VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Mokslo tiriamojo darbo projektas

OPTIMALIŲ MARŠRUTŲ PAIEŠKOS ALGORITMAI

Atliko: magistro 2 kurso, 10 grupės studentas
Karolis Šarapnickis

Darbo vadovas:
doc. Tadas Meškauskas

Vilnius
2015

Turinys

1. Įvadas

Optimalių maršrutų paieškos aktualumas kiekvieno iš mūsų kasdieniniame gyvenime buvo aptartas pirmame moksliniame tiriamajame darbe. Jame buvo susipažinta su keletų heuristinių optimalaus maršruto paieškos algoritmų ir spręstos keliaujančio pirklio problemos pilno grafo atveju [Šar14].

Yra nemažai mokslinių darbų sprendžiančių keliaujančio pirklio problemas naudojant heuristinius algoritmus, juos modifikuojant ir parenkant įvairiausias sąlygas. Tačiau didžioji dali tokių darbų analizuoja pilnus grafus (kiekvienas miestas turi tiesioginį kelią su bet kuriuo kitu miestu), o realiame gyvenime praktiškai nėra didesnių sausumos žemėlapių, kuriuose kiekvienas miestas būtų tiesiogiai susietais keliais su likusiais miestais.

Šio mokslinio tiriamojo darbo projekto tikslas yra naudojant heuristinius algoritmus ir jų modifikacijas ieškoti trumpiausio maršruto tarp miestų aplankant juos bent vieną kartą. Visi analizei naudojami grafai yra panašūs į sausumos miestų žemėlapius – juose įmanoma aplankyti visus miestus, o grafo kraštinės niekur nesusikerta.

2. Optimalaus maršruto, aplankančio visus paskirties taškus algoritmai

2.1. Skruzdėlių kolonijos sistemos algoritmas

Skruzdėlių kolonijos algoritmas (ACS) (angl. *ant colony system*) yra heuristinis, gamtos įkvėptas algoritmas. Algoritmas yra paremtas skruzdėlių elgesiu gamtoje joms keliaujant miško takais.

Skruzdėlių kolonijos algoritmo veikimo principas [YuHuZha09]:

1. Inicializacija. Nustatomi ACS parametrai, o skruzdėlės yra pastatomos ant atsitiktinių miestų.

2. Kelio sudarymas. Kiekviena skruzdėlė sudaro nuosavą kelią nuolat pritaikydama pseudo-atistiktinę proporcingumo taisyklę. Kai skruzdė *k* yra mieste *r*, ji pasirenka miestą *s* vadovaudamasi taisykle:

$$s=\begin{cases} arg\,max_{u\in J_k(r)}\{[\tau(r,u)]\cdot[\eta(r,u)]^{\beta}\}, & \text{jei} \quad q\le q_0 \\ S, & \text{kitu atveju} \end{cases} \tag{1}$$

kur $\tau(r,u)$ yra feromono lygis tarp miesto *r* ir *u*, $\eta(r,u)$ yra atstumo tarp miesto *r* ir *u* inversija, $J_k(r)$ yra rinkinys skruzdėlės *k* dar neaplankytų miestų, $\beta(\beta>0)$ yra heuristinis faktorius, kuris nulemia reliatyvę svarbą feromono ir atstumo, *q* yra atsitiktinis skaičius rėžiuose $[0,1]$, $q_0(0\le q_0\le 1)$ yra parametras, *S* yra atsitiktinis kintamasis parinktas pagal tikimybės pasiskirstymą:

$$p_k(r,s)=\begin{cases} \dfrac{[\tau(r,s)]\cdot[\eta(r,s)]^{\beta}}{\sum_{u\in J_k(r)}[\tau(r,u)]\cdot[\eta(r,u)]^{\beta}}, & \text{jei} \quad s\in J_k(r) \\ 0, & \text{kitu atveju} \end{cases} \tag{2}$$

Ši procedūra kartojama tol, kol kiekviena skruzdėlė aplanko visus miestus ir tada grįžta į pradinį miestą.

3. Lokalus feromono atnaujinimas. Kelio sudarymo metu skruzdėlės naudoja lokalaus feromono atnaujinimo taisyklę, kad pakeistų feromono lygį ant kraštinės (r,s), kurią

praėjo kaip:

$$\tau(r,s) \leftarrow (1-\rho)\cdot\tau(r,s) + \rho\cdot\tau_0 \tag{3}$$

kur $\rho(0<\rho<1)$ yra lokalaus feromono nykimo koeficientas, τ_0 yra pradinis feromono lygis.

4. Globalus feromono atnaujinimas. Globalus feromono atnaujinimas įvyksta kai visos skruzdėlės baigia savo maršrutus. Tik kraštai (r,s) priklausantys geriausiam maršrutui yra atnaujinti pagal:

$$\tau(r,s) \leftarrow (1-\alpha)\cdot\tau(r,s) + \alpha\cdot\Delta\tau_k(r,s) \tag{4}$$

kur $\alpha(0<\alpha<1)$ yra globalaus feromono nykimo parametras. $\Delta\tau_k(r,s)$ vertė yra nustatoma pagal

$$\Delta\tau_k(r,s) = \begin{cases} \frac{1}{L_{ib}}, & \text{jei } (r,s) \text{ priklauso geriausiam maršrutui} \\ 0, & \text{kitu atveju} \end{cases} \tag{5}$$

kur L_{ib} yra trumpiausio maršruto maršruto ilgis.

Algoritmas kartojamas nuo 2 žingsnio tol kol yra pasiekiama algoritmo sustabdymo sąlyga [YuHuZha09].

2.2. Simuliuoto atkaitinimo algoritmas

Simuliuoto atkaitinimo (SA) (angl. *simulated annealing*) algoritmas yra heuristinis, gamtos įkvėptas algoritmas. Jis yra paremtas metalo kietinimo procesu, kai jį lėtai vėsinant yra randami absoliutūs minimumai.

1. Sugeneruojamas pradinis maršrutas π ir apskaičiuojamas jo ilgis $f(\pi)$;
2. Nustatoma pradinė temperatūra $T:=T_0$;
3. Nustatomas iteracijos indeksas $t:=1$;
4. Kol nesustabdomas ciklas, vykdoma:
    1. Naudojant reprodukcijos operatorių yra sugeneruojamas naujas maršrutas π' ir apskaičiuojamas jo ilgis $f(\pi')$
    2. Nustatoma $\Delta f:=f(\pi')-f(\pi)$
    3. Jei $\Delta f<0$, tada π kelias pakeičiamas π' .
    4. Jei $e^{-\Delta f/T}<rand()$, tada π kelias pakeičiamas π' .
    5. Kitu atveju išmetamas π' kelias.
    6. Jei $t \bmod T_u=0$, tada atnaujinama temperatūra: $T:=\alpha T$
    7. Atnaujinamas iteracijos skaitiklis $t:=t+1$

1. pvz. Standartinė simuliuoto atkaitinimo procedūra [LiZhoGui11].

Standartinė simuliuoto atkaitinimo algoritmo struktūra pavaizduota 1 pav. Kiekvienos SA algoritmo iteracijos metu turimas kelias $\pi=(\pi_1\pi_2\cdots\pi_n)$. Algoritmo paaiškinimas [LiZhoGui11]:

- Inicializacija. Jos metu atsitiktinai sugeneruota maršrutas taip: visiems $i=1,2,\cdots,n$ yra rinkinys $\pi_i=i$; tada kiekvienam $i=1,2,\cdots,n-1$ atsitiktinai parenkamas $\pi_j\in\{\pi_i,\pi_{i+1},\cdots,\pi_n\}$ ir sukeičiamas π_i ir π_j .

- Ciklo sustabdymas. Kelio paieškos ciklas gali būti sustabdytas išsipildžius vienai iš dviejų sąlygų: pasiekus tam tikrą kartų skaičių, *maxt* , arba nerastas trumpesnis kelias tam tikrą skaičių kartų, *maxut* .

- Pasirinkimo procedūra. SA algoritmo pasirinkimo taisyklė yra pritaikoma 2 pav. 4.3 – 4.5 žingsniuose. Yra 3 situacijos kaip pasielgiama su naujai gautu maršrutu: 1) naujas kelias yra

trumpesnis ir yra pasirenkamas; 2) naujas kelias yra prastesnis, bet yra pasirenkamas remiantis tikimybe kur $rand()$ funkcija gražina atsitiktinį skaičių rėžiuose $[0,1]$; 3) naujas kelias yra prastesnis, todėl yra išmetamas.

- Reprodukcijos operatorius (2 pav.). Standartinio SA algoritmo reprodukcijos operatoriai būna du: 1) parinktas sub-maršrutas yra paimamas priešinga puse; 2) parinktas sub-maršrutas yra įstatomas į kitą maršruto poziciją.

1. Atsitiktinai iš maršruto π yra parenkamas sub-maršrutas ir pažymimas

$$(\pi_{i_1} \cdots \pi_{i_k})$$

kur π_{i_1} yra sujungtas su π_{i_1-1} , π_{i_k} yra sujungtas su π_{i_k+1} ir $1<k<n-1$

2. Jei $rand()<0.5$

   Apsukamas sub-maršrutas ir gaunamas naujas maršrutas π' pavidalu:

$$(\pi_1 \cdots \underline{\pi_{i_1-1}\, \pi_{i_k}} \cdots \underline{\pi_{i_1}\, \pi_{i_k+1}} \cdots \pi_n)$$

3. Jei $rand()\geq 0.5$

   Atsitiktinai parenkamas kraštas $\pi_j \pi_{j+1}$ iš galimų, ne sub-maršrute esančių, kraštų. Įterpiamas sub-maršrutas tarp π_j ir π_{j+1} ir gaunamas naujas maršrutas π' pavidalu:

$$(\pi_1 \cdots \underline{\pi_{i_1-1}\, \pi_{i_k+1}} \cdots \underline{\pi_j \pi_{i_1}} \cdots \underline{\pi_{i_k}\, \pi_{i+1}} \cdots \pi_n)$$

2. pvz. Standartinė simuliuoto atkaitinimo reprodukcijos procedūra [LiZhoGui11].

2.3. Genetiniai algoritmai

Genetiniai algoritmai (GA) (angl. *genetic algorithm*) – vieni iš heuristinių algoritmų, įkvėptų gamtos. Jie yra paremti evoliuciniu gamtos modeliu, kai keičiantis kartoms individai tampa vis tobulesni ir labiau prisitaikę prie aplinkos sąlygų.

Dėl keliaujančio pirklio problemos paprastos formuluotės yra bandoma įvairiausius algoritmus panaudoti sprendžiant šią problemą. Ne išimtis yra ir šie algoritmai, tačiau „grynieji“ genetiniai algoritmai, kurti J. H. Holland'o ir jo studentų Mičigano universitete 60-taisiais, 70-taisiais, nebuvo sugalvoti spręsti kombinatorinio optimizavimo problemas. Tai laikais genetiniai algoritmai

dažniausiai buvo naudojami sprendžiant įvairias skaitines funkcija, todėl norint rasti keliaujančio pirklio problemos sprendimą, reikia atlikti tam tikrus genetinio algoritmo pakeitimus [Pot96].

Genetinis algoritmas iš esmės operuoja su baigtine chromosomų arba bitų sekų populiacija. Paieškos mechanizmas susideda iš trijų skirtingų fazių: kiekvienos chromosomos tinkamumo (angl. *fitness*) įvertinimas, tėvinių chromosomų parinkimas ir mutacijos bei rekombinacijos operatorių pritaikymas tėvinėms chromosomoms. Naujos chromosomos, gautos pritaikius genetinius operatorius, dalyvauja tolimesnėje revoliucijos iteracijoje ir pati sistema tobulėja augant kartų skaičiui [Pot96]. Sistemos supaprastintas pseudo-kodas pavaizduotas 3 pavyzdyje.

5. Sukuriama pradinė P chromosomų populiacija (0 karta).
6. Įvertinamas kiekvienos chromosomos tinkamumas.
7. Pasirenkama P tėvų iš esamos populiacijos pasitelkiant proporcingumo taisykle (angl. *proportional selection*) (pasirinkimo tikimybė priklauso nuo tinkamumo vertės).
8. Dauginimuisi atsitiktinai pasirenkama pora chromosomų. Atliekama rekombinacijos operacija apkeičiant bitus pasirinktame taške, taip sukuriant vaikines chromosomas.
9. Kiekviena vaikinė chromosoma apdorojama mutacijos operatoriais ir grąžinama į populiaciją.
10. Kartojami 4 ir 5 žingsniai kol visos chromosomos būna parinktos ir paveiktos genetiniais operatoriais.
11. Sena chromosomų populiacija pakeičiama nauja.
12. Įvertinamas kiekvienos chromosomos tinkamumas.
13. Kartojama viskas nuo 3 žingsnio kol pasiekiamas atitinkamas kartų skaičius ar kitokia sąlyga.

3. pvz. Genetinio algoritmo pseudo-kodas [Pot96].

2.3.1. Parinkimo tikimybė

Genetinio algoritmo vykdymo metu yra parenkamos tėvinės chromosomos. Problema yra kokias chromosomas yra geriausia parinkti, kad iš jų gautos vaikinės chromosomos būtų geresnės. Yra nemažai metodų kaip yra parenkamos chromosomos atsižvelgiant į jų tinkamumo koeficientus ir kiekvienas metodas pasižymi tam tikrais privalumais ir trūkumais.

Vienas iš metodų parinkti tėvines chromosomas yra ruletės metodas. Ruletės parinkimo metodo principas:

1. Susumuojami visų populiacijos chromosomų tinkamumo koeficientai.
2. Sugeneruojamas atsitiktinis skaičius tarp 0 ir tinkamumo koeficientų sumos.
3. Atsitiktine tvarka sumuojami chromosomų tinkamumo koeficientai, kol suma yra lygi arba viršija 2 punkte gautą skaičių. Paskutinė sumuota chromosoma ir yra gražinama.

Šis metodas pasižymi vienos super-chromosomos trūkumu. Pavyzdžiui, esant tinkamumo koeficientams - [20, 10, 5, 847, 42, 1], dominuojanti chromosoma bus su koeficientu 847 ir praktiškai visada bus parinkta ruletės metodu. Todėl keičiantis kartoms populiacija praktiškai nesikeičia, kiekviena chromosoma būna panaši ir evoliucija toliau nebevyksta [Pot96].

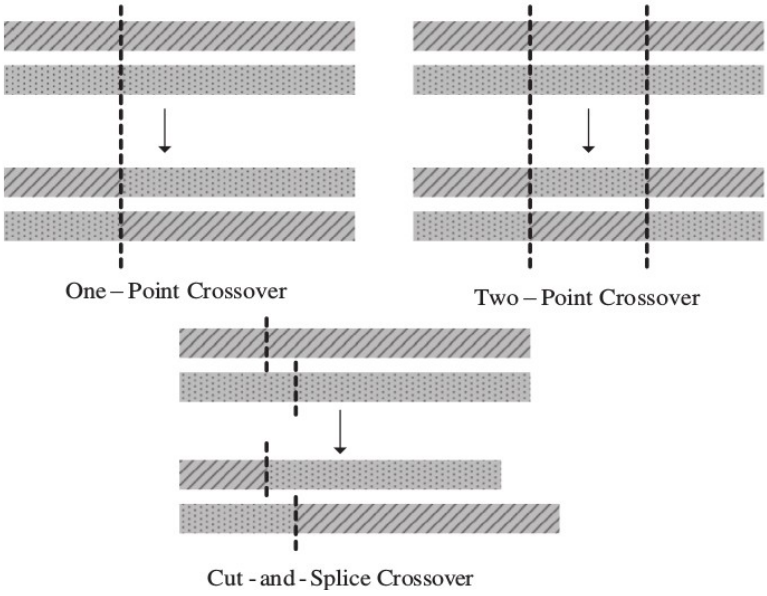
Galima alternatyva ruletės metodui yra rango parinkimo metodas. Šis metodas labai panašus į ruletės, tačiau vietoj tinkamumo koeficientų yra sumuojami tinkamumo koeficientų rangai. Tačiau šis metodas taip pat nėra idealus, jis pasižymi lėta konvergencija, nes geriausios chromosomos ne taip stipriai skiriasi nuo prastesnių chromosomų [Pot96].

2.3.2. Rekombinacija

Rekombinacijos metu yra generuojamos naujos chromosomos tol, kol atnaujinama visa populiacija. Tam tikru parinkimo metodu iš chromosomų populiacijos yra parenkamos 2 tėvinės chromosomos. Atsitiktinai sugeneruojamas skaičius tarp 0 ir 1. Jei tas skaičius yra didesnis nei nurodytas rekombinacijos koeficientas *CR* (kur $CR \in (0,1]$), tada yra vykdoma rekombinacija. Jei rekombinacija yra įvykdoma, tada yra gaunamos 2 vaikinės chromosomos. Šis metodas yra vykdomas tol, kol yra atnaujinama visa populiacija [CheChi11].

Rekombinacijos metodų taip pat yra ne vienas (4 pav). Vienas iš metodų yra vieno taško (angl. *one point*) rekombinacija, kurio metu yra apkeičiamos bitų eilutės tarp tėvinių chromosomų. Yra parenkamas atsitiktinis skaičius nuo 1 iki $L-1$, kur L yra chromosomos ilgis. Tada chromosomos yra perskiriamos parinktame taške ir jų galai yra apkeičiami tarp tėvinių

chromosomų, taip sukuriant 2 vaikines chromosomas [Pot96].



4. pvz. Vieno taško, dviejų taškų ir supjaustymo ir sujungimo rekombinacijos metodų [CheChi11].

Rekombinacijos tikimybė siejama su indeksu *CR*, kuris nulemia paieškos „agresyvumą“. Jei nėra pritaikomas rekombinacijos metodas, tada tėvinės chromosomas keliauja į naują populiaciją, kitu atveju tai daroma su naujai gautomis chromosomomis. Didelis *CR* indeksas sukuria daugiau naujų chromosomų, tačiau padidina tikimybę prarasti geresnes tėvines chromosomas [Pot96].

Keliaujančio pirklio problemos sprendimo metu įprastiniai vieno taško ir panašūs rekombinacijos metodai netinkami taikant juos tiesiogiai galimiems maršrutams. Šie operatoriai yra skirti bitų manipuliacijai, o juos pritaikius miestų sekoms yra nemaža tikimybė, kad gauti vaikai nebus validžios permutacijos. Pavyzdžiui turint dvi chromosomas 12564387 ir 14236578, ir atlikus vieno taško rekombinacijos operatorių antroje pozicijoje yra gaunami vaikai: 12236578, bei 14564387. Tokios chromosomos nėra validžios, nes kartojasi arba trūksta tam tikrų miestų. Šią problemą galima spręsti pakeitus kelio reprezentacinę išraišką, arba pasirinkus kitokį rekombinacijos metodą [Pot96].

2.3.2.1. Kelintinė išraiška

Viena iš galimų išraiškų yra kelintinė (angl. *ordinal*) (5 pav.). Kelintinę išraišką paveikus rekombinacijos operatoriumi visada yra gaunamos validžios vaikinės chromosomos, kurios yra sistemos perturbacijos. Atlikus vieno taško ar panašaus tipo rekombinacijos operaciją ant chromosomų, išreikštų kelintine išraiška, yra gaunamos vaikinės chromosomos, kurias galima iš kelintinės išraiška konvertuoti atgal į skaitinę [Pot96].

Kelio išraiška	Likę miestai	Kelintinė išraiška
1 2 5 3 4	1 2 3 4 5	1
1 2 5 3 4	2 3 4 5	1 1
1 2 5 3 4	3 4 5	1 1 3
1 2 5 3 4	3 4	1 1 3 1
1 2 5 3 4	4	1 1 3 1 1

5. pvz. Kelintinė išraiška.

2.3.2.2. Eilės (OX) rekombinacija

Viena iš alternatyvų tradiciniams rekombinacijos metodas (vieno taško ir t.t.) yra eilės (angl. order *crossover*) rekombinacija (OX). OX rekombinacija paremta modifikuotos rekombinacijos principu (6 pav.), tik vietoj 1 taško yra imami 2 taškai. Modifikuotos rekombinacijos metu vaikinė chromosoma yra sukuriama prie pirmosios chromosomos dalies pridedant antrosios chromosomos galą, eliminuojant dublikatus [Pot96].

Chromosoma 1 : **1 2** | 5 6 4 3 8 7
Chromosoma 2 : 1 4 | **2 3 6 5 7 8**

Vaikas 1 : 1 2 4 3 6 5 7 8

6. pvz. Modifikuota rekombinacija.

OX rekombinacija sukuria vaikines chromosomas padarydama 2 pjūvio taškus (7 pav.). Padalinus abi chromosomas į 3 dalis, į pirmą vaikinę chromosomą yra nukeliama 2-oji pirmosios chromosomos dalis. Tada iš antrosios chromosomos yra paimama pradžia ir pabaiga ir sujungiama su vaikinės vidurine dalimi. Panaikinus dublikatus yra gaunama nauja chromosoma [Pot96].

Chromosoma 1 : **1 2** | **5 6 4** | **3 8 7**
Chromosoma 2 : 1 4 | 2 3 6 | 5 7 8

Vaikas 1
1 žingsnis : - - **5 6 4** - - -
2 žingsnis : 2 3 **5 6 4** 7 8 1

7. pvz. OX rekombinacija.

2.3.3. Mutacija

Rekombinacijos metu gautų vaikinių chromosomų bitai, su labai maža tikimybe yra paveikiami mutacijos operatoriumi. Ši tikimybė apibrėžiama naudojant koeficientą *MR*, kur $MR \in (0,1]$. Jei atsitiktinai sugeneruotas skaičius nuo 0 iki 1 yra didesnis, nei *MR*, tada yra taikoma mutacijos operacija [CheChi11].

Tam tikroje vietoje pritaikius šį operatorių bitas iš 0 paverčiamas į 1, arba iš 1 į 0. Mutacijos operatoriaus tikslas yra į procesą įtraukti atsitiktinių perturbacijų tikimybę. Yra naudinga į genetinio algoritmo veikimo procesą įtraukti įvairovės, kuri sukurti chromosomas, kurios nebūtų gražinamos į ankstesnes būsenas vien rekombinacijos metodais. Taip pat yra naudinga didinti mutacijos tikimybę augant kartų skaičiui, norint išlaikyti tinkamą įvairovės lygį [Pot96].

Sprendžiant keliaujančio pirklio problema genetinio algoritmo pagalba galima naudoti apkeitimo (angl. *swap*) mutacija. Apkeitimo mutacijos metu chromosomoje atsitiktinai parinkti du miestai yra apkeičiami vietomis [Pot96].

2.3.4. Elitizmas

Paprasto genetinio algoritmo veikimo metu, kiekviena nauja chromosomų karta pakeičia senąją. Tačiau yra ir kitų galimų būdų kaip užbaigti algoritmo kartą. Galima pakeisti tik dalį senosios kartos naujomis chromosomomis. Elitizmo metodas kiekvienos kartos generavimo pabaigoje išsaugo geriausią senosios kartos chromosomą [Pot96]. Taip panaikinama tikimybė, kad keičiantis kartoms gali būti sunaikinta labai tinkama chromosoma.

3. Praktinė dalis

Mokslinio tiriamojo darbo praktinėje dalyje buvo įgyvendinti 3 algoritmai:

- Genetinis algoritmas (GA)
- Simuliuoto atkaitinimo algoritmas (SA)
- Skruzdėlių kolonijos algoritmas (ACS)

Praktinės dalies tikslas yra detaliai išanalizuoti GA algoritmo ypatybes, keičiant parametrus parinkimo, rekombinacijos ir mutacijos metodus, ACS ir SA algoritmu panaudojimą pačiame GA algoritme.

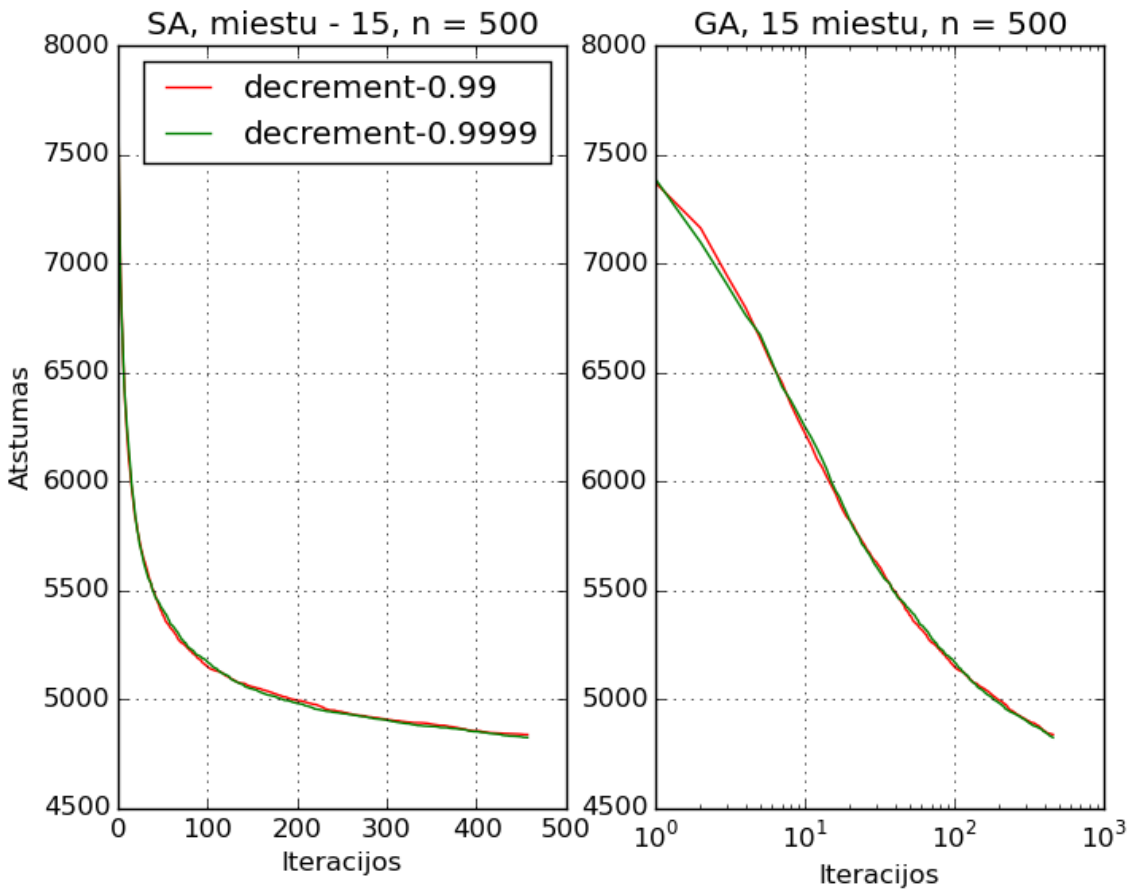
Heuristiniai algoritmai buvo sprendžiama modifikuota keliaujančio pirklio problema panašiuose į miestų žemėlapius idealiuose grafuose. Keliaujančio pirklio problemos taisyklė, kad kiekvienas miestas turi būti aplankytas tik vieną kartą buvo panaikinta. Buvo naudojami ne pilni grafai, kurių nei viena kraštinė nesusikerta su kita grafo kraštine.

Daugelis heuristinių algoritmų turėdami neribotą laiką galėtų rasti optimaliausią maršrutą. Tačiau skaičiavimo laikas yra ne ką mažiau svarbesnis matavimo vienetas už patį maršruto ilgį. Taip pat tokie algoritmai pasižymi įsisotinimu ties tam tikromis ribomis, ties kuriomis maršruto trumpėjimas labai sustoja. Todėl mokslo tiriamojo darbo metu buvo ieškoma ne tik optimalių parametrų su kuriais būtų gaunamas geriausias vidutinis rezultatas, bet ir su kokiais parametrais kokiose algoritmo gyvavimo laikotarpiuose maršrutas trumpėja greičiausiai.

Programinis kodas parašytas naudojant Python 2.7 programinę kalbą ir vykdytas Ubuntu operacinėje sistemoje.

3.1. ACS ir SA algoritmai

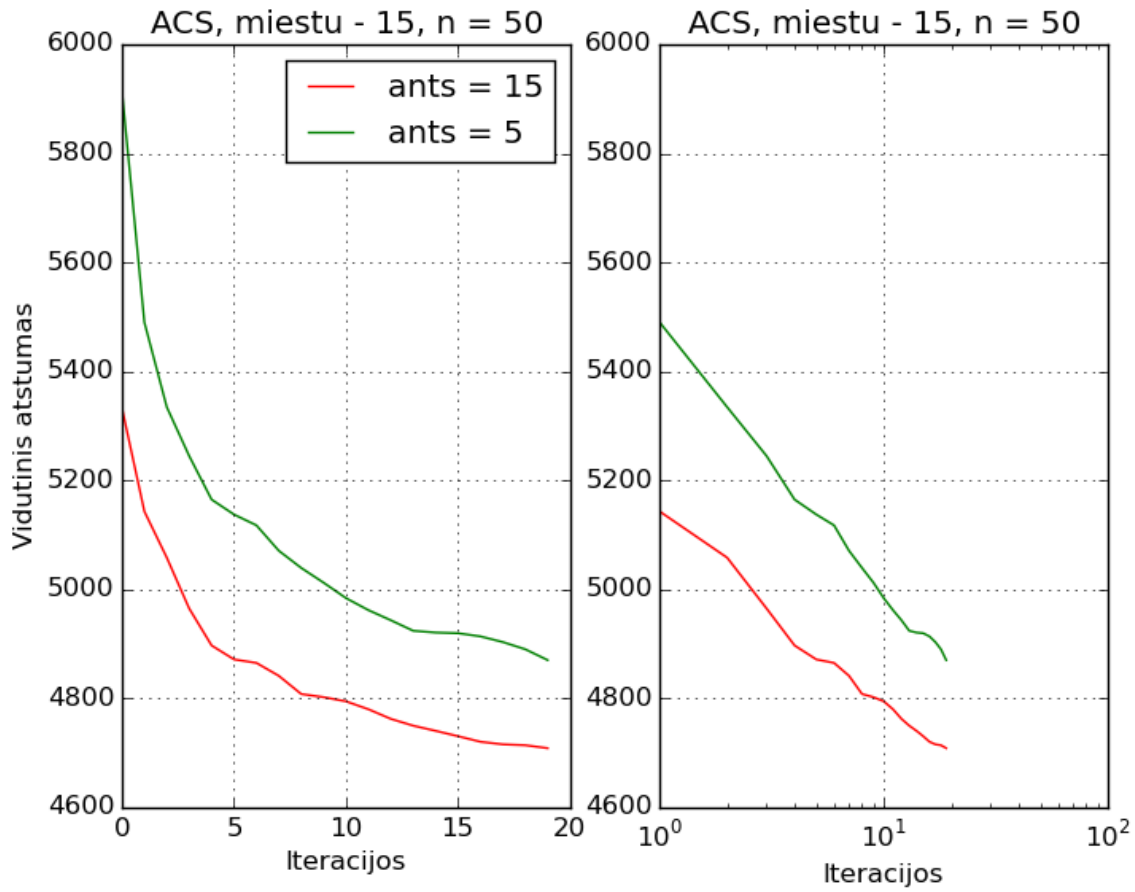
Skruzdėlių kolonijos ir simuliuoto atkaitinimo algoritmai nėra patys optimaliausi algoritmai ieškant trumpiausio maršruto, tačiau buvo analizuotos šių algoritmų savybės įvairiais algoritmo gyvavimo laikotarpiais, kad būtų galima padarytas išvadas panaudoti genetinio algoritmo modifikavimui. 8 Pav. pavaizduota SA algoritmo vidutiniai rezultatai analizuojant 15 miestų optimalaus maršruto problemą. Vidutinis rasto maršruto ilgis abiem temperatūros kritimo koeficientams yra panašus – apie 4830, o skaičiavimo laikas 0.112s.



8. pvz. 500 kartų vykdyto SA algoritmo vidutiniai rezultatai su temperatūros mažėjimo koeficientais 0.99 ir 0.9999.

Nors SA algoritmo pradinis sugeneruotas maršrutas yra labai ilgas, tačiau algoritmas tam tikrą iteracijų skaičių tą maršrutą trumpina gana greitai.

Taip pat buvo analizuotas ACS algoritmas (9 pav., 10 pav.), buvo keičiami β , α , ρ parametrai ir skruzdėlių skaičius.



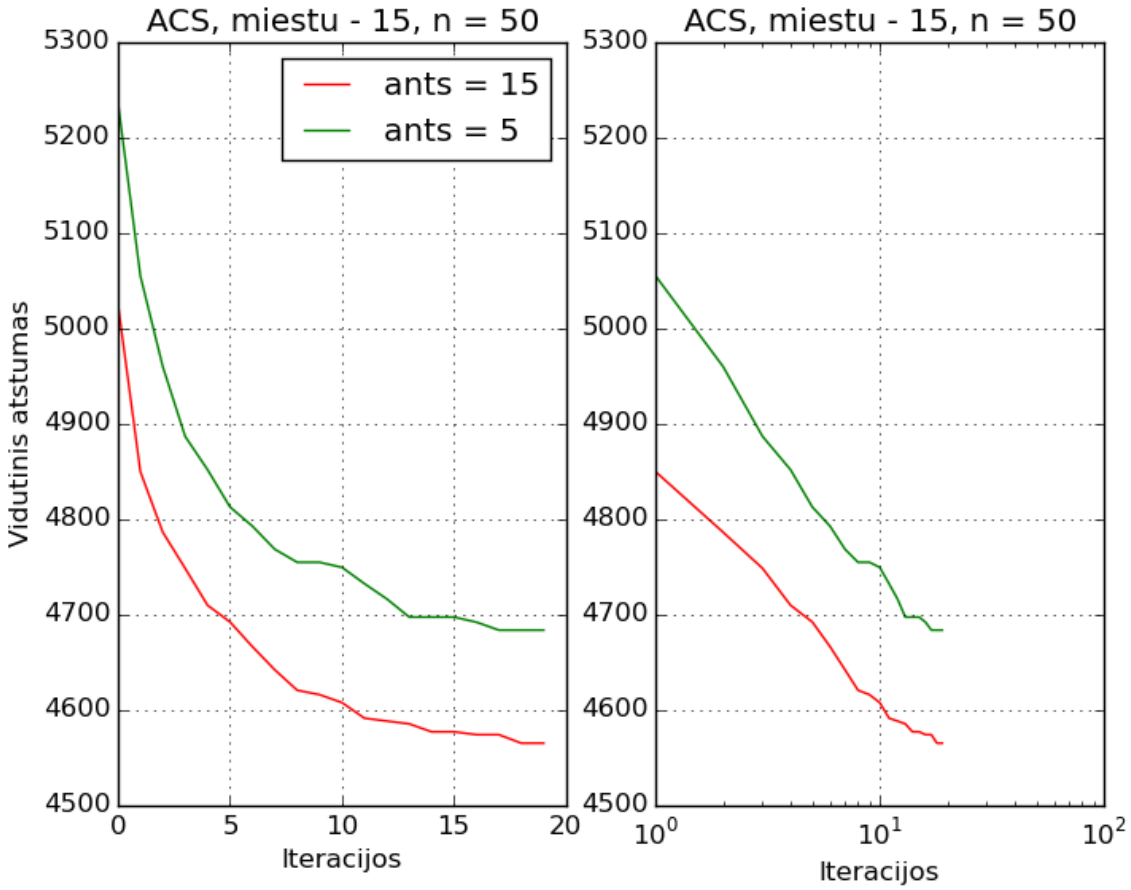
9. pvz. 50 kartų vykdyto ACS algoritmo vidutiniai rezultatai su parametrais $\alpha=0.1$, $\rho=0.9$, $\beta=1$, skruzdėlių skaičius 15 ir 5 , $iteracijų\,skaičius=20$ 15 miestų problemai.

Pastebėta, kad feromonų nykimo koeficientai (α , ρ) aiškios koreliacijos tarp jų verčių ir gautų rezultatų neturi. 9 Pav. naudojant $\beta=1$ buvo sprendžiama 15 miestų optimalaus maršruto problema naudojant 5 ir 15 skruzdėlių. Su 5 skruzdėlėm gauti vidutiniai rezultatai: atstumas – 4862, laikas – 0.32512s; su 15 skruzdėlių: atstumas – 4699, laikas – 0.96966. Didesnis skruzdėlių skaičius įtakoja trumpesnį vidutinį maršrutą, tačiau ir kur kas didesnį skaičiavimo laiką. 10 pav. buvo naudojamas $\beta=30$ ir gauti vidutiniai rezultatai naudojant 5 skruzdėles: atstumas – 4678, laikas – 0.477s; naudojant 15 skruzdėlių: atstumas – 4557, laikas – 1.407s. Pastebėta, kad esant didesniam β koeficientui algoritmas dirba apie vidutiniškai 40% ilgiau, o randamas atstumas yra apie 4%

trumpesnis. Galima daryti išvadą, kad prie didesnių β randamas atstumas trumpėja.

Kadangi prie didelio β koeficiento, kuris reiškia kad trumpas atstumas yra svarbiau už didelį feromono kiekį, gaunami vidutinio maršruto ilgiai yra trumpesni, galima daryti išvadą, kad feromono nykimo koeficientai didelės reikšmės galutiniam maršruto ilgiui neturi.

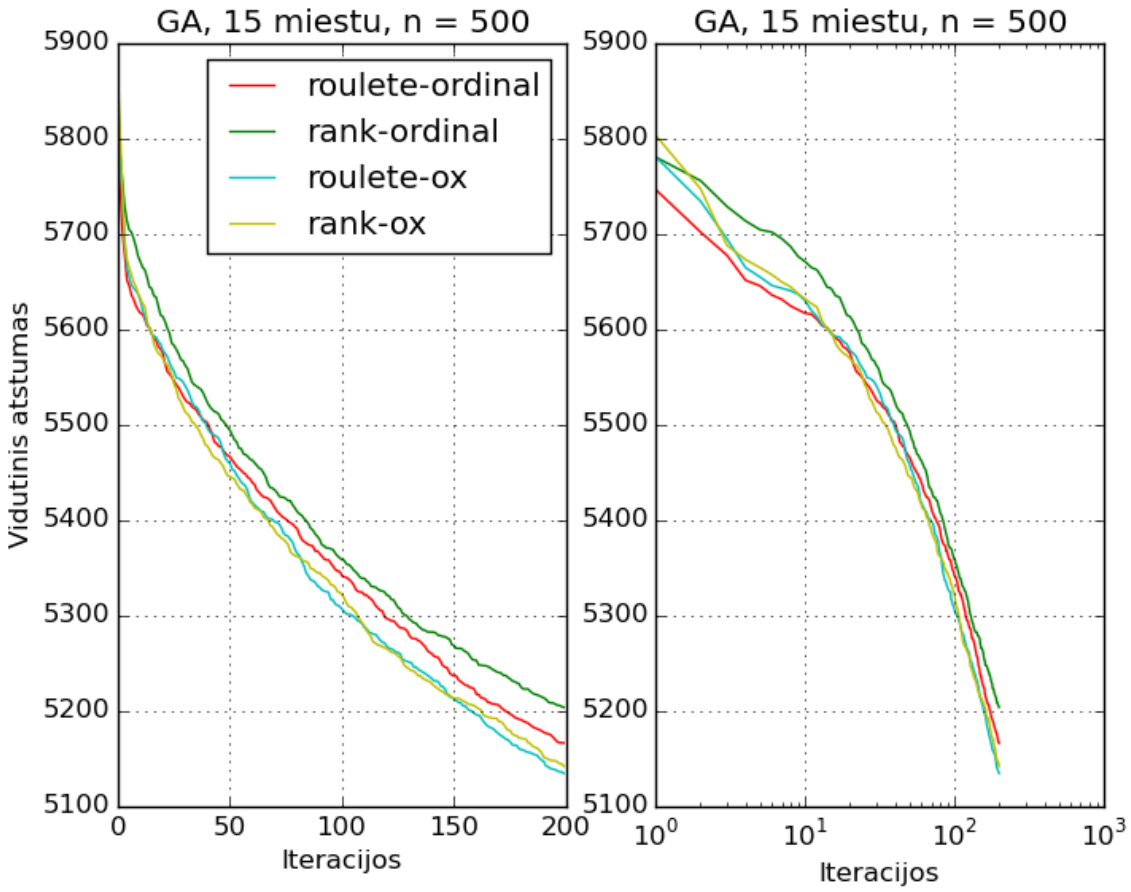
Lyginant ACS ir SA algoritmus, galima daryti išvadą, kad ACS algoritmas geba sugeneruoti geresnę pradinę rezultatų populiacija. Tačiau pastebėta, kad prie didesnių miestų skaičiaus, skruzdėlių maršrutų sudarymo laikas gerokai išauga.



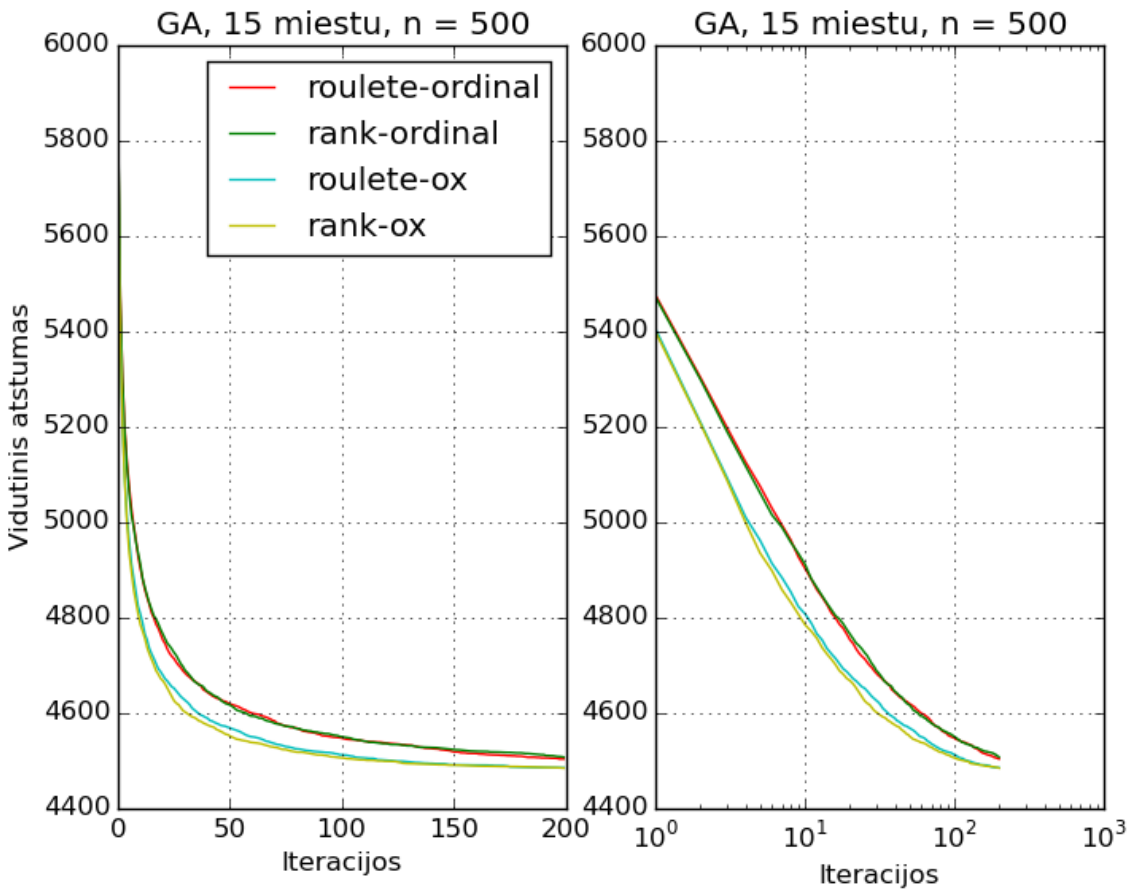
10. pvz. 50 kartų vykdyto ACS algoritmo vidutiniai rezultatai su parametrais $\alpha = 0.1$, $\rho = 0.9$, $\beta = 30$, skruzdėlių skaičius 15 ir 5, $iteracijų\ skaičius = 20$ 15 miestų problemai.

3.2. GA algoritmas

Pradžioje buvo analizuojama klasikinio GA algoritmo veikimą naudojant skirtingus parametrus, ruletės ir rango pasirinkimo bei OX ir kelintinės išraiškos rekombinacijos metodus (11 pav., 12 pav.).



11. pvz. 500 kartų vykdyto GA algoritmo vidutiniai rezultatai su parametrais $cr = 0.9$, $mr = 0.9$, $chromosomų = 14$, $iteracijų\ skaičius = 200$ 15 miestų problemai.



12. pvz. 500 kartų vykdyto GA algoritmo vidutiniai rezultatai su parametrais $cr = 0.1$, $mr = 0.1$, $chromosomų = 14$, $iteracijų\ skaičius = 200$ 15 miestų problemai.

Analizuojant visas GA algoritmo pasirinkimo ir rekombinacijos metodu kombinacijas (11 pav., 12 pav.) pastebėta, kad labiausiai maršruto kelio ilgį nulėmė rekombinacijos (cr) koeficientas, kuo šis koeficientas didesnis tuo prastesni rezultatai gaunami. 13 ir 14 vidutinių rezultatų lentelėse matyti, kad naudojant mažą cr koeficientą skaičiavimo laikas šiek tiek išauga, tačiau randamas trumpiausias atstumas taip pat sutrumpėja.

Naudojant koeficientą $cr = 0.1$ (12 pav.) matyti, kad OX rekombinacijos metodas duoda geresnius rezultatus nei kelintinės išraiškos rekombinacija. Naudojant OX rekombinacija, ruletės ir rango parinkimo metodų rezultatai praktiškai nesiskiria, gaunamas vidutinis atstumas – 4485, o laikas – 1.87s. Lyginant 11 pav. ir 12 pav. grafikus matyti, kad po 100 GA algoritmo iteracijų trumpiausi maršrutai yra apie 5300 ir apie 4580, todėl mažo cr koeficiento pranašumas yra

ryškus.

		Vidutinis atstumas	Vidutinis laikas (s)
9 miestai	roulete-ordinal	3146	0.55681
	roulete-ox	3146	0.65309
	rank-ordinal	3146	0.5702
	rank-ox	3146	0.69584
15 miestų	roulete-ordinal	4504	1.59314
	roulete-ox	4485	1.86061
	rank-ordinal	4507	1.57502
	rank-ox	4484	1.88258

13. pvz. 500 kartų vykdyto GA algoritmo vidutiniai rezultatai su parametrais $mr = 0.1$, $cr = 0.1$, $iteracijų\ skaičius = 200$, kai miestų skaičius 9 - $chromosomų = 8$, kai 15 - $chrom = 14$.

		Vidutinis atstumas	Vidutinis laikas (s)
9 miestai	roulete-ordinal	3410	0.02943
	roulete-ox	3401	0.02882
	rank-ordinal	3421	0.02956
	rank-ox	3420	0.02925
15 miestų	roulete-ordinal	5164	0.07802
	roulete-ox	5132	0.07761
	rank-ordinal	5202	0.07922
	rank-ox	5141	0.07862

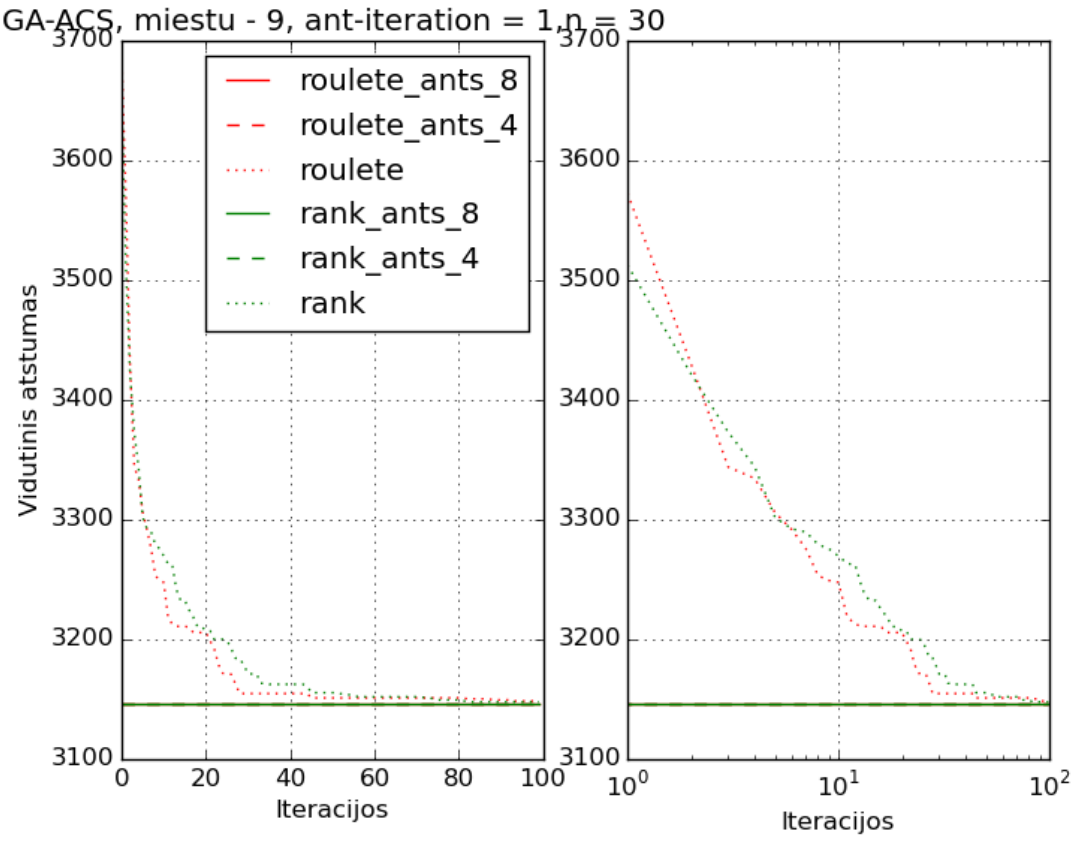
14. pvz. 500 kartų vykdyto GA algoritmo vidutiniai rezultatai su parametrais $mr = 0.9$, $cr = 0.9$, $iteracijų\ skaičius = 200$, kai miestų skaičius 9 - $chromosomų = 8$, kai 15 - $chrom = 14$.

3.3. GA algoritmo modifikacijos

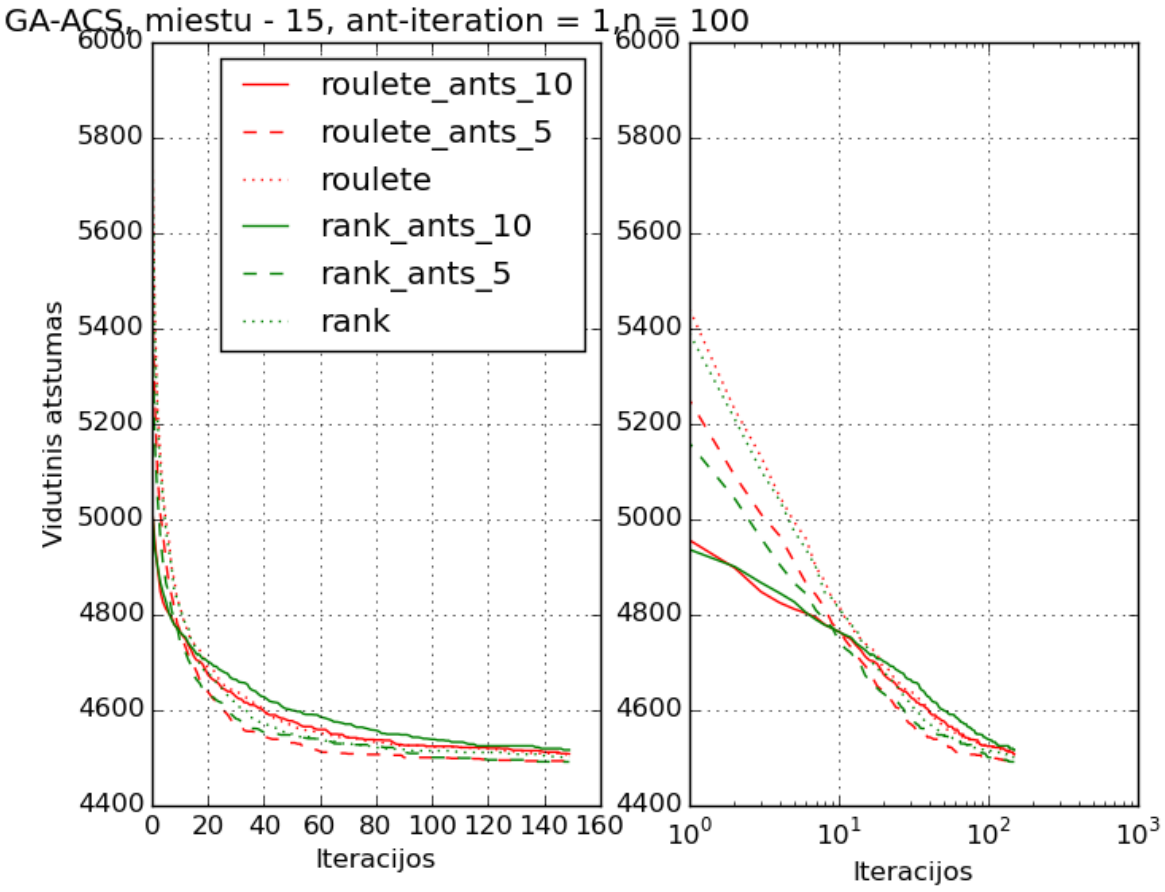
3.3.1. GA-ACS

GA algoritmas pradinę chromosomų šeimą sudaro atsitiktinai, o ACS naudojasi 2 formule, o iš rezultatų (10 pav., 12 pav.) matyti, kad būtent ACS turi geresnę pradinę populiaciją. Atsižvelgus į šią išvadą panaudotas šių dviejų algoritmų hibridinis junginys sprendžiant 9, 15 ir 23 miestų didžio optimaliausių maršrutų paieškos problemas.

Pradžioje buvo bandyta ieškoti optimalaus maršruto atlikus 1 ACS algoritmo iteraciją ir gautus skruzdėlių maršrutus panaudoti kaip genetinio algoritmo pradinę populiaciją. Taip pat buvo nurodyti skirtingi ACS skruzdėlių ir GA chromosomų skaičiai, jei sugeneruotu skruzdėlių maršrutų yra mažiau nei reikia chromosomų, likusios chromosomos sugeneruojamos įprastai – atsitiktiniu būdu. GA algoritmams parinkti ruletės ir rango pasirinkimo metodai, bei OX rekombinacija.

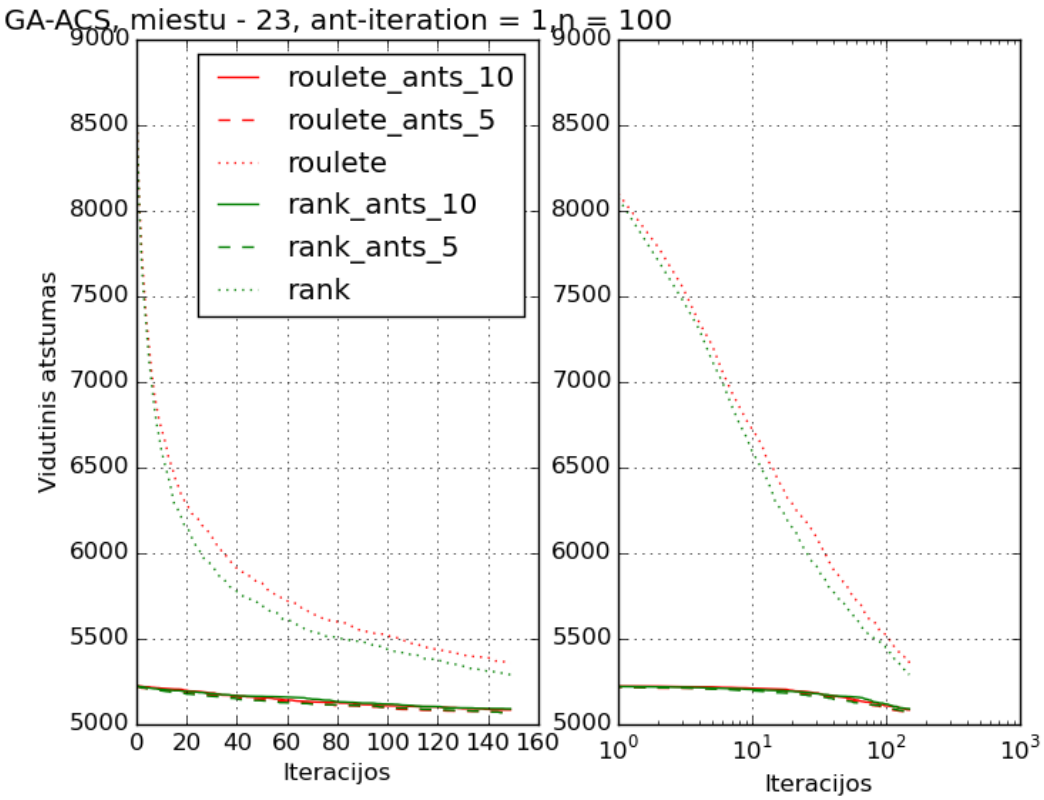


15. pvz. 30 kartų vykdyto GA-ACS hibridinio algoritmo vidutiniai rezultatai 9 miestų problemai. Parametrai: 8 chromosomos, 8 ir 4 skruzdėlės, 1 ACS iteracija, 100 GA iteracijų, $mr = 0.1$, $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$.



16. pvz. 100 kartų vykdyto GA-ACS hibridinio algoritmo vidutiniai rezultatai 15 miestų problemai. Parametrai: 14 chromosomos, 10 ir 5 skruzdėlės, 1 ACS iteracija, 100 GA iteracijų, $mr = 0.1$, $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$.

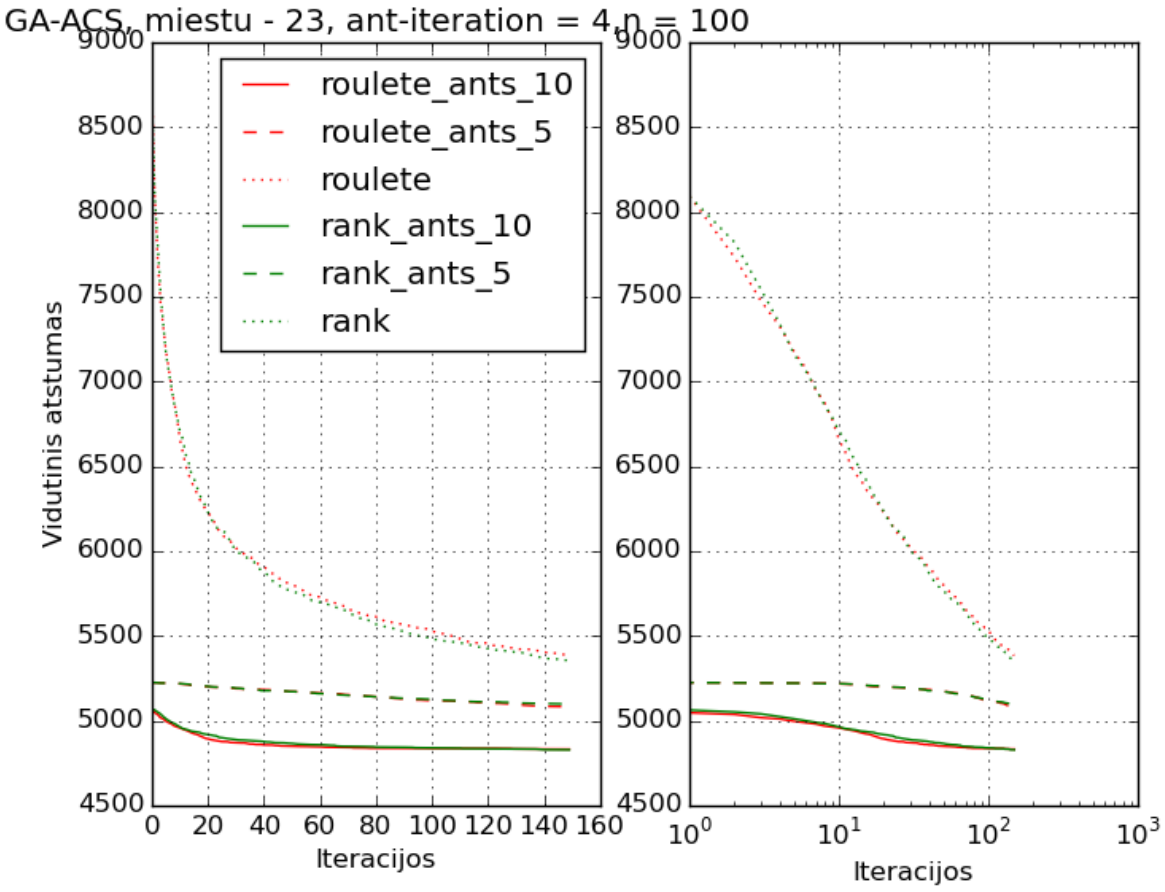
GA-ACS hibridinio algoritmo pranašumas 16 pav. matomas nežymiai, tačiau 15 ir 17 pav. šis pranašumas yra labai ryškus. 9 miestų atvejų visus 30 kartų naudojant GA-ACS hibridinį algoritmą vidutinis maršruto ilgis – 3147, kai perrinkimo būdu rastas maršruto ilgis yra 3146.



17. pvz. 100 kartų vykdyto GA-ACS hibridinio algoritmo vidutiniai rezultatai 23 miestų problemai. Parametrai: 30 chromosomų, 10 ir 5 skruzdėlės, 1 ACS iteracija, 100 GA iteracijų, $mr=0.1$, $cr=0.1$, $\alpha=0.1$, $\rho=0.1$, $\beta=30$.

23 miestų problemos atveju (17 pav) po 150 iteracijų rastų maršrutų vidutiniai ilgiai skiriasi nežymiai, o ir algoritmo veikimo laikai yra praktiškai vienodi, tačiau GA-ACS algoritmas jau pirmųjų iteracijų metu maršruto ilgis žymiai trumpesnis.

1 ACS iteracijos nepakako įžvelgti skruzdėlių ir chromosomų skaičiaus santykio įtaką GA-ACS algoritmo rezultatams. Padidinus ACS iteracijų skaičių iki 4, rezultatai pagerėjo (18 pav.). Matyti kad didesniam iteracijų skaičiui 10 skruzdėlių duoda geresnį rezultatą, nei 5. Padidinus ACS iteracijų skaičių pagerėja pradinių chromosomų populiacijos kokybė , tačiau algoritmo skaičiavimo laikas taip pat gerokai padidėja.



18. pvz. 100 kartų vykdyto GA-ACS hibridinio algoritmo vidutiniai rezultatai 23 miestų problemai. Parametrai: 30 chromosomų, 10 ir 5 skruzdėlės, 4 ACS iteracijos, 150 GA iteracijų, $mr = 0.1$, $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$.

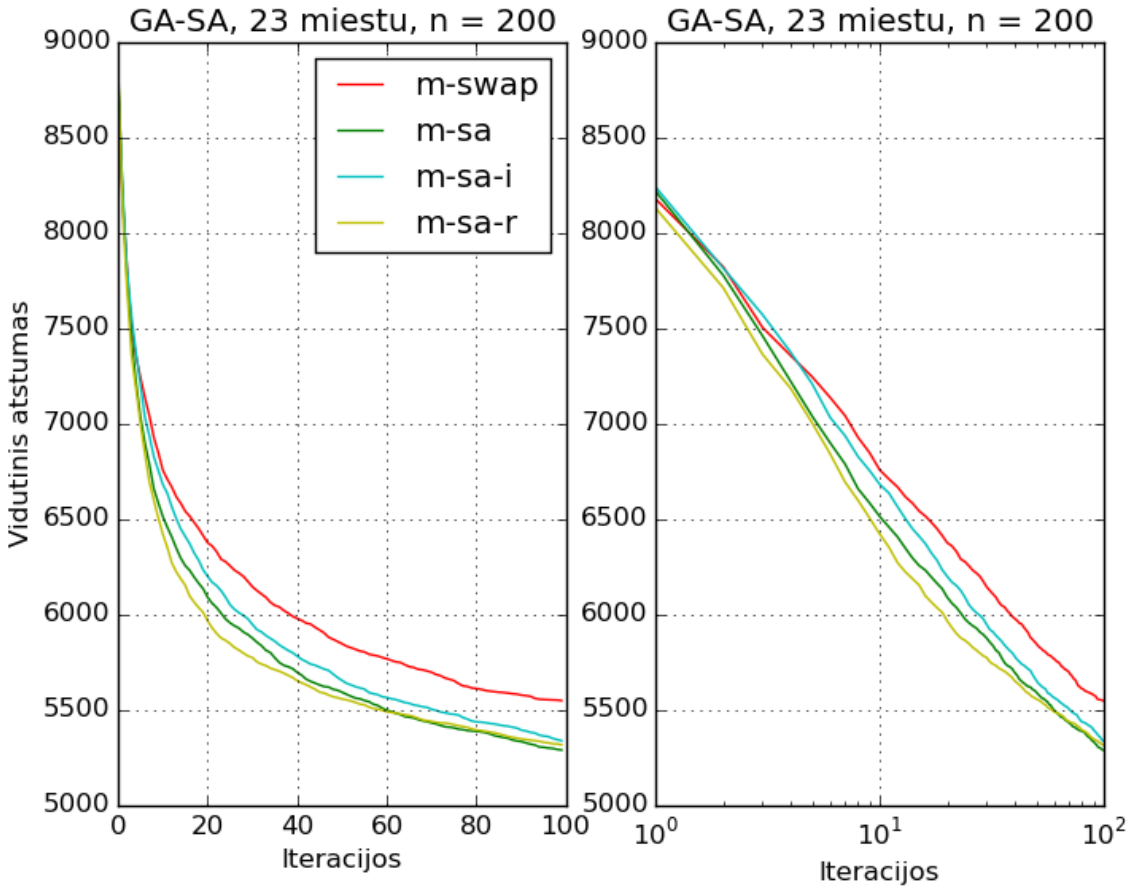
3.3.2. GA-SA

SA algoritmas, kad sugeneruotų naują maršrutą pagal tikimybę naudoja apvertimo, arba įterpimo operatorių (2 pav.). Kadangi iš ganėtinai prastų pradinių maršrutų SA algoritmas geba sugeneruoti ganėtinai neblogus maršrutus, buvo bandyta pritaikyti SA algoritmo maršruto modifikavimo operatorius kaip GA algoritmo mutacijos metodus.

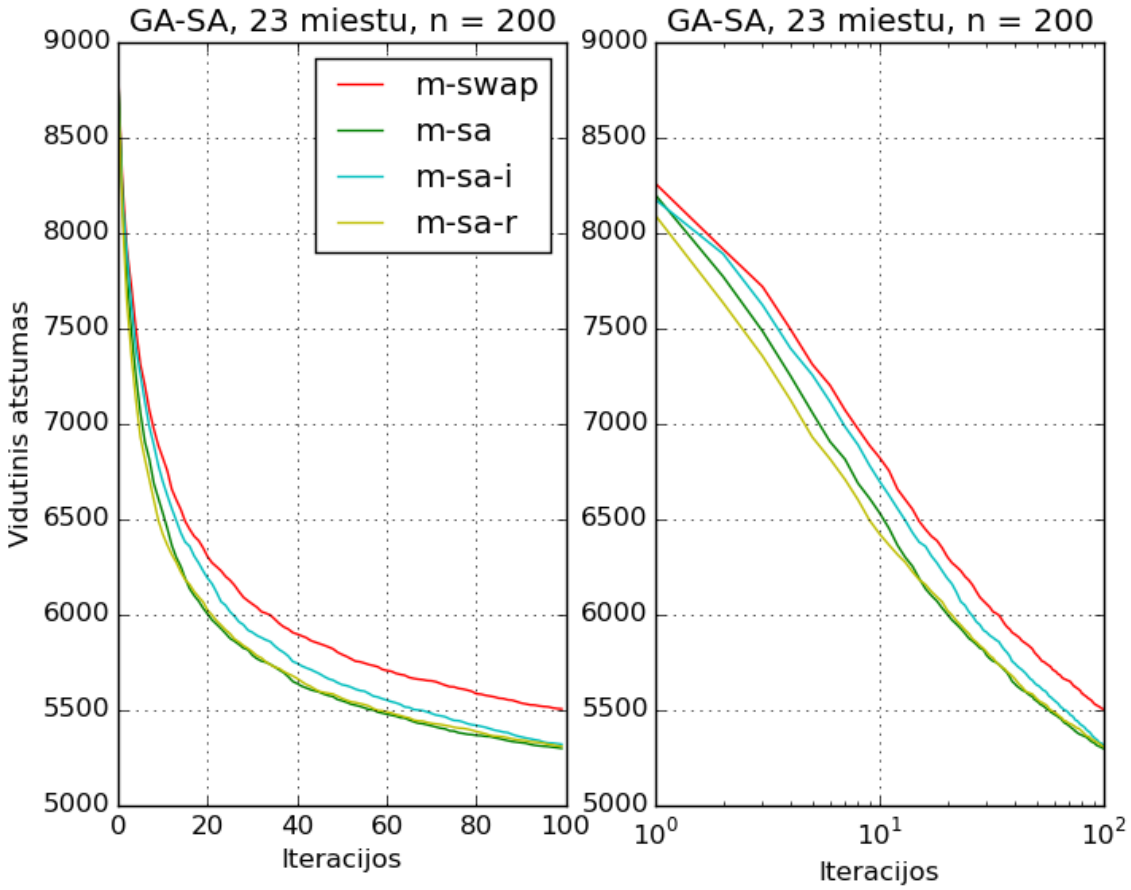
Su ruletės ir rango GA parinkimo metodais buvo kombinuojamos šios mutacijos:

- m-swap – klasikinė GA algoritmo mutacija, kuri buvo naudota visiems, viršuje esantiems, GA algoritmams.

- m-sa-i – SA algoritmo įterpimo modifikacija.
- m-sa-r – SA algoritmo apkeitimo modifikacija.
- m-sa – SA algoritmo tikimybė mutacija (2 pav.), kur atsitiktinai naudoja arba m-sa-i, arba m-sa-r modifikaciją.



19. pvz. 200 kartų vykdyto GA-SA hibridinio algoritmo rezultatai 23 miestų problemai, kai naudotas ruletės parinkimo metodas. Kiti parametrai: 100 iteracijų, OX rekombinacija, $mr = 0.5$, $cr = 0.1$, 22 chromosomos.

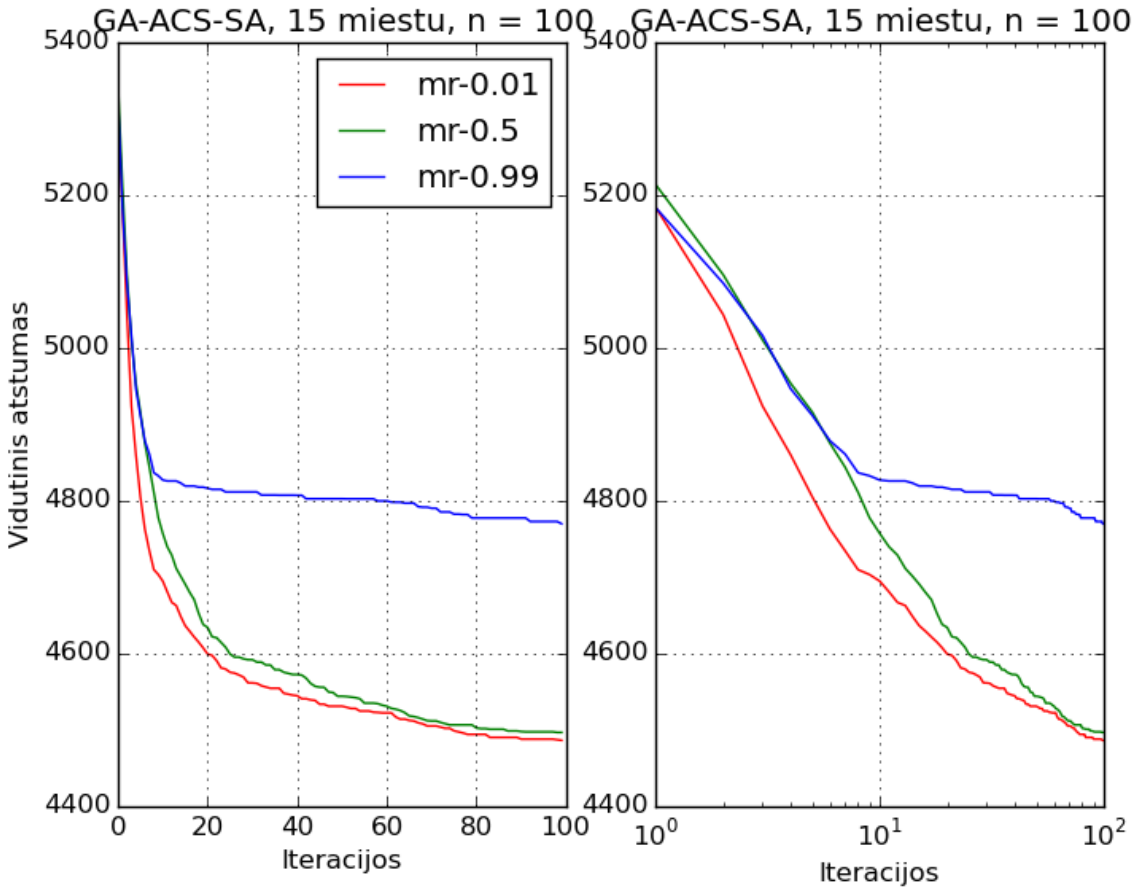


20. pvz. 200 kartų vykdyto GA-SA hibridinio algoritmo rezultatai 23 miestų problemai, kai naudotas rango parinkimo metodas. Kiti parametrai: 100 iteracijų, OX rekombinacija, $mr = 0.5$, $cr = 0.1$, 22 chromosomos.

19 pav. ir 20 pav. matyti kad tiek rango tiek ruletės GA algoritmo pasirinkimo atvejais GA algoritmas visų iteracijų metu trumpiausią atstumą turi naudodamas m-sa-r mutaciją. Analizuojant GA-SA hibridiniu algoritmu 9, 15 ir 23 miestų dydžio grafus m-sa-r mutacija geriausia, arba viena iš geriausių.

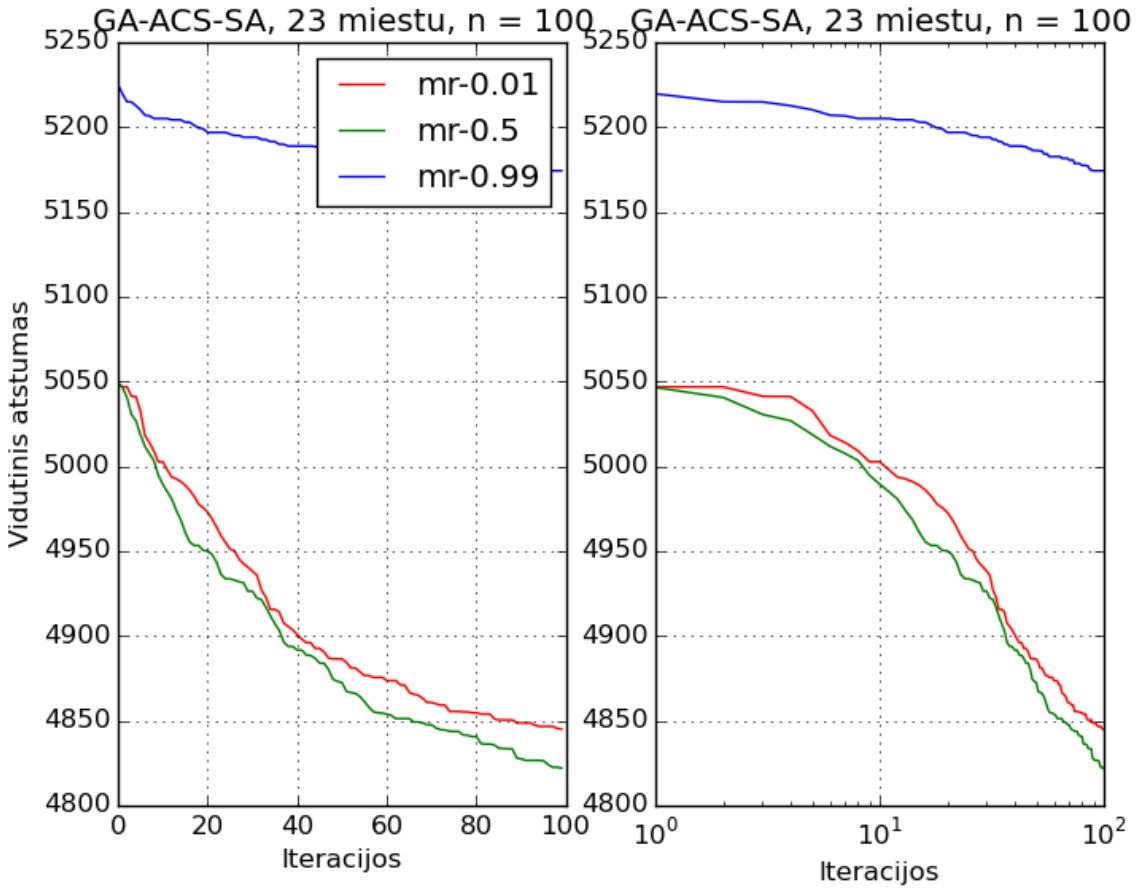
3.3.3. GA-ACS-SA

GA-ACS ir GA-SA algoritmus galima nesunkiai naudoti kartu. Buvo analizuoti GA-ACS-SA algoritmo rezultatai naudojant $mr = [0.01, 0.5, 0.99]$ koeficientus ir ruletės bei rango pasirinkimo metodus. GA-ACS-SA skruzdėlių skaičius sudarė 1/3 chromosomų skaičiaus, o kiti parametrai $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$.



21. pvz. 100 kartų vykdyto GA-ACS-SA hibridinio algoritmo rezultatai 15 miestų problemai, kai naudotas rango parinkimo metodas. Kiti parametrai: 100 iteracijų, OX rekombinacija, $cr = 0.1$, 14 chromosomų.

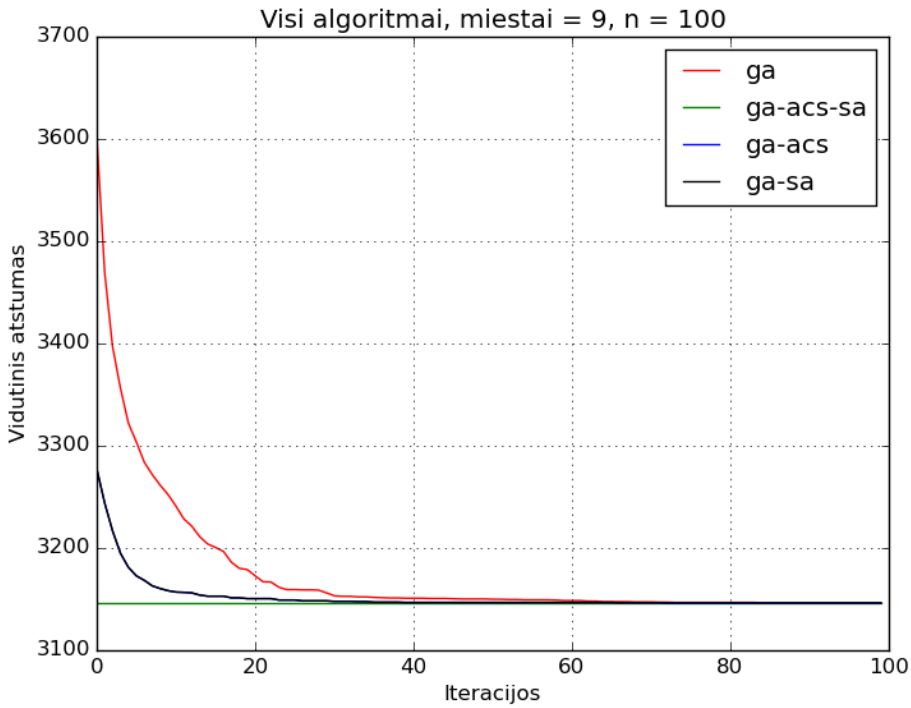
Iš 21 pav. ir 22 pav. grafikų matyti, kad greičiausiai optimaliausio maršruto link vystosi algoritmas kai $mr \leq 0.5$.



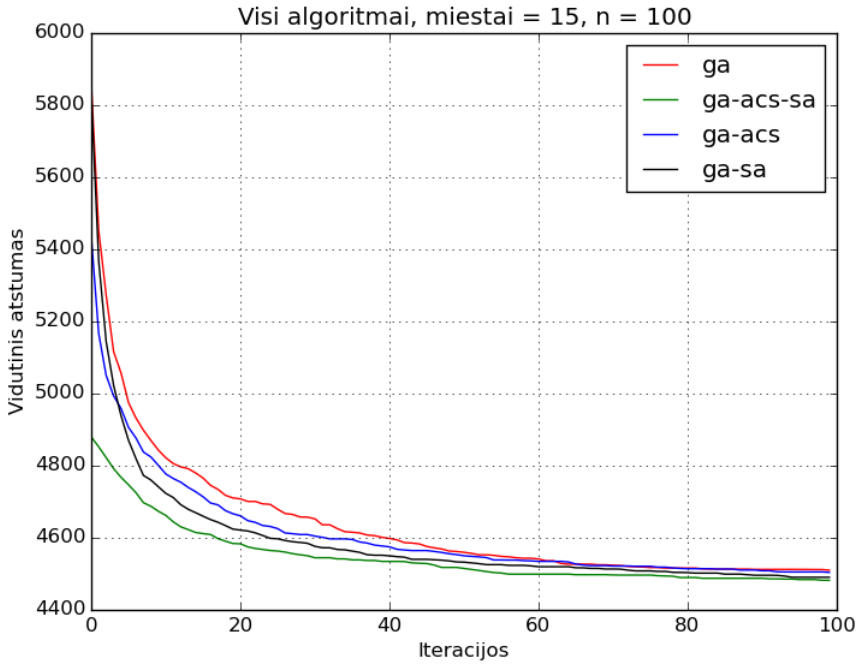
22. pvz. 100 kartų vykdyto GA-ACS-SA hibridinio algoritmo rezultatai 23 miestų problemai, kai naudotas ruletės parinkimo metodas. Kiti parametrai: 100 iteracijų, OX rekombinacija, $cr = 0.1$, 22 chromosomų.

3.3.4. Visi GA hibridai

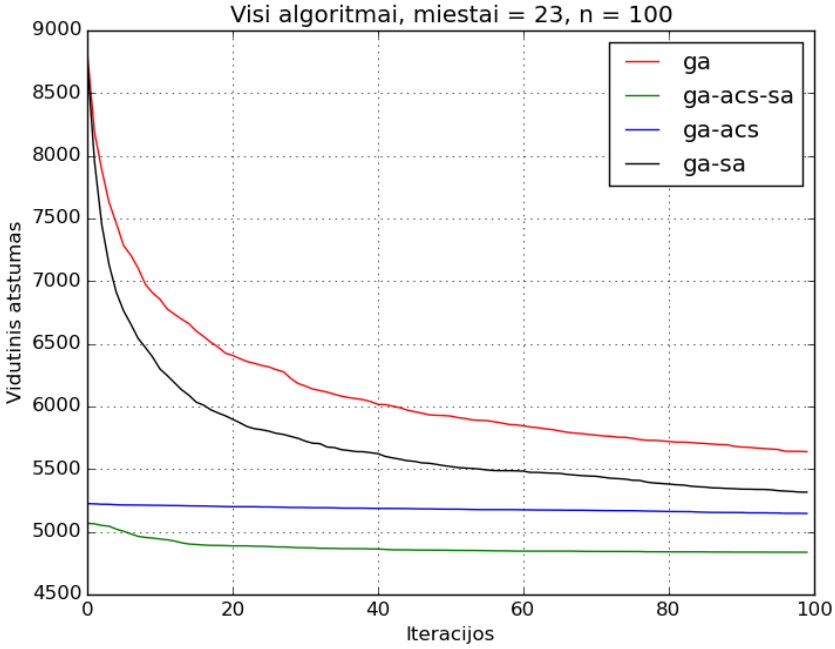
Naudojant visas GA algoritmo variacija buvo spręsto 9, 15 ir 23 miestų optimalaus maršruto problemos (23 pav., 24 pav., 25 pav.). Žvelgiant į visus grafikus matyti kad greičiausiai įsisotina GA algoritmo hibridas su ACS kolonijos generuotomis pradinėmis chromosomomis ir SA algoritmo kelio apvertimo funkcija.



23. pvz. 100 kartų vykdyto GA algoritmų variacijos 9 miestų problemai spręsti. Naudoti parametrai $mr=0.25$, $cr=0.1$, $\alpha=0.1$, $\rho=0.1$, $\beta=30$, 8 chromosomos, 2 skruzdėlės, 1 ACS iteracija. SA algoritmų atmainos naudotos su m-sa-r mutacija, kitos atmainos su m-swap.



24. pvz. 100 kartų vykdyto GA algoritmų variacijos 15 miestų problemai spręsti. Naudoti parametrai $mr = 0.25$, $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$, 14 chromosomos, 4 skruzdėlės, 2 ACS iteracijos. SA algoritmų atmainos naudotos su m-sa-r mutacija, kitos atmainos su m-swap.



25. pvz. 100 kartų vykdyto GA algoritmų variacijos 23 miestų problemai spręsti. Naudoti parametrai $mr = 0.25$, $cr = 0.1$, $\alpha = 0.1$, $\rho = 0.1$, $\beta = 30$, 22 chromosomos, 7 skruzdėlės, 3 ACS iteracijos. SA algoritmų atmainos naudotos su m-sa-r mutacija, kitos atmainos su m-swap.

4. Ateities darbai

Išanalizuoti heuristinių algoritmų veikimą grafuose, kuriuose viršūnių yra daugiau nei ieškomų (ieškoti trumpiausią maršrutą tarp n miestų grafe, kuriame miestų yra m, kai $m>n$).

Išanalizuoti kokios ACS algoritmo pradinio skruzdės maršruto parinkimo algoritmo savybės įtakoja GA algoritmo veikimą, bei išmėginti kitas pradinio maršruto generavimo alternatyvas.

Įvesti parametrus į SA maršruto apsukimo mutacijos funkciją ir išanalizuoti parametrų poveikį algoritmo veikai.

Išanalizuoti naujesnių GA algoritmo rekombinacijos bei mutacijos metodų įtaką algoritmui, dinamiškai keičiant parametrus ir pačius metodus.

5. Išvados

Naudojant panašius į sausumos žemėlapių grafus ir sprendžiant trumpiausio maršruto per n miestų uždavinį GA algoritmo veikimą galima pastebimai pagerinti jei dalis pradinės populiacijos chromosomų yra sugeneruojamos ACS algoritmu.

ACS algoritmo pradinės populiacijos, naudojamos GA algoritme, kokybę labiausiai įtakoja β koeficientas. Kuo jis didesnis tuo gauta populiacija yra geresnė. Kiti ACS feromonų mažėjimo koeficientais pastebimos reikšmės nesukelia.

SA algoritmo dalies maršruto apsukimo (angl. *reverse*) metodas panaudotas kaip GA algoritmo mutacijos operatorius pastebimai pagerina tiek paprasto GA algoritmo, tiek GA algoritmo su pradine ACS chromosomų populiacija veikimą, lyginant su miestų apkeitimo vietomis (angl. *swap*) ir SA algoritmo maršruto dalies įterpimo (angl. *insert*) funkcija.

6. Literatūros sąrašas

[CheChi11]	Shyi-Ming Chen, Chih-Yao Chien, Solving the traveling salesman problem based on the genetic simulatedannealing ant colony system with particle swarm optimization techniques. 2011
[Pot96]	Jean-Yves Potvin, Genetic algorithms for thetraveling salesman problem, 1996
[LiZhoGui11]	Yang Li, Aimin Zhou, Guixu Zhang, Simulated Annealing with ProbabilisticNeighborhood for Traveling Salesman Problems, 2011
[YuHuZha09]	Wei-jie Yu, Xiao-min Hu, Jun Zhang, Self-Adaptive Ant Colony System for the TravelingSalesman Problem, 2009
[Šar14]	Karolis Šarapnickis, Mokslo tiriamasis darbas I: optimalių maršrutų paieškos algoritmai, 2014