KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

Viktoras Kovaliovas

Genomų palyginimo algoritmų tyrimas

Magistro darbas

Darbo vadovas
prof. hab. dr. J. Mockus

Kaunas, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PROGRAMŲ INŽINERIJOS KATEDRA

TVIRTINU
Katedros vedėjas
doc. dr. E. Bareiša
2005 05

Viktoras Kovaliovas

Genomų palyginimo algoritmų tyrimas

Informatikos mokslų magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių katedros lekt.
dr. J. Mikelionienė
2005 05

Recenzentas
dr. G. Palubeckis
2005 05

Vadovas
prof. hab. dr. J. Mockus
2005 05

Atliko
IFM-9/1 gr. stud.
V. Kovaliovas
2005 05

Kaunas, 2005

Summary

To understand evolution, and to discover how different species are related, gene order analysis is a useful tool. Problems in this area can usually be formulated in a combinatorial language. We regard genomes as signed, or unsigned permutations, and thus evolutionary operations like inversions (reversing the order of a segment of genes) are easy to describe combinatorially. A commonly studied problem is to determine the evolutionary distance between two species. This is estimated by several combinatorial distances between gene order permutations, for instance the inversion distance.

The main objective of this work was to survey the existing algorithms for genome comparison and to present new approach for solving this problem. The work led to these results:

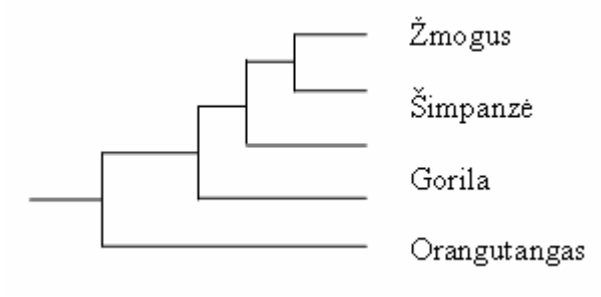
- We have surveyed existing approaches of genome comparison, namely comparison by inversion distance in signed and unsigned cases. It appeared that sorting signed genomes by inversions is done in quadratic time, but sorting unsigned genomes by inversions is NP-hard.
- We have proposed the method of how to apply heuristic algorithms for sorting unsigned genomes by inversions.
- We have applied tabu search and genetic algorithm to solve the sorting unsigned genomes by inversions problem.
- We have experimentally proven, that the worst case solutions to sorting unsigned genomes by inversions found by heuristics (tabu search and genetic algorithm) are better then ones expected from best known approximating algorithm used for sorting unsigned genomes by inversions.

Turinys

1 ĮVADAS

Pastarąjį dešimtmetį pasaulyje pastebimas susidomėjimo skaičiuojamąja biologija (bioinformatika) augimas. Per tą laiką šioje srityje buvo suformuluota nemažai problemų, kurių sprendimas tiesiogiai daro įtaką biologijos mokslo vystymuisi. Viena problemų grupė yra susijusi su molekulinės biologijos nagrinėjamais rūšių genomais. Atsiradus dideliam kiekiui genominių duomenų, tampa įmanoma palyginti rūšių genomus.

Norint suprasti gyvybės evoliuciją, reikia suvokti, kaip skirtingos gyvybės rūšys siejasi tarpusavyje. Mokslo pasiekimai leidžia manyti, kad kiekvieno gyvo organizmo struktūra yra apibrėžta jo genome. Kiekvienos rūšies genomas yra unikalus, tačiau ne pastovus, t. y. lėtai keičiasi. Todėl rūšys vystosi, o šis procesas vadinamas evoliucija. Manoma, kad vienai rūšiai evoliucionuojant keičiasi jos genomas ir atsiranda nauja rūšis. Taigi palyginus dviejų rūšių genomus galima įvertinti, kada tos rūšys atsiskyrė (1 pav.). Šis įvertis vadinamas dviejų rūšių evoliuciniu atstumu.



1 pav. *Evoliucinis žmogaus ir kai kurių beždžionių medis*

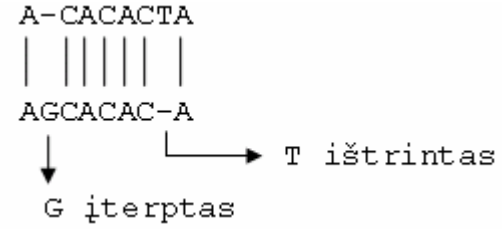
Genomas yra sudarytas iš vieno ar keleto ilgų dvigubų nukleotidų spiralių. Šie nukleotidai yra sugrupuoti į funkcinius blokus, kurie vadinami genais. Genome genas gali būti orientuotas tiesiogiai, arba apverstas. Dažnai skirtingų rūšių genomai turi tuos pačius genus, tačiau jų padėtis ir orientacija gali skirtis.

Genomo kitimas gali būti aprašytas naudojant baigtinę aibę tokių žinomų evoliucinių įvykių, kaip inversija, perkėlimas, įterpimas, dubliavimasis, šalinimas, transpozicija ir kt. Taigi genomų palyginimo matu galėtų būti juos skiriančių evoliucinių įvykių kiekis. Bendru atveju genomų palygimo problema formuluojama taip:

Duoti du genomai. Reikia rasti evoliucinių įvykių seką, kurią pritaikius pirmajam genomui būtų gautas antrasis.

Pastarojo dešimtmečio tyrinėjimai parodė, kad labai sunku išspręsti šią problemą atsižvelgiant net ir tik į kurį nors vieną evoliucinį įvykį, ignoruojant kitus. Skirtingos šios problemos formuluotės dažniausiai susiveda į NP sunkius kombinatorinio optimizavimo uždavinius. Taigi patikimų matematinių modelių ir tikslių ar euristinių algoritmų radimas šios problemos sprendimui yra aktualus uždavinys.

Iš esmės egzistuoja du genomų kitimo būdai. Vienas būdas yra lokalieji genomo pasikeitimai, t. y. vieno nukleotido pasikeitimas, naujo nukleotido atsiradimas ar dingimas (žr. 2 pav.).



2 pav. *Vienos nukleotidų sekos gavimas iš kitos atliekant lokaliuosius pakeitimus*

Kitas būdas yra genų išsidėstymo genome tvarkos kitimas. Pastarasis genomų kitimo būdas, genų pertvarkymas genome, vadinamas globaliuoju kitimu. Šiame darbe nagrinėjamas tik globalusis genomų kitimas. 3 pav. pavaizduotų nukleotidų sekų panašumą sunku įžvelgti pasitelkus lokaliuosius genomo kitimus. Tačiau vienintelis globalusis įvykis (šiuo atveju inversija) gali vieną seką paversti kita.

s_1 = GGAATGGTTTCACTTCCC

s_2 = GGCCCTTCACTTTGGTAA

3 pav. *Dvi nukleotidų sekos besiskiriančios vieninteliu globaliuoju įvykiu*

Genomų pertvarkymų tyrimą 1930 metais pradėjo Dobzhansky ir Sturtevant [1]. Tačiau tikri pasiekimai šioje srityje prasidėjo tik tada, kai Palmer ir Herbon [2] pastebėjo, kad kopūsto ir ropės genomai turi labai panašias genų sekas, kurios buvo skirtingai pertvarkytos jų genomuose. Nuo tada genų tvarka paremti atstumo tarp genomų įverčiai tapo plačiai naudojami evoliuciniam atstumui nustatyti.

Taigi suteikus kiekvienam genui unikalų numerį genomus galime užrašyti kaip skaičių seką. Tada nesunku pamatyti, kad vienas genomas yra perstatymas kito genomo atžvilgiu. Jeigu nekreipsime dėmesio į genų orientaciją genome gausime neženklintą perstatymą. O

jeigu apverstam genui prie jo numerio dar pridėsime minuso ženklą, o neapverstam pliuso, tai gausime ženklintą perstatymą. Toliau darbe naudosime genomų aprašymus ženklintais ir neženklintais perstatymais.

Manoma, kad labiausiai genų tvarką genome keičia trys pagrindiniai evoliuciniai įvykiai: inversija, perkėlimas ir invertuotas perkėlimas (jų apibrėžimas pavaizduotas 4 pav.). Tačiau nėra žinoma kokiomis proporcijomis vienas kito atžvilgiu šie įvykiai veikia genų tvarką. Dėl šios priežasties praktikoje nagrinėjami tiek modeliai, kur kiekvienam įvykiui priskiriamas skirtingos tikimybės, tiek modeliai, kuriuose visi trys įvykiai laikomi vienodai tikėtinais.

$$\dots \pi_i \, \pi_{i+1} \, \pi_{i+2} \dots \pi_{j-1} \pi_j \, \pi_{j+1} \dots \rightarrow \dots \pi_i \, -\pi_j \, -\pi_{j-1} \dots \, -\pi_{i+2} - \pi_{i+1} \, \pi_{j+1} \dots$$

$$\dots \pi_i \, \pi_{i+1} \dots \pi_j \, \pi_{j+1} \dots \pi_k \, \pi_{k+1} \dots \rightarrow \dots \pi_i \, \pi_{j+1} \dots \pi_k \, \pi_{i+1} \dots \pi_j \, \pi_{k+1} \dots$$

$$\dots \pi_i \, \pi_{i+1} \dots \pi_j \, \pi_{j+1} \dots \pi_k \, \pi_{k+1} \dots \rightarrow \dots \pi_i \, \pi_{j+1} \dots \pi_k \, -\pi_j \dots \, -\pi_{i+1} \, \pi_{k+1} \dots$$

4 pav. *Inversijos, perkėlimo ir invertuoto perkėlimo apibrėžimai ženklintiems perstatymams. Jeigu nuimtume ženklus gautume tų pačių įvykių apibrėžimus neženklintiems perstatymams*

Bendru atveju genomų palyginimo problema formuluojama taip. Duoti du genomai. Reikia rasti minimalią evoliucinių įvykių seką, kurią pritaikius pirmajam genomui būtų gautas antrasis. Tarkime, kad $\pi = (\pi_1 \dots \pi_n)$ ir $o = (o_1 \dots o_n)$ yra vienodo ilgio genomai, kurių ilgis yra n. Egzistuoja toks genų sunumeravimas, kad $o = \iota = (1\ 2\ \dots\ n)$. ι vadinsime tapatingu perstatymu.

Taip pat pagrindinius evoliucinius įvykius pavadinkime operacijomis su perstatymais ir žymėkime taip: $\rho(i, j)$ yra inversija nuo i iki j; $\tau(i, j, k)$ yra j ilgio segmento perkėlimas iš pozicijos i į k; $\omega(i, j, k)$ yra j ilgio segmento invertuotas perkėlimas iš pozicijos i į k.

Atsižvelgiant į ankščiau pateiktus apibrėžimus, problema suvedama į π rikiavimą naudojant operacijas iš visų leistinų operacijų aibės poaibio. Šiame darbe bus nagrinėjamas tik rikiavimas panaudojant inversijas. Taigi šiame darbe nagrinėjama problema formuluojama taip:

Problema: *reikia rasti minimalaus ilgio inversijų seką surikiuojančią π.*

Šio darbo tikslai:

1. Ištirti šiuo metu egzistuojančius algoritmus naudojamus rikiavimo inversijomis uždaviniui spręsti.
2. Sukurti ir ištirti metodologija euristinių algoritmų taikymo rikiavimo inversijomis uždaviniui spręsti.

2 GENOMŲ VAIZDAVIMAS PERSTATYMAIS

Toliau nagrinėjamuose uždaviniuose bus duoti du genomai sudaryti iš tos pačios baigtinės genų aibės $G = \{g_1, g_2, g_3, \ldots, g_n\}$ genų. Kiekvienas genomas bus šių genų seka. Pagal nuo uždavinio tipą genai bus orientuotieji arba ne orientuotieji.

Kiekvienam genui $g \in G$ iš genomo $\Gamma = (g_1, g_2, g_3, \ldots, g_n)$ priskirsime unikalų sveiką skaičių *i*, kur $i \in \{1,2,\ldots n\}$. Tada genomą Γ pakeisime skaičių perstatymu π taip, kad kiekvieną geną genome Γ atitiks sveikas skaičius perstatyme π ir skaičiai perstatyme π turės tą patį eiliškumą kaip ir jų atitinkami genai genome Γ. Jeigu genas $g \in G$ iš genomo Γ yra orientuotas, tada jį atitinkantis skaičius perstatyme π turės pliuso ženklą, kai *g* orientacija yra tiesioginė, ir minuso ženklą, kai *g* orientacija yra atvirkštinė.

Tegul duoti du genomai $\Gamma_1 = (g_1^1, g_2^1, g_3^1, \ldots, g_n^1)$ ir $\Gamma_2 = (g_1^2, g_2^2, g_3^2, \ldots, g_n^2)$, sudaryti iš genų $g \in G$. Genomo Γ_2 genus sunumeruokime taip, kad $g_i^2 := i$, kai $i = 1,2,\ldots,n$. Tada genomą Γ_2 galėsime vaizduoti perstatymu $\pi_2 = (1,2,\ldots,n)$. Perstatymą, kurio visi elementai tenkina lygybę $\pi_{i+1} - \pi_i = 1$, kai $i = 1,2,\ldots,n-1$, vadinsime tapačiuoju perstatymu ir žymėsime ι. Matome, kad $\pi_2 = \iota$. Pritaikę tą pačią genų numeraciją genomui Γ_1 gausime jį vaizduojantį perstatymą $\pi_1 = (\pi_1, \pi_2, \ldots, \pi_n)$. Matematiniu požiūriu genu numeracija nėra svarbi. Taigi duotus genomus galime vaizduoti perstatymu $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ atitinkančiu pirmąjį genomą antrojo genomo atžvilgiu (antrąjį genomą atitinka tapatusis perstatymas).

Pavyzdžiui, tegul turime genų aibę $G = \{g_1, g_2, g_3, g_4\}$. Taip pat turime du genomus $\Gamma_1 = (g_3, g_2, g_1, g_4)$ ir $\Gamma_2 = (g_4, g_3, g_1, g_2)$ sudarytus iš genų $g \in G$. Genus $g \in G$ sunumeruokime pagal antrąjį genomą taip, kaip buvo apibrėžta aukščiau. Gausime $g_1 = 3$, $g_2 = 4$, $g_3 = 2$, $g_4 = 1$. Tada genomą Γ_2 galėsime vaizduoti tapačiuoju perstatymu ι. Pagal gautą genų numeraciją sunumeruokime genomą Γ_1. Gausime genomą Γ_1 vaizduojantį perstatymą $\pi = (2,4,3,1)$.

Nuo šiol darbe naudosime genomų aprašymą perstatymais.

3 ŽENKLINTŲ PERSTATYMŲ RIKIAVIMAS INVERSIJOMIS (*SSBR*)

Literatūroje šis uždavinys žymimas *SSBR*. Tarkime, kad turime baigtinę genų aibę $G = \{g_1, g_2, g_3, \ldots, g_n\}$. Tada kiekvienas genomas yra šių genų seka, kur kiekvienas genas turi savo orientaciją: teigiamą (g_i), arba neigiamą $(-g_i)$. Tegul turime genomą $\Gamma = (g_1, g_2, g_3, \ldots, g_n)$. Tada **inversija** tarp indeksų *i* ir *j*, kur $i \le j$, sukuria naują genomą $\Gamma\rho(i, j) = (g_1, g_2, \ldots, g_{i-1}, -g_j, -g_{j-1}, \ldots, -g_i, g_{j+1}, \ldots, g_n)$.

Inversiniu atstumu tarp dviejų genomų, vadinamas minimalus inversijų skaičius, kurį pritaikius pirmajam genomui gaunamas antrasis. Trumpiausios inversijų sekos, atitinkančios inversinį atstumą, radimas vadinamas **rikiavimu inversijomis**, o pati seka – **minimali inversijų seka**.

Kadangi genomus vaizduosime perstatymais, tai inversinį atstumą žymėsime $d(\pi)$, kur π genomus atitinkantis perstatymas. Kadangi rikiavimo inversijomis tikslas yra iš perstatymo π gauti tapatųjį perstatymą ι, tai nesunku suprasti, kodėl šį uždavinį galima vadinti rikiavimu. 5 pav. pateiktas rikiavimo inversijomis pavyzdys.

Pradinis perstatymas:	$\pi := (5,1,3,2,4)$
Inversija nuo indekso 1 iki indekso 2:	$\pi := \pi\rho(1,2) = (-1,-5,3,2,4)$
Inversija nuo indekso 3 iki indekso 5:	$\pi := \pi\rho(3,5) = (-1,-5,-4,-2,-3)$
Inversija nuo indekso 1 iki indekso 1:	$\pi := \pi\rho(1,1) = (1,-5,-4,-2,-3)$
Inversija nuo indekso 2 iki indekso 4:	$\pi := \pi\rho(2,4) = (1,2,4,5,-3)$
Inversija nuo indekso 3 iki indekso 4:	$\pi := \pi\rho(3,4) = (1,2,-5,-4,-3)$
Inversija nuo indekso 3 iki indekso 5:	$\pi := \pi\rho(3,5) = (1,2,3,4,5)$
Perstatymas surikiuotas ($\pi = \iota$). **Inversinis atstumas** $d(\pi)$=6. **Minimali inversijų seka**: (1,2), (3,5), (1,1), (2,4), (3,4), (3,5).	

5 pav. *Ženklinto perstatymo rikiavimas inversijomis*

Ženklintų perstatymų rikiavimas inversijomis yra sudėtinga problema, kuri pastaraisiais metais buvo intensyviai tiriama. Buvo įrodytą, kad minimalią inversijų seką

galima rasti per laiką $O(n^2)$ [3]. Šiame tyrime itin svarbus yra D.A Bader, B. Moret ir M. Yan darbas [4], kuriame pateiktas tiesinis inversinį atstumą apskaičiuojantis algoritmas.

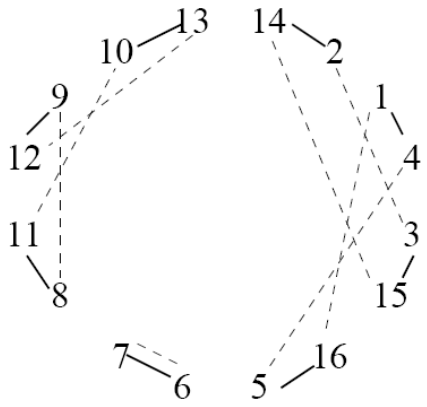
3.1 Lūžio taškų grafas

Genomo π lūžio taškų grafą 1996 metais sugalvojo Bafna ir Pevzner [6]. Du genai *a* ir *b* genome π vadinami **gretimais**, jei *b* eina iškart po *a*, arba jei *a* eina iškart po *b*. Svarbu atkreipti dėmesį į tai, kad *a* ir *b* reikia vertinti kaip surikiuotą porą, tad jeigu *a* ir *b* yra gretimi, tada *b* ir *a* nėra gretimi. Sakoma, kad tarp a ir b genome π genomo *o* atžvilgiu yra **lūžis**, jei *a* ir *b* yra gretimi π, bet ne *o*. Toliau seką, kad lūžių skaičius genome π genomo *o* atžvilgiu yra lygus lūžių skaičiui genome *o* genomo π atžvilgiu. Lūžių skaičių genome π genomo *o* atžvilgiu žymėsime *b(π, o)*. Taip pat naudosime tokį žymėjimą *b(π) = b(π, ι)*.

Tegul U_{2n} žymi visų 2n ilgio neženklintų genomų aibę. Atsižvelgiant į [7] mes apibrėžiame genomų transformacijos funkciją $gtm : G_n \rightarrow U_{2n}$. Kiekvienas genas *a* iš genomo $\pi \in G_n$ yra vaizduojamas kaip genų pora $(2a - 1, 2a)$, kai $a > 0$, arba į genų porą $(-2a, -2a - 1)$, kai $a < 0$. Kairįjį geną genų poroje gautoje iš *a* žymėsime a_L, o dešinįjį a_R. Toliau gautas poras imame ta pačia tvarka, kuria genai yra π. Pavyzdžiui genomas $\pi = (1, -5, 3, 2, -4)$ bus atvaizduojamas neženklintu genomu $gtm(\pi) = (1, 2, 10, 9, 5, 6, 3, 4, 8, 7)$. Svarbu pastebėti, kad abiejų genomo lūžio taškų skaičius yra tas pats: $b(\pi) = b(gtm(\pi))$.

Genomo $\pi \in G_n$ **lūžio taškų grafo** $G(\pi)$ viršūnės yra $gtm(\pi)$ genai. Jei a ir b yra gretimi genome π, tuomet briauna tarp a_R ir b_L yra ištisinė, o briauna tarp $2k$ ir $2k+1$ bei $2n$ ir 1 yra punktyrinė. Lūžio taškų grafo pavyzdys parodytas 6 pav.

Lengva pastebėti, kad kiekvienos grafo $G(\pi)$ viršūnės laipsnis yra 2 ir kad nei viena viršūnė neturi dviejų briaunų, kurios būtų vienodos spalvos, taigi briaunos suformuoja kintančiuosius ciklus. Ištisinių briaunų skaičius cikle toliau bus vadinamas ciklo ilgiu. Ciklai, kurių ilgis lygus 1 bus vadinami trivialiaisiais, o visi kiti – ne trivialiaisiais. Ciklai, kurių ilgis lygus nelyginiam skaičiui bus vadinami nelyginiais ciklais.



6 pav. *Lūžio taškų grafas, kai* $\pi = (1\ -7\ -5\ -6\ -4\ -3\ -8\ 2)$

Nuo šiol genomo π lūžio taškų grafas bus piešiamas taip, kad jo viršūnės būtų išdėstytos ratu, prieš laikrodžio rodyklę, tokia tvarka, kokia jos surašytos genome $gtm(\pi)$. Ciklas yra orientuotasis, jei jis yra trivialusis arba jei eidami juo nepereisime visų jo ištisinių briaunų ta pačia kryptimi (pagal laikrodžio rodyklę arba prieš laikrodžio rodyklę). Kitu atveju, ciklas yra vadinamas neorientuotuoju.

Dabar apibrėšime ciklų lygiavertiškumą. Genomo intervalu vadinamas gretimų genų segmentas. Ciklus vadinsime ekvivalentiškais, jei vienas intervalas, turintis visas pirmojo ciklo viršūnes, ir kitas intervalas, turintis visas antrojo ciklo viršūnes, visada susikerta. Ekvivalentiškumo klasės yra vadinamos komponentais. Komponentas yra orientuotasis, jei turi bent vieną orientuotąjį ciklą, kitu atveju ─ ne orientuotasis. Komponentas vadinamas nelyginiu, jei visi ciklai sudarantys komponentą yra nelyginiai.

Jei yra intervalas, kuriame yra neorientuotasis komponentas τ, bet nėra jokių kitų neorientuotųjų komponentų, tada τ vadinamas kliūtimi. Jei yra intervalas, kurį sudaro būtinai du neorientuotieji komponentai ir galbūt keli orientuotieji komponentai, ir būtinai vienas iš šių neorientuotųjų komponentų yra kliūtis, tada ši kliūtis vadinama visiška kliūtimi. Jei lūžio grafas turi nelyginį skaičių kliūčių, iš kurių visos yra visiškos kliūtys, tada šis grafas vadinamas tvirtove.

Genomui $\pi \in G_n$ apibrėšime keletą funkcijų. Funkcijos $c_{rev} : G_n \to \mathrm{N}$ ir $c_{odd} : G_n \to \mathrm{N}$ operuoja genomu π, o grąžina atitinkamai ciklų skaičių ir nelyginių ciklų skaičių lūžio taškų grafe $G(\pi)$. Taip pat $h : G_n \to \mathrm{N}$ grąžina kliūčių skaičių lūžio taškų grafe $G(\pi)$. Funkcija $f : G_n \to \{0,1\}$ lygi vienetui kai $G(\pi)$ yra tvirtovė ir nuliui – priešingu atveju.

Bafna ir Pevzner [6] įrodė teoremą:

Teorema 1. Kiekvienam ženklintam perstatymui π teisinga lygybė:

$$d(\pi) = n - c_{rev}(\pi) + h(\pi) + f(\pi).$$

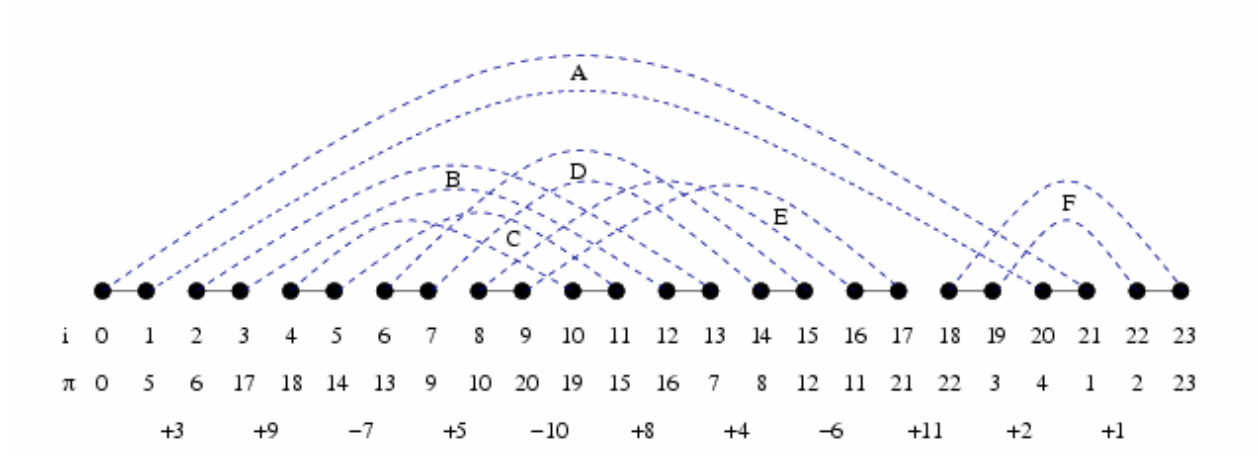
Pastaroji teorema yra svarbiausia nagrinėjant ženklintų perstatymų rikiavimą ir todėl naudojama praktiškai visuose algoritmuose, sprendžiančiuose *SSBR*.

3.2 Persidengimų grafas ir miškas

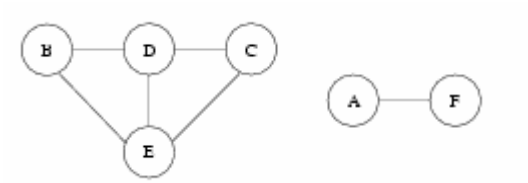
Duotą ženklintą perstatymą (1, ..., n) pertvarkome į neženklintą perstatymą π=(1, …, 2n) pakeičiant teigiamą elementą x į surikiuotą porą $(2x-1, 2x)$, o neigiamą elementą x į surikiuotą porą $(2x, 2x-1)$, tada išplečiam perstatymą π iki (0, 1, …, 2n, 2n+1) nustatydami $\pi(0) = 0$ ir $\pi(2n+1) = 2n+1$. Pagal susitarimą manysime, kad du ženklinti perstatymai, kuriems reikia skaičiuoti atstumą, bus pertvarkyti būtent šiuo būdu į ne ženklintus perstatymus, o tada abu perstatinėjami taip, kad pirmasis perstatymas taptų tapačiuoju (0, 1, ..., 2n, 2n+1); šios manipuliacijos nedaro įtakos atstumui. Pateikiame išplėstą neženklintą perstatymą spalvotų briaunų grafu, perstatymo ciklų grafu. Grafas turi $2n+2$ viršūnes. Kiekvienam i, $0 \le i \le n$ jungiame viršūnes $\pi(2i)$ ir $\pi(2i+1)$ pilka briauna, o viršūnes $2i$ ir $2i+1$ juoda briauna, kaip pavaizduota 7 pav. Gautas grafas sudarytas iš nesusikertančių ciklų, kuriuose briaunų spalvos kaitaliojasi. Pašaliname iš grafo ilgio 2 ciklus, nes šie ciklai atitinka dalį perstatymo, kuris jau surikiuotas ir nesikerta su jokiu kitu ciklu. Sakome, kad pilkos briaunos $(\pi(i), \pi(j))$ ir $(\pi(k), \pi(t))$ persidengia, kai du intervalai $[i, j]$ ir $[k, t]$ persidengia, bet ne tada kai vienas iš jų telpa į kitą. Taip pat, sakome kad ciklai C_1 ir C_2 persidengia, jei egzistuoja persidengiančios pilkos briaunos $e_1 \in C_1$ ir $e_2 \in C_2$.

Perstatymo π persidengiantis grafas turi vieną viršūnę kiekvienam ciklui ciklų grafe ir briauną tarp bet kokių dviejų viršūnių, kurios atitinka persidengiančius ciklus.

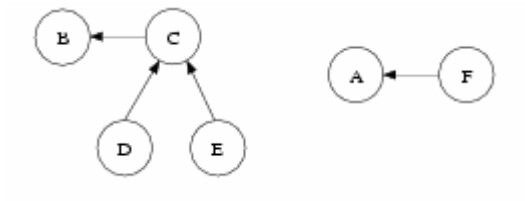
Ciklų grafo pavyzdys duotas 8 pav. Ciklo C išplėtimas yra intervalas [*C.B*, *C.E*], kur $C.B = \min\{i \mid \pi(i) \in C\}$ ir $C.E = \max\{i \mid \pi(i) \in C\}$. Ciklų aibės $\{C_1, ..., C_k\}$ išplėtimas yra $[B, E]$, kur $B = \min_{i=1}^{k} C_i.B$ ir $E = \max_{i=1}^{k} C_i.E$. Pavyzdyje ciklo A išplėtimas yra [0,21], ciklo F - [18, 23], o aibės {A,F} išplėtimas [0,23].



7 pav. *Ženklintas perstatymas, jo neženklintas atvaizdas bei ciklų grafas*



8 pav. *Persidengimų grafas*



9 pav. Persidengimų medis

Neegzistuoja tiesinio sudėtingumo algoritmas, generuojantis persidengiantį grafą, kadangi grafas gali būti kvadratinio dydžio. Taigi mūsų tikslas sukonstruoti persidengiantį mišką tokį, kad dvi viršūnės f ir g priklausytų tam pačiam miško medžiui tik tada kai jos priklauso tam pačiam jungiajam komponentui persidengiančiame grafe. Persidengiantis miškas turi vieną ir tik vieną medį persidengiančio grafo jungiajame komponente. Tokiu būdu gauname tiesinį sudėtingumą.

3.3 Algoritmas randantis inversinį atstumą

Šiame skyriuje nagrinėjamas D.A Bader, B. Moret ir M. Yan darbas [4], t. y. tiesinio sudėtingumo algoritmas, randantis inversinį atstumą. Šis algoritmas peržiūri perstatymą du kartus. Pirmo peržiūrėjimo metu randamas trivialus miškas, kuriame visi medžiai sudaryti iš vienos viršūnės, pažymėtos kaip ciklo pradžia. Antro peržiūrėjimo metu vykdomas pirmojo miško iteracinis pagerinimas pridedant briaunas, kurios sujungia atskirus medžius miške.

Primename kad viršūnės persidengiančiame grafe (ar miške) atitinka ciklus ciklų grafe. Persidengiančio miško viršūnės f laipsnis $[f.B, f.E]$ yra viršūnių laipsnis pomedyje, kurio šaknis yra f. Tegul F_0 bus trivialus miškas, rastas pirmo peržiūrėjimo metu, tarkime,

kad algoritmas apdorojo perstatymo elementus nuo 0 iki $j-1$ ir pateikė mišką F_{j-1}. Mes sudarysime F_j iš F_{j-1}. Tegul f bus ciklas turintis perstatymo elementą j. Jei j yra savo paties ciklo f pradžia, tada šis elementas turi būti medžio, sudaryto iš vienos viršūnės, šaknis. Kitu atveju, jei f perdengtas kitu ciklu g, tada įtraukiame kitą lanką (g, f) ir skaičiuojame jungtinį g ir medžio, kurio šaknis f, laipsnį. Sakome, kad medis, kurio šaknis f yra aktyvus srityje j visada, kai j priklauso nuo f laipsnio. Įrašome į steką aktyvių medžių laipsnius.

10 pav. apibendrina algoritmą sudarantį persidengiančius miškus. Algoritme kintamasis *top* žymi viršutinį steko elementą. Viršutinių miško medžių pavertimas į jungiuosius komponentus įvykdomas per tiesinį laiką, panaudojant masyvus. Svarbu, kad *i* elemento protėvis būtų įrašytas masyve prieš *i* elementą

Lema: Algoritmo (3) žingsnio iteracijoje *i*, jei medžio šaknis *top* aktyvi ir *i* priklauso ciklui f ir $f.B < top.B$, tada medyje, kurio šaknis *top*, egzistuoja toks h, kuris persidengia su f.

Įrodymas. Kadangi *top* yra aktyvusis, tai jis turėjo būti įdėtas į steką prieš esamą iteraciją $(top.B < i)$ ir mes nepasiekėme *top* laipsnio $(i < top.E)$. Taigi $top.B < i < top.E$. Kadangi i priklauso ciklui f, kuris prasideda prieš *top* $(f.B < top.B)$, tai cikle f turi būti briauna, kuri persidengtų su *top*.

Teorema. Algoritmas sukuria tokius miškus, kuriuose medžiai sudaryti būtent iš tų viršūnių, kurios sudaro jungųjį komponentą.

Įrodymas. Užtenka parodyti kad, po kiekvienos iteracijos, miško medis tiksliai atitinka jungųjį komponentą, kurį apibrėžia perstatymo reikšmės tyrinėjamas iki to taško.

Tarkime, kad invariantas nusistovi po $(i-1)$-osios iteracijos ir tegul i priklauso ciklui f. Įrodyta, kad medžio, kuriame yra elementas i, viršūnės suformuoja tokią pačią aibę kaip ir viršūnės, sudarančios jungųjį komponentą, kuriame yra elementas i. Kiti medžiai ir jungieji komponentai nėra įtakojami taigi jie atitinka invariantą.

Įvestis: ženklintas perstatymas

Išvestis: *parent[i]*, *i*-tojo elemento protėvis persidengimų miške

1. Peržiūrime perstatymą, kiekvienai pozicijai i pažymime $C[i].B$ ir nustatome $[C[i].B, C[i].E]$
2. Inicijuojame tuščią steką
3. **for** $i \leftarrow 0$ **to** $2n+1$
   (a) **if** $i = C[i].B$ **then push** $C[i]$
   (b) $extent \leftarrow C[i]$
      while $top.B > C[i].B$
         $extent.B \leftarrow \min\{extent.B, top.B\}$
         $extent.E \leftarrow \max\{extent.E, top.E\}$
         pop *top*
         $parent[top.B] \leftarrow C[i].B$
      endwhile
      $top.B \leftarrow \min\{extent.B, top.B\}$
      $top.E \leftarrow \max\{extent.E, top.E\}$
   (c) **if** $i = top.E$ **then pop** *top*
4. Konvertuoti kiekvieną medį į jo viršūnės sužymėjimą.

10 pav. *Persidengimo miškų konstravimo algoritmas įvykdomas per tiesinį laiką*

Įrodyta, kad viršūnė, priklausanti medžiui, kuriame yra i, turi priklausyti tam pačiam jungiajam komponentui kaip ir i. Jei $i = f.B$, tada, kaip minėjome anksčiau, niekas nesikeičia persidengiančiame grafe, o taip pat ir jungiajame komponente. Nuo (3) žingsnio taip pat aišku, kad miškas išlieka nepakitęs, taigi invariantas išlaikomas.

Kita vertus, jei $i > f.B$, tai (3) žingsnyje briauna (top, f) bus priskirta miškui kai tik $f.B < top.B$. Ši briauna sujungs du skirtingus pomedžius, kurių šaknys f ir top, į vieną bendrą pometį. Iš ankstesnės lemos mes žinome, kad kai tik medyje, kurio šaknis top, $f.B < top.B$, tada turi egzistuoti toks h, kad h ir f persidengtų. Briauna (h, f) būtinai priklausys persidengiančiam grafui,, o taip pat sujungs komponentą, turintį f, su tuo, kuriame yra top, į vieną bendrą jungųjį komponentą, kuris patvirtina invariantą.

Įrodyta, kad viršūnė esanti tame pačiame jungiajame komponente kaip ir i turi priklausyti tam pačiam medžiui, kuriame yra i. Bet kada, kai (j, i) ir (k, l) yra pilkosios

ciklų f ir h briaunos, kur $j < k < i < l$, tada briauna (f, h) turi priklausyti persidengiančiam grafui i pirmųjų perstatymo elementų. Tokiu būdu algoritmas garantuoja, kad briauna (h, f) priklausys persidengiančiam miškui.

Akivaizdu, kad kiekvienas algoritmo žingsnis yra tiesinio sudėtingumo, taigi visas algoritmas įvykdomas per tiesinį laiką.

3.4 Algoritmas randantis minimalią inversijų seką

Šiame skyriuje nagrinėjamą algoritmą pasiūlė H. Kaplan, R. Shamir ir R. E. Tarjan [5]. Jis yra kvadratinio sudėtingumo. Šiuo metu tai geriausias žinomas algoritmas, randantis minimalią inversijų seką. 11 pav. pateikti pagrindiniai algoritmo žingsniai.

Įvestis: ženklintas perstatymas π.
Išvestis: minimali inversijų seka, surikiuojanti π.
1. Rasti grafo $OV(\pi)$ jungiuosius komponentus. 2. Išvalyti kliūtis. 3. **while** π nesurikiuotas (a) grafe $OV(\pi)$ surasti tinkamą kliką C. (b) surasti viršūnę $e \in C$, turinčią didžiausią laipsnį, ir atlikti bei išsaugoti saugiąją inversiją. (c) atnaujinti π ir $OV(\pi)$.

11 pav. *Algoritmas randantis minimalią inversijų seką*

Pirmame algoritmo žingsnyje grafo $OV(\pi)$ jungieji komponentai randami naudojant algoritmą aprašytą 3.3 skyriuje.

Antrajame algoritmo žingsnyje išvalomos kliūtys. Remiantis teorija [3] matome, kad duotam perstatymui π, kur $h(\pi) > 0$, reikia atlikti $t = \lceil h(\pi)/2 \rceil$ perstatymų, kad π pertvarkyti į tokį perstatymą π', kad $h(\pi') = 0$, o $d(\pi') = d(\pi) - t$. Jei persidengimų grafas $OV(\pi)$ turi neorientuotųjų komponentų, tada algoritmas pirmiausia randa t inversijų, kurios pertvarko π į π'. π' turi tik orientuotuosius komponentus.

Trečiajame algoritmo žingsnyje, perstatymo π' persidengimų grafas $OV(\pi')$ turi tiktai orientuotuosius komponentus. Įrodyta, kad bet kokios orientuotosios pilkosios briaunos e kaimynystėje yra orientuotoji pilkoji briauna e_1 (e_1 gali būti ta pati briauna e) tokia, kad inversija veikianti e_1 nesukuria naujų kliūčių. Tokią inversiją vadinsime saugiąja inversija.

Šiame algoritmo etape ieškoma saugiųjų inversijų ir tai daroma tol, kol persidengimų grafas $OV(\pi)$ nepasidaro tuščias.

4 NEŽENKLINTŲ PERSTATYMŲ RIKIAVIMAS INVERSIJOMIS (*SBR*)

Literatūroje šis uždavinys žymimas *SBR*. Tarkime, kad turime baigtinę genų aibę $G = \{g_1, g_2, g_3, \ldots, g_n\}$. Tada kiekvienas genomas yra šių genų seka. Tegul turime genomą $\Gamma = (g_1, g_2, g_3, \ldots, g_n)$. Tada **inversija** tarp indeksų *i* ir *j*, kur $i \le j$, sukuria naują genomą $\Gamma\rho(i, j) = (g_1, g_2, \ldots, g_{i-1}, g_j, g_{j-1}, \ldots, g_i, g_{j+1}, \ldots, g_n)$.

Inversiniu atstumu tarp dviejų genomų (dviejų neženklintų perstatymų sudarytų iš tos pačios genų aibės), vadinamas minimalus inversijų skaičius, kurį pritaikius pirmajam genomui gaunamas antrasis. Trumpiausios inversijų sekos, atitinkančios inversinį atstumą, radimas vadinamas **rikiavimu inversijomis**, o pati seka – **minimali inversijų seka**. 12 pav. pateiktas neženklinto perstatymo rikiavimo inversijomis pavyzdys.

Pradinis perstatymas:	$\pi := (5,1,3,2,4)$
Inversija nuo indekso 3 iki indekso 4:	$\pi := \pi\rho(3,4) = (5,1,2,3,4)$
Inversija nuo indekso 2 iki indekso 5:	$\pi := \pi\rho(2,5) = (5,4,3,2,1)$
Inversija nuo indekso 1 iki indekso 5:	$\pi := \pi\rho(1,5) = (1,2,3,4,5)$
Perstatymas surikiuotas ($\pi = \iota$). **Inversinis atstumas** $d(\pi)$=3. **Minimali inversijų seka**: (3,4), (2,5), (1,5).	

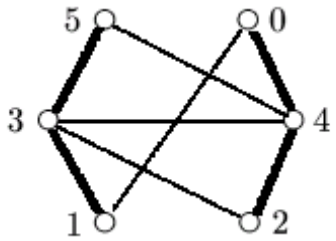
12 pav. *Neženklinto perstatymo rikiavimas inversijomis*

Nors ženklinta ir neženklinta rikiavimo inversijomis problemų versijos atrodo panašios, tačiau skaičiuojamuoju požiūriu jos yra visiškai skirtingos. Kaip parodėme praeitame skyriuje ženklinto perstatymo rikiavimas yra kvadratinio sudėtingumo uždavinys. Tačiau neženklinto perstatymo rikiavimas yra NP-sunkus [8].

4.1 Lūžio taškų grafas

Kaip ir sprendžiant *SSBR* uždavinį, taip ir čia naudojamas lūžio taškų grafas. Kaip apibrėžta [6], perstatymo π lūžio taškų grafas yra $G(\pi) = (V, B \cup Y)$. Norint sukonstruoti $G(\pi)$, reikia išplėsti π elementais $\pi_0 := 0$ ir $\pi_{n+1} := n+1$ taip, kad

$\pi = (0, \pi_1, \pi_2, \ldots, \pi_n, \pi_{n+1})$. Taip pat apibrėžkime perstatymui π atvirkštinį perstatymą π^{-1} taip, kad $\pi^{-1}_{\pi_i} := i$ kai $i := 0,1,\ldots,n+1$. Tada $V = \{0,1,2,\ldots,n,n+1\}$, kur kiekviena viršūnė $v \in V$ atstovaus vieną π elementą. Grafas $G(\pi)$ yra dvispalvis, t. y. jo briaunų aibė sudalinta į dvi, kurių kiekvienai priskirta sava spalva. B yra juodųjų briaunų aibė, kurių kiekviena apibrėžiama viršūnėmis (π_i, π_{i+1}), visoms $i \in \{0 \ldots n\}$ reikšmėms su kuriomis $|\pi_i - \pi_{i+1}| \neq 1$, t. y. elementais kurie yra gretimi perstatyme π, bet nėra gretimi tapačiajame perstatyme ι. Tokia pora (π_i, π_{i+1}) yra vadinama π lūžio tašku, o elementas π_i yra vadinamas pavieniu, jei (π_i, π_{i-1}) ir (π_i, π_{i+1}) yra π lūžio taškais. Tegul $b(\pi) := |B|$ bus π lūžio taškų skaičius. Y yra pilkųjų briaunų aibė, kurių kiekviena apibrėžiama viršūnėmis $(i, i+1)$, visoms $i \in \{0 \ldots n\}$ reikšmėms su kuriomis $|\pi_i^{-1} - \pi_{i+1}^{-1}| \neq 1$, t. y. elementais kurie yra gretimi tapačiajame perstatyme ι, bet nėra gretimi perstatyme π. Reikia pastebėti, kad kiekvienos viršūnės $v \in V$ laipsnis yra 0, 2 arba 4, o taip pat ji turi vienodą juodųjų ir pilkųjų incidentinių briaunų skaičių. Todėl $|B| = |Y|$. 13 pav. pavaizduotas lūžio taškų grafas sudarytas perstatymui (4, 2, 1, 3).



13 pav. *Perstatymo (4, 2, 1, 3) lūžio taškų grafas*

Kintančiuoju lūžio taškų grafo $G(\pi)$ ciklu vadinsime tokį ciklą, kurio bet kurios dvi gretimos briaunos yra skirtingų spalvų ir kuriame gali kartotis viršūnės, bet negali briaunos. Formaliai kintantysis ciklas yra tokia briaunų seka $b_1, y_1, b_2, y_2, \ldots, b_m, y_m$, kad:

1. $b_i \in B$, $y_i \in Y$ kai $i = 1,2,\ldots,m$.
2. b_i ir y_j turi bendrą viršūnę, kai $i = j = 1,2,\ldots,m$ ir kai $i = j+1$, $j = 1,2,\ldots,m$.
3. b_i ir b_{i+1} (y_i ir y_{i+1}) neturi bendros viršūnės, kai $i = 1,2,\ldots,m$.
4. $b_i \neq b_j$ ($y_i \neq y_j$), kai $1 \leq i < j \leq m$.

Remiantis anksčiau pateiktu apibrėžimu gauname, kad 13 pav. pavaizduotas lūžio taškų grafas turi kintančiuosius ciklus (0, 4), (4, 3), (3, 1), (1, 0) ir (4, 2), (2, 3), (3, 5), (5, 4).

Lūžio taškų grafo $G(\pi)$ suskaidymas kintančiaisiais ciklais yra toks kintančiųjų ciklų rinkinys, kad kiekviena grafo $G(\pi)$ briauna priklauso vienam ir tik vienam kintančiajam ciklui iš to rinkinio. Lengva pamatyti, kad $G(\pi)$ visada galima suskaidyti kintančiaisiais ciklais. 13 pav. kintantieji ciklai (0, 4), (4, 3), (3, 1), (1, 0) ir (4, 2), (2, 3), (3, 5), (5, 4) sudaro lūžio taškų grafo suskaidymą kintančiais ciklais. Tegul $c(\pi)$ bus maksimalus lūžio taškų grafo $G(\pi)$ suskaidymo kintančiais ciklais narių skaičius. Bafna ir Pevzner [6] įrodė sekančią teoremą.

Teorema 2. Kiekvienam perstatymui π teisinga nelygybė:

$$b(\pi) - c(\pi) \leq d(\pi).$$

4.2 *SBR* sprendimo algoritmai

Caprara ir Lancia [9] sukūrė tikslų algoritmą *SBR* sprendimui, kuris sprendžia uždavinius su $n < 200$. Nors praktikoje nepasitaiko daug genomų, kurių rikiavimui nepakaktų pastarojo algoritmo pajėgumo, vis vien yra kuriami aproksimuojantys algoritmai. Taip yra todėl, kad praktikoje nepakanka nagrinėti tik vieno tipo globalius genomų kitimus – reikalingi algoritmai atsižvelgiantys į daugiau globalių genomų kitimų. Žinomi keli aproksimuojantys algoritmai: Christie [10] pasiūlytas algoritmas *SBR* aproksimuoja faktoriumi 3/2, o P. Berman su kolegomis [11] pasiūlė algoritmą, kuris *SBR* aproksimuoja faktoriumi 1.375.

5 EURISTINIAI ALGORITMAI

Žinoma daug uždavinių, kurių sprendimo sudėtingumas yra eksponentinis. Tokiems uždaviniams spręsti dažnai negalima naudoti tikslų sprendinį garantuojančių algoritmų, nes jie dirba labai ilgai. Tokiais atvejais praktikoje naudojami euristiniai algoritmai , kurie sprendinį optimizuoja atsižvelgiant į ribotus skaičiuojamuosius pajėgumus. Euristiniai algoritmai negarantuoja optimalaus sprendinio, bet jų gaunami sprendiniai dažnai būna pakankamai geri, kad juos būtų galima naudoti praktikoje. Žinoma daug bendro pobūdžio euristinių metodų (meta-euristikų), kurias galima taikyti įvairaus tipo uždaviniams spręsti.

5.1 Tabu paieška

Šį algoritmą sugalvojo F. Glover [12] Tabu paieška pagrįsta kaimyninių sprendinių paieška pereinant nuo vieno lokaliojo optimumo prie kito. Pagrindinė tabu paieškos idėja yra leidimas atlikti perėjimus net ir tais atvejais, kai nėra pagerinamas gretimas sprendinys. Tačiau kai kurie perėjimai yra draudžiami, siekiant išvengti ciklų.

Tabu paieška pradedama nuo pradinio sprendinio $s_0 \in S$, kur S yra galimų sprendinių aibė. Pradinis sprendinys gali būti sugeneruotas atsitiktiniu būdu. Algoritmo vykdymo eigoje analizuojama sprendinio $s \in S$ aplinkos (kaimyninių) sprendinių aibė $N(s)$. Baigus analizę, pereinama į tą sprendinį $s' \in N(s)$, kuriam tikslo funkcijos reikšmė f yra mažiausia. Perėjimas atliekamas ir tuo atveju, kai tikslo funkcijos pokytis yra teigiamas (t. y. tikslo funkcijos reikšmė pablogėja) – taip galima pereiti nuo vieno lokaliai optimalaus sprendinio prie kito. Grįžimas į anksčiau nagrinėtą sprendinį turi būti uždraudžiamas tam tikram laikotarpiui, kad būtų išvengta paieškos kartojimo nuo to paties sprendinio. Taigi nagrinėtieji sprendiniai tam tikrais momentais tampa „tabu“, t. y. įtraukiami į specialų sąrašą T vadinamą tabu sąrašu. Tokiu būdu perėjimas į sprendinį $s' \in N(s)$ draudžiamas, jeigu tas sprendinys duotu momentu yra sąraše T. Sprendinių draudimas gali apriboti paiešką kai kuriose sprendinių aibės S srityse. Siekiant to išvengti naudojamas vadinamasis aspiracijos kriterijus. Šis kriterijus anuliuoja tabu būseną esant tam tikroms sąlygoms. Viena iš standartinių sąlygų yra kai $f(s') < f(s^*)$, kur s^* yra geriausias iki šiol paieškos metu surastas sprendinys. Siekiant pagerinti rezultatus gali būti keičiamas tabu sąrašo ilgis. Tabu paieškos algoritmo vykdymas baigiamas atlikus iš anksto nustatytą iteracijų skaičių.

Klasikinė tabu paieškos algoritmo schema pateikta 14 pav. [14].

Įvestis: s_0 – pradinis sprendinys. f – tikslo funkcija.

Išvestis: $s*$ – geriausias rastas sprendinys.
1. **[Pradžia]** $s := s_0$ ir $s^* := s$. Sukuriamas tuščias tabu sąrašas T. 2. **[Sprendinio gerinimas]** gerinti sprendinį tol, kol netenkinama pabaigos sąlyga. 2.1 Rasti geriausią $s' \in N(s)$ atsižvelgiant į esamą tabu sąrašą T ir pasirinktą aspiracijos kriterijų. 2.2 Pakeisti esamą sprendinį nauju $s = s'$. 2.3 Įtraukti s į tabu sąrašą T. 2.4 Jei $f(s) < f(s^*)$, prisiminti geriausią sprendinį $s^* = s'$. 2.5 Atnaujinti tabu sąrašą T.

14 pav. *Tabu paieškos algoritmo schema*

5.2 Genetiniai algoritmai

Genetiniai algoritmai yra vienas iš evoliucinio skaičiavimo metodų [15]. Apskritai evoliuciniai skaičiavimo metodai iš kitų išsiskiria tuo, kad pagrindiniai jų konstrukcijos ir realizacijos elementai remiasi gerai žinomomis evoliucijos savybėmis. Paprastai tariant, problemos šiuo metodu yra sprendžiamos evoliuciniu procesu, kurio metu gaunamas geriausias sprendinys, t. y. sprendinys evoliucionuoja. Žinomos šios evoliucinių skaičiavimo metodų klasės: genetiniai algoritmai, evoliucinis programavimas, evoliucinės strategijos bei klasifikavimo sistemos. Šiame darbe bus taikomi tik genetiniai algoritmai, todėl kiti evoliucinio skaičiavimo metodų nagrinėjami nebus.

Genetinius algoritmus atrado ir išvystė J.Holland. Jis studijavo natūralios adaptacijos fenomeną ir ieškojo būdų jį perkelti į kompiuterį. Genetiniai algoritmai buvo pristatyti kaip biologinės evoliucijos modeliavimo priemonė.

Klasikinė genetinio algoritmo schema pateikta 15 pav. Pirmame pateiktos schemos žingsnyje (*Pradžia*) sukuriama pradinė sprendinių aibė (populiacija). Paprasčiausias pradinių sprendinių parinkimo būdas yra jų generavimas atsitiktine tvarka. Populiacijos dydis nesikeičia algoritmo vykdymo metu, taigi jis yra vienas iš genetinio algoritmo parametrų.

Toliau algoritmas vykdo ciklą, kuris gali būti apribotas maksimaliu iteracijų skaičiumi arba vykdymo laiku. Kiekvienoje ciklo iteracijoje stengiamasi pereiti prie geresnės populiacijos. Visų pirma įvertinama esama populiacija (*Tinkamumas*). Toliau kuriama nauja (*Nauja populiacija*). Nauja populiacija kuriame iš senosios pritaikius genetinius operatorius: kryžminimą (*Išrinkimas*, K*ryžminimas*) ir mutaciją (*Mutacija*).

Įvestis: tikslo funkcija f.
Išvestis: geriausias rastas sprendinys.
1. **[Pradžia]** generuoti pradinių sprendinių populiaciją, kurios dydis yra *n*.

2. **[Tinkamumas]** apskaičiuoti kiekvieno sprendinio tinkamumą pagal tikslo funkciją *f(s)*.
3. **[Nauja populiacija]** sukurti naują populiaciją kartojant sekančius žingsnius tol, kol populiacija prisipildys.
   - 3.1 **[Išrinkimas]** Pagal tinkamumą išrinkti du tėvinius sprendinius iš populiacijos (kuo didesnis tinkamumas, tuo sprendinys turi didesnę tikimybę būti išrinktas).
   - 3.2 **[Kryžminimas]** Iš tėvynių sprendinių sukuriamas naujas sprendinys (vaikas).
   - 3.3 **[Mutacija]** Pagal mutacijos tikimybę keisti sprendinį (vaiką) atskirose jo dalyse.
   - 3.4 **[Priėmimas]** Įdėti sprendinį (vaiką) į naują populiaciją.
4. **[Pakeitimas]** Tolimesniame darbe naudoti naujai sugeneruotą populiaciją.
5. **[Tikrinti]** Jei pabaigos sąlyga tenkinama, sustabdyti algoritmą ir grąžinti geriausią sprendinį iš einamos populiacijos.
6. **[Ciklas]** Grįžti į žingsnį 2.

15 pav. *Genetinio algoritmo schema*

Tėvų parinkimą kryžminimui galima atlikti naudojant įvairiausias strategijas. Galima naudoti ruletės metodą. Šiuo metodu kiekvienam sprendiniui populiacijoje priskiriama išrinkimo tikimybė, kuri tiesiogiai priklauso nuo sprendinio gerumo: $p(s) := \frac{f(s)}{\sum_s f(s)}$. Tada pagal apskaičiuotas tikimybes tėvai parenkami kryžminimui. Kryžminime dalyvauja mažiausiai du tėvai. Atsitiktiniu būdų parenkamas vienas arba daugiau pjūvio taškų. Tada naują sprendinį gauname paėmus tėvinių sprendinių dalis gautas perkitus juos per pjūvio taškus. 16 pav. pateiktas kryžminimo pavyzdys su dvejais tėvais ir vienu pjūvio tašku. 17 pav. pateiktas kryžminimo pavyzdys su dvejais tėvais ir dvejais pjūvio taškais.

tėvas A	A	A	A	A	A	A	A	A	A	A
tėvas B	B	B	B	B	B	B	B	B	B	B
vaikas	A	A	A	A	A	A	B	B	B	B

16 pav. *Kryžminimas per vieną tašką*

tėvas A	A	A	A	A	A	A	A	A	A	A
tėvas B	B	B	B	B	B	B	B	B	B	B
vaikas	B	B	A	A	A	A	B	B	B	B

17 pav. *Kryžminimas per du taškus*

Toliau gautiems naujiems sprendiniams pritaikoma mutacija. Mutacijos metu tam tikros naujų sprendinių dalys yra pakeičiamos. Taip daroma norint išvengti populiacijos užstrigimo lokaliame optimume. Mutacijos pavyzdys pateiktas 18 pav. Svarbu atkreipti dėmesį į tai, kad mutacijos tikimybė neturėtų būti labai didelė. Kuo didesnė ši tikimybė, tuo labiau genetinis algoritmas panašėja į paprasčiausia atsitiktinį algoritmą. Klasikinėje schemoje mutacijos tikimybė nesikeičia algoritmo vykdymo metu, todėl galima tarti, kad mutacijos tikimybė yra vienas iš genetinio algoritmo parametrų.

senas	A	A	A	A	A	A	A	A	A	A
naujas	A	A	A	A	A	B	A	A	A	A

18 pav. Vieno geno mutacija

5.3 Bayes'o optimizavimo metodai

Tradicinė skaitinė analizė apibrėžia optimizavimo algoritmus, kurie garantuoja tam tikrą visų optimizuojamų funkcijų tikslumą. Tai apima tikslius algoritmus. Tai yra blogiausio atvejo analizė. Norint apriboti klaidų skaičių reikia daugiau skaičiavimo bandymų, dažniausiai didėjančių eksponentiškai. Tai pagrindinis blogiausio atvejo analizės trūkumas.

Kaip alternatyva yra naudojama vidutinio atvejo analizė. Čia vidutinė paklaida ne apribojama, bet sumažinama tiek kiek įmanoma. Vidurkis imamas optimizuojamų funkcijų rinkiniui. Tokia vidutinio atvejo analizė vadinama Bayes'o metodu [16].

Bayes'o optimizacijos taikymui yra keli būdai. Pirmasis yra tiesioginis Bayes'o algoritmas. Jis apibrėžiamas fiksuojant ankstesnio išsibarstymo P funkcijų f(x) rinkiniui ir minimizuojant Bayes'o rizikos funkciją. Rizikos funkcija R(x) yra planuojamas skirtumas iš globalaus minimumo fiksuotame taške x. P išsibarstymas yra apgalvotas kaip stochastinis modelis funkcijai f(x), $x \in R$, kur f(x) gali būti determinuota arba stochastinė funkcija. Gauso atveju, laikant kad (n+1) stebėjimas yra paskutinis

$$R(x) = \frac{1}{\sqrt{2\pi s_n(x)}} \times \int_{-\infty}^{+\infty} min(c_n, z) e^{-\frac{1}{2}\left(\frac{y - m_n(x)}{s_n(x)}\right)^2} dz$$

Čia $c_n = \min_i z_i$ -, $z_i = f(x_i)$. $m_n(x)$ yra sąlyginė tikimybė atsižvelgiant į stebėjimų reikšmes z_i, i = 1, ..., n. $s_n^2(x)$ yra sąlyginis pasiskirstymas ir $\varepsilon > 0$ yra taisymo parametras. Rizikos funkcijos R(x) minimumas yra surandamas taške:

$$x_{n+1} = \arg\max_x \frac{s_n(x)}{m_n(x) - c_n}$$

Tiesioginio Bayes'o algoritmo tikslas, naudojamas daugeliu atvejų, yra pateikti kiek įmanomai mažą paklaidą laikantis konvergavimo sąlygų.

Wienerio procesas yra bendras stochastinis modelis vienmačiu atveju m = 1. Šis modelis laiko, kad skirtumai $f(x_4) - f(x_3)$ ir $f(x_2) - f(x_1)$, $x_1<x_2<x_3<x_4$, yra stochastiškai nepriklausomi. Čia f(x) yra Gauso $(0, \sigma x)$ kiekviename fiksuotame x > 0.

Taip pat šis modelis yra išplėstas į daugiamatį atvejį. Tačiau paprasti apytiksliai stochastiniai modeliai pageidautini, jei m>1. Apytiksliai modeliai yra suprojektuoti pakeičiant tradicines Kolmogorovo tikslumo sąlygoms. Šios sąlygos reikalauja n-tojo laipsnio matricų inversijos, kad surasti sąlyginę tikimybę $m_n(x)$ ir pasiskirstymą $s_n^2(x)$.

Keičiant įprastas tikslumo sąlygas (rizikos funkcijos R(x) tęstinumu, x_n konvergavimu iki globalaus minimumo, išraiškų $m_n(x)$ ir $s_n(x)$ supaprastinimu) R(x) išraiška apskaičiuojama naudojant formulę:

$$R(x) = \min_{1 \le i \le n} z_i - \min_{1 \le i \le n} \frac{x - x_i^2}{z_i - c_n}.$$

Kitas būdas yra Bayes'o euristinis metodas. Jis fiksuoja ankstesnį pasiskirstymą P pagalbinėms funkcijoms $f_K(x)$. Šios funkcijos apibrėžia geriausias reikšmes gautas K kartų, naudojant kokią nors euristiką h(x). Laikoma kad euristika h(x) priklauso nuo tęstinių parametrų $\rightarrow x \in R^m$. Euristika padeda optimizuoti originalią funkciją C(y) kintamiesiems $y \in R^n$, kur m < n. Paprastai, komponentai y yra diskretinės reikšmės. Euristika yra paremta ekspertų nuomonėmis apie sprendimo prioritetus.

Bayes'o euristinis metodas naudojamas optimizuojant funkcijas su dideliais kintamųjų skaičiais. Tai daugelio diskretinių optimizavimo problemų atvejai. Ekspertų nuomonėmis pagrįstos euristikos dažnai įtraukia atsitiktinių skaičių generavimo procedūras.

6 NAUJASIS *SBR* SPRENDIMO BŪDAS

6.1 *SBR* sprendimas naudojant *SSBR*

Kaip matyti iš ankstesnių skyrių *SBR* yra NP-sunkus, o *SSBR* išsprendžiamas per laiką $\mathrm{O}(n^2)$. Būtent pastaruoju skirtumu mes ir pasinaudosime. Tarkime turime neženklintą perstatymą π, kurį ir reikia surikiuoti inversijomis. Pažymėkime aibę visų galimų π suženklinimų $S(\pi)$, t. y. aibę tokių ženklintų perstatymų, kurių elementų moduliai sutampa su π elementais, o skiriasi tik jų ženklai. Pavyzdžiui, jeigu turime $\pi = (1,2)$, tada $S(\pi) = ((1,2),(-1,2),(1,-2),(-1,-2))$.

Teorema 3. Kiekviena inversijų seka $\rho(\pi')$, surikiuojanti $\pi' \in S(\pi)$, surikiuoja ir π.

Įrodymas. Pagal mūsų apibrėžimą $\pi_i = |\pi'_i|$ su visais $i = 1,2,\ldots,n$. Tada nesunku įsitikinti, kad pritaikius bet kokią inversiją ρ perstatymams π ir π' gautume naujus perstatymus, kurie tenkintų lygybę $(\pi\rho)_i = |(\pi'\rho)_i|$ su visais $i = 1,2,\ldots,n$. Duota, kad $\rho(\pi')$ surikiuoja π', t. y. $\pi'\rho(\pi') = \iota$. Bet taip pat parodėme, kad $|\pi'\rho(\pi')| = \pi\rho(\pi')$. Įvertinus tai, kad $|\iota| = \iota$, gauname $\pi\rho(\pi') = \iota$, t. y. kad $\rho(\pi')$ surikiuoja π.

Teorema 4. Egzistuoja toks π suženklinimas $\pi' \in S(\pi)$, kurio minimali inversijų seka $\rho_{\min}(\pi')$ yra taip pat minimali inversijų seka surikiuojanti π.

Įrodymas. Tarkime, kad žinome minimalią inversijų seką $\rho_{\min}(\pi) = \rho_1, \rho_2, \ldots, \rho_{d(\pi)}$, kuri surikiuoja π. Apvertę šią seką gauname naują seką $\rho^{-1}(\pi) = \rho_{d(\pi)}, \rho_{d(\pi)-1}, \ldots, \rho_1$. Pritaikę seką $\rho^{-1}(\pi)$ tapačiajam neženklintam perstatymui gausime π, o pritaikę ją tapačiajam ženklintam perstatymui gausime $\pi' \in S(\pi)$. Vadinasi $\rho_{\min}(\pi) = \rho_{\min}(\pi')$.

Įvertinę 3 ir 4 teoremas neženklinto perstatymo rikiavimo inversijomis uždavinį galime suformuluoti taip:

Reikia rasti tokį π ženklinimą $\pi' \in S(\pi)$, kurio minimali inversijų seka $\rho_{\min}(\pi')$ yra trumpiausia. Tada $\rho_{\min}(\pi) = \rho_{\min}(\pi')$.

Nauja uždavinio sprendimo strategija nepakeičia uždavinio sudėtingumo. Nesunku įsitikinti, kad $|S(\pi)| = 2^n$. Taigi norint rasti π suženklinimą $\pi' \in S(\pi)$, kurio minimali inversijų seka yra trumpiausia, reiktų apskaičiuoti visų aibės $S(\pi)$ elementų inversinį

atstumą. Geriausias žinomas algoritmas inversinį atstumą ženklintiems perstatymams randa per laiką $O(n)$. Vadinasi rasti optimalų suženklinimą reiktų $O(2^n n)$ laiko.

6.2 Universali *SBR* sprendimo schema

Pagrindinis naujos strategijos privalumas yra tas, kad ji leidžia *SBR* sprendimui taikyti klasikines euristinių paieškos algoritmų schemas. Taip yra todėl, kad dabar norint rasti *SBR* sprendinį nėra būtina operuoti inversijų seka, o pakanka operuoti suženklinimu $\pi' \in S(\pi)$. Suženklinimą π' patogu vaizduoti ilgio *n* bitų masyvu s, kurio *i*-oje pozicijoje esantis bitas reiškia suženklinimo π' *i*-ojo elemento ženklą. Taigi $s_i = 0$, kai $\pi'_i < 0$, ir $s(\pi')_i = 1$, kai $\pi'_i > 0$, su visais $i = 1,2,\ldots,n$. Akivaizdu, kad pritaikius suženklinimą s neženklintam perstatymui π, gausime π'. Taigi galime tarti, kad suženklinimas s yra *SBR* sprendinys, kuris atitinką inversinį atstumą $d(s) = d(\pi s) = d(\pi')$. 19 pav. pateikta universali *SBR* sprendimo schema.

Įvestis: neženklintas perstatymas π.
Išvestis: minimali inversijų seka $\rho_{\min}(\pi) = \rho_1, \rho_2, \ldots, \rho_{d(\pi)}$, kurią pritaikius perstatymui π gautume tapatųjį perstatymą.
1. **[Suženklinimas]** rasti suženklinimą s, kurio atitinkamas inversinis atstumas yra minimalus. 2. **[Sprendinys]** gautam ženklintam sprendiniui $\pi' = \pi s$ išspręsti *SSBR* uždavinį. Gauta minimali inversijų seka bus norimas rezultatas ($\rho_{\min}(\pi) = \rho_{\min}(\pi')$).

19 pav. *Universali SBR sprendimo schema*

Kaip parodėme anksčiau universalios *SBR* sprendimo schemos pirmojo žingsnio sudėtingumas yra $O(2^n n)$, o antrojo $O(n^2)$. Taigi akivaizdu, kad norint rasti optimalų π suženklinimą, kai *n* yra pakankamai didelis, prireiktų labai daug laiko. Todėl praktikoje suženklinimo paieškai teks naudoti euristinius metodus. Taip pat svarbu pastebėti, kad universali *SBR* sprendimo schema negarantuoja optimalaus sprendinio. Sprendinio gerumas priklauso nuo rasto suženklinimo gerumo, kuris priklauso nuo panaudotos euristikos tinkamumo.

7 EKSPERIMENTINIS NAUJOJO *SBR* SPRENDIMO BŪDO TYRIMAS

Šiame darbe nėra svarbu sukurti optimalų ir patikimą euristinį algoritmą. Svarbiausia ištirti įvairių klasikinių euristinių algoritmų tinkamumą *SBR* sprendimui. Toliau apžvelgsime tabu paieškos bei genetinio algoritmo taikymą problemos sprendimui.

Eksperimentiniams tyrimams naudosime duomenis, kurių optimalios reikšmės $d(s*)$ rastos Caprara darbe [9]. Kiekvienas duomenų rinkinys turi savo pavadinimą, kuris yra unikalus. Toliau lentelėse bus naudojami unikalūs duomenų rinkinių pavadinimai. Pavadinimai prasidedantys *p25*, reikš testinius duomenis, kuriuose $n = 25$. Pavadinimai prasidedantys *p100*, reikš testinius duomenis, kuriuose $n = 100$. Pavadinimai prasidedantys *p200*, reikš testinius duomenis, kuriuose $n = 200$.

Eksperimento metu stengsimės surasti optimalius tiriamo algoritmo parametrus. Parametrų optimizavimą atliksime rankiniu būdu, bei pasinaudoję *GMJ* paketu [17]. *GMJ* paketas, naudoją Bayes'o metodą euristinių algoritmų parametrų optimizavimui. Tada pritaikę gautus parametrus rasime turimų duomenų rinkinių sprendinius ir palyginsime gautus rezultatus su kitų žinomų *SBR* aproksimuojančių algoritmų rastais sprendiniais. Kadangi mūsų metodas nėra tikslus, tai rasto sprendinio gerumą vertinsime pagal vidutinį jo nuokrypį nuo optimalaus sprendinio. Kiekvienam testiniam duomenų rinkiniui algoritmą leisime m kartų ir skaičiuosime normuotą vidutinį nuokrypį:

$$\sigma = \frac{\sum_{i=1}^{m} d(s_i) - d(s*)}{m d(s*)}$$

Darbe naudosime $m = 10$. Abu realizuoti euristiniai algoritmai naudos mūsų pačių realizuotą tiesinio sudėtingumo algoritmą randanti inversinį ženklinto perstatymo atstumą [4].

7.1 Tabu paieškos taikymas *SBR* sprendimui

Šiame darbe realizavome klasikinį tabu paieškos algoritmą [14]. Sprendinio s kaimyninių sprendinių aibę $N(s)$ sudaro visi suženklinimai, kurie nuo suženklinimo s skiriasi tik vienos pozicijos ženklu. Kadangi tikslo funkcijos sudėtingumas yra $O(n)$, o kiekvienos iteracijos metų peržiūrimi $n-1$ kaimynai, tai bendras vienos iteracijos sudėtingumas yra $O(n^2)$.

Tabu laikas nusako, kiek iteracijų negalima keisti ženklo tam tikroje pozicijoje.

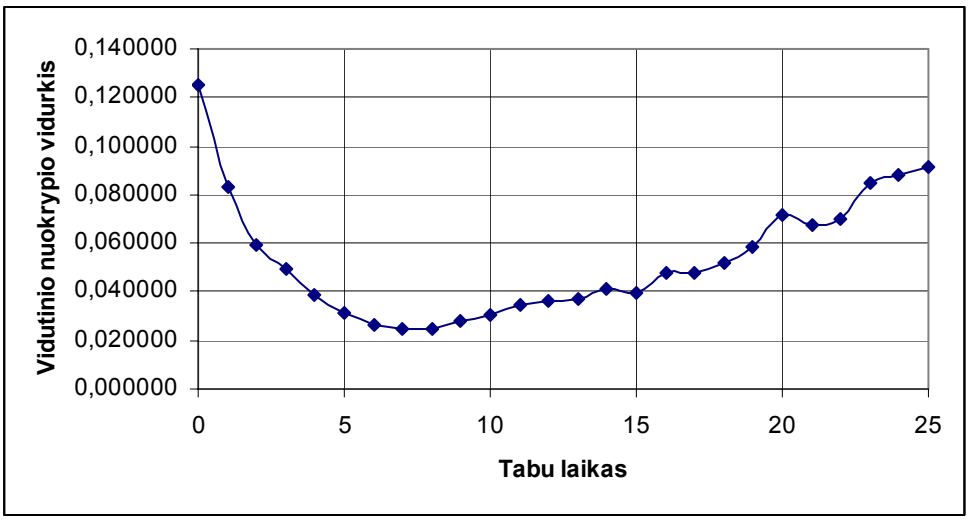
7.1.1 Optimalių algoritmo parametrų radimas

Vienintelis mūsų realizuotos tabu paieškos algoritmo parametras yra tabu laikas. Tiriant tabu laiko įtaką rastų sprendinių gerumui naudosime sąlyginai mažą iteracijų skaičių tam, kad algoritmas nekonverguotų į optimalius sprendinius.

1 lentelėje pavaizduota, kaip tabu laikas įtakoja tabu paieškos rezultatų gerumą naudojant testinius duomenis, kai $n = 25$. 20 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko. Iš šių duomenų galime daryti išvadą, kad kai $n = 25$ optimalus tabu laikas yra 8. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

1 lentelė *Tabu paieškos rezultatų priklausomybė nuo tabu laiko, kai* $n = 25$

Tabu laikas	Iteracijų skaičius	Normuotas vidutinis nuokrypis					Vidutinio nuokrypio vidurkis
		p25_1	p25_2	p25_3	p25_4	p25_5	
0	100	0,0909	0,1500	0,1579	0,1414	0,0860	0,1252
1	100	0,0728	0,0503	0,0698	0,1371	0,0860	0,0832
2	100	0,0278	0,0230	0,0419	0,1371	0,0659	0,0591
3	100	0,0141	0,0000	0,0533	0,1500	0,0286	0,0492
4	100	0,0278	0,0116	0,0419	0,0909	0,0230	0,0390
5	100	0,0278	0,0230	0,0533	0,0116	0,0395	0,0310
6	100	0,0278	0,0058	0,0533	0,0058	0,0395	0,0265
7	100	0,0345	0,0116	0,0476	0,0116	0,0173	0,0245
8	100	0,0345	0,0058	0,0533	0,0058	0,0230	0,0245
9	100	0,0345	0,0058	0,0419	0,0058	0,0503	0,0277
10	100	0,0411	0,0058	0,0476	0,0058	0,0503	0,0301
11	100	0,0476	0,0116	0,0588	0,0058	0,0503	0,0348
12	100	0,0604	0,0000	0,0588	0,0173	0,0449	0,0363
13	100	0,0604	0,0116	0,0588	0,0058	0,0503	0,0374
14	100	0,0728	0,0230	0,0643	0,0000	0,0449	0,0410
15	100	0,0604	0,0230	0,0476	0,0230	0,0449	0,0398
16	100	0,0667	0,0286	0,0643	0,0116	0,0659	0,0474
17	100	0,0789	0,0286	0,0588	0,0173	0,0556	0,0478
18	100	0,0728	0,0341	0,0643	0,0173	0,0710	0,0519
19	100	0,0789	0,0503	0,0698	0,0230	0,0710	0,0586
20	100	0,0909	0,0659	0,0698	0,0449	0,0860	0,0715
21	100	0,0850	0,0503	0,0643	0,0556	0,0811	0,0672
22	100	0,0667	0,0608	0,0643	0,0710	0,0860	0,0698
23	100	0,0909	0,0811	0,0805	0,0860	0,0860	0,0849
24	100	0,1026	0,0811	0,0909	0,0811	0,0860	0,0883
25	100	0,1026	0,0860	0,0960	0,0860	0,0860	0,0913

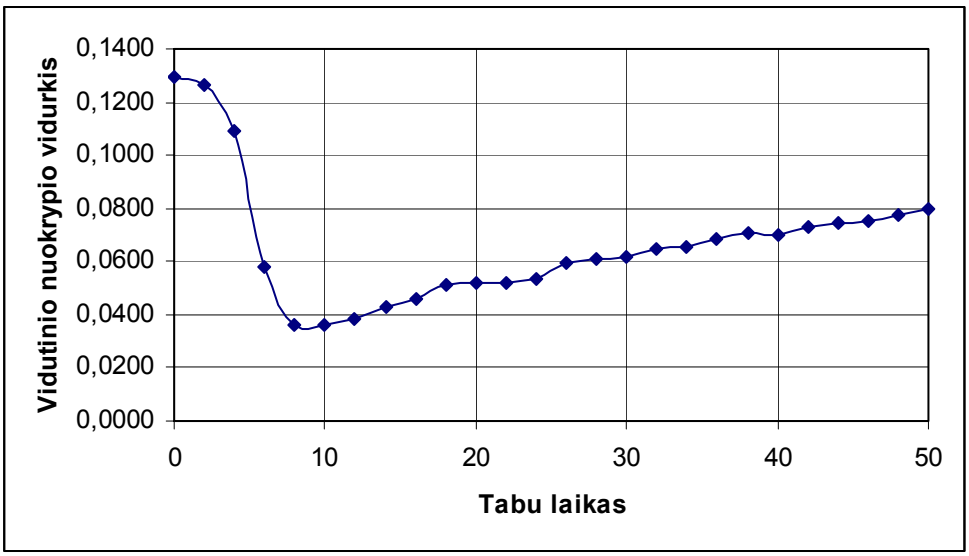


20 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko, kai* $n = 25$

2 lentelėje pavaizduota, kaip tabu laikas įtakoja tabu paieškos rezultatų gerumą naudojant testinius duomenis, kai $n = 100$. 21 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko. Iš šių duomenų galime daryti išvadą, kad kai $n = 100$ optimalus tabu laikas yra 10. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

2 lentelė *Tabu paieškos rezultatų priklausomybė nuo tabu laiko, kai* $n = 100$

Tabu laikas	Iteracijų skaičius	Normuotas vidutinis nuokrypis					Vidutinio nuokrypio vidurkis
		p100_1	p100_2	p100_3	p100_4	p100_5	
0	1000	0,1244	0,0983	0,1461	0,1246	0,1553	0,1297
2	1000	0,1173	0,1003	0,1489	0,1136	0,1514	0,1263
4	1000	0,1080	0,0877	0,1508	0,0659	0,1325	0,1090
6	1000	0,0732	0,0499	0,1090	0,0238	0,0352	0,0582
8	1000	0,0440	0,0441	0,0343	0,0286	0,0314	0,0365
10	1000	0,0343	0,0406	0,0343	0,0311	0,0390	0,0358
12	1000	0,0306	0,0418	0,0392	0,0323	0,0476	0,0383
14	1000	0,0476	0,0453	0,0440	0,0311	0,0452	0,0426
16	1000	0,0452	0,0441	0,0535	0,0394	0,0488	0,0462
18	1000	0,0524	0,0465	0,0535	0,0418	0,0609	0,0510
20	1000	0,0512	0,0511	0,0500	0,0465	0,0609	0,0519
22	1000	0,0524	0,0511	0,0524	0,0406	0,0633	0,0519
24	1000	0,0524	0,0545	0,0500	0,0429	0,0668	0,0533
26	1000	0,0571	0,0625	0,0606	0,0476	0,0680	0,0592
28	1000	0,0582	0,0602	0,0582	0,0545	0,0727	0,0608
30	1000	0,0606	0,0580	0,0617	0,0557	0,0738	0,0620

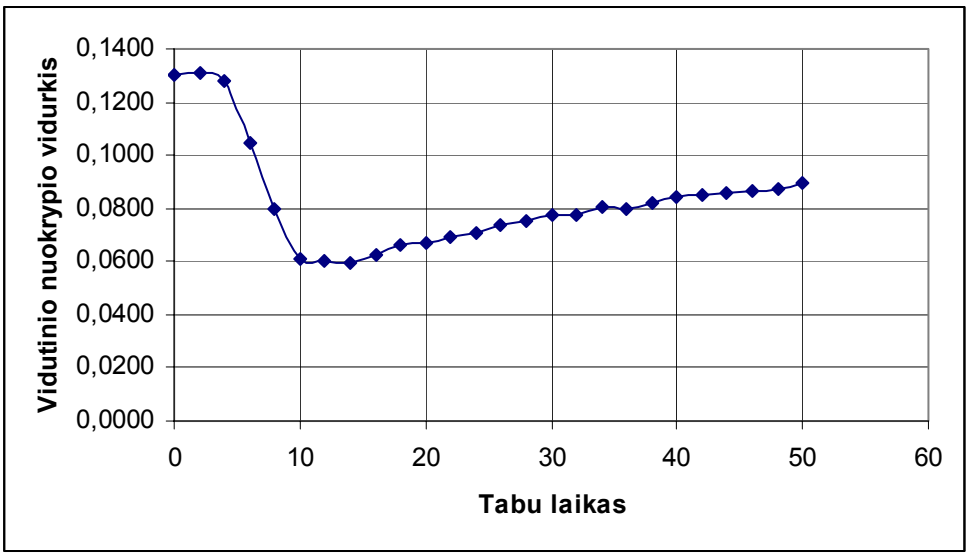


21 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko, kai* $n = 100$

3 lentelėje pavaizduota, kaip tabu laikas įtakoja tabu paieškos rezultatų gerumą naudojant testinius duomenis, kai $n = 200$. 22 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko. Iš šių duomenų galime daryti išvadą, kad kai $n = 200$ optimalus tabu laikas yra 14. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

3 lentelė *Tabu paieškos rezultatų priklausomybė nuo tabu laiko, kai* $n = 200$

Tabu laikas	Iteracijų skaičius	Normuotas vidutinis nuokrypis					Vidutinio nuokrypio vidurkis
		p200_1	p200_2	p200_3	p200_4	p200_5	
0	1000	0,1120	0,1359	0,1337	0,1408	0,1273	0,1300
2	1000	0,1195	0,1363	0,1252	0,1376	0,1363	0,1310
4	1000	0,1135	0,1297	0,1323	0,1408	0,1249	0,1283
6	1000	0,0951	0,1042	0,1066	0,1222	0,0950	0,1046
8	1000	0,0610	0,0723	0,0836	0,1048	0,0767	0,0797
10	1000	0,0571	0,0625	0,0638	0,0695	0,0519	0,0609
12	1000	0,0548	0,0581	0,0665	0,0651	0,0581	0,0605
14	1000	0,0514	0,0564	0,0638	0,0618	0,0631	0,0593
16	1000	0,0548	0,0575	0,0671	0,0684	0,0653	0,0626
18	1000	0,0627	0,0669	0,0687	0,0722	0,0619	0,0665
20	1000	0,0677	0,0642	0,0687	0,0728	0,0625	0,0672
22	1000	0,0649	0,0647	0,0708	0,0760	0,0696	0,0692
24	1000	0,0671	0,0713	0,0714	0,0722	0,0702	0,0704
26	1000	0,0737	0,0729	0,0762	0,0739	0,0723	0,0738
28	1000	0,0716	0,0750	0,0789	0,0766	0,0729	0,0750
30	1000	0,0754	0,0767	0,0799	0,0819	0,0745	0,0777

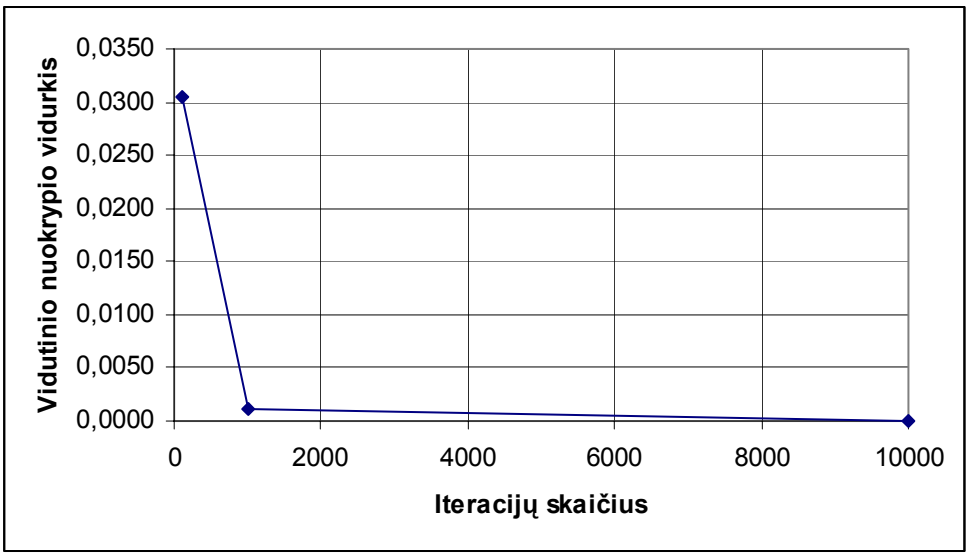


22 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo tabu laiko, kai* $n = 200$

Apibendrinus gautus rezultatus matome, kad optimali tabu laiko reikšmė nėra vienoda bet kokio dydžio perstatymams. Kuo perstatymas didesnis, tuo optimalus tabu laikas ilgesnis. Taip pat pastebima, kad per mažas tabu laikas sprendinio gerumui turi didesnę neigiamą įtaką nei per didelis.

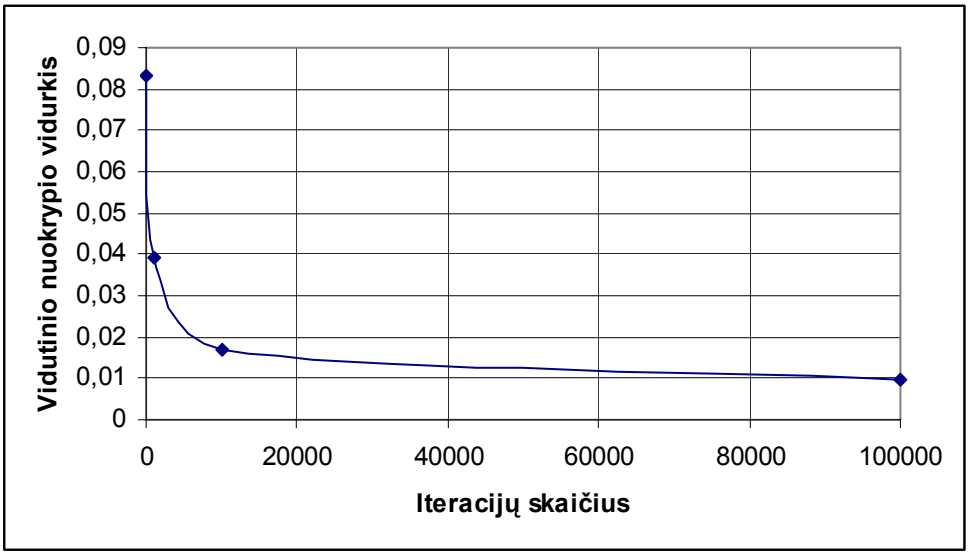
7.1.2 Algoritmo tyrimas

Toliau tabu paieškos tyrime naudosime anksčiau nustatytas optimalias parametrų reikšmes. Euristinių, o tuo pačiu ir tabu paieškos, algoritmų gaunamo sprendinio gerumas tiesiogiai priklauso nuo parinktų parametrų optimalumo ir atliktų iteracijų skaičiaus. 23 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 25$. Matome, kad rezultatai žymiai gerėja didinant iteracijų skaičių, o kai iteracijų skaičius yra 10000 tabu paieška visą laiką gražina optimalias reikšmes.



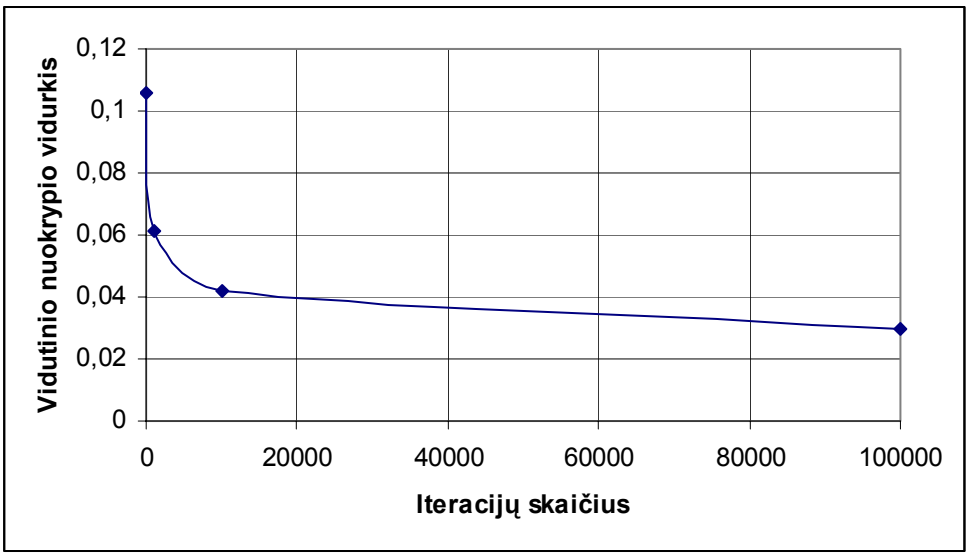
23 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 25$

24 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 100$. Čia taip pat matosi, kad rezultatai žymiai gerėja didinant iteracijų skaičių.



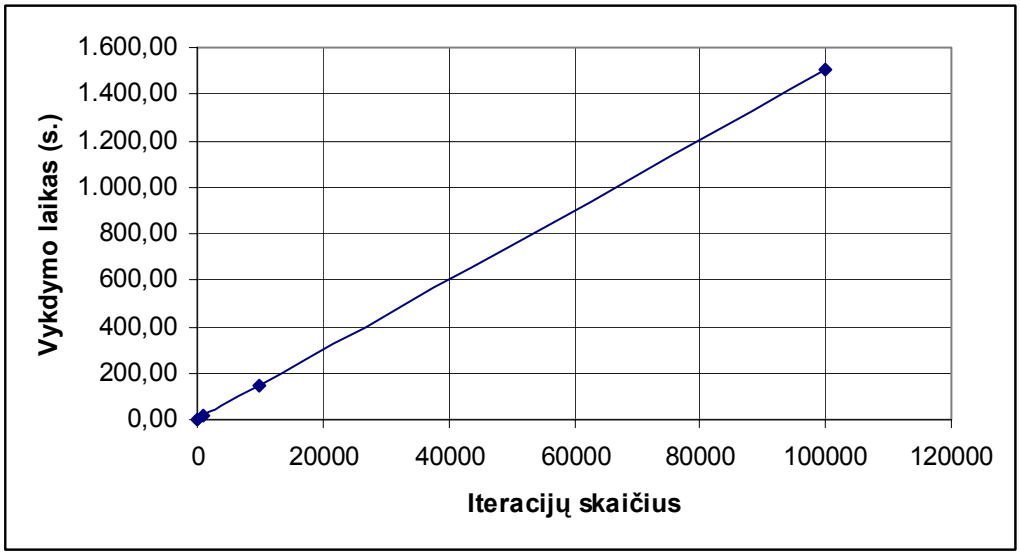
24 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 100$

25 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 200$. Čia taip pat matosi, kad rezultatai žymiai gerėja didinant iteracijų skaičių.

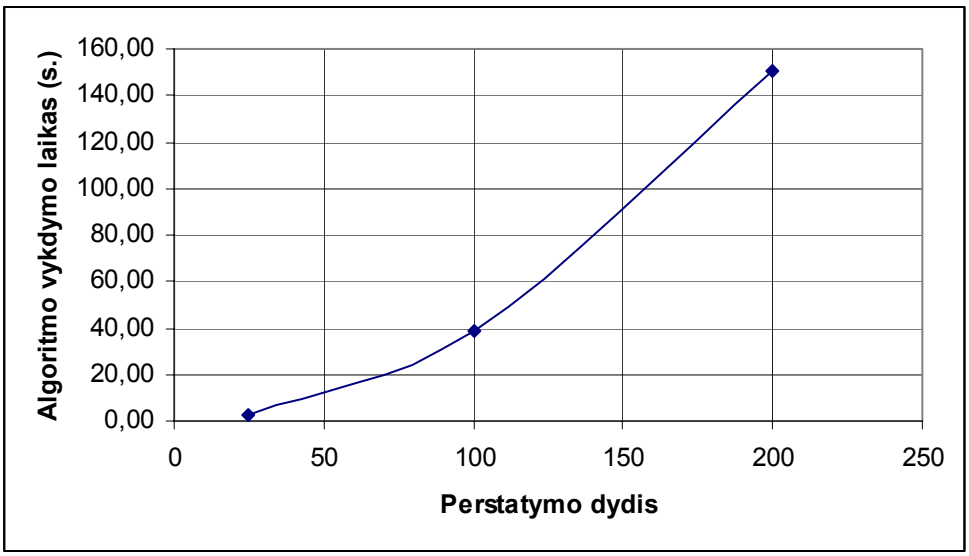


25 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 200$

Toliau ištirkime algoritmo laikines charakteristikas. 26 pav. pavaizduota algoritmo vykdymo laiko priklausomybė nuo iteracijų skaičiaus. Galima daryti išvada, kad algoritmo vykdymo laikas tiesiogiai priklauso nuo iteracijų skaičiaus. 27 pav. pavaizduota algoritmo vykdymo laiko priklausomybė nuo perstatymo dydžio n. Čia matosi polinominė priklausomybė, kuri patvirtina ankstesnį mūsų teiginį, kad algoritmo iteracijos sudėtingumas yra $O(n^2)$.



26 pav. Algoritmo vykdymo laiko priklausomybė nuo iteracijų skaičiaus



27 pav. Algoritmo vykdymo laiko priklausomybė nuo perstatymo dydžio

Taigi apibendrinant tabu paieškos algoritmo taikymo *SBR* uždaviniui spręsti tyrimą, galime pasakyti, kad algoritmas neblogai susidorojo su problema. Nors prie didesnių *n* reikšmių optimumas nebuvo dažnai pasiekiamas, tačiau matomos tendencijos, kad didinant iteracijų skaičių sprendinys gerėja. 4 lentelėje pateiktos reikšmės rastos tabu paieškos testiniams duomenims. Lentelėje paryškintos reikšmės atitinka optimumus.

4 lentelė *Tabu paieškos algoritmo rasti minimalūs inversiniai atstumai*

Perstatymas	Dydis	Minimalus inversinis atstumas	Geriausias ir blogiausias rastas inversinis atstumas							
			100 iteracijų		1000 iteracijų		10000 iteracijų		100000 iteracijų	
			Ger.	Blog.	Ger.	Blog.	Ger.	Blog.	Ger.	Blog.
p25_1	25	**14**	**14**	15	**14**	**14**	**14**	**14**	-	-
p25_2	25	**17**	**17**	18	**17**	**17**	**17**	**17**	-	-
p25_3	25	**16**	**16**	17	**16**	17	**16**	**16**	-	-
p25_4	25	**17**	**17**	18	**17**	**17**	**17**	**17**	-	-
p25_5	25	**17**	**17**	18	**17**	**17**	**17**	**17**	-	-
p100_1	100	**76**	80	85	79	80	77	78	77	77
p100_2	100	**78**	83	85	80	81	79	80	**78**	79
p100_3	100	**76**	83	85	78	81	77	78	77	77
p100_4	100	**78**	83	88	79	82	79	80	**78**	79
p100_5	100	**74**	79	84	77	78	**74**	76	**74**	75
p200_1	200	**157**	169	174	163	168	162	165	161	162
p200_2	200	**159**	180	181	167	172	165	166	163	164
p200_3	200	**160**	176	180	168	173	166	169	164	166
p200_4	200	**158**	177	180	168	172	165	166	161	164
p200_5	200	**159**	175	181	169	171	166	167	164	165

7.2 Genetinio algoritmo taikymas *SBR* sprendimui

Šiame darbe realizavome genetinį algoritmą pagal [12] darbe pateiktą schemą. Sukurtas algoritmas kryžminimui naudoja du tėvus ir vieną pjūvio tašką. Jo parametrai yra populiacijos dydis ir mutacijos tikimybė. Parametrų prasmė nusakytą skyriuje 5.2.

7.2.1 Optimalių algoritmo parametrų radimas

Visų pirma genetinio algoritmo parametrus nustatėme naudojant *GMJ* paketą. Taip padarėme todėl, kad sunku nustatyti dviejų parametrų optimalias reikšmes rankiniu būdu – reiktų perrinkti visas poras, kas užimtų daug laiko. 5 lentelėje pateiktos *GMJ* paketo rastos optimalios parametrų reikšmės skirtingų dydžiu perstatymams.

5 lentelė Optimalios genetinio algoritmo parametrų reikšmės rasto *GMJ* paketo

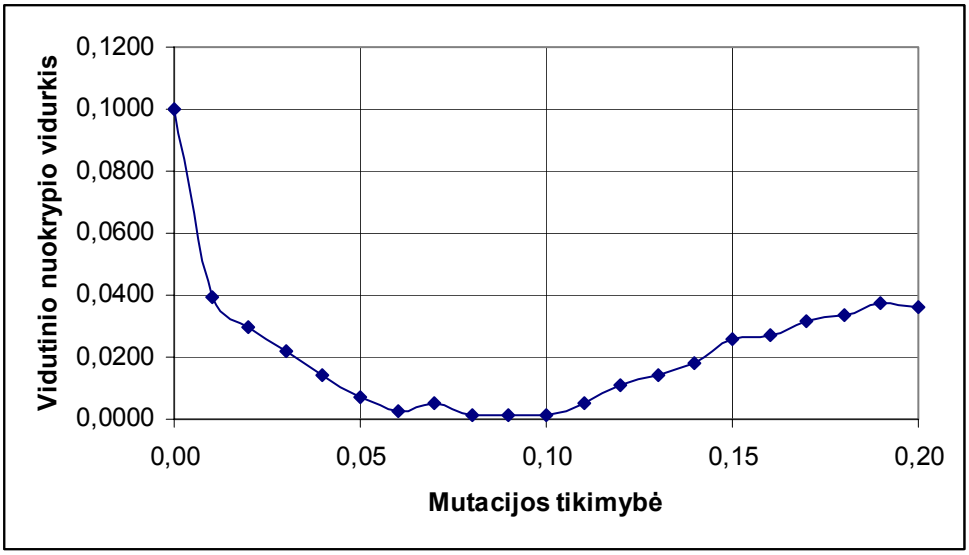
Dydis	Populiacijos dydis	Mutacijos tikimybė
25	30	0,09
100	110	0,05
200	230	0,03

Tyrimo metu buvo pastebėta, kad tikroji optimali populiacijos dydžio reikšmė yra begalybė. Kuo didesnė populiacija, tuo geresnė randama reikšmė. Taip yra todėl, kad kuo populiacija didesnė, tuo daugiau peržiūrima galimų sprendinių. Taigi darbe populiacijos dydis bus naudojamas šiek tiek didesnis už perstatymo dydį.

6 lentelėje pavaizduota, kaip mutacijos tikimybė įtakoja genetinio algoritmo rezultatų gerumą naudojant testinius duomenis, kai $n = 25$. 28 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės. Iš šių duomenų galime daryti išvadą, kad kai $n = 25$ optimali mutacijos tikimybė yra 0,09. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

6 lentelė *Genetinio algoritmo rezultatų priklausomybė nuo mutacijos, kai* $n = 25$

Populiacijos dydis	Mutacijos tikimybė	Iteracijų skaičius	p25_1	p25_2	p25_3	p25_4	p25_5	Vidutinio nuokrypio vidurkis
30	0,00	1000,0000	0,1716	0,0761	0,1011	0,0556	0,0957	0,1000
30	0,01	1000,0000	0,0604	0,0058	0,0588	0,0116	0,0608	0,0395
30	0,02	1000,0000	0,0278	0,0116	0,0533	0,0116	0,0449	0,0298
30	0,03	1000,0000	0,0210	0,0000	0,0476	0,0058	0,0341	0,0217
30	0,04	1000,0000	0,0071	0,0000	0,0533	0,0000	0,0116	0,0144
30	0,05	1000,0000	0,0000	0,0000	0,0361	0,0000	0,0000	0,0072
30	0,06	1000,0000	0,0000	0,0000	0,0123	0,0000	0,0000	0,0025
30	0,07	1000,0000	0,0000	0,0000	0,0244	0,0000	0,0000	0,0049
30	0,08	1000,0000	0,0000	0,0000	0,0062	0,0000	0,0000	0,0012
30	0,09	1000,0000	0,0000	0,0000	0,0062	0,0000	0,0000	0,0012
30	0,10	1000,0000	0,0000	0,0000	0,0062	0,0000	0,0000	0,0012
30	0,11	1000,0000	0,0000	0,0000	0,0244	0,0000	0,0000	0,0049
30	0,12	1000,0000	0,0000	0,0000	0,0476	0,0000	0,0058	0,0107
30	0,13	1000,0000	0,0071	0,0000	0,0533	0,0000	0,0116	0,0144
30	0,14	1000,0000	0,0141	0,0000	0,0533	0,0000	0,0230	0,0181
30	0,15	1000,0000	0,0476	0,0000	0,0588	0,0000	0,0230	0,0259
30	0,16	1000,0000	0,0476	0,0000	0,0588	0,0000	0,0286	0,0270
30	0,17	1000,0000	0,0604	0,0000	0,0588	0,0000	0,0395	0,0318
30	0,18	1000,0000	0,0476	0,0116	0,0588	0,0058	0,0449	0,0338
30	0,19	1000,0000	0,0667	0,0116	0,0588	0,0000	0,0503	0,0375
30	0,20	1000,0000	0,0667	0,0058	0,0588	0,0000	0,0503	0,0363

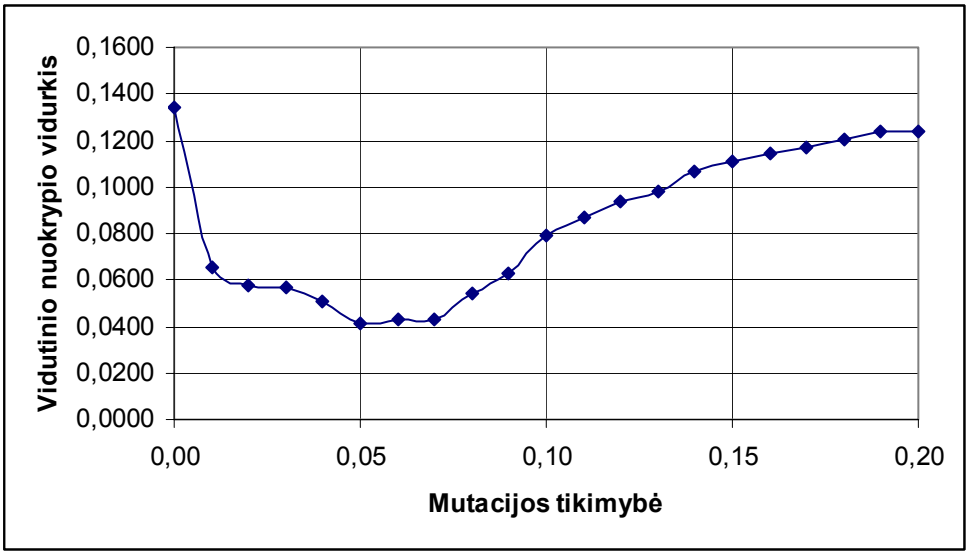


28 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės, kai* $n = 25$

7 lentelėje pavaizduota, kaip mutacijos tikimybė įtakoja genetinio algoritmo rezultatų gerumą naudojant testinius duomenis, kai $n = 100$. 29 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės. Iš šių duomenų galime daryti išvadą, kad kai $n = 100$ optimali mutacijos tikimybė yra 0,05. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

7 lentelė *Genetinio algoritmo rezultatų priklausomybė nuo mutacijos, kai* $n = 100$

Populiacijos dydis	Mutacijos tikimybė	Iteracijų skaičius	p100_1	p100_2	p100_3	p100_4	p100_5	Vidutinio nuokrypio vidurkis
110	0,00	1.000	0,1373	0,1216	0,1304	0,1275	0,1533	0,1340
110	0,01	1.000	0,0617	0,0568	0,0754	0,0511	0,0819	0,0654
110	0,02	1.000	0,0524	0,0647	0,0559	0,0488	0,0680	0,0580
110	0,03	1.000	0,0559	0,0591	0,0535	0,0476	0,0668	0,0566
110	0,04	1.000	0,0524	0,0488	0,0464	0,0453	0,0597	0,0505
110	0,05	1.000	0,0440	0,0418	0,0380	0,0370	0,0452	0,0412
110	0,06	1.000	0,0392	0,0476	0,0488	0,0335	0,0439	0,0426
110	0,07	1.000	0,0440	0,0488	0,0440	0,0347	0,0439	0,0431
110	0,08	1.000	0,0547	0,0580	0,0594	0,0499	0,0488	0,0542
110	0,09	1.000	0,0675	0,0625	0,0582	0,0625	0,0645	0,0630
110	0,10	1.000	0,0743	0,0802	0,0810	0,0780	0,0819	0,0791
110	0,11	1.000	0,0821	0,0888	0,0898	0,0780	0,0954	0,0868
110	0,12	1.000	0,0909	0,0951	0,0942	0,0856	0,1030	0,0938
110	0,13	1.000	0,0952	0,0951	0,1027	0,0941	0,1030	0,0980
110	0,14	1.000	0,1017	0,1086	0,1111	0,0951	0,1159	0,1065
110	0,15	1.000	0,1111	0,1136	0,1090	0,0983	0,1243	0,1113
110	0,16	1.000	0,1121	0,1116	0,1121	0,1055	0,1284	0,1140
110	0,17	1.000	0,1173	0,1146	0,1173	0,1086	0,1294	0,1174
110	0,18	1.000	0,1163	0,1186	0,1224	0,1116	0,1345	0,1207
110	0,19	1.000	0,1224	0,1226	0,1264	0,1146	0,1345	0,1241
110	0,20	1.000	0,1244	0,1146	0,1264	0,1156	0,1405	0,1243

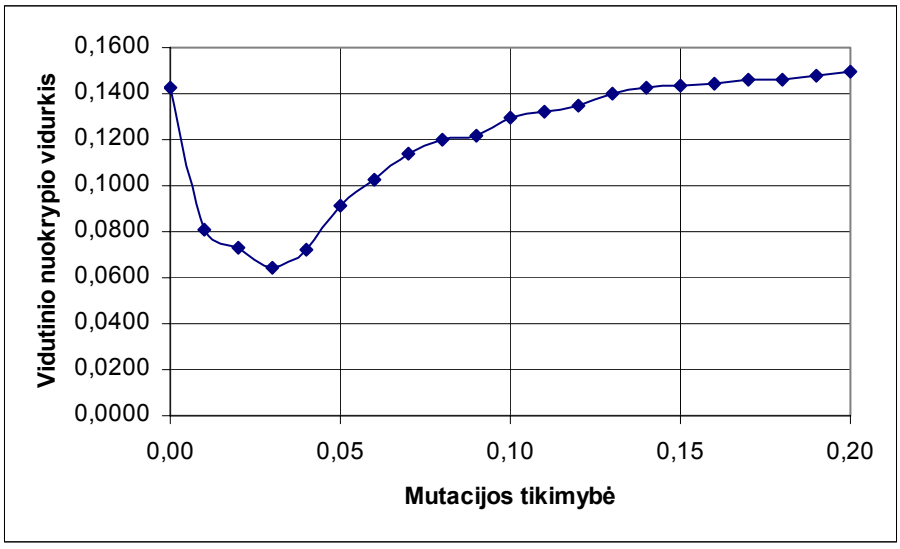


29 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės, kai* $n = 100$

8 lentelėje pavaizduota, kaip mutacijos tikimybė įtakoja genetinio algoritmo rezultatų gerumą naudojant testinius duomenis, kai $n = 200$. 30 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės. Iš šių duomenų galime daryti išvadą, kad kai $n = 200$ optimali mutacijos tikimybė yra 0,03. *GMJ* paketas patvirtino gautą optimalią parametro reikšmę.

8 lentelė *Genetinio algoritmo rezultatų priklausomybė nuo mutacijos, kai* $n = 200$

Populiacijos dydis	Mutacijos tikimybė	Iteracijų skaičius	p200_1	p200_2	p200_3	p200_4	p200_5	Vidutinio nuokrypio vidurkis
100	0,00	1.000	0,1374	0,1497	0,1367	0,1459	0,1452	0,1430
100	0,01	1.000	0,0819	0,0792	0,0787	0,0849	0,0809	0,0811
100	0,02	1.000	0,0801	0,0684	0,0698	0,0742	0,0720	0,0729
100	0,03	1.000	0,0485	0,0702	0,0751	0,0706	0,0554	0,0640
100	0,04	1.000	0,0710	0,0774	0,0734	0,0724	0,0665	0,0721
100	0,05	1.000	0,0783	0,0880	0,0960	0,0971	0,0966	0,0912
100	0,06	1.000	0,1011	0,0983	0,1095	0,1023	0,1034	0,1029
100	0,07	1.000	0,1130	0,1101	0,1111	0,1190	0,1150	0,1136
100	0,08	1.000	0,1163	0,1134	0,1241	0,1271	0,1183	0,1198
100	0,09	1.000	0,1147	0,1199	0,1241	0,1271	0,1215	0,1215
100	0,10	1.000	0,1294	0,1296	0,1289	0,1303	0,1296	0,1295
100	0,11	1.000	0,1326	0,1296	0,1336	0,1335	0,1311	0,1321
100	0,12	1.000	0,1358	0,1374	0,1336	0,1350	0,1327	0,1349
100	0,13	1.000	0,1405	0,1390	0,1413	0,1413	0,1374	0,1399
100	0,14	1.000	0,1436	0,1390	0,1459	0,1444	0,1405	0,1427
100	0,15	1.000	0,1452	0,1436	0,1459	0,1429	0,1405	0,1436
100	0,16	1.000	0,1467	0,1421	0,1459	0,1429	0,1436	0,1442
100	0,17	1.000	0,1467	0,1482	0,1429	0,1490	0,1421	0,1458
100	0,18	1.000	0,1467	0,1452	0,1459	0,1505	0,1421	0,1461
100	0,19	1.000	0,1498	0,1482	0,1474	0,1490	0,1452	0,1479
100	0,20	1.000	0,1514	0,1512	0,1474	0,1505	0,1482	0,1498

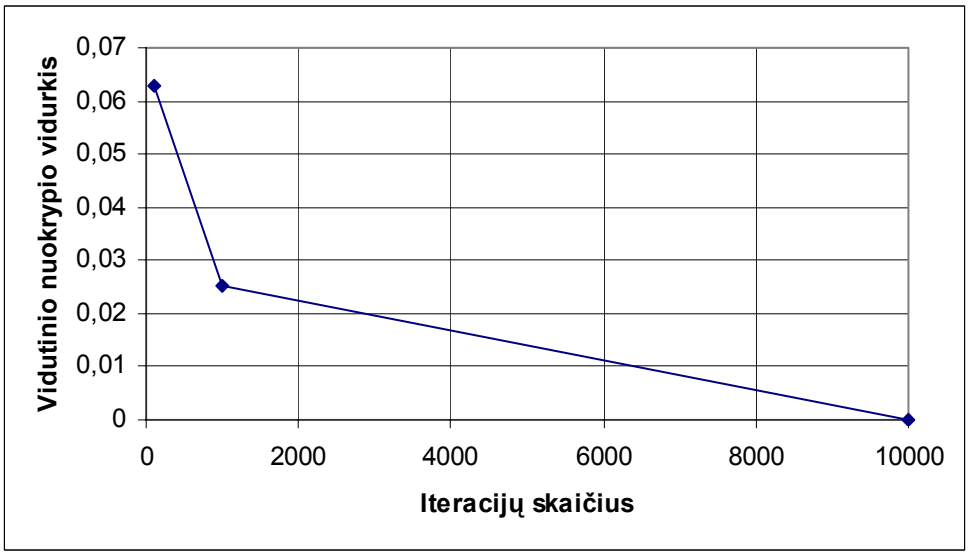


30 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo mutacijos tikimybės, kai* $n = 200$

Apibendrinus tyrimo rezultatus matome, kad mutacijos tikimybė turi būti pakankamai maža, norint gauti gerus rezultatus. Taip yra todėl, kad esant per dideliai mutacijos tikimybei, genetinio algoritmo darbo rezultatai prastėja dėl per didelės randomizacijos. Esant per mažai mutacijos tikimybei, genetinio algoritmo darbo rezultatai prastėja dėl to, kad jis dažniau užstringa lokaliame optimume.

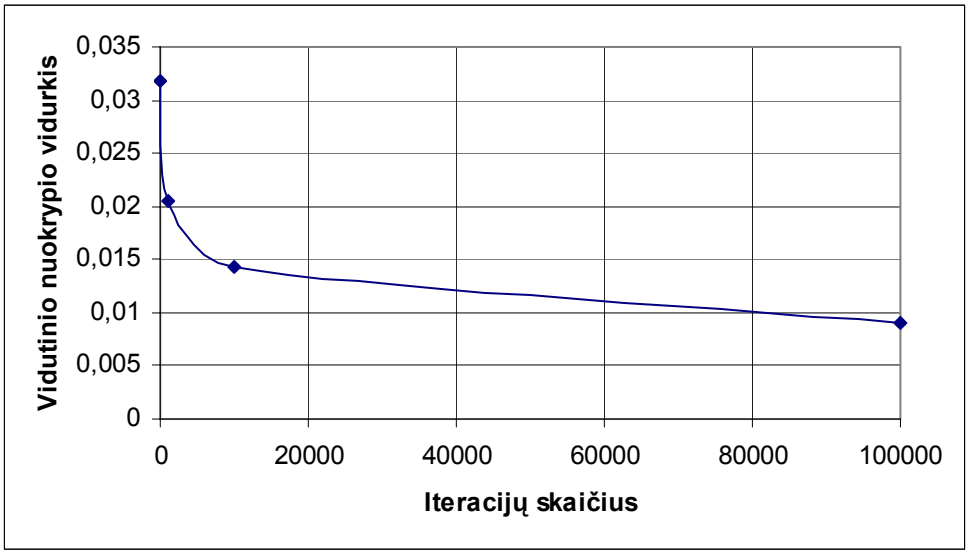
7.2.2 Algoritmo tyrimas

Toliau genetinio algoritmo tyrime naudosime anksčiau nustatytas optimalias parametrų reikšmes. Kaip minėjome, euristinių, o tuo pačiu ir genetinio, algoritmų gaunamo sprendinio gerumas tiesiogiai priklauso nuo parinktų parametrų optimalumo ir atliktų iteracijų skaičiaus. 31 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 25$. Matome, kad rezultatai žymiai gerėja didinant iteracijų skaičių, o kai iteracijų skaičius yra 10000 tabu paieška visą laiką gražina optimalias reikšmes.



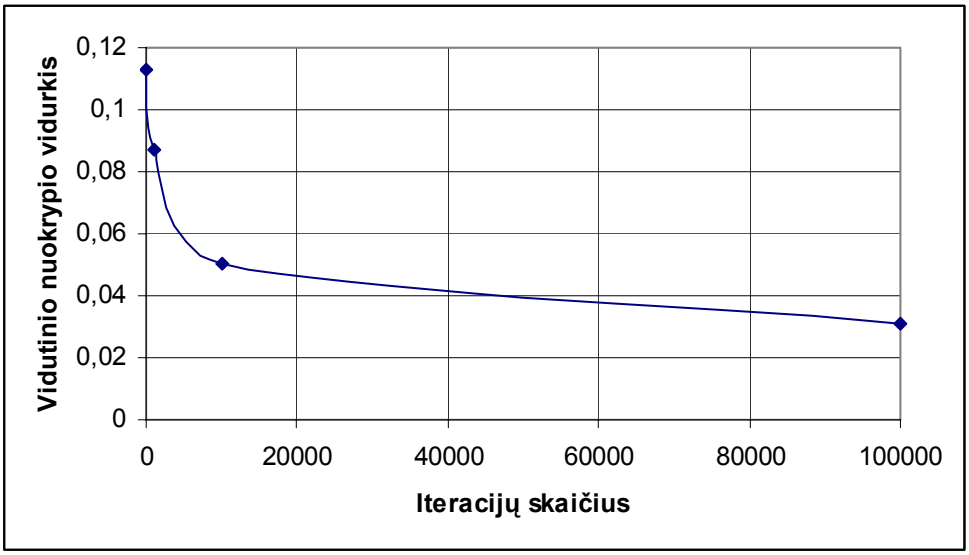
31 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 25$

32 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 100$. Čia taip pat matosi, kad rezultatai žymiai gerėja didinant iteracijų skaičių.



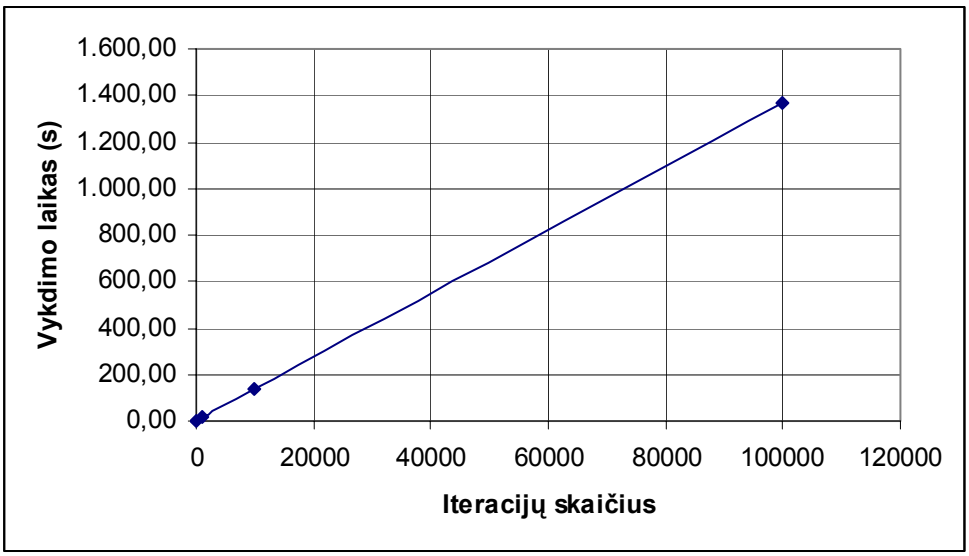
32 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 100$

33 pav. pavaizduotame grafike matome normuoto vidutinio nuokrypio vidurkio priklausomybe nuo iteracijų skaičiaus, kai $n = 200$. Čia taip pat matosi, kad rezultatai žymiai gerėja didinant iteracijų skaičių.

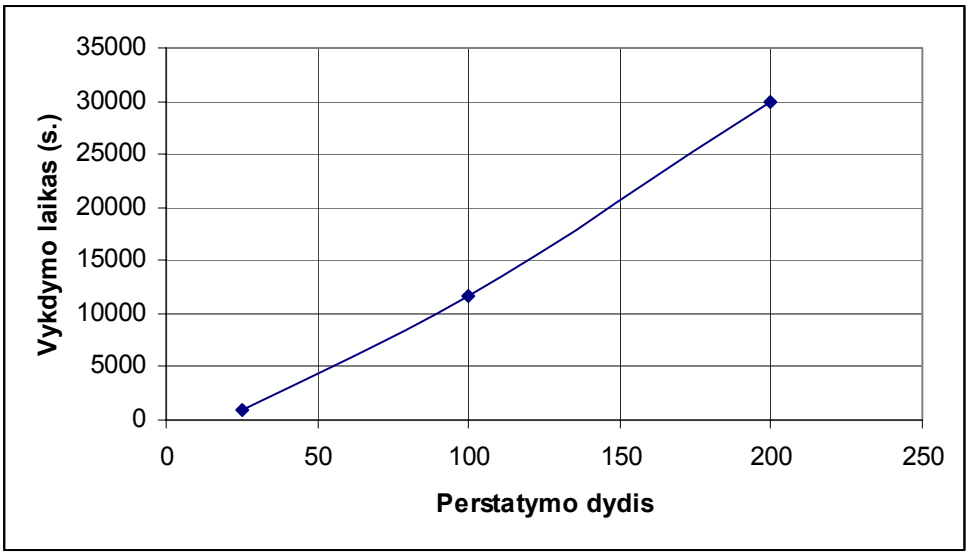


33 pav. *Vidutinio nuokrypio vidurkio priklausomybę nuo iteracijų skaičiaus, kai* $n = 200$

Toliau ištirkime algoritmo laikines charakteristikas. 34 pav. pavaizduota algoritmo vykdymo laiko priklausomybė nuo iteracijų skaičiaus. Galima daryti išvada, kad algoritmo vykdymo laikas tiesiogiai priklauso nuo iteracijų skaičiaus. 35 pav. pavaizduota algoritmo vykdymo laiko priklausomybė nuo perstatymo dydžio *n*.



34 pav. *Algoritmo vykdymo laiko priklausomybė nuo iteracijų skaičiaus*



35 pav. *Algoritmo vykdymo laiko priklausomybė nuo perstatymo dydžio*

Taigi apibendrinant genetinio algoritmo taikymo *SBR* uždaviniui spręsti tyrimą, galime pasakyti, kad algoritmas neblogai susidorojo su problema. Prie didesnių *n* reikšmių optimumas buvo pakankamai dažnai pasiekiamas, o taip pat matomos tendencijos, kad didinant iteracijų skaičių sprendinys gerėja. 9 lentelėje pateiktos reikšmės rastos genetinio algoritmo testiniams duomenims. Lentelėje paryškintos reikšmės atitinka optimumus.

9 lentelė *Genetinio algoritmo rasti minimalūs inversiniai atstumai*

Perstatymas	Dydis	Minimalus inversinis atstumas	Geriausias ir blogiausias rastas inversinis atstumas							
			100 iteracijų		1000 iteracijų		10000 iteracijų		100000 iteracijų	
			Ger.	Blog.	Ger.	Blog.	Ger.	Blog.	Ger.	Blog.
p25_1	25	**14**	**14**	16	**14**	15	**14**	**14**	-	-

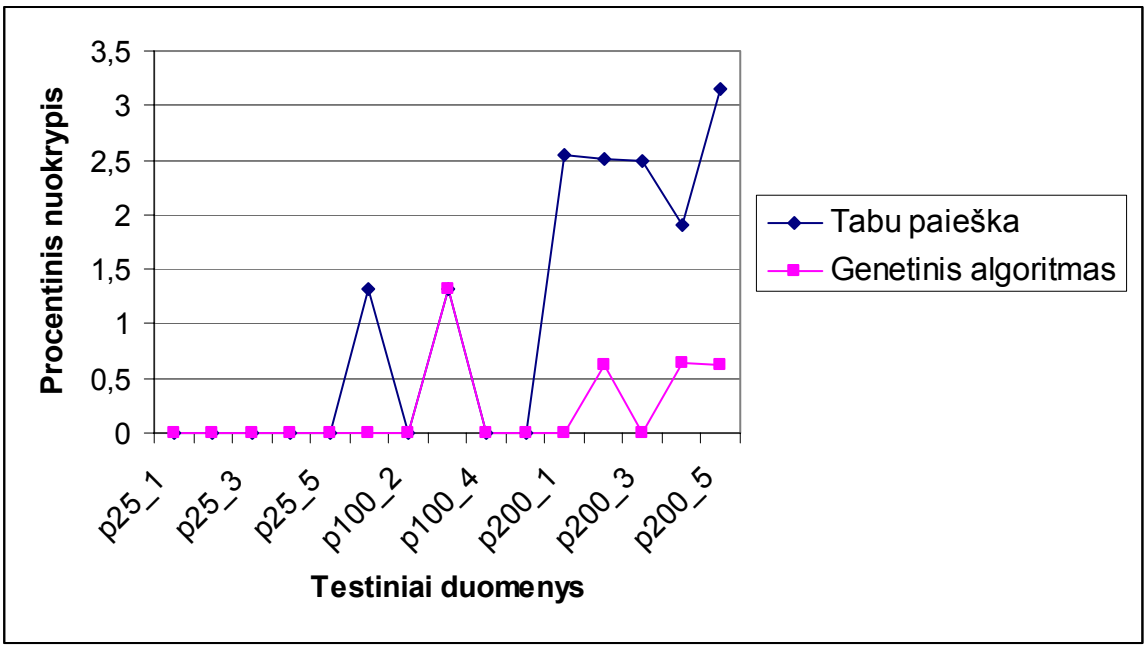
p25_2	25	**17**	**17**	**17**	**17**	**17**	**17**	**17**	-	-
p25_3	25	**16**	17	17	**16**	17	**16**	**16**	-	-
p25_4	25	**17**	**17**	18	**17**	**17**	**17**	**17**	-	-
p25_5	25	**17**	**17**	18	**17**	**17**	**17**	**17**	-	-
p100_1	100	**76**	77	80	77	79	**76**	78	**76**	77
p100_2	100	**78**	80	83	79	82	**78**	80	**78**	79
p100_3	100	**76**	77	81	77	80	77	78	77	77
p100_4	100	**78**	80	82	79	81	**78**	80	**78**	79
p100_5	100	**74**	79	82	77	78	**74**	75	**74**	75
p200_1	200	**157**	167	172	163	165	160	163	**157**	160
p200_2	200	**159**	175	177	167	172	163	164	160	162
p200_3	200	**160**	174	178	174	176	164	167	**160**	163
p200_4	200	**158**	174	176	164	169	161	164	159	161
p200_5	200	**159**	174	178	170	171	165	167	160	163

7.3 Eksperimento apibendrinimas

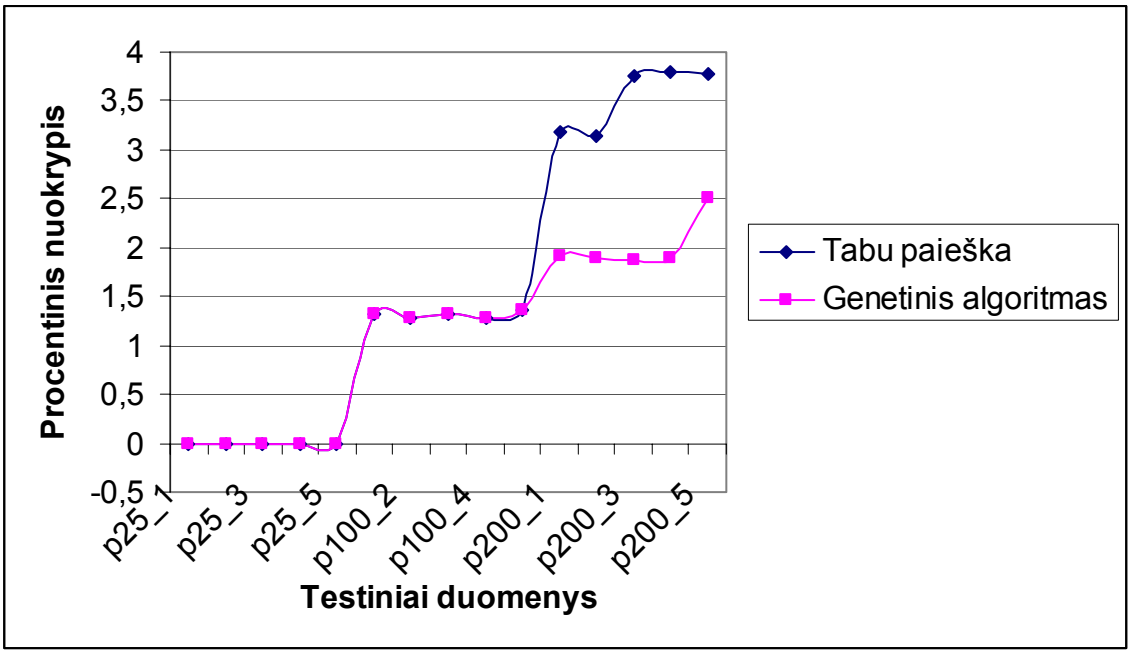
Buvo ištirtas dviejų euristinių algoritmų taikymas *SBR* uždaviniui spręsti. 36 pav. esančiame grafike visiems testiniams duomenims pavaizduotas geriausio rasto sprendinio procentinis nuokrypis nuo optimumo. 37 pav. esančiame grafike visiems testiniams duomenims pavaizduotas blogiausio rasto sprendinio procentinis nuokrypis nuo optimumo. Kai turime sprendinį s ir žinome optimalų sprendinį $s*$, procentinis nuokrypis skaičiuojamas pagal formule:

$$\sigma' = \frac{d(s)*100}{d(s*)} - 100$$

Matome, kad genetinis algoritmas rado geresnes reikšmes, tačiau šitas faktas neturi didelės įtakos mūsų darbui, kadangi pagrindinis tikslas buvo parodyti, kad euristiniai algoritmai tinka *SBR* uždaviniui spręsti. Tiek tabu paieška, tiek genetinis algoritmas rado pakankamai gerus sprendinius, kas leidžia daryti išvada, jog mūsų universali *SBR* sprendimo schema pasiteisina. Taip pat verta pastebėti, kad prasčiausio rasto sprendinio procentinis nuokrypis nuo optimumo yra 3.8%, o geriausias žinomas SBR aproksimuojantis algoritmas užtikrina sprendinį nuo optimumo nukrypusi ne daugiau nei 37.5%. Iš eksperimento galime daryti išvada, kad mūsų realizuoti euristiniai algoritmai randa geresnius sprendinius nei geriausias žinomas aproksimuojantis algoritmas.



36 pav. *Geriausio rasto sprendinio procentinis nuokrypis nuo optimumo*



37 pav. *Blogiausio rasto sprendinio procentinis nuokrypis nuo optimumo*

8 IŠVADOS

- Išnagrinėtas genomų palyginimo pagal inversinį atstumą (rikiavimo inversijomis) uždavinys tiek ženklintiems, tiek neženklintiems perstatymams. Nustatyta, kad ženklintų perstatymų rikiavimo inversijomis uždavinys yra kvadratinio sudėtingumo, o neženklintų perstatymų rikiavimo inversijomis uždavinys yra NP-sunkus.
- Darbe pateikta metodologija, kaip euristinių algoritmų pagalba spręsti neženklintų perstatymų rikiavimo inversijomis uždavinį.
- Darbe ištirtas tabu paieškos ir genetinio algoritmo taikymas neženklintų perstatymų rikiavimo inversijomis uždaviniui spręsti. Rastos optimalios šių algoritmų parametrų reikšmės.
- Eksperimentiškai nustatyta, kad euristinių algoritmų pagalba gaunami sprendiniai yra geresni už geriausio žinomo neženklintų perstatymų rikiavimą inversijomis aproksimuojančio algoritmo gaunamus sprendinius. Blogiausio euristinių algoritmų rasto sprendinio procentinis nuokrypis nuo optimumo yra 3.8%, o blogiausias tikėtinas 1.375 faktoriumi aproksimuojančio algoritmo randamo sprendinio procentinis nuokrypis nuo optimumo yra 37.5%.

9 LITERATŪRA

[1] DOBZHANSKY, T., STURTEVANT, A.H. *Inversions in the chromosomes of drosophila pseudoobscura.* Genetics, Volume 23, 1938, p28–64.

[2] PALMER, J.D., HERBON, L.A. *Plant mitochondrial DNA evolves rapidly in structure, but slowly in sequence*. Journal of Molecular evolution, Volume 28, 1988, p87–97.

[3] HANNENHALLI, S., PEVZNER, P. *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals)*. In Proceedings of the 27th Annual Symposium on Theory of Computing. Las Vegas, NV. ACM Press, 1995, p78-189.

[4] BADER, D.A., MORET, B.M., YAN, M. *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study*. Journal of Computational Biology, 8(5), 2001, p483-491.

[5] KAPLAN, H., SHAMIR, J., TARJA, R.E. Faster and simpler algorithm for sorting signed permutations by reversals [interaktyvus]. 1998 [žiūrėta 2004 m. balandžio 18 d.]. Prieiga per internetą: <http://www.math.tau.ac.il/~shamir/papers/reversals.ps>

[6] BAFNA, V., PEVZNER, P. *Genome rearrangements and sorting by Reversals*. SIAM Journal of Computing, Volume 25, 1996, p272–289.

[7] HANNENHALLI, S., PEVZNER, P. *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations with reversals)*. Journal of the ACM, Volume 46, 1999, p1–27.

[8] CAPRARA, A. *Sorting permutations by reversals and Eulerian cycle decompositions*. SIAM Journal on Discrete Mathematics, Volume 12, Number 1, 1999, p91-110.

[9] CAPRARA, A., LANCIA, G. *Sorting permutations by reversals through branch and price*. INFORMS Journal on Computing, Volume 13, Number 3, 2001, p224-244.

[10] CHRISTIE, D. *A 3/2-approximation algorithm for sorting by reversals*. Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California, 25-27 January 1998, p244-252.

[11] BERMAN, P., HANNENHALLI, S., KARPINSKI, M. *1.375-approximation algorithm sorting by reversals*. In Proceedings of Annual European Symposium on Algorithms (ESA), Volume 2461 of Lecture Notes in Computer Science, 2002, Springer, p200-210.

[12] VOß, S. *Meta-heuristics: The State of the Art*. Lecture Notes in Computer Science, 2001, Springer.

[13] GLOVER, F. *Future Paths for Integer Programming and Links to Artificial Intelligence.* Computers and Operations Research, Volume 13, 1986, p533-549.

[14] HERTZ, A., TAILLARD, E., WERRA, D. *A tutorial on Tabu Search* [interaktyvus]. [žiūrėta 2003 m. gruodžio 16 d.]. Prieiga per internetą: <http://www.cs.colostate.edu/~whitley/CS640/hertz92tutorial.pdf>

[15] BAECK, T., FOGEL, D.B. MICHALEWICZ, Z., BACK, T. *Evolutionary Computation 1: Basic Algorithms and Operators*. IOP Publishing Ltd, 2000.

[16] MOCKUS, J., EDDY, W., MOCKUS, A., MOCKUS, L., REKLAITIS, G. *Bayesian heuristic approach to discrete and global optimization* [interaktyvus]. 2000, [žiūrėta 2005 m. vasario 11 d.]. Prieiga per internetą: < http://soften.ktu.lt/~mockus/docj/book.pdf >

[17] MOCKUS, J. *A Set of Examples of Global and Discrete Optimization* [interaktyvus]. [žiūrėta 2005-04-17]. Prieiga per internetą: <http://soften.ktu.lt/~mockus>.