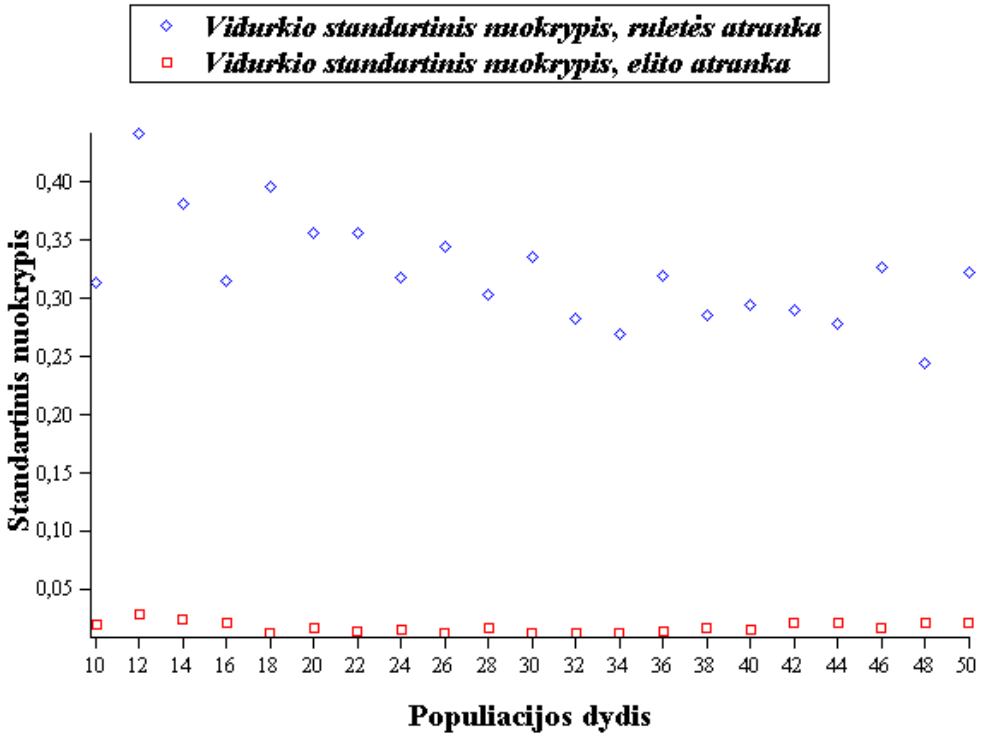
4.5 pav. Minimumų priklausomybė nuo populiacijos dydžio

4.6 paveiksle apatinė kreivė vaizduoja trečią 4.5 lentelės stulpelį, o viršutinė – trečią 4.2 lentelės stulpelį.

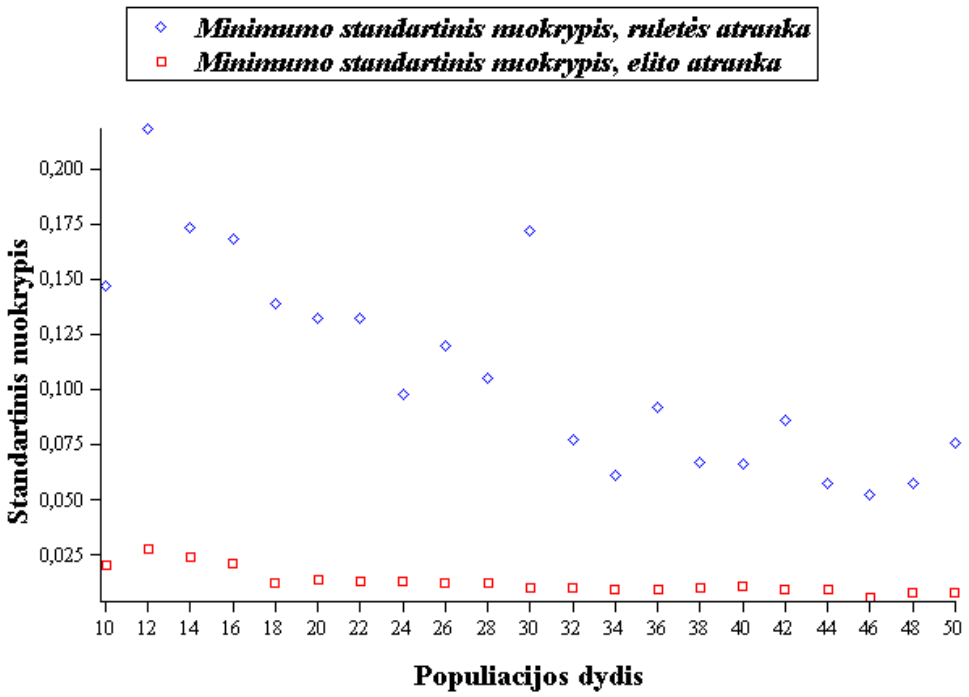


4.6 pav. Maksimumų priklausomybė nuo populiacijos dydžio

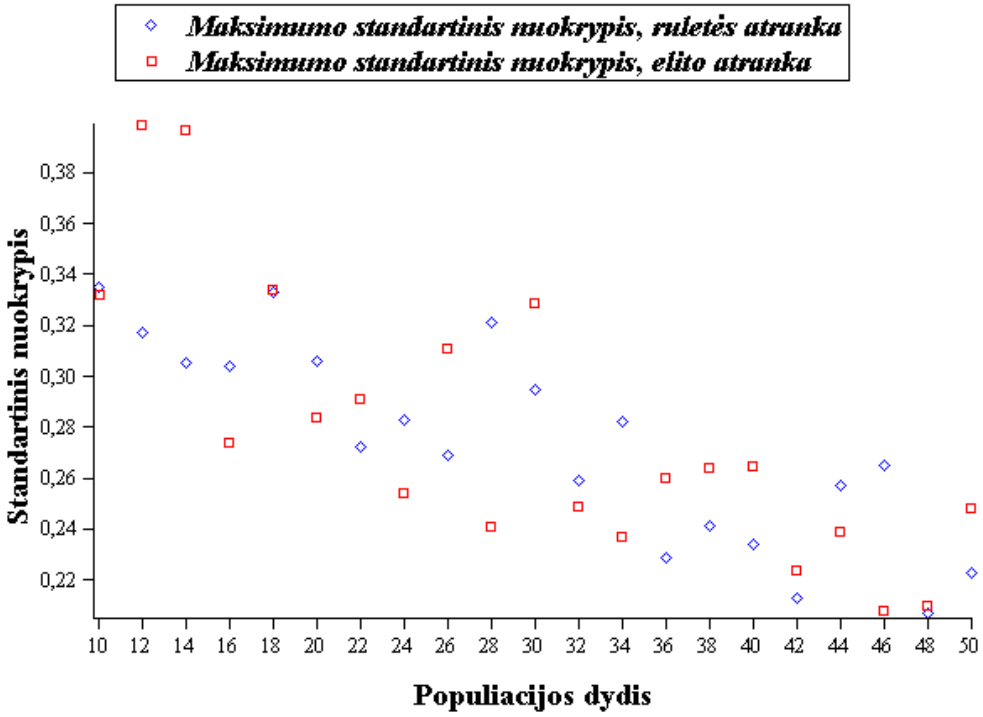
Vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių standartinių nuokrypių skirtumai pavaizduoti 4.7, 4.8 ir 4.9 paveiksluose.



4.7 pav. Vidurkio standartinių nuokrypių priklausomybė nuo populiacijos dydžio



4.8 pav. Minimumo standartinių nuokrypių priklausomybė nuo populiacijos dydžio



4.9 pav. Maksimumo standartinių nuokrypių priklausomybė nuo populiacijos dydžio

4.6 lentelėje pateikta vidurkio pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio. Šiuo atveju iš testo gavome, kad eksperimentų rezultatai nėra pasiskirstę pagal normalųjį skirstinį, kai populiaciją sudaro 10, 12, 16, 20, 28, 30, 34, 36 ir 44 individai. Kaip jau minėjome, dėl šios priežasties pasikliautinieji intervalai negali būti apskaičiuojami.

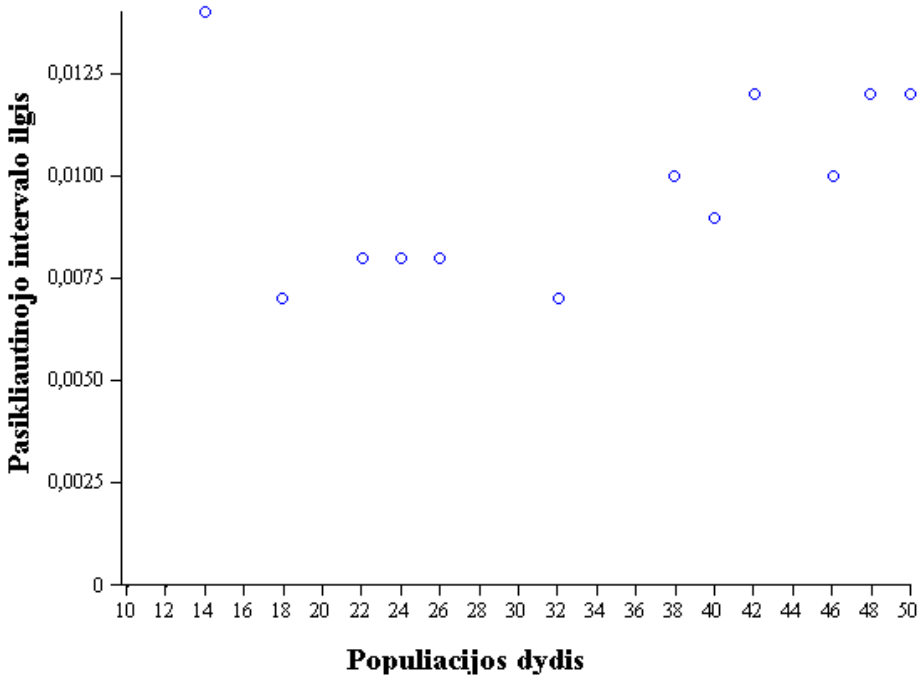
4.6 lentelė. Pasikliautinųjų intervalų priklausomybė nuo populiacijos dydžio

Populiacijos dydis	Vidurkio pasikliautinasis intervalas
10	-
12	-
14	[11,257; 11,271]
16	-
18	[11,245; 11,252]
20	-
22	[11,248; 11,256]
24	[11,246; 11,254]
26	[11,246; 11,254]
28	-
30	-
32	[11,249; 11,256]
34	-
36	-
38	[11,252; 11,262]
40	[11,256; 11,264]
42	[11,259; 11,271]
44	-
46	[11,260; 11,270]
48	[11,273; 11,285]
50	[11,266; 11,278]

Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio pavaizduota 4.10 paveiksle.

Iš gautų eksperimentų grafikų (4.4, 4.5 ir 4.6) matome, kad elito atrankos metodo taikymas leido sumažinti tikslo funkcijos reikšmes. Vidurkį sumažinome apie 4-7 %, minimumą – 1-3 %, maksimumą – 1-2 %. Vidurkio ir minimumo standartiniai nuokrypiai sumažėjo apie 10 %. Vidurkio pasikliautinieji intervalai susiaurėjo apie 5 %.

Remiantis gautais eksperimentų rezultatais pasirenkame optimaliausią populiacijos dydį – 46 individai. Tirsime tikslo funkcijos reikšmių priklausomybę nuo elitinių individų skaičiaus, kai populiaciją sudaro 46 individai. 4.7 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai naudojami trečiame tyrime, o 4.8 lentelėje pateikta tikslo funkcijos reikšmių priklausomybė nuo elitinių individų skaičiaus.



4.10 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo populiacijos dydžio

4.7 lentelė. Trečio tyrimo GA parametrai

Mutacijos tikimybė	0,01
Kryžminimo tikimybė	0,8
Iteracijų skaičius	250
Individų kiekis populiacijoje	46
Atrankos metodas	elito
Kryžminimo metodas	vieno taško

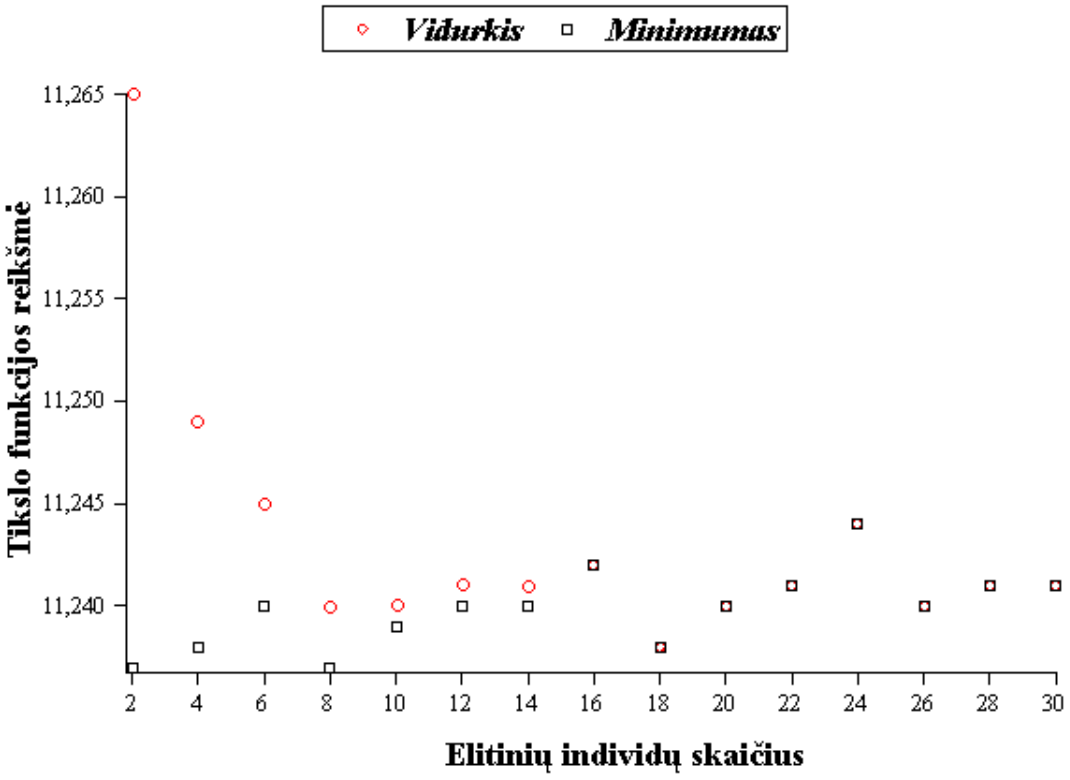
4.8 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo elitinių individų skaičiaus

Elitinių individų skaičius	**Vidurkis**	**Minimumas**	**Maksimumas**
2	11,265	11,237	13,942
4	11,249	11,238	13,882
6	11,245	11,240	13,772
8	11,240	11,237	13,704
10	11,240	11,239	13,779
12	11,241	11,240	13,783
14	11,241	11,240	13,692
16	11,242	11,242	13,645
18	11,238	11,238	13,664
20	11,240	11,240	13,694
22	11,241	11,241	13,712

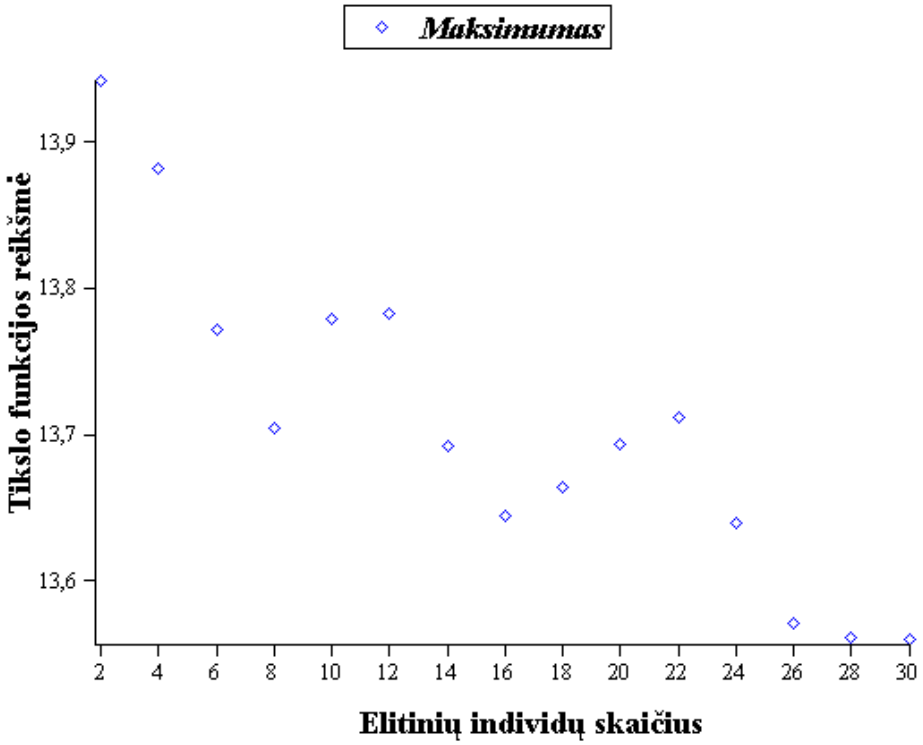
4.8 lentelės pabaiga

Elitinių individų skaičius	Vidurkis	Minimumas	Maksimumas
24	11,244	11,244	13,640
26	11,240	11,240	13,571
28	11,241	11,241	13,561
30	11,241	11,241	13,560

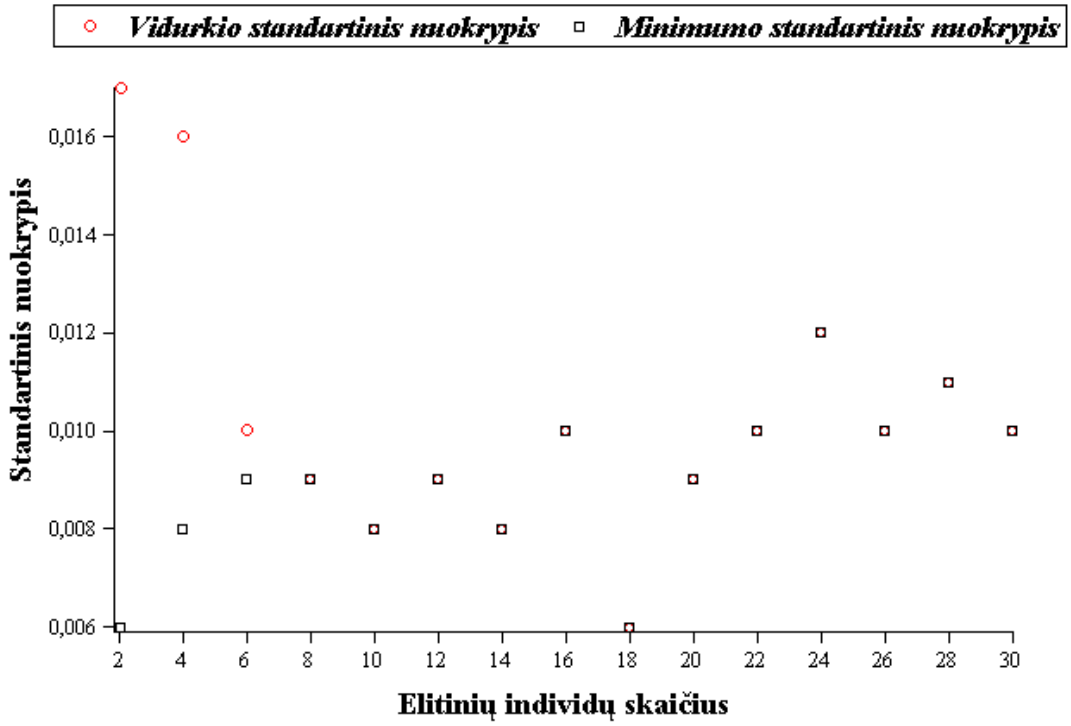
Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės. Grafiškai 4.8 lentelės rezultatai pavaizduoti 4.11 ir 4.12 paveiksluose. Vidutinių ir minimalių tikslo funkcijos reikšmių standartiniai nuokrypiai pavaizduoti 4.13 paveiksle. Apatinė kreivė vaizduoja minimalių tikslo funkcijos reikšmių sklaidą apie vidurkį, viršutinė – vidutinių reikšmių sklaidą apie vidurkį. Kai populiaciją sudaro 8 elitiniai individai ir daugiau standartiniai nuokrypiai tampa lygūs.



4.11 pav. Vidurkių ir minimumų priklausomybė nuo elitinių individų skaičiaus



4.12 pav. Maksimumų priklausomybė nuo elitinių individų skaičiaus



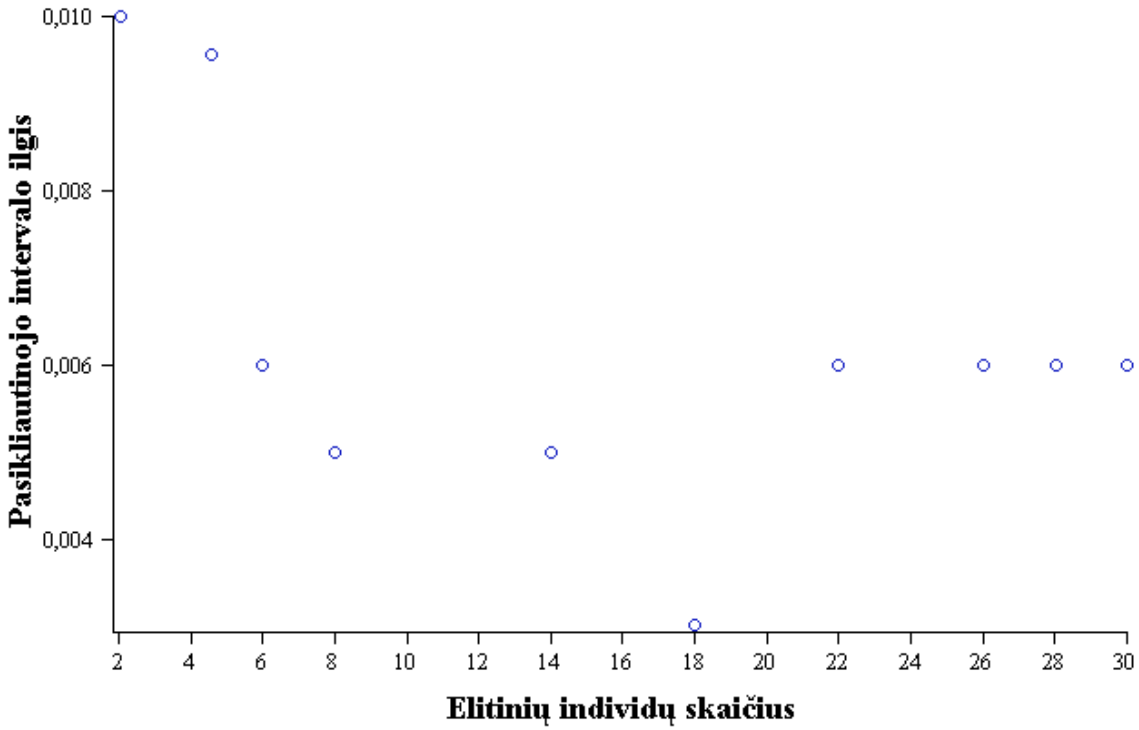
4.13 pav. Standartinio nuokrypio priklausomybė nuo elitinių individų skaičiaus

4.9 lentelėje pateikta vidurkio pasikliautinųjų intervalų priklausomybė nuo elitinių individų skaičiaus. Šiuo atveju iš testo gavome, kad eksperimentų rezultatai nėra pasiskirstę pagal normalųjį skirstinį, kai populiaciją sudaro 4, 10, 12, 16, 20 ir 24 elitiniai individai.

4.9 lentelė. Pasikliautinųjų intervalų priklausomybė nuo elitinių individų skaičiaus

Populiacijos dydis	Vidurkio pasikliautinasis intervalas
2	[11,260; 11,270]
4	-
6	[11,242; 11,248]
8	[11,238; 11,243]
10	-
12	-
14	[11,239; 11,243]
16	-
18	[11,236; 11,240]
20	-
22	[11,238; 11,244]
24	-
26	[11,238; 11,243]
28	[11,237; 11,244]
30	[11,238; 11,244]

Pasikliautinųjų intervalų ilgių priklausomybė nuo elitinių individų skaičiaus pavaizduota 4.14 paveiksle.



4.14 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo elitinių individų skaičiaus

Iš trečio tyrimo gautų eksperimentų rezultatų (4.11 pav., 4.12 pav.) matome, kad pasitvirtina teoriniai teiginiai. Susidūrėme su dviem pagrindinėmis, ankščiau minėtomis problemomis. Pirmoji – tikslo funkcija konverguoja į lokalųjį minimumą. Dėl šios priežasties, kai populiaciją sudaro net 18 elitinių individų, gauname atsitiktinį, pakankamai gerą sprendinį. Antroji problema – individų supanašėjimas. Kaip matome iš 4.11 paveikslo, kai populiaciją sudaro 16 ir daugiau elitinių individų, t.y. 35 % ir daugiau, minimalios tikslo funkcijos reikšmės sutampa su vidutinėmis tikslo funkcijos reikšmėmis. Tokius rezultatus gavome dėl to, kad geriausias individas buvo parenkamas labai dažnai ir po tam tikro iteracijų skaičiaus visą populiaciją sudarė beveik vienodi individai.

Padidinus elitinių individų skaičių populiacijoje, minimalią tikslo funkcijos reikšmę sumažinti nepavyko, tad tolesniuose tyrimuose leisime pereiti į kitą populiaciją tik dviem sėkmingiausiems individams be jokių pakeitimų.

4.3. GA kryžminimo strategijų tyrimas

Kaip jau minėjome, kryžminimas ir mutacija yra pagrindiniai genetinio algoritmo operatoriai lemiantys uždavinio sprendinių raidą. Šiame poskyryje tirsime tikslo funkcijos reikšmių priklausomybę nuo kryžminimo taškų skaičiaus. Taikysime vieno, dviejų ir trijų taškų kryžminimo metodus. Atliksime po 50 nepriklausomų bandymų su kiekvienu kryžminimo metodu. 4.10 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai naudojami ketvirtame tyrime, o 4.11 lentelėje pateikta tikslo funkcijos reikšmių priklausomybė nuo kryžminimo taškų skaičiaus.

4.10 lentelė. Ketvirto tyrimo GA parametrai

Mutacijos tikimybė	0,01
Kryžminimo tikimybė	0,8
Iteracijų skaičius	250
Individų kiekis populiacijoje	46
Atrankos metodas	elito
Elitinių individų skaičius	2

4.11 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo kryžminimo taškų skaičiaus

Kryžminimo taškų skaičius	Vidurkis	Minimumas	Maksimumas
1	11,265	11,237	13,942
2	11,270	11,235	13,861
3	11,271	11,236	13,810

Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės. Kaip matome iš 4.11 lentelės, vidutinės, minimalios ir maksimalios tikslo funkcijos reikšmės yra pakankamai stabilios.

4.12 lentelėje pateikiami vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių standartiniai nuokrypiai. Pirmame lentelės stulpelyje pateikta vidutinių tikslo funkcijos reikšmių sklaida apie vidurkį, antrame – minimalių tikslo funkcijos reikšmių sklaida apie vidurkį, trečiame – maksimalių tikslo funkcijos reikšmių sklaida apie vidurkį.

4.12 lentelė. Standartinio nuokrypio priklausomybė nuo kryžminimo taškų skaičiaus

Kryžminimo taškų skaičius	Vidurkio standartinis nuokrypis	Minimumo standartinis nuokrypis	Maksimumo standartinis nuokrypis
1	0,017	0,006	0,208
2	0,022	0,006	0,233
3	0,024	0,008	0,195

Eksperimentų rezultatai nėra pasiskirstę pagal normalųjį skirstinį, kai taikome dviejų ir trijų taškų kryžminimo metodus. Šiais atvejais pasikliautinieji intervalai negali būti apskaičiuojami. Pritaikius vieno taško kryžminimo metodą vidurkio pasikliautinasis intervalas [11,260; 11,270] .

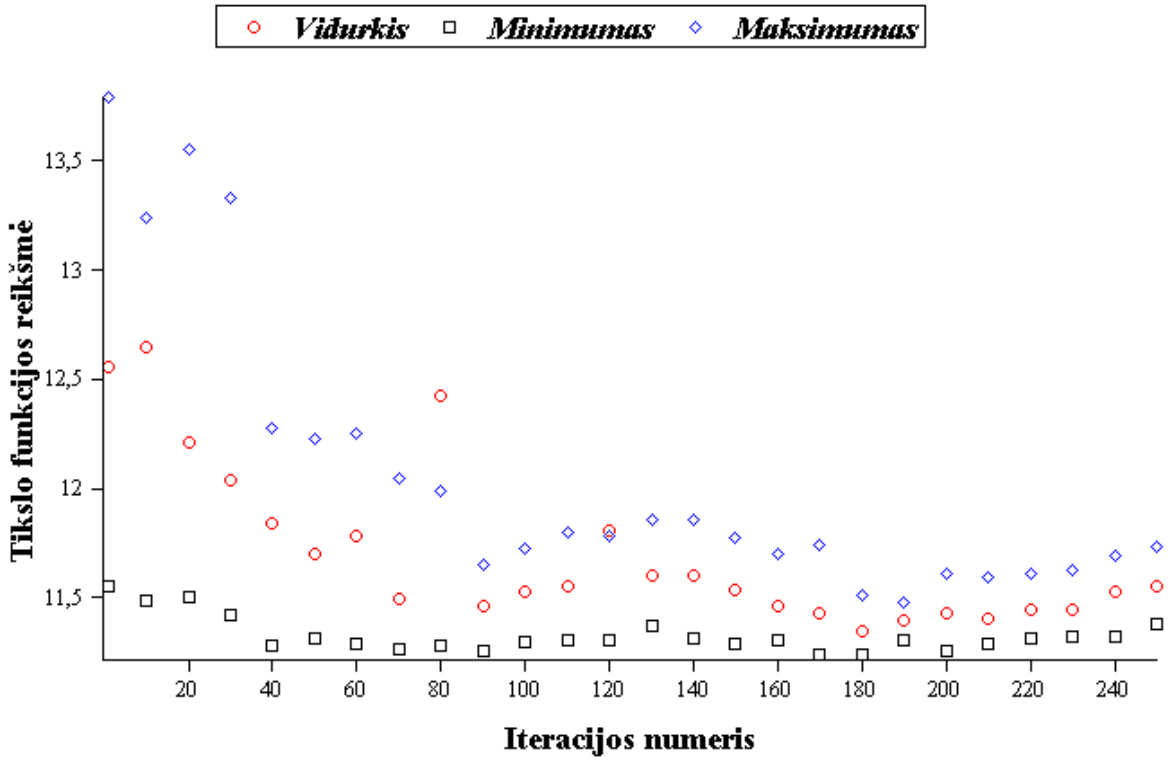
Remiantis ketvirto tyrimo metu gautais rezultatais galime teigti, kad tikslo funkcijos reikšmės yra pakankamai stabilios ir beveik nepriklauso nuo kryžminimo taškų skaičiaus. Geriausią minimalią tikslo funkcijos reikšmę gavome, kai taikėme dviejų taškų kryžminimo metodą. Dėl šios priežasties tolesniuose tyrimuose taikysime būtent šį metodą.

4.4. GA iteracijų skaičiaus tyrimas

GA evoliucijai stabdyti mūsų parašytoje programoje yra naudojamas iteracijų skaičius. Tai vienas iš svarbiausių parametrų bet kuriame optimizavimo uždavinyje. Didesnis iteracijų

skaičius padidina tikimybę rasti geresnį sprendinį, tačiau didesnis iteracijų skaičius gali reikalauti nerealių kompiuterio resursų. Be to, jei tikslo funkcija jau konvergavo, iteracijų skaičiaus didinimas tik apsunkins skaičiavimus ir neduos norimo rezultatų pagerėjimo. Šiame poskyryje tirsime tikslo funkcijos reikšmių priklausomybę nuo iteracijų skaičiaus.

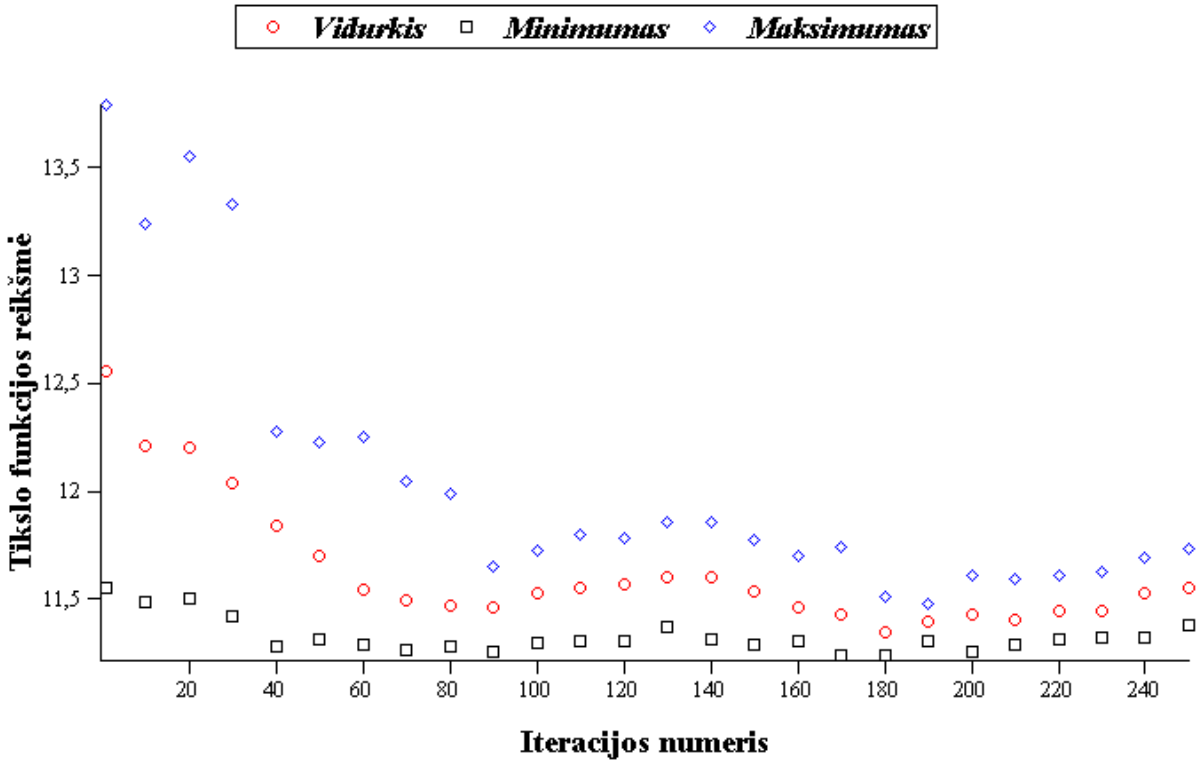
Vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių, gautų taikant ruletės atrankos metodą, priklausomybė nuo iteracijų skaičiaus pavaizduota 4.15 paveiksle.



4.15 pav. Ruletės atranka. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus

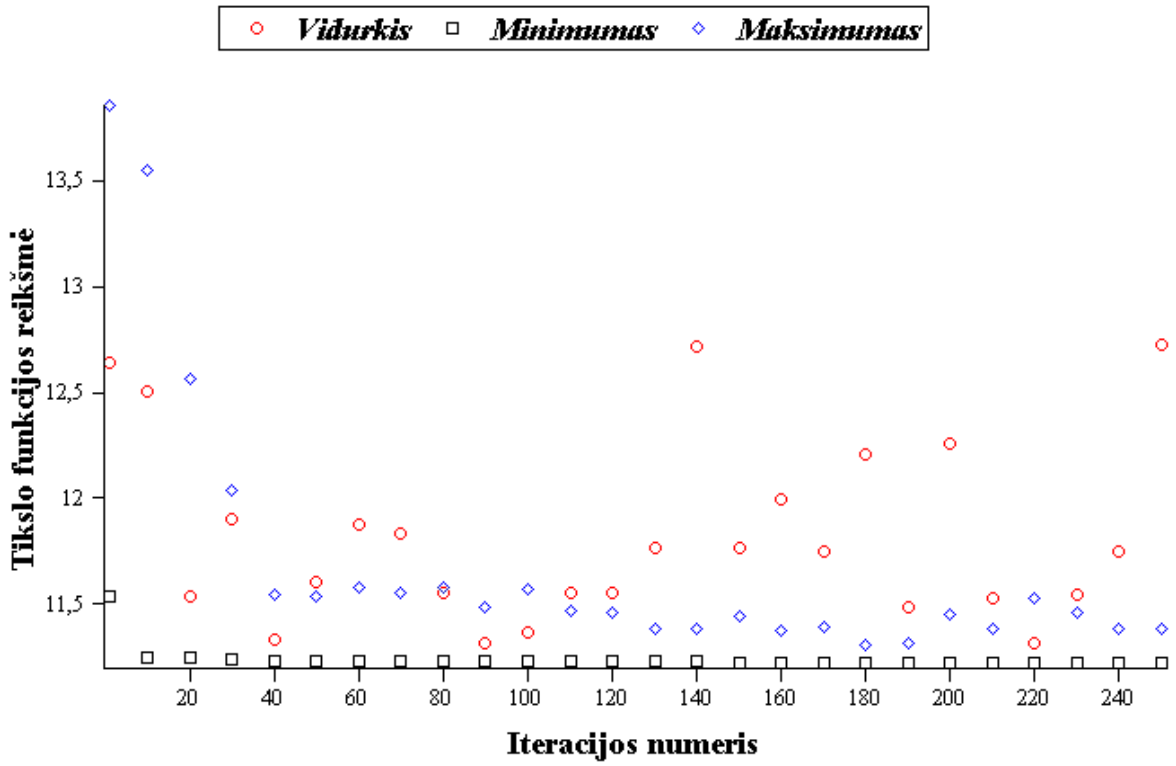
Apatinė kreivė vaizduoja mažiausias rastas tikslo funkcijos reikšmes, vidurinė – vidutines tikslo funkcijos reikšmes, viršutinė – maksimalias tikslo funkcijos reikšmes. 4.15 paveiksle pavaizdavome vieno iš eksperimentų gautus rezultatus, konvergavimą. Kitų eksperimentų rezultatai panašūs. Maždaug iki 100 iteracijos tikslo funkcijos reikšmės mažėja, o vėliau stabilizuojasi.

Kaip matome iš grafiko (4.15 pav.) vidutinės tikslo funkcijos reikšmės yra išsibarsčiusios. Taip yra dėl to, kad į vidurkio skaičiavimus įtraukiame individus, kurie netenkina maksimalaus įtempio apribojimo, o jų tikslo funkcijos reikšmės yra su bauda, t.y. padaugintos iš dviejų. Baudą pridedame tam, kad individas, kuris netenkina apribojimų, liktų populiacijoje, bet su labai maža tikimybe galėtų pereiti į kitą generaciją. Vidutinių tikslo funkcijos reikšmių, gautų atmetus individus, kurie netenkina maksimalaus įtempio apribojimo, priklausomybė nuo iteracijų skaičiaus pavaizduota 4.16 paveiksle.



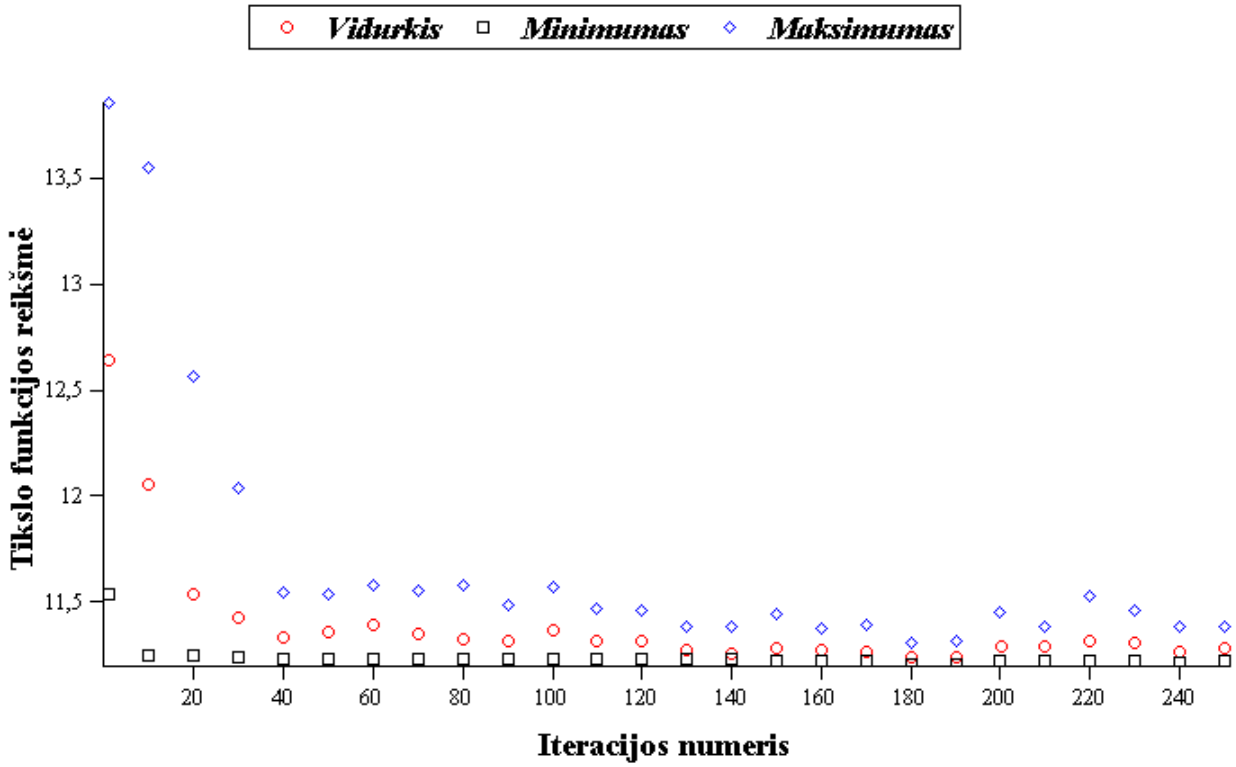
4.16 pav. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus

Vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių, gautų taikant elito atrankos metodą, priklausomybė nuo iteracijų skaičiaus pavaizduota 4.17 paveiksle.



4.17 pav. Elito atranka. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus

Kaip matome iš grafiko (4.17 pav.) minimalios tikslo funkcijos reikšmės monotoniškai mažėja skirtingai nei ankščiau nagrinėtame pavyzdyje. Kitas skirtumas – vidutinės tikslo funkcijos reikšmės yra išsibarsčiusios dar daugiau. Taip yra dėl to, kad taikant elito atrankos metodą gauname mažesnes tikslo funkcijos reikšmes ir pasitaiko daugiau individų, kurie netenkina apribojimų. Vidutinių tikslo funkcijos reikšmių, gautų atmetus individus, kurie netenkina maksimalaus įtempio apribojimo, priklausomybė nuo iteracijų skaičiaus pavaizduota 4.18 paveiksle.



4.18 pav. Tikslo funkcijos reikšmių priklausomybė nuo iteracijų skaičiaus

Iš gautų eksperimentų grafikų (4.15, 4.16, 4.17 ir 4.18) galime teigti, kad uždavinys konverguoja prie mažesnio nei 250 iteracijų skaičiaus nepriklausomai nuo atrankos metodo. Vidutiniškai geriausias vidutinis sprendinys surandamas tarp 100 ir 150 iteracijų, geriausias minimalus – tarp 150 ir 200 iteracijų, maksimalus – tarp 1 ir 10 iteracijų, kai taikome elito atrankos metodą. Geriausias minimalus sprendinys surandamas tarp 40 ir 100 iteracijų, kai taikome ruletės atrankos metodą.

Remiantis penkto tyrimo metu gautais rezultatais galime teigti, kad 250 iteracijų pakanka su atsarga tam, kad uždavinys konverguotų.

4.5. GA mutacijos strategijų tyrimas

Šiame poskyryje pirmiausia tirsime tikslo funkcijos reikšmių priklausomybę nuo mutacijos tikimybės. Kaip matėme 4.4 poskyryje tikslo funkcijos reiškės algoritmo pradžioje kinta labai greitai, o vėliau šis procesas sulėtėja. Didinsime mutacijos tikimybę nuo 100 iteracijos tikintis, kad tai turės teigiamos įtakos individų tinkamumui ir įneš naujovių į populiaciją. Atliksime po 50 nepriklausomų bandymų su kiekviena mutacijos tikimybe. 4.13 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai naudojami šeštame tyrime, o 4.14 lentelėje pateikta tikslo funkcijos reikšmių priklausomybė nuo mutacijos tikimybės.

4.13 lentelė. Šešto tyrimo GA parametrai

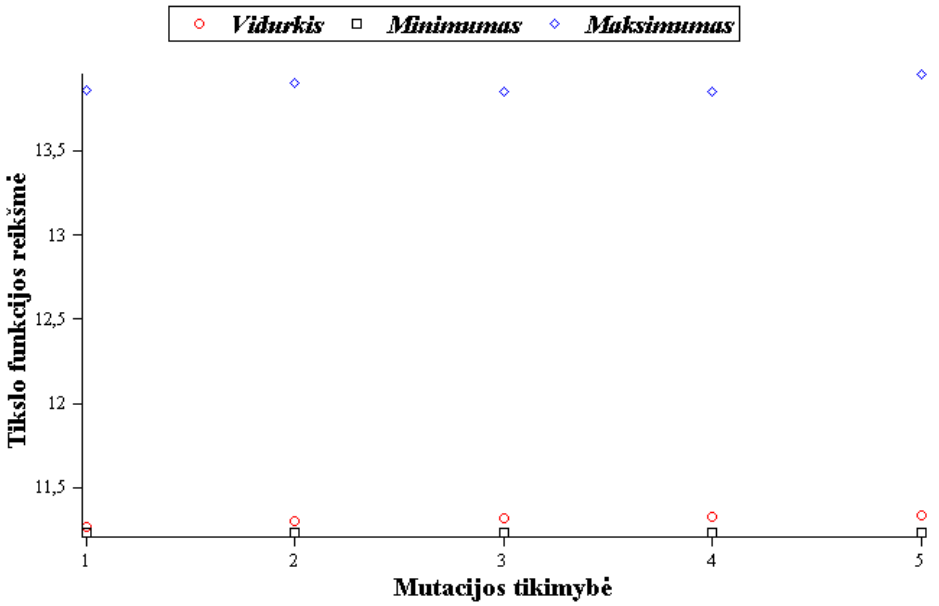
Kryžminimo tikimybė	0,8
Individų kiekis populiacijoje	46
Iteracijų skaičius	250
Kryžminimo metodas	dviejų taškų
Atrankos metodas	elito
Elitinių individų skaičius	2

4.14 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo mutacijos tikimybės

Mutacijos tikimybė	**Vidurkis**	**Minimumas**	**Maksimumas**
0,01	11,270	11,235	13,861
0,02	11,298	11,235	13,906
0,03	11,318	11,233	13,851
0,04	11,326	11,233	13,851
0,05	11,332	11,233	13,955

Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės. Grafiškai 4.14 lentelės rezultatai pavaizduoti 4.19 paveiksle. Apatinė kreivė vaizduoja antrą 4.14 lentelės stulpelį, vidurinė – pirmą 4.14 lentelės stulpelį, viršutinė – trečią 4.14 lentelės stulpelį.

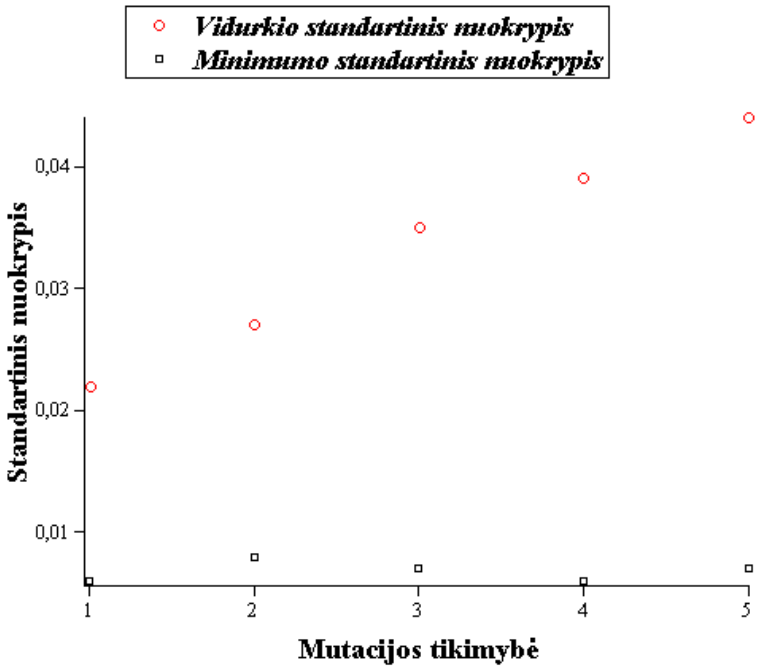
4.15 lentelėje pateikiami vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių standartiniai nuokrypiai. Pirmame lentelės stulpelyje pateikta vidutinių tikslo funkcijos reikšmių sklaida apie vidurkį, antrame – minimalių tikslo funkcijos reikšmių sklaida apie vidurkį, trečiame – maksimalių tikslo funkcijos reikšmių sklaida apie vidurkį. Grafiškai vidutinių ir minimalių tikslo funkcijos reikšmių standartiniai nuokrypiai pavaizduoti 4.20 paveiksle.



4.19 pav. Tikslo funkcijos reikšmių priklausomybė nuo mutacijos tikimybės

4.15 lentelė. Standartinio nuokrypio priklausomybė nuo mutacijos tikimybės

Mutacijos tikimybė	Vidurkio standartinis nuokrypis	Minimumo standartinis nuokrypis	Maksimumo standartinis nuokrypis
0,01	0,022	0,006	0,233
0,02	0,027	0,008	0,202
0,03	0,035	0,007	0,216
0,04	0,039	0,006	0,264
0,05	0,044	0,007	0,233



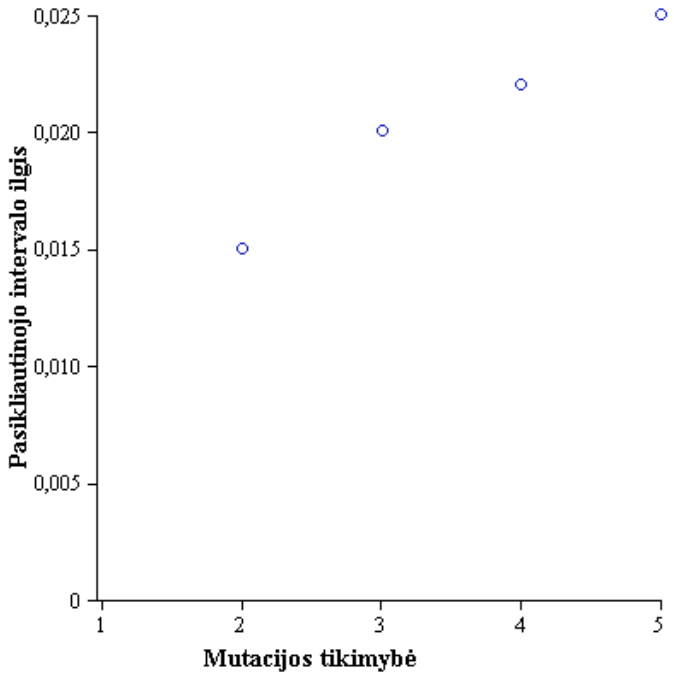
4.20 pav. Standartinio nuokrypio priklausomybė nuo mutacijos tikimybės

4.16 lentelėje pateikta vidurkio pasikliautinųjų intervalų priklausomybė nuo mutacijos tikimybės. Šiuo atveju iš testo gavome, kad eksperimentų rezultatai nėra pasiskirstę pagal normalųjį skirstinį, kai mutacijos tikimybė 1 %.

4.16 lentelė. Pasikliautinųjų intervalų priklausomybė nuo mutacijos tikimybės

Mutacijos tikimybė	Vidurkio pasikliautinasis intervalas
0,01	-
0,02	[11,290; 11,305]
0,03	[11,308; 11,328]
0,04	[11,315; 11,337]
0,05	[11,320; 11,345]

Pasikliautinųjų intervalų ilgių priklausomybė nuo mutacijos tikimybės pavaizduota 4.21 paveiksle.



4.21 pav. Pasikliautinųjų intervalų ilgių priklausomybė nuo mutacijos tikimybės

Iš šešto tyrimo gautų eksperimentų rezultatų (4.19 pav.) matome, kad minimalios ir maksimalios tikslo funkcijos reikšmės beveik nepriklauso nuo mutacijos tikimybės, o populiacijos vidurkis didėja didėjant tikimybei.

Toliau tirsime tikslo funkcijos reikšmių priklausomybę nuo δ (žr. 2.2.4 poskyrių). 4.17 lentelėje pateikiami pagrindiniai genetinio algoritmo parametrai naudojami septintame tyrime, o 4.18 lentelėje pateikta tikslo funkcijos reikšmių priklausomybė nuo δ.

4.17 lentelė. Septinto tyrimo GA parametrai

Kryžminimo tikimybė	0,8
Mutacijos tikimybė	0,01
Individų kiekis populiacijoje	46
Iteracijų skaičius	250
Kryžminimo metodas	dviejų taškų
Atrankos metodas	elito
Elitinių individų skaičius	2

4.18 lentelė. Tikslo funkcijos reikšmių priklausomybė nuo δ

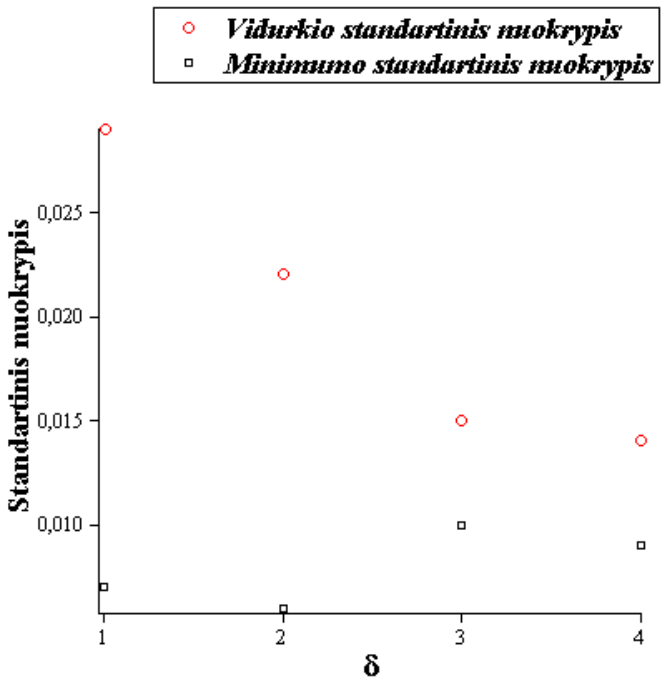
δ	**Vidurkis**	**Minimumas**	**Maksimumas**
$\boldsymbol{d/5}$	11,287	11,237	13,898
$\boldsymbol{d/10}$	11,270	11,235	13,861
$\boldsymbol{d/20}$	11,261	11,239	13,904
$\boldsymbol{d/40}$	11,253	11,240	13,850

Kiekviena šios lentelės reikšmė apskaičiuota remiantis 50 nepriklausomų eksperimentų rezultatais. Pirmame lentelės stulpelyje pateikti surastų tikslo funkcijos reikšmių vidurkiai, antrame – mažiausios reikšmės, trečiame – didžiausios reikšmės.

4.19 lentelėje pateikiami vidutinių, minimalių ir maksimalių tikslo funkcijos reikšmių standartiniai nuokrypiai. Pirmame lentelės stulpelyje pateikta vidutinių tikslo funkcijos reikšmių sklaida apie vidurkį, antrame – minimalių tikslo funkcijos reikšmių sklaida apie vidurkį, trečiame – maksimalių tikslo funkcijos reikšmių sklaida apie vidurkį. Grafiškai vidutinių ir minimalių tikslo funkcijos reikšmių standartiniai nuokrypiai pavaizduoti 4.22 paveiksle.

4.19 lentelė. Standartinio nuokrypio priklausomybė nuo δ

Eil. Nr.	δ	**Vidurkio standartinis nuokrypis**	**Minimumo standartinis nuokrypis**	**Maksimumo standartinis nuokrypis**
1	$\boldsymbol{d/5}$	0,029	0,007	0,253
2	$\boldsymbol{d/10}$	0,022	0,006	0,233
3	$\boldsymbol{d/20}$	0,015	0,010	0,291
4	$\boldsymbol{d/40}$	0,014	0,009	0,265



4.22 pav. Standartinio nuokrypio priklausomybė nuo δ

4.20 lentelėje pateikta vidurkio pasikliautinųjų intervalų priklausomybė δ.

4.20 lentelė. Pasikliautinųjų intervalų priklausomybė nuo δ

δ	**Vidurkio pasikliautinasis intervalas**
$\boldsymbol{d/5}$	[11,279; 11,295]
$\boldsymbol{d/10}$	-
$\boldsymbol{d/20}$	[11,256; 11,265]
$\boldsymbol{d/40}$	-

Remiantis septinto tyrimo metu gautais rezultatais galime teigti, kad minimalios tikslo funkcijos reikšmės yra pakankamai stabilios ir beveik nepriklauso nuo δ. Populiacijos vidurkis mažėja mažėjant δ reikšmei.

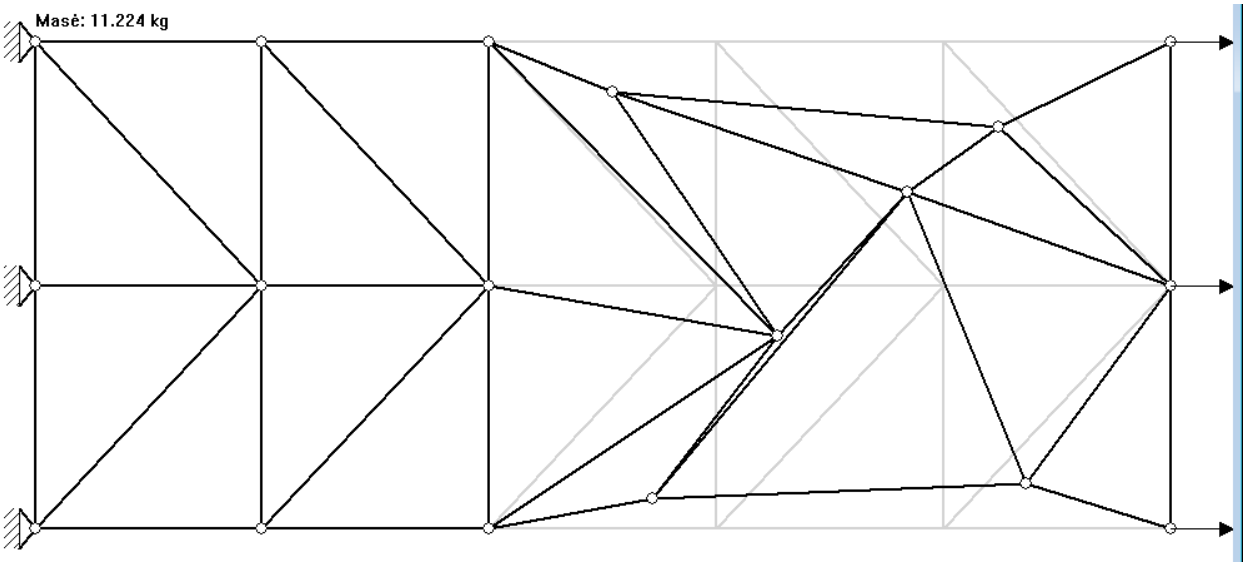
4.6. Eksperimentų apibendrinimas

Kaip jau minėjome, tiriant GA strategijas ir parametrų reikšmes iš viso buvo atlikta 4600 nepriklausomų eksperimentų. Remiantis šiais eksperimentais 4.21 lentelėje pateikiame optimalius GA parametrus bei GA strategijas, kurios leidžia efektyviausiai atlikti mechaninės struktūros masės optimizavimą.

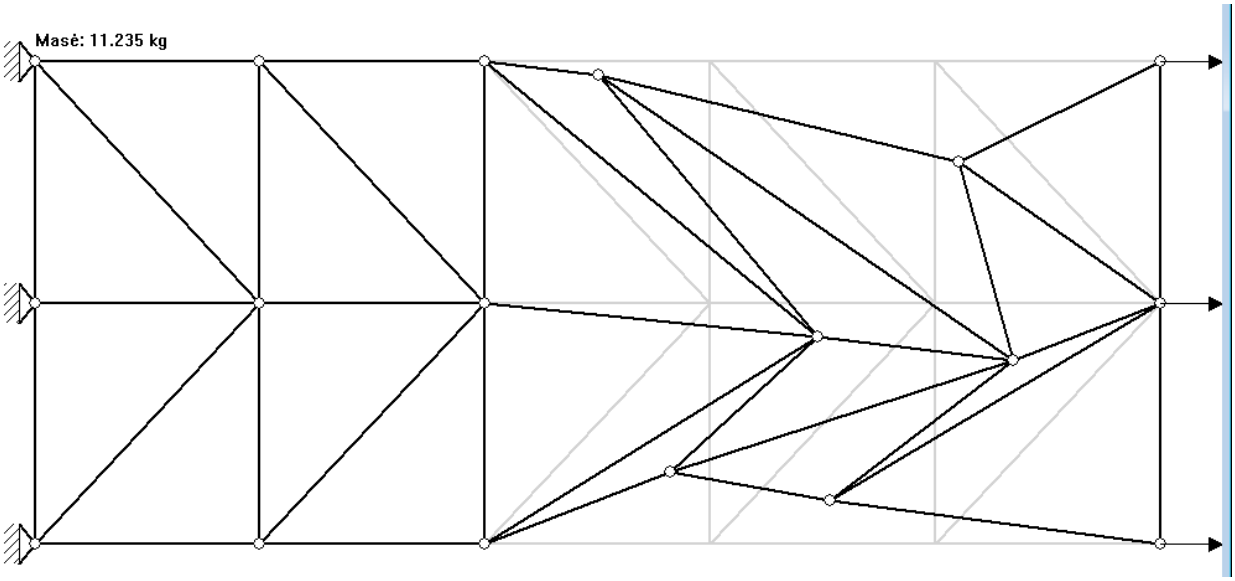
4.21 lentelė. Optimalūs GA parametrai

Kryžminimo tikimybė	0,8
Mutacijos tikimybė	0,01
Individų kiekis populiacijoje	46
Iteracijų skaičius	250
Kryžminimo metodas	dviejų taškų
Atrankos metodas	elito
Elitinių individų skaičius	2

Pasinaudoję šiuo parametrų rinkiniu radome geriausią minimalią tikslo funkcijos reikšmę 11.224 (4.23 pav.) ir geriausią vidutinę – 11.235 (4.24 pav.).



4.23 pav. Mažiausios masės strypas



4.24 pav. Artimiausias vidurkiui sprendinys.

Išvados

Kaip daugelio tyrėjų patirtis rodo, šiame darbe minėto uždavinio sprendimas reikalauja nerealių kompiuterio resursų, todėl optimizavimui buvo pasirinkti GA, kurie priskiriami stochastinių (tikimybinių) algoritmų klasei. Buvo sukurta technologija, kuri naudodama GA koncepciją leidžia atlikti mechaninės struktūros masės minimizavimą.

Autorė išanalizavo, apibendrino ir pateikė virš 4500 nepriklausomų skaitinių eksperimentų gautus rezultatus. Atliekant šiuos eksperimentus buvo pastebėta:

1. Individų skaičiaus populiacijoje didinimas leidžia tikėtis geresnio rezultato. Optimalus populiacijos dydis yra apie 32-48 individų;
2. Elito strategija leido surasti 4-7 % geresnį populiacijos vidurkį ir 1-3 % geresnį minimalų sprendinį negu ruletės atrankos strategija;
3. Pritaikius elito strategiją vidurkio ir minimumo standartiniai nuokrypiai sumažėjo apie 10 %, o vidurkio pasikliautinieji intervalai susiaurėjo apie 5 %. Tai parodo didesnį šios strategijos stabilumą;
4. Elitinių individų skaičius neturėtų sudaryti daugiau nei 5 % visos populiacijos;
5. Uždavinys konverguoja mažiau nei per 250 iteracijų;
6. Kryžminimo taškų skaičiaus padidinimas didelės įtakos mūsų sprendžiamo uždavinio rezultatams neturėjo;
7. Mutacija turi būti vykdoma su nedidele tikimybe apie 1 %;
8. GA negarantuoja, kad bus surastas globalus sprendinys. Būtina kelis kartus spręsti uždavinį su tais pačiais duomenimis.

Literatūros sąrašas

1. Barauskas, R.; Belevičius, R.; Kačianauskas, R. *Baigtinių elementų metodo pagrindai.* Vilnius: Technika, 2004. 610 p.
2. Beasley, D.; Bull, D. R.; Martin, R. R. An overview of genetic algorithms: Part 1. Fundamentals. *University Computing.* nr. 15(2) (1993), p. 58-69.
3. Beasley, D.; Bull, D. R.; Martin, R. R. An overview of genetic algorithms: Part 2. Reseach topics. *University Computing.* nr. 15(4) (1993), p. 170-181.
4. Dobilas M. *Genetinių algoritmų taikymas imituojant sistemas aprašytas agregatiniu metodu.* Kaunas, 2010.
5. Dzemyda, G.; Šaltenis, V.; Tiešis, V. *Optimizavimo metodai.* Vilnius: Mokslo aidai, 2007. 166 p.
6. *Genetic algorithm.* [žiūrėta 20013-04-29]. Prieiga per internetą: http://en.wikipedia.org/wiki/Genetic_algorithm.
7. *Genetic Algorithms and Evolutionary Computation.* [žiūrėta 20013-04-29]. Prieiga per internetą: http://www.talkorigins.org/faqs/genalg/
8. Goldberg D. E. Genetic algorithms in search, optimization, and machine learning. Addison-Wesley, 2005.
9. Goldberg, D. E.; Deb, K. *A Comparative Analysis of Selection Schemes Used in Genetic Algorithms.* San Mateo, California: Morgan Kaufmann Publishers, 1991.
10. Haupt, R. L.; Haupt, S. E. *Practical genetic algorithms*. 2 rd. ed. Hoboken, New Jersey: John Wiley & Sons, Inc., 2004. 253 p.
11. Luke, Sean. *Essentials of Metaheuristics*. Online version 1.3. Fairfax, Virginia: George Mason University, 2012 [žiūrėta 2013-04-29]. Prieiga per internetą: http://cs.gmu.edu/~sean/book/metaheuristics/.
12. Mitchell M. *An introduction to genetic algorithms.* 5 rd. ed. Cambridge, Mass.: MIT Press, 1999. 159 p.
13. Obitko M. *Introduction to Genetic Algorithms.* Czech Technical University, 1998 [žiūrėta 2013-04-29]. Prieiga per internetą: http://www.obitko.com/tutorials/genetic-algorithms/
14. Olsson, U.; Engstrand, U.; Rupšys, P. *Statistiniai metodai SAS ir MINITAB*: mokomoji knyga. Akademija, 2007. 138 p.
15. Reeves, Colin. *Genetic algorithms*. Coventry, United Kingdom: Coventry University [žiūrėta 2013-04-29]. Prieiga per internetą: http://homes.ieu.edu.tr/~agokce/Courses/.

16. Sakalauskas, L.; Felinskas, G. Tvarkaraščių su ribotais ištekliais sudarymo euristiniai algoritmai, jų tyrimas ir taikymai. *Informacijos mokslai*. nr. 38 (2006), p. 90-103. ISSN 1392-0561.
17. Šešok, D.; Belevičius, R. Globalusis rostverkinių pamatų optimizavimas hibridiniu genetiniu algoritmu. *Statybinės konstrukcijos ir technologijos*, 2009, p. 80-88. ISSN 2029-2317.
18. Vasiliauskienė L. *Prisitaikančiosios baigtinių elementų strategijos plokštuminiams tamprumo teorijos uždaviniams*: daktaro disertacija. Vilnius, 2006.
19. Weise T. *Global optimization algorithms. Theory and application.* 2 rd. ed. 2009, birželis [žiūrėta 2013-04-29]. Prieiga per internetą: http://www.it-weise.de/.
20. Панченко Т. В. *Генетические алгоритмы*. Издательский дом «Астраханский университет», 2007. ISBN 5-88200-913-8.

PRIEDAI

A priedas. Programos kodas

```
#include <windows.h>
#include "Arrow.h"
#include <iostream>
#include <stdio.h>
#include <iomanip>
#include <fstream>
#include <ctime>
#include <cstdlib>
#include <math.h>

using namespace std;

double **piesk_prad_koord = NULL;
double **piesk_susk_koord = NULL;
int **rysiai_piesk = NULL;

double mase_piesk = 0.;
char pavadinimas_piesk[120] = "";

int** br_rysiai;
double** koordinates;
double** vid_ind; double** min_ind;
double** max_ind;

int str_sk, nm;
double minbmas, maxbmas, vidbmas;

void ivedimas(double** koordinates, int** rysiai, int** br_rysiai, int** itvirtinimai, double D[][3],
              double& mtik, double** jegos, int& nm, int& ne, int& ni, int& nf, double& E,
              double& pk, double& ktik, double& t, ifstream& is, int& nmin, int& nind, int& msky,
              int& str_sk, double& max_itemp, int& elitn );
//
void pradindivid( double** koordinates, int& nmin, int& nind, int& nr, int& nm, ofstream& os,
                  double** populiacija, double** kintkoord, double& a, double& b, int& msky);
//
void BEM ( int& nm, int& ne, int** rysiai, double& E, double& pk, int& nind, int& nmin,
```

```
                int& temp, double** jegos, int& nf, double& t, int& ni, int** itvirtinimai, double D[][3],
                ofstream& op, double& M, double** kintkoord, double* atvmase, double& sum_atvmase,
                double& minmas, double& maxmas, int& nr, double& max_itemp, double& minbmas,
                double& maxbmas, int& nr_max_ind, int& nr_min_ind );
//
void GAlgoritmas ( ifstream& is, double** populiacija, double& mtik, double& a, double& b, int& nind,
                double* atvmase, double& t, double& ktik, double& sum_atvmase, int& nm, int& nmin,
                double** newpopuliacija, int** rysiai, double** koordinates, int& ne, int& msky, int&
elitn );
//
void elitatranka (double* atvmase, double** populiacija, int& elitn, int& nind, int& nmin);

void Delete()
{
        for ( int i=0; i<str_sk; i++)
        {
            delete[] br_rysiai[i];
        }
        delete[]br_rysiai;

        for ( int i=0; i<nm; i++)
        {
            delete[] koordinates[i];
        }
        delete[]koordinates;

         for ( int i=0; i<nm; i++ )
        {
            delete[] vid_ind[i];
         }
        delete[]vid_ind;

         for ( int i=0; i<nm; i++ )
        {
            delete[] min_ind[i];
        }
        delete[]min_ind;
```

```
            for ( int i=0; i<nm; i++ )
            {
                delete[] max_ind[i];
             }
            delete[]max_ind;
}
void Skaiciuok()
{
                Delete();

                int ne, ni, nf, nmin, nind, msky, elitn;
                double** kintkoord; double** jegos;
                double** populiacija; double** newpopuliacija;
                 double* atvmase; double sum_atvmase;
                double a, b, M, minsk, skirtumas;
                double minmas, mtik, ktik, maxmas;
                int** rysiai; int** itvirtinimai;
                double E, t, pk, max_itemp;
                double D[3][3];
                int temp, nr_vid_ind, nr_min_ind, nr_max_ind, l;
                double pradindsum, pradvidmas;
                ifstream is; ofstream os, op, om, vid, mases, min, max;
                int itmax, itmin;
//
            is.open("Duomenys_1.txt");
            if (is.fail())
            {
              cout<<" ***Failas nerastas!***\n";
              exit(1);
            }
            srand(time(NULL));
//
            is>>nm>>ne>>ni>>nf>>nmin>>nind>>str_sk;

            koordinates= new double* [nm];
            for ( int i=0; i<nm; i++)
            {
               koordinates[i]= new double [2];
```

```
}
rysiai= new int* [ne];
for ( int i=0; i<ne; i++)
{
   rysiai[i]= new int [3];
}
br_rysiai= new int* [str_sk];
for ( int i=0; i<str_sk; i++)
{
   br_rysiai[i]= new int [2];
}
itvirtinimai= new int* [ni];
for ( int i=0; i<ni; i++)
{
   itvirtinimai[i]= new int [3];
}
jegos= new double* [nf];
for (int i=0; i<nf; i++)
{
   jegos[i]= new double [3];
}
populiacija= new double* [nind];
for ( int i=0; i<nind; i++ )
{
populiacija[i]= new double [nmin*2+1];
}
newpopuliacija= new double* [nind];
for ( int i=0; i<nind; i++)
{
   newpopuliacija[i]= new double [nmin*2+1];
}
kintkoord= new double* [nm];
for ( int i=0; i<nm; i++ )
{
   kintkoord[i]= new double [2];
}
vid_ind= new double* [nm];
for ( int i=0; i<nm; i++ )
```

```
{
   vid_ind[i]= new double [2];
}
min_ind= new double* [nm];
for ( int i=0; i<nm; i++ )
{
   min_ind[i]= new double [2];
}
max_ind= new double* [nm];
for ( int i=0; i<nm; i++ )
{
   max_ind[i]= new double [2];
}
atvmase= new double [nind];

sum_atvmase=0;
minmas=1000;
maxmas=0.1;
minbmas=1000;
maxbmas=0.1;
vidbmas=1000;
//
ivedimas(koordinates, rysiai, br_rysiai, itvirtinimai, D, mtik, jegos, nm, ne, ni, nf, E, pk, ktik, t,
      is, nmin, nind, msky, str_sk, max_itemp, elitn);
for (int i=0; i<nm; i++)
{
   for (int j=0; j<2; j++)
   {
      kintkoord[i][j]=koordinates[i][j];
   }
}
for (int i=0; i<nm; i++)
{
   for (int j=0; j<2; j++)
   {
      vid_ind[i][j]=koordinates[i][j];
   }
}
```

```
            for (int i=0; i<nm; i++)
            {
               for (int j=0; j<2; j++)
               {
                  min_ind[i][j]=koordinates[i][j];
               }
            }
            for (int i=0; i<nm; i++)
            {
               for (int j=0; j<2; j++)
               {
                  max_ind[i][j]=koordinates[i][j];
               }
            }
//
            int nr=0;
            pradindsum=0;

            do
            {
              temp=0;
              pradindivid( koordinates, nmin, nind, nr, nm, os, populiacija, kintkoord, a, b, msky);
              BEM ( nm, ne, rysiai, E, pk, nind, nmin, temp, jegos, nf, t, ni, itvirtinimai, D, op, M,
                    kintkoord, atvmase, sum_atvmase, minmas, maxmas, nr, max_itemp, minbmas,
          maxbmas,
                   nr_max_ind, nr_min_ind );
              if (temp==0)
              {
                  pradindsum+=M;
                  nr++;
              }else
              {
                   sum_atvmase-=atvmase[nr];
              }
            }while (nr!=nind);

            pradvidmas=pradindsum/nind;
//
```

```
double minvid=1000;  int itvid=0;
for ( int iter=0; iter<250; iter++ )
{
        minmas=1000; maxmas=0.1;

       elitatranka (atvmase, populiacija, elitn, nind, nmin);
        GAlgoritmas ( is, populiacija, mtik, a, b, nind, atvmase, t, ktik, sum_atvmase, nm, nmin,
                    newpopuliacija, rysiai, koordinates, ne, msky, elitn );
        sum_atvmase=0;
        double sum=0;
        for (int nr=0; nr<nind; nr++)
        {
            l=nm-nmin-msky;
            for (int i=1; i<nmin*2+1; i++)
            {
              if (i%2!=0)
              {
                    kintkoord[l][0]=newpopuliacija[nr][i];
              }
              if (i%2==0)
              {
                 kintkoord[l][1]=newpopuliacija[nr][i];
                 l++;
               }
           }
          BEM ( nm, ne, rysiai, E, pk, nind, nmin, temp, jegos, nf, t, ni, itvirtinimai, D, op, M,
                     kintkoord, atvmase, sum_atvmase, minmas, maxmas, nr, max_itemp,
     minbmas,
                    maxbmas, nr_max_ind, nr_min_ind );
          sum+=M;
 }

 double vidmas=sum/nind;
 minsk=1000;
 if (minvid>vidmas)
 {
          minvid=vidmas;
          itvid=iter+1;
```

```
            for (int i=0; i<nind; i++)
            {
                  skirtumas=fabs(1/atvmase[i]-minvid);
                  if (minsk>skirtumas)
                  {
                     minsk=skirtumas;
                     nr_vid_ind=i;
                     vidbmas=1/atvmase[nr_vid_ind];
                  }
            }
            l=nm-nmin-msky;
            for (int i=1; i<nmin*2+1; i++)
            {
                  if (i%2!=0)
                  {
                    vid_ind[l][0]=newpopuliacija[nr_vid_ind][i];
                  }
                  if (i%2==0)
                  {
                    vid_ind[l][1]=newpopuliacija[nr_vid_ind][i];
                    l++;
                  }
   }
}

          if (minbmas>minmas)
          {
             minbmas=minmas;
             itmin=iter+1;
             l=nm-nmin-msky;
            for (int i=1; i<nmin*2+1; i++)
            {
                  if (i%2!=0)
                  {
                    min_ind[l][0]=newpopuliacija[nr_min_ind][i];
                  }
                  if (i%2==0)
                  {
```

```
                    min_ind[l][1]=newpopuliacija[nr_min_ind][i];
                    l++;
                  }
                }
              }
              if (maxbmas<maxmas)
              {
                maxbmas=maxmas;
                itmax=iter+1;
                l=nm-nmin-msky;
                for (int i=1; i<nmin*2+1; i++)
                {
                  if (i%2!=0)
                  {
                    max_ind[l][0]=newpopuliacija[nr_max_ind][i];
                  }
                  if (i%2==0)
                  {
                    max_ind[l][1]=newpopuliacija[nr_max_ind][i];
                    l++;
                  }
                }
              }
            }
          is.close();
//
          for ( int i=0; i<ne; i++)
          {
            delete[] rysiai[i];
          }
          delete[]rysiai;

          for ( int i=0; i<ni; i++)
          {
            delete[] itvirtinimai[i];
          }
          delete[]itvirtinimai;
```

```
 for ( int i=0; i<nf; i++)
 {
    delete[] jegos[i];
 }
 delete[]jegos;

 for ( int i=0; i<nind; i++ )
 {
    delete[] populiacija[i];
 }
 delete[]populiacija;

 for ( int i=0; i<nind; i++)
 {
    delete[] newpopuliacija[i];
 }
 delete[]newpopuliacija;

 for ( int i=0; i<nm; i++ )
 {
    delete[] kintkoord[i];
 }
 delete[]kintkoord;

 delete[]atvmase;
}
static const int OFFSET = 50;
static const int APSKRITIMO_SKERSMUO = 5;
static const int APSKRITIMO_STORIS = 1;
static const int LINIJOS_STORIS = 2;
static const COLORREF PILKA = RGB(211,211,211);
static const COLORREF JUODA = RGB(0,0,0);

double MastelisX(int width)
{
        double dMin = 1e10;
        double dMax = -1e10;
```

```
	for(int i = 0; i < nm; i++)
	{
			dMin = dMin > piesk_susk_koord[i][0] ? piesk_susk_koord[i][0] : dMin;
			dMax = dMax > piesk_susk_koord[i][0] ? dMax : piesk_susk_koord[i][0];
	}

	return width / (dMax - dMin);
}

double MastelisY(int width, double &dMin)
{
	dMin = 1e10;
	double dMax = -1e10;

	for(int i = 0; i < nm; i++)
	{
	 dMin = dMin > piesk_susk_koord[i][1] ? piesk_susk_koord[i][1] : dMin;
			dMax = dMax > piesk_susk_koord[i][1] ? dMax : piesk_susk_koord[i][1];
	}

	return width / (dMax - dMin);
}

void Draw(HDC hdc, RECT &rc)
{
	if(NULL == piesk_prad_koord || NULL == piesk_susk_koord ||
	NULL == rysiai_piesk || 0 == nm || 0 == str_sk )
	return;

   double dMinY;
   const double mastelisX = MastelisX(rc.right-rc.left);
   const double mastelisY = MastelisY(rc.bottom-rc.top, dMinY);

   int startX = rc.left;
   int startY = -rc.bottom - dMinY * mastelisY;

	HGDIOBJ oldPen=SelectObject(hdc,CreatePen(PS_SOLID, LINIJOS_STORIS, PILKA));
```

```
	for(int i = 0; i < str_sk; i++)
	{
			MoveToEx(hdc,
				startX+piesk_prad_koord[rysiai_piesk[i][0]-1][0]* mastelisX,
				startY+piesk_prad_koord[rysiai_piesk[i][0]-1][1]* mastelisY,
				NULL);

			LineTo(hdc,
				startX+piesk_prad_koord[rysiai_piesk[i][1]-1][0]* mastelisX,
				startY+piesk_prad_koord[rysiai_piesk[i][1]-1][1]*
mastelisY);
	}

   SelectObject(hdc,CreatePen(PS_SOLID, LINIJOS_STORIS, JUODA));
	for(int i = 0; i < str_sk; ++i)
	{
			MoveToEx(hdc,
				startX+piesk_susk_koord[rysiai_piesk[i][0]-1][0]*mastelisX,
				startY+piesk_susk_koord[rysiai_piesk[i][0]-1][1]*mastelisY,
				NULL);

			LineTo(hdc,
				startX+piesk_susk_koord[rysiai_piesk[i][1]-1][0]*mastelisX,
				startY+piesk_susk_koord[rysiai_piesk[i][1]-
1][1]*mastelisY);
	}

	for(int i = 0; i < 3; ++i)
	{
			SupportIn(hdc,
				startX+piesk_susk_koord[i][0]* mastelisX-OFFSET/2,
				startY+piesk_susk_koord[i][1]* mastelisY-OFFSET/3,
				startX+piesk_susk_koord[i][0]* mastelisX,
				startY+piesk_susk_koord[i][1]* mastelisY+OFFSET/3);
	}

	SelectObject(hdc,CreatePen(PS_SOLID, APSKRITIMO_STORIS, JUODA));
	for(int i = 0; i < nm; ++i)
```

```
{
		Ellipse(hdc,
				startX+piesk_susk_koord[i][0]*mastelisX-
APSKRITIMO_SKERSMUO,
				startY+piesk_susk_koord[i][1]*mastelisY-
APSKRITIMO_SKERSMUO,
	startX+piesk_susk_koord[i][0]*mastelisX+APSKRITIMO_SKERSMUO,

	startY+piesk_susk_koord[i][1]*mastelisY+APSKRITIMO_SKERSMUO);
	}

	HGDIOBJ oldBrush=SelectObject(hdc,CreateSolidBrush(JUODA));

	for(int i = 0; i < 3; ++i)
	{
		ARROWSTRUCT arrow;
		arrow.nWidth = 10;
		arrow.fTheta = 45. / 180. * 3.14;
		arrow.bFill = true;

		MoveToEx( hdc,
				startX+piesk_susk_koord[nm-1-i][0]*mastelisX,
				startY+piesk_susk_koord[nm-1-i][1]*mastelisY,
				NULL);

		ArrowTo(
				hdc,
				startX+piesk_susk_koord[nm-1-i][0]*mastelisX+OFFSET,
				startY+piesk_susk_koord[nm-1-i][1]*mastelisY,
				&arrow);
	}

	char buffer [50];
	long nLength = sprintf (buffer, "Masė: %.3f kg", mase_piesk);

	TextOut(
		hdc,
		startX,
```

```
                    startY- OFFSET / 2,
                    buffer,
                    nLength);

        SetTextAlign(hdc, TA_CENTER);
        TextOut( hdc,
                    (rc.left + rc.right) / 2,
                    -rc.top + OFFSET / 2,
                    pavadinimas_piesk,
                    strlen(pavadinimas_piesk));

        SelectObject(hdc,oldPen);
        SelectObject(hdc,oldBrush);
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
   switch (uMsg)
   {
   case WM_DESTROY:
        Delete();
      PostQuitMessage(0);
      return 0;

   case WM_SIZE:
        {
                    InvalidateRect(hwnd, NULL, TRUE);
                    break;
        }

   case WM_PAINT:
      {
         PAINTSTRUCT ps;
         HDC hdc = BeginPaint(hwnd, &ps);

                                RECT rect;
                                GetClientRect(hwnd, &rect);
```

```
                    FillRect(hdc, &rect, (HBRUSH) RGB(255,255,255));
                    SetWindowExtEx(hdc, rect.right, rect.bottom, NULL);
                    SetViewportOrgEx(hdc, 0, rect.bottom, NULL);

                    rect.left   += OFFSET;
                    rect.top    += OFFSET;
                    rect.right  -= OFFSET;
                    rect.bottom -= OFFSET;

                    Draw(hdc, rect);

        EndPaint(hwnd, &ps);
        break;
    }
case WM_KEYUP:
    {
            switch(wParam)
            {
               case '1':
                        piesk_prad_koord  = koordinates;
                        piesk_susk_koord = vid_ind;
                        rysiai_piesk      = br_rysiai;
                        mase_piesk        = vidbmas;
                        strcpy(pavadinimas_piesk, "Vidutinis sprendinys");
                        break;
               case '2':
                        piesk_prad_koord  = koordinates;
                        piesk_susk_koord          = min_ind;
                        rysiai_piesk      = br_rysiai;
                        mase_piesk        = minbmas;
                        strcpy(pavadinimas_piesk, "Minimalus sprendinys");
                        break;
               case '3':
                        piesk_prad_koord  = koordinates;
                        piesk_susk_koord          = max_ind;
                        rysiai_piesk      = br_rysiai;
                        mase_piesk        = maxbmas;
```

```
                                    strcpy(pavadinimas_piesk, "Maksimalus sprendinys");
                                    break;
                            default:
                                    piesk_prad_koord    = NULL;
                                    piesk_susk_koord            = NULL;
                                    rysiai_piesk                = NULL;
                                break;
                        }
                        InvalidateRect(hwnd, NULL, TRUE);
                        break;
        }


    }
    return DefWindowProc(hwnd, uMsg, wParam, lParam);
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
{
    // Register the window class.
    const char CLASS_NAME[]  = "Sample Window Class";

    WNDCLASS wc = { };

    wc.lpfnWndProc   = WindowProc;
    wc.hInstance     = hInstance;
    wc.lpszClassName = CLASS_NAME;

    RegisterClass(&wc);

    // Create the window.

    HWND hwnd = CreateWindowEx(
        0,                      // Optional window styles.
        CLASS_NAME,             // Window class
        "Brezinys",    // Window text
        WS_OVERLAPPEDWINDOW,            // Window style
```

```
            // Size and position
            CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

            NULL,       // Parent window
            NULL,       // Menu
            hInstance,  // Instance handle
            NULL        // Additional application data
            );

          if (hwnd == NULL)
          {
             return 0;
          }
         ShowWindow(hwnd, nCmdShow);
         Skaiciuok();
         MSG msg = { };
         while (GetMessage(&msg, NULL, 0, 0))
          {
             TranslateMessage(&msg);
             DispatchMessage(&msg);
          }
        return 0;
}
void ivedimas(double** koordinates, int** rysiai, int** itvirtinimai, double D[][3], double& mtik,
        double** jegos, int& nm, int& ne, int& ni, int& nf, double& E, double& pk, double& ktik,
        double& t, ifstream& is, int& nmin, int& nind, int& msky, double& max_itemp){

        int i, j;
        is>>E>>pk>>t>>msky>>max_itemp>>ktik>>mtik;

         cout<<endl;
        cout<<setw(44)<<"Duomenys"<<endl<<endl;
cout<<"\t*************************************************************"<<endl;
cout<<"\t* Mazgai                 *    "<<nm<<"                *"<<endl;
cout<<"\t* CST elementai          *    "<<ne<<"                *"<<endl;
cout<<"\t* Itvirtinimai           *    "<<ni<<"                *"<<endl;
cout<<"\t* Jegos                  *    "<<nf<<"                *"<<endl;
```

```
cout<<"\t*****************************************************************"<<endl;
cout<<"\t* E                        *    "<<E<<"            *"<<endl;
cout<<"\t* Puasono koef. pk          *    "<<pk<<"               *"<<endl;
cout<<"\t* Elemento storis t         *    "<<t<<"               *"<<endl;
cout<<"\t*****************************************************************"<<endl; cout<<"\t*
Minimizuojamu mazgu sk.     *    "<<nmin<<"                  *"<<endl;
cout<<"\t* Populiacijos individu sk.   *    "<<nind<<"                *"<<endl;
cout<<"\t* Mazgu sk. vertikaliai       *    "<<msky<<"                *"<<endl;
cout<<"\t* Max el. itempimas          *    "<<max_itemp<<"              *"<<endl;
cout<<"\t*****************************************************************"<<endl;
cout<<"\t* Kryzminimo tik.            *    "<<ktik<<"                *"<<endl;
cout<<"\t* Mutacijos tik.             *    "<<mtik<<"                *"<<endl;
cout<<"\t*****************************************************************"<<endl;

          for ( i=0; i<nm; i++){
             for ( j=0; j<2; j++){
                is>> koordinates[i][j];
             }
          }
cout<<"\t* Koordinates"<<setw(19)<<"*"<<setw(5);
for ( i=0; i<nm; i++){
                cout<<setw(8);
                for ( j=0; j<2; j++){
                                cout<<koordinates[i][j]<<setw(8);
                }
                 cout<<setw(12)<<"*"<<endl<<"\t*"<<setw(31)<<"*";
}
cout<<setw(28)<<"*"<<endl;
cout<<"\t*****************************************************************"<<endl;
//
  for ( i=0; i<ne; i++){
             for ( j=0; j<3; j++){
                is>> rysiai[i][j];
             }
  }
cout<<"\t* Rysiai"<<setw(24)<<"*"<<setw(8);
 for ( i=0; i<ne; i++){
                cout<<setw(8);
```

```
                for ( j=0; j<3; j++){
                                cout<<rysiai[i][j]<<setw(6);
                }
                cout<<setw(8)<<"*"<<endl<<"\t*"<<setw(31)<<"*";
}
cout<<setw(28)<<"*"<<endl;
          cout<<"\t*****************************************************************"<<endl;
//
          for ( i=0; i<ni; i++){
             for ( j=0; j<3; j++){
                is>> itvirtinimai[i][j];
             }
           }
cout<<"\t* Itvirtinimai"<<setw(18)<<"*"<<setw(5);
for ( i=0; i<ni; i++){
                cout<<setw(8);
                for ( j=0; j<3; j++){
                cout<< itvirtinimai[i][j]<<setw(6);
                }
                cout<<setw(8)<<"*"<<endl<<"\t*"<<setw(31)<<"*";
}
cout<<setw(28)<<"*"<<endl;
          cout<<"\t*****************************************************************"<<endl;
//
for ( i=0; i<nf; i++){
             for ( j=0; j<3; j++){
                is>> jegos[i][j];
             }
          }
          cout<<"\t* Jegos"<<setw(25)<<"*"<<setw(5);
          for ( i=0; i<nf; i++){
                                cout<<setw(8);

                                for ( j=0; j<3; j++){
                                cout<< jegos[i][j]<<setw(8);
                                }
                                cout<<setw(4)<<"*"<<endl<<"\t*"<<setw(31)<<"*";
}
```

```
        cout<<setw(28)<<"*"<<endl;
cout<<"\t*************************************************************"<<endl<<endl;
 cout<<"\t*** Rezultatu ieskokite faile 'Rezultatai.txt'! ***"<<endl<<endl;

        for ( i=0; i<3; i++){
            for ( j=0; j<3; j++){
               is>>D[i][j];
            }
         }
}
void pradindivid( double** koordinates, int& nmin, int& nind, int& nr, int& nm, ofstream& oz,
            double** populiacija, double** kintkoord, double& a, double& b, int& msky){

             double m, x, y, ykr;
             int l=nm-nmin-msky;
             a=(koordinates[msky][0]-koordinates[0][0])/2;
             b=(koordinates[1][1]-koordinates[0][1])/2;
             ofstream os;
             ofstream ok;

             for(int j=0; j<nmin*2+1; j++){
                  if(j%2!=0){os.open ("Pradinis individas.txt",ios::app);
                        m=rand()%1000;
                        x=-a+m/(999./(a*2.));
                        populiacija[nr][j]=koordinates[l][0]+x;
                        kintkoord[l][0]=populiacija[nr][j];
                        ykr=sqrt((1-pow(x,2)/pow(a,2))*pow(b,2));
                  }
                  if(j%2==0 && j!=0 ){
                        if (ykr==0){
                              populiacija[nr][j]=koordinates[l][1];
                              kintkoord[l][1]=populiacija[nr][j];
                              l++;
                        } else {

                              do{
                                    m=rand()%1000;
                                    y=-b+m/(999./(b*2.));
```

```
                        }while( y > ykr || y < -ykr);
                        populiacija[nr][j]=koordinates[l][1]+y;
                        kintkoord[l][1]=populiacija[nr][j];
                        l++;
                    }
                }

                if (j==0)
                    populiacija[nr][j]=nr+1;
            }
}
void tikslofunk (double& t, double** kintkoord, int** rysiai, int& ne, int& nr, double& M,
        double* atvmase, double& sum_atvmase, double& minmas, int& nm, ofstream& oz);

void ansamblis(double Ke[][6], int& nm, int& ne, int** rysiai, double** kintkoord,
        double& E, double& pk, double** jegos, double* Fa, int& nf, double** K, double& t);

void krastsalygos(double** K, int& ni, int** itvirtinimai, int& nm);

void gauso (double** K, double* Fa, int& nm, ofstream& op);

void sandaugam (double B[][6], double& E, double& pk, double u[][1], double itemp[][1], double D[][3],
        int** rysiai, double** kintkoord, double* Fa, int& ne, double& t, double* elitemp);

void BEM ( int& nm, int& ne, int** rysiai, double& E, double& pk, int& nind, int& nmin, ofstream& oz,
        int& temp, double** jegos, int& nf, double& t, int& ni, int** itvirtinimai, double D[][3],
        ofstream& op, double& M, double** kintkoord, double* atvmase, double& sum_atvmase,
        double& minmas, int& nr, double& max_itemp ){

   double Ke[6][6]; double B[3][6]; double u[6][1];
   double itemp[3][1];
   double* Fa; double* elitemp;
   double** K;
   int g=0;

   Fa= new double [nm*2];

K= new double* [nm*2];
```

```
for (int i=0; i<nm*2; i++){
      K[i]= new double [nm*2];
}

elitemp= new double [ne];
tikslofunk (t, kintkoord, rysiai, ne, nr, M, atvmase, sum_atvmase, minmas, nm, oz);

ansamblis( Ke, nm, ne, rysiai, kintkoord, E, pk, jegos, Fa, nf, K, t);

krastsalygos(K, ni, itvirtinimai, nm);

gauso (K, Fa, nm, op);

sandaugam (B, E, pk, u, itemp, D, rysiai, kintkoord, Fa, ne, t, elitemp);

   for (int i=0; i<ne; i++){
      if ( elitemp[i]>max_itemp ){
          g++;
      }
   }
   if (g!=0){
            atvmase[nr]/=100;
            M*=100;
            cout<<" Elemento itempimas virsija max_itemp "<<nr<<" "<<g<<endl;
            temp++;
   }
   if ( minmas > M){
         minmas=M;
   }
   delete[]Fa;

          for (int i=0; i<nm*2; i++){
                  delete[] K[i];
          }
          delete[]K;

   delete[]elitemp;
}
```

```
void tikslofunk (double& t, double** kintkoord, int** rysiai, int& ne, int& nr, double& M,
            double* atvmase, double& sum_atvmase, double& minmas, int& nm, ofstream& oz ){
      double trm, trpl;
                  double b[3],c[3];
                  int n;
      ofstream os;
                  M=0;

                  os.open ("Minimalios mases strypas.txt", ios::app);

for( n=0; n < ne; n++){
          int i= rysiai[n][0], j= rysiai[n][1], k= rysiai[n][2];
          double xi= kintkoord[i-1][0],
                                    yi= kintkoord[i-1][1],
                                    xj= kintkoord[j-1][0],
                                    yj= kintkoord[j-1][1],
                                    xk= kintkoord[k-1][0],
                                    yk= kintkoord[k-1][1];

                    b[0]=yj-yk;
                    b[1]=yk-yi;
                    b[2]=yi-yj;
                    c[0]=xk-xj;
          c[1]=xi-xk;
                    c[2]=xj-xi;
                    trpl=(xj*yk+xi*yj+yi*xk-xj*yi-xi*yk-xk*yj)/2.;
                    trm=trpl*t*7800.;
                    M+=trm;
  }
     atvmase[nr]=1/M;
     sum_atvmase+=atvmase[nr];
    oz<<M<<endl;
     os.close();
}
//
void stm (double Ke[][6], double& E, double& pk,double& t, double** kintkoord, int** rysiai, int& nr);
//
```

```
void ansamblis(double Ke[][6], int& nm, int& ne, int** rysiai, double** kintkoord, double& E, double&
        pk, double** jegos, double* Fa, int& nf, double** K, double& t){

            int i,j, nr, il, lnr, el, ig, sl, lnrs, jl, jg, n;

            for( i=0; i<nm*2; i++){
                        for( j=0; j<nm*2; j++){
                        K[i][j]=0;
      }
   }
            for( nr=0; nr<ne; nr++){
                        stm(Ke, E, pk,t, kintkoord, rysiai, nr);
                        il=0;
                        for(int e=0; e<3; e++){
                                    lnr=(rysiai[nr][e]-1)*2;
                                    for( el=0; el<2; el++){
                                                il+=1;
                                                ig=lnr+el;
                                                jl=0;
                                                for(int s=0; s<3; s++){
                                                            lnrs=(rysiai[nr][s]-1)*2;
                                                            for( sl=0; sl<2; sl++){
                                                                        jl++;
                                                                        jg=lnrs + sl;
                                                                K[ig][jg]+=Ke[il-1][jl-1];
                                                            }
                                                }
                                    }
                        }
            }

            for( i=0; i<nm*2; i++){
                        Fa[i]=0;
            }

            for( i=0; i<nf; i++){

                        n=static_cast<int>((jegos[i][0]-1)*2);
```

```
                    Fa[n]= jegos[i][1];
                    Fa[n+1]= jegos[i][2];
            }
}
void stm (double Ke[][6], double& E, double& pk, double& t, double** kintkoord, int** rysiai, int& nr){

            double kof,trpl;
            double b[3],c[3];
            int n,m;

            int i= rysiai[nr][0], j= rysiai[nr][1], k= rysiai[nr][2];
            double xi= kintkoord[i-1][0],
                    yi= kintkoord[i-1][1],
                    xj= kintkoord[j-1][0],
                    yj= kintkoord[j-1][1],
                    xk= kintkoord[k-1][0],
                    yk= kintkoord[k-1][1];
            b[0]=yj-yk;
            b[1]=yk-yi;
            b[2]=yi-yj;
            c[0]=xk-xj;
            c[1]=xi-xk;
            c[2]=xj-xi;

            trpl=2.*(xj*yk+xi*yj+yi*xk-xj*yi-xi*yk-xk*yj);
            kof=E*t/(trpl*(1-pk*pk));

            m=0; n=0;
            for( i=0; i<3; i++){
                    for(int j=0; j<3; j++){
                            Ke[m][n]=kof*(b[i]*b[j]+(1-pk)/2.*c[i]*c[j]);
                            Ke[m][n+1]=kof*(pk*b[i]*c[j]+(1-pk)/2.*c[i]*b[j]);
                            Ke[m+1][n]=kof*(pk*c[i]*b[j]+(1-pk)/2.*b[i]*c[j]);
                            Ke[m+1][n+1]=kof*(c[i]*c[j]+(1-pk)/2.*b[i]*b[j]);
                            n+=2;
                    }
                    n=0;
                    m+=2;
```

```
		}
}
void krastsalygos(double** K, int&ni, int** itvirtinimai, int& nm){

	int i,n,k;
	k=0;
	for( i=0; i<ni; i++){
		n=(itvirtinimai[i][0]-1)*2;
		if (itvirtinimai[i][1]==1){
				K[n][n]*= 1e6;
		}
		if (itvirtinimai[i][2]==1){
				K[n+1][n+1]*= 1e6;
		}
	}
}
void gauso (double** K, double* Fa, int& nm, ofstream& os ){

	int k, i,j;
	double temp;

	for( k=0; k<nm*2; k++){
		//Tikrinamas pagrindinis elementas
		if(K[k][k]==0){
		    temp=0;
				for( i=k+1;i<nm*2; i++){
					if(fabs(K[i][k])>temp){
						temp=fabs(K[i][k]);
						j=i;
					}
				}
				for(i=k; i<nm*2; i++){
					temp=K[k][i];
					K[k][i]=K[j][i];
					K[j][i]=temp;
				}
				temp=Fa[k];
				Fa[k]=Fa[j];
```

```
                Fa[j]=temp;
        }
        //eliminuojami elementai zemiau pagrindinio

        for(i=k+1; i<nm*2; i++){
                temp=K[i][k]/K[k][k];
                for(j=k+1; j<nm*2; j++){
                        K[i][j]-=temp*K[k][j];
                }
                Fa[i]-=temp*Fa[k];
        }
    }
    //Atbuline eiga
    Fa[nm*2-1]/=K[nm*2-1][nm*2-1];
    for(i=nm*2-2; i>=0; i--){
        for(j=i+1; j<nm*2; j++)
                Fa[i]-=K[i][j]*Fa[j];
        Fa[i]/=K[i][i];
    }
    //
    int z=0;
    os<<"1"<<setw(6)<<"x";
    for(i=0; i<nm*2; i++){
        if (i%2==0 && i!=0){
                os<<i-z<<setw(6)<<"x";
                z++;
        }else
        {
                if(i!=0)
                os<<setw(8)<<"y";
        }
        os<<setw(16)<<Fa[i]<<endl;
        if (i%2!=0)
        os<<endl;
    }
    os<<endl;
}
```

```
void matricab ( double B[][6], double& E, double& pk, double& t, double** kintkoord, int** rysiai, int&
        nr);

void sandaugam ( double B[][6], double& E, double& pk, double u[][1], double itemp[][1], double
        D[][3],  int** rysiai,  double** kintkoord, double* Fa, int& ne, double& t, double* elitemp){

        int n, v, m, w, nr;
        double koef;
        double R[3][1], Z[3][3];
        koef=E/(1-pk*pk);

        for (int i=0; i<3; i++){
                for (int j=0; j<3; j++){
                                Z[i][j]=koef*D[i][j];
                }
        }
        n=3;v=6;m=1;w=3;
        for( nr=0; nr <ne; nr++){
                int i= rysiai[nr][0], j= rysiai[nr][1], k= rysiai[nr][2];
                u[0][0]=Fa[i*2-2];
                u[1][0]=Fa[i*2-1];
                u[2][0]=Fa[j*2-2];
                u[3][0]=Fa[j*2-1];
                u[4][0]=Fa[k*2-2];
                u[5][0]=Fa[k*2-1];
                matricab (B, E, pk, t, kintkoord, rysiai, nr);
                for (int t=0; t<n; t++){
                                for(int h=0; h<m; h++){
                                                R[t][h]=0;
                                                for (int l=0; l<v; l++){
                                                                R[t][h]+=B[t][l]*u[l][h];
                                                }
                                }
                }
                for (int t=0; t<n; t++){
                                for(int h=0; h<m; h++){
                                                itemp[t][h]=0;
                                                for (int l=0; l<w; l++){
```

```
                                                    itemp[t][h]+=Z[t][l]*R[l][h];
                                                }
                                            }
                                        }
elitemp[nr]=sqrt(pow(itemp[0][0],2)+pow(itemp[1][0],2)-
itemp[0][0]*itemp[1][0]+3*pow(itemp[2][0],2));
                    }
}
int atranka(int& nind, double* atvmase, double& sum_atvmase);

void kryzminimas (double* ind_1, double* ind_2, double* newind_1, double* newind_2, int& nmin,
                  double& a, double& b, double** koordinates, int& k);

void mutacija (double** newpopuliacija, double** koordinates, int& i,
                int& j, double& a, double& b, int& k );

void GAlgoritmas ( ifstream& is, double** populiacija, double** kintkoord, double& mtik,
                   double& a, double& b, int& nind, double* atvmase, double& t, double& ktik,
                   double& sum_atvmase, int& nm, int& nmin,  double** newpopuliacija, int** rysiai,
                   double** koordinates, int& ne, double& minmas, int& msky ){

                  int test, k;
                  double* ind_1; double* ind_2;
                  double* newind_1; double *newind_2;
                  double ask;
                  ofstream os;
                  os.open ("GA.txt",ios::app);

                  k=nm-nmin-msky;

                  ind_1= new double [nmin*2];
                  ind_2= new double [nmin*2];
                  newind_1= new double [nmin*2];
                  newind_2= new double [nmin*2];

                  for(int nr=0; nr<nind; nr++){

                        int j =atranka(nind, atvmase, sum_atvmase);
```

```
        for (int i=0; i<nmin*2+1; i++){
           if (i==0){
              newpopuliacija[nr][i]=nr+1;
           }
           else{
              newpopuliacija[nr][i]=populiacija[j][i];
           }
        }
}

for (int i=0; i<nind; i++){

        if ( i%2==0 ){
            test=0;
            ask=rand()%1000;
            if ( ask/1000<ktik ){
                  test=1;
                  for (int j=0; j<nmin*2; j++){
                      ind_1[j]=newpopuliacija[i][j+1];
                  }
            }
        }
        if ( i%2!=0 && test==1 ){
                                  for ( int j=0; j<nmin*2; j++){
                   ind_2[j]=newpopuliacija[i][j+1];
            }
            kryzminimas ( ind_1, ind_2, newind_1, newind_2, nmin, a, b, koordinates, k);
            for( int p=0; p<nmin*2+1; p++){
                  if (p==0){
                     newpopuliacija[i-1][p]=i;
                     newpopuliacija[i][p]=i+1;
                  }
                  else{
                      newpopuliacija[i-1][p]=newind_1[p-1];
                      newpopuliacija[i][p]=newind_2[p-1];
                  }
            }
```

```
                }
            }

            for (int i=0; i<nind; i++){
               for(int j=1; j<nmin*2+1; j++){
                    ask=rand()%1000;
                    if (ask/1000 < mtik){
                            mutacija(newpopuliacija, koordinates, i, j, a, b, k );
                    }
               }
            }

            for (int nr=0; nr<nind; nr++){
               for (int i=1; i<nmin*2+1; i++){
                  populiacija[nr][i]=newpopuliacija[nr][i];
               }
            }

            for ( int i=0; i<nind; i++ ){
                                    for ( int j=0; j<nmin*2+1; j++ ){
                  os<<newpopuliacija[i][j]<<" ";
               }
                        os<<endl<<endl;
            }

            os.close();

            delete[] ind_1;
            delete[] ind_2;
            delete[] newind_1;
            delete[] newind_2;
}
int atranka(int& nind, double* atvmase, double& sum_atvmase){
            double skrand;
            double tempsum=0;
            int j=-1;

            skrand=rand()%1000;
```

```
            skrand=skrand/1000*sum_atvmase;
            while((tempsum <= skrand )&&(j!=nind-1)){
                    j++;
                    tempsum+=atvmase[j];
            }
            return j;
}
void kryzminimas (double* ind_1, double* ind_2, double* newind_1, double* newind_2, int& nmin,
             double& a, double& b, double** koordinates, int& k){

    int poz;
    int l, j;
    double x1, y1, x2, y2;
    l=nmin*2-1;
    poz=rand()%l+1;

    for(int i=0; i<nmin*2; i++){
        if (i<poz){
                newind_1[i]=ind_1[i];
                newind_2[i]=ind_2[i];
        }
        else{
                newind_1[i]=ind_2[i];
                newind_2[i]=ind_1[i];
        }
    }

    if ( poz%2!=0 ){
        j=(poz-1)/2;

        x1=koordinates[k+j][0]-newind_1[poz-1];
        y1=sqrt((1-pow(x1,2)/pow(a,2))*pow(b,2));

        x2=koordinates[k+j][0]-newind_2[poz-1];
        y2=sqrt((1-pow(x2,2)/pow(a,2))*pow(b,2));

        if ( y1+koordinates[k+j][1] < newind_1[poz] ){
            newind_1[poz]=y1+koordinates[k+j][1];
```

```
        }
        if ( koordinates[k+j][1]-y1 > newind_1[poz] ){
            newind_1[poz]=koordinates[k+j][1]-y1;
        }
        if ( y2+koordinates[k+j][1] < newind_2[poz] ){
            newind_2[poz]=y2+koordinates[k+j][1];
        }
        if ( koordinates[k+j][1]-y2 > newind_2[poz] ){
            newind_2[poz]=koordinates[k+j][1]-y2;
        }
    }
}
void mutacija (double** newpopuliacija, double** koordinates, int& i, int& j, double& a, double& b,
            int& k ){
    double ask, x, y;
    ask=rand()%1000;
    if ( ask/1000 <= 0.5 ){
        if (j%2!=0){
          newpopuliacija[i][j]+=a/10;
        }
        if (j%2==0){
          newpopuliacija[i][j]+=b/10;
        }
    }
    else{
        if (j%2!=0){
          newpopuliacija[i][j]-=a/10;
        }
        if (j%2==0){
          newpopuliacija[i][j]-=b/10;
        }
    }
    if ( j%2!=0 ){
```

```
        int l=(j-1)/2;
        y=koordinates[k+l][1]-newpopuliacija[i][j+1];
        if (y > b){
            y=b;
        }
        x=sqrt((1-pow(y,2)/pow(b,2))*pow(a,2));
        if ( x+koordinates[k+l][0] < newpopuliacija[i][j] ){
            newpopuliacija[i][j]=x+koordinates[k+l][0];
        }
        if ( koordinates[k+l][0]-x > newpopuliacija[i][j] ){
            newpopuliacija[i][j]=koordinates[k+l][0]-x;
        }
    }
    if ( j%2==0 ){
        int l=(j-2)/2;
        x=koordinates[k+l][0]-newpopuliacija[i][j-1];
        if (x > a){
            x=a;
        }
        y=sqrt((1-pow(x,2)/pow(a,2))*pow(b,2));
        if ( y+koordinates[k+l][1] < newpopuliacija[i][j] ){
            newpopuliacija[i][j]=y+koordinates[k+l][1];
        }
        if ( koordinates[k+l][1]-y > newpopuliacija[i][j] ){
            newpopuliacija[i][j]=koordinates[k+l][1]-y;
        }
    }
}
```