VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Keliaujančių pirklių uždavinys

Multiple traveling salesman problem

Magistro baigiamasis darbas

Atliko: Gžegož Jurgo (parašas)

Darbo vadovas: Doc. Algirdas Bastys (parašas)

Recenzentas: Doc. G. Skersys (parašas)

Vilnius – 2012

Santrauka

Atliekant magistrinį darbą pagrindinis tikslas buvo išnagrinėti keliaujančių pirklių uždavinį su papildomais apribojimais. Darbo metu buvo pridėtas pirklio keliamosios galios apribojimas. Išanalizuoti įmanomi sprendimo budai. Darbo metu buvo realizuotas genetinis algoritmas gebantis spręsti iškeltą uždavinį. Sugalvoti ir realizuoti uždavinio sprendimui reikalingi genetiniai operatoriai. Realizuoti lokalaus optimizavimo algoritmai. Atlikti testavimo darbai bei gauti galimi sprendiniai.

Summary

The main goal of the master's thesis was to analyze travelling salesmen problem with additional limitations.

The limitation of salesman's lifting force was entered during study. Possible calculation methods were analyzed. During the study genetic algorithm was applied, possible of handling current problem. Genetic operators, needed for solving travelling salesmen problem were created and applied. Besides that, local route optimization algorithms were implemented. Tests were accomplished and possible solutions found.

Turinys

1. Įvadas ... 7
1.2 Pritaikymo sritys ... 8
1.3 Darbo tikslas ... 8
1.4 Darbo uždaviniai ... 9
2. Literatūros apžvalga ... 10
2.1 Sprendimo būdai ... 10
2.1.1 Tikslieji algoritmai ... 10
2.1.2 Euristiniai algoritmai ... 10
2.1.2.1 K – centrų algoritmas ... 11
2.1.2.2 Artimiausio kaimyno algoritmas ... 12
2.1.2.3 Įtraukimo algoritmas ... 12
2.1.2.4 Dviejų briaunų apkeitimo algoritmas ... 13
2.1.2.5 „K“ – briaunų apkeitimo algoritmas ... 14
2.1.2.6 Lin – Kernigham algoritmas ... 14
2.1.3 Genetiniai algoritmai ... 15
2.1.3.1 Vienos chromosomos technika ... 16
2.1.3.2 Dviejų chromosomų technika ... 17
2.1.3.3 Dviejų dalių chromosomos technika ... 17
2.1.3.4 Chromosomų sudarymo technikų palyginimas ... 18
2.1.4 Neuroniniai tinklai ... 20
2.1.4.1 Elastingo tinklo modelis ... 21
2.1.4.2 Savarankiškai organizuojantis ypatybių žemėlapis ... 22
3. Uždavinio sprendimas genetinio algoritmo pagalba ... 23
3.1 Pagrindinė genetinio algoritmo schema ... 23
3.2 Chromosomos sandara ... 24
3.3 Algoritmo veikimo nutraukimo sąlyga ... 24
3.3.1 Viršyto laiko saugiklis ... 25

3.3.2 Viršyto iteracijų skaičiaus saugiklis ... 25
3.3.3 Tinkamumo funkcijos įvertinimas ... 25
3.3.4 Pasirinkta sustojimo sąlyga ... 25
3.4 Pradinės populiacijos sudarymas ... 26
3.4.1 Atsitiktinis populiacijos sudarymas ... 26
3.4.2 Populiacijos sudarymas kitais grubiais algoritmais ... 27
3.4.3 Pasirinktas pradinės populiacijos generavimo modelis ... 27
3.4 Tinkamumo funkcija ... 27
3.5 Lokalus optimizavimas ... 28
3.6 Genetiniai operatoriai ... 29
3.6.1 Pasirinkimo operatorius ... 29
3.6.1.1 Reitingavimo metodas ... 29
3.6.1.2 Ruletės metodas ... 30
3.6.1.3 Turnyro metodas ... 30
3.6.1.4 Pasirinktas išrinkimo metodas ... 30
3.6.2 Kryžminimo operatorius ... 30
3.6.2.1 Paprastas atsitiktinis kryžminimas ... 30
3.6.3 Mutacijos operatorius ... 31
4. Genetinių algoritmų tyrimas panaudojant "Symphony" duomenų rinkinį ... 32
4.1 Pradiniai duomenys ... 32
4.2 Chromosomos modelis ... 32
4.3 Pradinės populiacijos sudarymas ... 33
4.3.1 Atsitiktinis individų generavimas ... 33
4.3.2 Individų generavimas euristiniu algoritmu ... 34
4.4 Tinkamumo funkcija ... 34
4.5 Mutacijų operatoriai ... 36
4.6 Kryžminimo operatoriai ... 39
4.6.1 Persikertančių miestų apkeitimas ... 39
4.6.2 Uždaros dėžutės ... 40
5. Rezultatai ... 42

6. Išvados 45

7. Šaltinių sąrašas 46

8. Santrumpos 49

9. Priedai 50

1. Įvadas

Keliaujančių pirklių toliau (MTSP) uždavinys yra apibendrinimas gerai žinomo keliaujančio pirklio uždavinio, toliau (TSP). TSP veikimas yra toks: vienas pirklys aplanko visus pristatymo taškus tik po vieną kartą, pradėdamas savo kelionę pradiniame taške ir užbaigdamas tame pačiame taške. Atliktų mokslinių tyrimų apie TSP yra nemažai, pateikiama daug sprendimo būdų, tačiau MTSP nėra plačiai nagrinėjamas, jam neskiriama labai daug dėmesio. Nepaisant to, MTSP uždavinys turi nemažai pritaikymo sričių šiuolaikiniame pasaulyje. MTSP uždavinį galima apibrėžti taip: yra taškų aibė, ir “m” skaičius pirklių, kurie visi yra viename taške, šis taškas vadinamas pradiniu kelionės tašku, likusieji taškai vadinami kelionės taškais. Uždavinys yra rasti “m” skaičių skirtingų kelių kiekvienam pirkliui, aplankant kiekvieną kelionės tašką tik vieną kartą, kartu minimizuojant bendrą kelionės kainą. Kelionės kainą galima apibrėžti kaip atstumą, reikalingą nukeliauti laiką, būtiną kelionei įveikti [Bek05]. Uždavinys gali turėti skirtingus sprendimo variantus:

1. Vieno arba kelių pradinių taškų variantas.

Uždavinys gali būti apibrėžtas vienu pradiniu tašku, šiuo atveju visi “m” pirklių iškeliauja iš vieno taško ir turi grįžti į jį. Kelių pradinių taškų atveju turima “n” pradinių taškų ir “m” pirklių kiekviename iš jų. Galima apibrėžti sąlygą, kad kiekvienas pirklys turi grįžti į savo pradinį tašką arba gali apsistoti kitame pradiniame taške, bet turi būti išpildyta sąlyga, kad pirklių skaičius kiekviename taške turi likti pabaigoje toks pat, kaip buvo judėjimo pradžioje .

2. Pirklių skaičiaus apribojimo variantas.

Uždavinys gali būti sprendžiamas apibrėžus tam tikrą pirklių skaičių kaip konstantą, tuomet bet kokiu atveju visi pirkliai bus panaudoti sprendimo gavimui, tačiau įmanomas ir toks atvejis: kai turima “n” skaičių pirklių, bet sprendimui norima panaudoti ne daugiau kaip “x” skaičių pirklių, tai nereiškia, kad visi “x” pirkliai dalyvaus sprendime, gali būti, kad bus gauta minimizuota kelionė jau su “y” skaičiumi pirklių, kur: $y<x<n$

3. Apribotos keliamosios galios variantas.

Galima spręsti uždavinį, kai mes turime nelimituotą skaičių pirklių, bet kiekvienas pirklys turi apribotą keliamąją galią, tada uždaviniui išspręsti reikia surasti kuo mažesnį skaičių pirklių, reikalingą tikslui pasiekti [Bek05].

4. Laiko apribojimo (angl. Time windows) variantas.

Kiekvienas aplankomas taškas gali turėti laiko apribojimus, tai reiškia kad į jį galima atvykti tik tam tikrame laiko intervale. Šis atvejis yra vienas svarbesnių MTSP uždaviniuose, kadangi turi nemažai pritaikymo sričių, kurios bus aptartos kitame skyriuje.

5. Kitokie apribojimų variantai.

Pirkliams galima priskirti įvairių apribojimų, pavyzdžiui: kiekvienas pirklys gali nukeliauti tik į tam tikrą skaičių taškų arba nukeliauti tik tam tikrą atstumą.

TSP uždavinį galima traktuoti kaip MTSP uždavinio vienetinį atvejį, kai turime vieną pradinį tašką, vieną pirklį ir jokių kitų apribojimų.

1.2 Pritaikymo sritys

1. Įgulos tvarkaraštis.

Centrinio banko pavyzdys, kai kasdien banko darbuotojai turi aplankyti visus padalinius, surinkti ten gautus pinigus ir kelionės gale pristatyti atgal. Tai MTSP minimizuoja bendrą kainą, reikalingą padaliniams aplankyti. Pašto pristatymo tarnyba kasdien turi pristatyti tam tikrą skaičių siuntinių, turėdama fiksuotą skaičių resursų (automobilių, paštininkų).

2. Mokyklos autobusų.

Uždavinys yra sprendžiamas panašiai kaip ir su aukščiau paminėtu įgulos tvarkaraščiu, kai yra pridedama papildomų tikrinimo sąlygų, pavyzdžiui, atsižvelgti į tai kad keleivių skaičius neviršytų leistino, taip pat bandoma mažinti reikalingų resursų kiekį, kad nevažiuotų pustuščiai autobusai [Bek05].

3. Turto brokerių.

Įmonės, užsiimančios nekilnojamu turtu, kasdien sutaria su pardavėjais ir pirkėjais, kada susitikti, tuomet laukiantys asmenys turi apribotą susitikimo laiką. Papildę MTSP papildoma sąlyga, kad reikia atsižvelgti ir atvykimo laikus, programa planuotų maršrutus taip, kad kiekvienas laukiantis asmuo būtų aplankytas jam tinkamu metu, ir bendra kelionės kaina būtų minimizuota. Variantų, kur galima pritaikyti laiko apribojimus, yra nemažai. Lietuvos rinkai labai aktualu būtų transportavimo paslaugomis užsiimančios įmonės, kadangi yra aplankoma nemažai punktų, dažniausia yra apribojama laike, iki kada galima pristatyti, kada galima pasiimti naują krovinį.

1.3 Darbo tikslas

Pagrindinis darbo tikslas yra sukurti veikiantį algoritmą, kuris sugebėtų spręsti optimizavimo uždavinį. Optimizuoti keliones kelioms transporto priemonėms, išvykstančioms iš vieno pradinio taško, kiekviena transporto priemonė turi apribotą keliamąją galią.

1.4 Darbo uždaviniai

Atliekant darbą buvo suformuoti pagrindiniai uždaviniai:

1. Išanalizuoti esamą literatūrą apie dominančios problemos sprendimą.
2. Išsirinkti vieną iš literatūroje rastų realizavimo būdų.
3. Įgyvendinti algoritmą. Pritaikyti jį VRP uždaviniui.
4. Įtraukti darbo eigoje sugalvotus patobulinimus.
5. Atlikti programos efektyvumo analizę.
6. Pasiūlyti patobulinimų tolesniam algoritmo vystymui.

2. Literatūros apžvalga

Matematinė uždavinio formuluotė: tegul A bus visų reikalingų aplankyti taškų aibė, C bus bazinis taškas, kuris priklauso aibei A, $C \in A$. Pirklių aibę pažymėkime raide P, taip kad $P < A$. Reikia rasti aibę S, kuri bus kelionių aibė kiekvienam pirkliui iš aibės P, taip kad kiekvienas aibės C elementas priklausytų tik vienam iš poaibių S. Tokiu būdu yra gaunama S_p skaičius kelionių kiekvienam pirkliui [Bek05].

2.1 Sprendimo būdai

Sprendžiant kelių keliaujančių pirklių uždavinį, nagrinėtoje literatūroje yra siūlomi keli skirtingi sprendimo būdai, kurie yra suskirstyti pagal algoritmų klases:

1. Tikslieji algoritmai.
2. Euristiniai algoritmai
3. Gentiniai algoritmai.
4. Neuroninių tinklų algoritmai.

2.1.1 Tikslieji algoritmai

Spręsti uždavinį galima pasitelkiant tiksliuosius algoritmus. Uždavinys sprendžiamas perrenkant visas įmanomas kelionių kombinacijas kiekvienam pirkliui. Toks sprendimo būdas yra tinkamas problemoms, kuriose yra nedaug aplankytinų taškų, ir pirkliui, su kuriuo reikia aplankyti tuos taškus, kadangi uždavinio sudėtingumas yra O(n + m - 1)!, kur „n“ yra aplankytinu taškų skaičius, o „m“ yra pirklių skaičius, ir n > m . Tai yra labai didelis skaičius kombinacijų net ir šiuolaikiniams kompiuteriams, kadangi turėdami bent 10 aplankytinų taškų ir 3 pirklius mes gauname 12, kas yra lygu: 479001600 palyginimo kombinacijų. Algoritmo laikas auga eksponentiškai su kiekvienu nauju tašku arba pridėjus naują pirklį, taigi praktiškai tokie algoritmai nėra taikomi, kadangi apskaičiuoti uždavinį su 20 aplankytinų taškų tampa praktiškai neįmanoma per normalų laiko tarpą [Hus89].

2.1.2 Euristiniai algoritmai

Euristiniai algoritmai yra taikomi tada, kai tikslieji algoritmai nėra praktiški. Euristiniai algoritmai sprendžia uždavinius mokymosi, atradimo būdais, ir žymiai pagreitina sprendimo procesą, tačiau jie nėra tikslieji algoritmai, todėl sprendžia uždavinius su tam tikrom paklaidom. Pagrindinis klausimas – kokia ta paklaida: ar nėra ji labai didelė, ir ar didinant

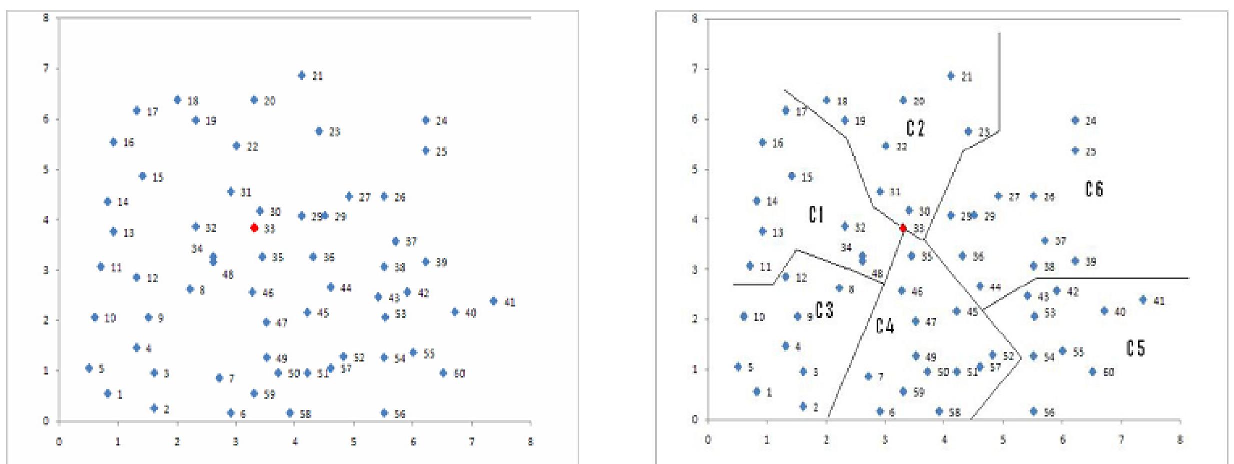
tikslumą nėra prarandama labai daug laiko, kuris reikalingas sprendimui rasti. Jiems tenka balansuoti tarp tikslumo ir laiko, reikalingo uždaviniui išspręsti. Apžvelgsime euristinius algoritmus, kurių pagalba sprendžiamas kelių keliaujančių pirklių uždavinys, iš karto transformuojant MTSP į TSP.

2.1.2.1 K – centrų algoritmas

K – centrų algoritmas atlieka objektų, mūsų atveju taškų suskaidymą į pasirinktas grupes, pagal tam tikrą objekto atributą, mūsų nagrinėjamame uždavinyje bus naudojamas atstumas tarp taškų. Grupavimas atliekamas minimizuojant kiekvienos grupės atstumų sumą, darant poslinkį grupės centriniam taškui [HaW79]. Algoritmo veikimą galima aprašyti 4 žingsniais:

1. Patalpinti pradinius „k“ centrinius taškus, kur „k“ yra norimų sudaryti grupių skaičius.
2. Suskirstyti likusius taškus į grupes pagal jų atstumą iki pradinių centrinių taškų.
3. Kai visi taškai yra priskirti į grupes, perskaičiuoti centrinį tašką, kad jis būtų grupės centre.
4. Kartoti 2 ir 3 žingsnius iki tol, kol centriniai taškai daugiau nejudės.

K – centrų algoritmas tinka uždavinių sprendimui, nes jis padalina taškus ne pagal lygų skaičių, o pagal tai, kad būtų gauti panašūs minimalūs atstumai tarp taškų. Skaidant taškus į grupes, gaunamos kelionės kiekvienam pirkliui, taip yra gaunamas tam tikras skaičius taškų, kuriems reikia paskaičiuoti kelionę. Šis skaičius yra žymiai mažesnis, nei tektų skaičiuoti per visus skaičius [NDDP09]. Tai vizualiai parodyta paveikslėlyje numeris 1 [NDDP09].



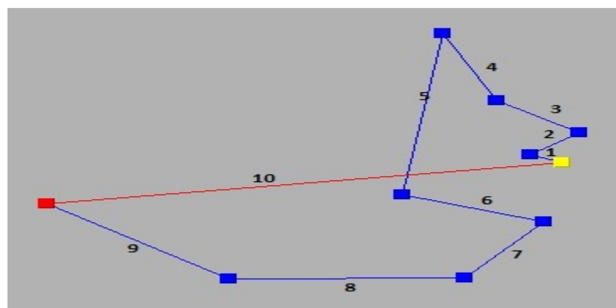
1. Taškų suskirstymas į grupes

Kaip matome iš pirmo paveikslėlio, 60 pradinių taškų buvo suskirstyta į 6 grupes. Kiekvienai grupei tenka maždaug po 10 taškų, kas leidžia mums nustatyti kelionę daug greičiau, kadangi kiekvienam pirkliui reikia paskaičiuoti jau jo tikslią kelionę ne su 60 taškų, o su maždaug 10. Kai taškai jau yra suskirstyti į grupes, galima pasinaudoti vienu iš euristinių algoritmų vieno pirklio kelionei rasti, tai yra – išspręsti TSP uždavinį. Jo sprendimui yra keli žinomiausi euristiniai algoritmai:

1. Artimiausio kaimyno;
2. Įtraukimo;
3. Dviejų briaunų apkeitimo;
4. „K“ – briaunų apkeitimo;
5. Lin – Kernigham.

2.1.2.2 Artimiausio kaimyno algoritmas

Artimiausio kaimyno algoritmas – pats paprasčiausias iš euristinių algoritmų. Jo pagrindinė mintis – pradėti nuo tam tikro taško, sekantį tašką pasirinkti artimiausią nuo jo esantį tašką ir juos apjungti. Prijungtam taškui surandam jo artimiausią tašką, bet jis negali buti iš jau apjungtų taškų aibės. Ir taip yra kartojama tol, kol sujungiami visi taškai. Paskutinis taškas sujungiamas su pirmu: dėl to dažniausiai gaunasi, kad kelionė iš paskutinio taško į pirmą turi labai didelį atstumą [Che03] – žiūrėti paveiksliuką numeris 2. Paveiksliuke pradinis taškas pažymėtas geltonai, o paskutinis raudonai, skaičiai ant briaunų žymi iteracijas.



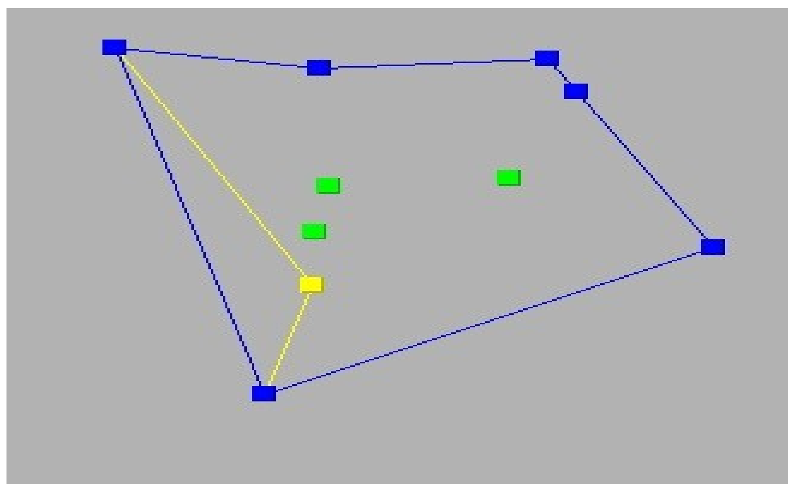
2. Artimiausio kaimyno algoritmo veikimas.

2.1.2.3 Įtraukimo algoritmas

Įtraukimo algoritmas irgi yra vienas iš paprasčiausių euristinių algoritmų. Jo veikimo principas yra toks: iš karto sujungiami labiausiai nutolę taškai – taip formuojasi pradinė kelionė [Mer99]. Įtraukti likusius taškus egzistuoja 2 būdai:

- Ieškant artimiausio taško, nutolusio nuo pradinės kelionės. Iteracijos kartojamos tol, kol visi taškai būna sujungti; kiekvienos iteracijos metu yra mažinamas pradinės kelionės atstumas.
- Ieškant tolimiausio taško, nutolusio nuo pradinės kelionės. Iteracijos kartojamos tol, kol visi taškai būna sujungti.

Būdas pasirenkamas tas, kurio pradinių iteracijų pagalba gaunamas mažiausias atstumas. Žiūrėti paveiksliuką numeris 3.



3. Įterpimo algoritmo veikimas

Mėlyni taškai, sujungti linijomis, žymi pradinę kelionę. Taškai, kuriuos reikia apjungti, pažymėti žalia spalva. Pavyzdyje nagrinėjamas artimiausio taško įtraukimo algoritmas. Geltona spalva žymimas operuojamas taškas ir kaip jis bus apjungtas.

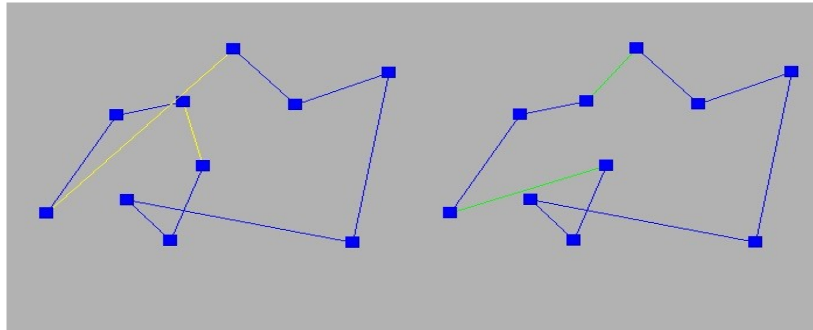
2.1.2.4 Dviejų briaunų apkeitimo algoritmas

Algoritmo veikimas yra gana paprastas ir efektyvus. Pradinis kelionės sudarymas gali būti dviejų būdų:

- Pradinė kelionė yra sudaroma pasinaudojant kitų euristinių algoritmų, dažniausiai yra pasirenkamas artimiausio kaimyno principas, kadangi jis yra pats paprasčiausias ir greičiausias, o dviejų briaunų apkeitimo algoritmas tiesiog atlieka patobulinimus, taip sutrumpindamas kelionę.
- Pradinė kelionė yra sudaroma atsitiktiniu būdu apjungiant taškus. Toks būdas yra rečiau naudojamas, nes reikalauja daugiau apkeitimo operacijų.

Briaunų apjungimo principas: pasirenkamos 2 poros viršūnių, bandoma jas apjungti kitu būdu, nei jos yra apjungtos dabar, ir tikrinama, ar kelionės atstumas sumažėja.

Jei sumažėja, yra paliekamas pakeitimas, pasirenkamos sekančios dvi poros ir atliekama ta pati operacija [Mer99]. Iteracijos atliekamos kol egzistuoja, kurios dar nebuvo keičiamos, žiūrėti paveiksliuką numeris 4.



4. Dviejų briaunų apkeitimo algoritmas.

Pavyzdžiui, pradinių algoritmų buvo pasirinktas artimiausio kaimyno algoritmas, kuris sudarė kelionę, kuri yra pažymėta mėlyna spalva. Geltonos briaunos yra 2 atsitiktiniu būdu pasirinktos briaunos, matoma kad viršūnių briaunos yra apjungtos ne efektyviai. Dešiniajame paveiksliuke matoma, kaip viršūnės buvo perjungtos, taip sumažinant bendrą kelionės atstumą.

2.1.2.5 „K“ – briaunų apkeitimo algoritmas

Algoritmo veikimo principas yra lygiai toks pats, kaip ir dviejų briaunų, tik yra pasirenkamos ne 2, o „k“ briaunų, bandant jas apjunginėti skirtingais būdais, taip mažinant atstumą. Skaičiaus „k“ didinimas, taip didina kelionės tikslumą, bet tai yra brangi operacija, kadangi artinant skaičių prie turimų taškų skaičius yra artėjamas prie pradinio tikslaus algoritmo – visų įmanomų kombinacijų perrinkimo. Dažniausiai praktikoje yra naudojamas dviejų arba trijų briaunų apkeitimo principas.

2.1.2.6 Lin – Kernigham algoritmas

Algoritmas yra paremtas ankščiau išvardintų „k“ – briaunų apjungimo principu. Vienintelis skirtumas yra tas, kad „Lin – Kernigham“ algoritmas adaptyvus, ir kiekviename žingsnyje yra parenkamas vis kitas skaičius „k“ [LiK73]. Šis algoritmas laikomas vienu iš efektyviausių euristinių algoritmų, su gana maža paklaida ir trumpu vykdymo laiku.

2.1.3 Genetiniai algoritmai

Genetiniai algoritmai yra ganėtinai naujas sprendimo būdas sprendžiant MTSP ir TSP uždavinius. Genetiniai algoritmai iš esmės nėra skirti spręsti uždavinių su dideliais apribojimais, apie kuriuos buvo kalbama įžangoje (svorio arba laiko langų). Jie dažniausiai taikomi rasti sprendimams, kuriems yra svarbus rūšiavimas, mūsų atveju tai yra tvarka, kuria aplankomi taškai MTSP ir TSP uždaviniuose. Pagrindinis genetinio algoritmo veikimo principas yra generuoti populiacijas iš numeruotų vektorių, kuriuos toliau vadinsime chromosomom. Kiekviena chromosoma vaizduos galimą sprendimą. Atskiri chromosomos komponentai, skaitinės vertės yra vadinamos genais. Nauja chromosoma yra sukuriama chaotiškai ar tikimybių pagalba apkeičiant reikšmes vektoriuose. Taip apkeičiant reikšmes ir kuriant vis naujas chromosomas yra padidinama paieškos sritis ir sumažinama tikimybė, kad iš anksto konverguos prie lokalaus optimalaus sprendimo. Kitame žingsnyje chromosomos yra įvertinamos pagal tam tikrą kainos funkciją, ir blogiausią rezultatą parodžiusios chromosomos yra eliminuojamos iš imties. Rezultatas – gaunamos vis naujos imtys chromosomų su geresniais rezultatais, o paiešką galima tęsti iki kol visos įmanomos kombinacijos bus pabandytos. Tačiau dažniausiai yra naudojami tam tikri iš anksto sugalvoti saugikliai, kaip pavyzdžiui: iš anksto nustatyta kainos funkcijos reikšmė yra pasiekiama, pasiektas tam tikras iteracijų skaičius, arba kitas mūsų sugalvotas saugiklis yra pasiekiamas [Tan00]. Pagrindinis tikslas, kuriant genetinius algoritmus, – kaip sukurti chromosomos struktūrą taip, kad būtų kuo mažiau pasikartojančių chromosomų. Pasikartojančios chromosomos žymiai padidina paieškos aibę, taip kartu padidindamos ir laiką, reikalingą mūsų užsibrėžtam atsakymui rasti. Gerai suprojektuota chromosoma neturi turėti pasikartojimo galimybės, taip kad iteracinis evoliucinis procesas, vykdantis paiešką, kuo greičiau pasiektų tikslą. Projektuojant chromosomą reikėtų atkreipti dėmesį ir į tai, kaip sudėtinga yra paskaičiuoti kainos funkciją chromosomai, kadangi skaičiavimas atliekamas didelį skaičių kartų, kad būtų galima nustatyti, kuri iš chromosomų turi pasilikti populiacijoje. Kuriant genetinį algoritmą nereikia pamiršti, kokiu būdu mes kursime naujas chromosomas, tai yra kaip vystysis mūsų populiacija. MTSP ir TSP uždaviniams jau yra sukurtas nemažas skaičius metodikų (algoritmų), kaip tai galima atlikti. Vieni dažniausiai naudojamų yra [Bja04]:

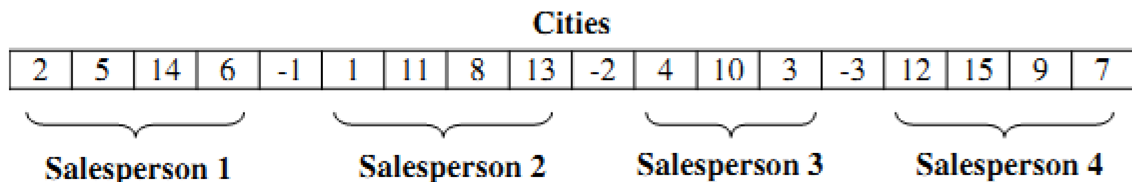
1. Rūšiuotas kryžminis apkeitimas. (Goldberg 1989, Davis 1985, Stark-weather 1991, Oliver 1987).
2. Dalinai pažymėtas kryžminis apkeitimas. (Goldberg 1989, Starkweatheret 1991, Goldberg and Lingle, 1985, Oliveret 1987).
3. Ciklinis kryžminis apkeitimas. (Goldber 1989, Starkweather 1991, Oliver 1987).
4. Hibridinių operatorių. (Liaw , 2000)
5. Paprastas kryžminis apkeitimas. (Knosala and Wal, 2001).
6. Rūšiuotas kryžminis apkeitimas 2. (Syswerda, 1989).
7. Moon kryžminis apkeitimas. (Moon , 2002).

Dažniausiai sprendžiant kelių pirklių uždavinį yra naudojama dviejų chromosomų technika. Apžvelgsime 3 technikas, kurių pagalba yra sprendžiamas MTSP uždavinys:

1. Vienos chromosomos technika.
2. Dviejų chromosomų technika.
3. Dviejų dalių chromosomos technika.

2.1.3.1 Vienos chromosomos technika

Pavyzdyje bus naudojama 15 taškų, kuriuos reikia aplankyti ir 4 pirkliai, kuriems reikia apkeliauti visus taškus. Vienos chromosomos techniką iliustruoja paveiksliukas 5 [CaR05].



5. Vienos chromosomos technikos pavaizdavimas.

Kaip matome iš 5 paveiksliuko, chromosoma turi n + m – 1 ilgį, kuris mūsų pavyzdžio atveju yra 18. Kur 15 aplankytinų taškų yra sudėlioti atsitiktine tvarka chromosomoje ir sunumeruoti nuo 1 iki „n“. Visa chromosoma yra suskirstytas i „m“ dalių įdedant į chromosomą „m“ neigiamų skaičių, kurių kiekvienas skaido keliones tarp pirklių. Pirkliai sužymėti pagal -(m) taisyklę. Sudarant bet kokią permaišą tarp šitų skaičių gaunamas vis skirtingas atsakymas, todėl iš viso skirtingų kombinacijų gaunasi (n+m-1)!. Iliustracijoje nr. 5, pateiktoje kaip pavyzdys, yra matomos 4 kelionės. Pirmam pirkliui taškai: 2, 5, 14, 6 antram: 1, 11, 8, 13 ir taip toliau. Kaip matome, toks sprendimo būdas nėra efektyvus, kadangi gaunama daug dublikatų, pavyzdžiui: apkeičiant pirmus penkis genus, su kitais

penkiais gaunamas lygiai toks pats bendras rezultatas, ir mes vykdysime daug nereikalingų ciklų skaičiavimų. Taigi vienos chromosomos technika nėra labai efektyvi, ir ji beveik identiška tiksliųjų algoritmų naudojimui. Vienos chromosomos techniką 2000 metais pasiūlė (Tangas).

2.1.3.2 Dviejų chromosomų technika

Kaip ir praeitame pavyzdyje, naudosime 15 taškų, kuriuos reikia aplankyti, ir 4 pirklius jiems aplankyti. Dviejų chromosomų techniką pasiūlė (Malmborgas) 1996 metais, bet 2001 metais (Parkas) ją patobulino ir dabar ji yra naudojama. Kad geriau suprasti, kaip atrodo dviejų chromosomų technika, pateikiamas paveiksliukas 6 [CaR05].

Cities

2	5	14	6	1	11	8	13	4	10	3	12	15	9	7

Salespersons

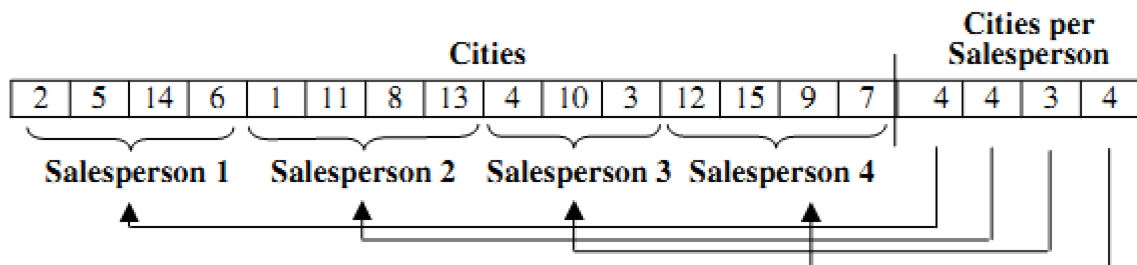
2	1	1	3	4	3	2	4	4	1	3	2	1	2	3

6. Dviejų chromosomų technikos pavaizdavimas.

Sprendimui yra sudaromos dvi chromosomos, kurių kiekvienos ilgis yra „n“, kur „n“ yra aplankytinų taškų skaičius, o „m“ – pirklių skaičius. Pirmoje chromosomoje matome kombinacijas aplankytinų taškų, o kitoje yra kiekvienam taškui priskirtas pirklys. Kiekvienas pirklys turi savo eilės numerį ir savo poziciją, pagal kurią ir yra gaunamos kelionės kiekvienam pirkliui. Pirmas pirklys apkeliauja: 5, 14, 10, 15 taškus, sekantis: 2, 8, 12, 9 tašką ir taip toliau. Naudojant dviejų chromosomų techniką egzistuoja $n!\, m^n$ įmanomų kombinacijų. Tai yra žymiai mažiau už vienos chromosomos technikos siūlomą sprendimo būdą, taigi dviejų chromosomų technika yra daug geresnė, bet ir ji nėra tobula, kadangi vis dar pastebima, kad egzistuoja pasikartojančios kombinacijos, pavyzdžiui: apkeitus kiekvienos chromosomos du pirmus genus vietomis, rezultatas nepasikeis, bet jau yra sumažinamas skaičius pasikartojamų, kurių nepavyko išvengti naudojant tik vieną chromosomą, bet bendru atveju skaičiavimų padaugėjo, kadangi jau naudojama ne viena, o dvi chromosomos.

2.1.3.3 Dviejų dalių chromosomos technika

Spręsime MTSP kur n=15, ir m=4. Uždavinys yra panašus į (Falkenauerio) spręstą 1998, grupuojant vieną chromosomą. Kaip atrodo tokiu būdu sugrupuota chromosoma yra parodyta paveikslėlyje 7 [CaR05].



7. Dviejų dalių chromosomos technikos pavaizdavimas.

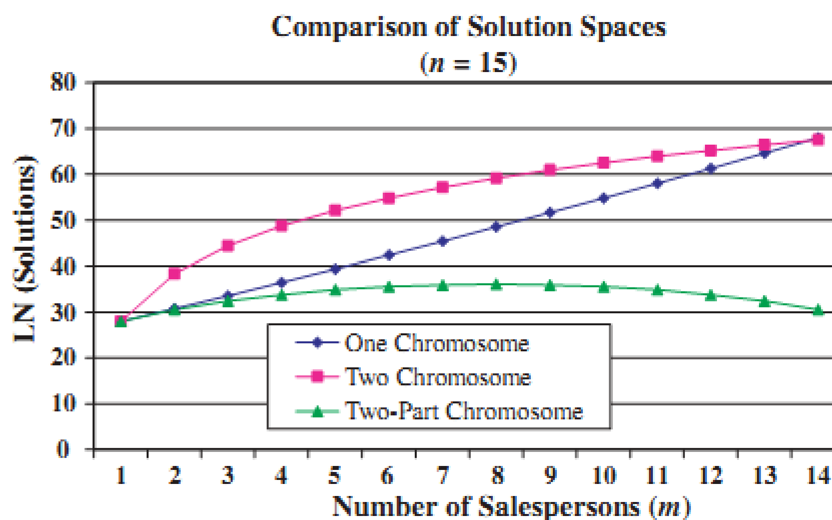
Pirmoje chromosomos pusėje yra sudėlioti „n“ aplankytinų taškų, mūsų pavyzdžio atveju tai yra 15 taškų. Kitoje chromosomos dalyje yra „m“ genų, mūsų pavyzdyje m = 4, kurių kiekvienas nurodo kiekvieno pirklio aplankomų taškų skaičių, visų jų suma turi būti lygi „n“. Pagal 7 paveikslėlyje pateiktą pavyzdį pirmas pirklys turi apkeliauti 4 taškus, todėl pradedama juos skaičiuoti nuo chromosomos pradžios, taigi pirmo pirklio kelionė gaunasi: 2, 5, 14, 6. Kitame žingsnyje imamas antro pirklio aplankomų taškų skaičius ir skaičiuojama nuo taško, kuriame pabaigė skaičiuoti pirmo pirklio kelionę, todėl antrojo pirklio kelionė yra: 1, 11, 8, 13. Procesas yra tęsiamas taip ir toliau, kol visi taškai priskiriami visiems pirkliams. Toks sprendimo būdas žymiai sumažina pasikartojimų galimybę, bet dar palieka neišspręstus kelis klausimus. Pasikartojimas gali atsirasti, nes kelionės kiekvienam pirkliui eina iš eilės, ir jos gali persideginėti, nesudarydamos optimalios kelionės, bet paieškos aibė yra sumažinama labai ženkliai, lyginant su prieš tai buvusiais sprendimo būdais. Dviejų dalių chromosomos technikos taikymas turi sudėtingumą: $n!\binom{n-1}{m-1}$. Pirma chromosomos dalis turi (n!) kombinacijų apkeisti „n“ aplankytinų taškų, antroje chromosomos dalyje turime vektorių sveikų skaičių $(x_1, x_2, x_3 \dots x_m)$ kurių suma turi būti lygi „n“. Egzistuoja tik $\binom{n-1}{m-1}$ kombinacijų, kurių pagalba galima gauti skaičių „n“ keičiant vektorius „x“.

2.1.3.4 Chromosomų sudarymo technikų palyginimas

Lyginant TSP rezultatus yra svarbu atsižvelgti į algoritmo tikslumą ir algoritmo veikimo greitį. Palyginant MTSP uždavinio rezultatus labai svarbu palyginti, koks yra gaunamas rezultatas keičiant pirklių skaičių „m“, kiek padidėja kelionė nuo optimalios rastos

TSP algoritmo. Lyginant chromosomų sudarymo technikas svarbu pažymėti tai, kad kai pirklių skaičius yra lygus vienetui, yra sprendžiamas paprastas TSP uždavinys ir visos mūsų ankščiau išvardintos chromosomų sudarymo technikos sprendžia ekvivalentiškai, kadangi perrenka visas įmanomas kombinacijas iš „n“ aplankytinų taškų – tai yra sprendžia uždavinį su sudėtingumu O(n!). Verta pažymėti, kad didinant pirklių skaičių „m“ prie , kuriuos turim aplankyti „n“, suminė visų pirklių kelionė irgi labai ženkliai išaugs, kadangi kiekvienas pirklys pradeda ir pabaigia savo kelią tame pačiame pradiniame taške.

Tikrinant chromosomų sudarymo technikas, baziniai genetinio algoritmo rodikliai ir perrinkimo funkcijos buvo paliktos tokios pačios, buvo keičiamas tik pats chromosomos sudarymo būdas. Kaip perrinkimo metodas buvo parinktas paprastas kryžminis apkeitimas, chromosomų populiacijos dydis 100, ir tikimybė pasikeisti 5%, kadangi tokie rodikliai buvo dažniausiai sutinkami anksčiau atliktuose darbuose kaip optimaliausi. Paveikslėlyje numeris 8 [CaR05] pateiktas grafikas apie tai, kokia yra paieškos erdvė naudojant skirtingas chromosomų sudarymo technikas.



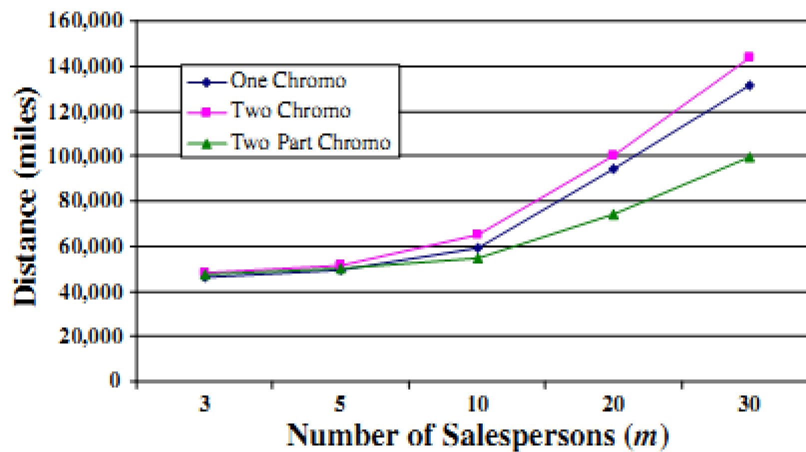
8. Chromosomų sudarymų technikų, paieškos aibių palyginimai.

Kaip ir buvo kalbama ankščiau, vienos chromosomos technikos paieškos aibės dydis auga eksponentiškai didėjant pirklių skaičiui, kadangi ji turi tiesinę priklausomybę nuo šito parametro.

Dviejų chromosomų technika turi logaritminę priklausomybę, taigi jos grafikas yra aukštesnis už visų kitų, o dviejų dalių chromosomos grafikas parodo tikrai įspūdingus rezultatus. Taip sudarydami chromosomą mes ženkliai galime sumažinti paieškos aibę.

Sekantis bandymas buvo padarytas bandant nustatyti kokia technika sudaryta chromosoma suras tiksliausią sprendimo būdą. Bandymas buvo atliekamas parinkus 150

taškų, kuriuos reikia aplankyti, ir didinant pirklių skaičių nuo 3 iki 30. Rezultatai pateikti paveikslėlyje numeris 9 [CaR05].



9. Suminės kelionės grafikas, naudojant skirtingas chromosomų sudarymo technikas.

Iš rezultatų puikiai matosi, kad dviejų dalių chromosomos sudarymo technika ženkliai lenkia kitas, to ir galima buvo tikėtis, kadangi ji turi mažiausią paieškos aibę ir gali greičiau rasti tikslesnį atsakymą. Dviejų chromosomų technika turėdama didžiausią paieškos aibę lieka paskutinėje vietoje. Verta pažymėti, kad visos technikos, su mažu skaičiumi pirklių, surasdavo labai panašias keliones.

2.1.4 Neuroniniai tinklai

Vystantis naujoms technologijoms, dirbtinis intelektas užima vis svarbesnę vietą optimizavimo uždaviniuose. Optimizavimo uždaviniuose yra naudojami genetiniai algoritmai, „Tabu“ paieška, modeliuojamo atkaitinimo ir aišku pasinaudojant neuroniniais tinklais. Algoritmai paremti neuroniniais tinklais yra save užsirekomendavę kombinatorikos ir tikimybių teorijos uždavinių sprendimuose. Neuroniniai tinklai yra pritaikomi dideliam kiekiui uždavinių, mokslininkai kasmet atranda vis naujų pritaikymo sričių. Neuroninio tinklo pagalba sukurtas algoritmas moka pats mokytis, taip pagreitindamas rezultato radimą, ir laukiamo rezultato tikslumą, analizuojant pradinius duomenis. Vienas iš pliusų naudoti neuroninius tinklus yra tai, kad jie iš esmės pasižymi lygiagretiškumu, kas yra labai svarbu šiais laikais, kai kompiuterių aparatūriniai mazgai įgauna vis daugiau lygiagretiškumo. Kaip pavyzdį galime pastebėti tai, kaip per kelis metus iš vieno branduolio procesoriaus peraugome į erą, kai 4 branduoliai jau yra mažai, o naujoviški serveriai operuoja su 8 ir daugiau skaičiavimo branduoliais. Pirmas naudoti neuroninius tinklus optimizavimo uždaviniams spręsti pasiūlė (Hopfieldas) ir (Tankas) dar 1985 metais. Jie pasiūlė idėją su

neuroninių tinklų ir aktyvavimo funkcija taip, kad tinklas konverguotu prie optimalaus atsakymo. Jų pasiūlytame variante tinklas pasiima reikšmę iš aktyvavimo funkcijos ir visą laiką ją mažina iki tol, kol yra pasiekiamas lokalus minimumo taškas. Kad pritaikyti tokį modelį mūsų nagrinėjamam uždaviniui buvo į aktyvavimo funkciją papildomai įvesta kainos funkcija. Aktyvavimo funkciją galima papildyti ir papildomais suvaržymais, jeigu tai yra būtina sprendžiamame uždavinyje, kaip pavyzdžiui, svorio apribojimas ir kito pobūdžio apribojimai. Toks neuroninio tinklo modelis buvo tobulinamas daugelio mokslininkų, bet dėl savo sudėtingumo nelabai konkurencingas, lyginant su naujausiais euristiniais algoritmais. Neseniai atsirado prisitaikantys neuroninių tinklų modeliai, kurie sugeba labai efektyviai mokytis iš pradinių duomenų, ir jie yra laikomi vieni perspektyviausių optimizavimo srityje. Tokie neuroninių tinklų modeliai gali būti skaidomi į grupes:

1. Elastingo tinklo.
2. Savarankiškai organizuojantis ypatybių žemėlapis.

Kiekvienas iš aukščiau išvardintų metodų yra labai panašus vienas į kitą, skirtumas pasireiškia tik tame, kaip kiekvienas iš jų daro atnaujinimo procedūrą su mazgais. Elastingo tinklo metodas gražina rezultatą atsižvelgdamas į visus mazgus, o savarankiškai organizuojantis žemėlapis atsižvelgia į palyginimo rezultatus. Elastingo tinklo ir savarankiškai organizuojančio žemėlapio metodai skiriasi nuo (Hopfieldo) ir (Tanko) 1985m. sukurto metodo labiausiai tuo, kad jų modelis naudojo iš anksto apibrėžtas taisykles, taip vadinamą konfigūraciją, o naujausi algoritmai startuoja neturėdami pradinių konfigūracijų, o tik operuoja su pradiniais duomenimis, ir patys sau veikimo eigoje nusistato konfigūraciją.

2.1.4.1 Elastingo tinklo modelis

Elastingo tinklo modelį pirmas pasiūlė (Durbin) ir (Willshaw) 1987 metais. Metodas yra labiau geometrinis, ir jį galima apibrėžti kaip rinkinį taisyklių, kurių pagalba grubi geometrinė linija yra apibrėžta aplinkui visus aplankytinus „n“ taškus. Algoritmais pradžioje apibrėžę apskritimą su M mazgų, kur M = 2,5n, kur „n“ aplankytinų taškų skaičius, toks apskritimas apibrėžia visus taškus, nes jo centras yra patalpintas centre visų turimų taškų. Naudojant iteracinį metodą, pradinis apskritimas, kuris yra laikomas pirmine kelione, yra įtraukiamas taip, kad kiekvienas taškas, kuris turi būti aplankytas, atsidurs kelionėje. Kiekvienos iteracijos metu, kelionės pritraukimo prie taško sąlygoje yra du veiksmai. Pirmas

traukia kelionę prie artimiausio taško, o antra sąlyga bando pritraukti tašką prie artimiausio kaimyninio taško. Mazgo poslinkis yra nustatomas pagal formulę [MSE99] :

$$\Delta Y_j = \alpha \sum w_{ij}(X_i - Y_j) + \beta K(Y_{j+1} - 2Y_j + Y_{j-1}).$$

Koeficientas W_{ij} yra skaičiuojamas pagal formulę [MSE99] :

$$w_{ij} = \frac{\phi(d_{X_i Y_j}, K)}{\sum_k \phi(d_{X_i Y_K}, K)}.$$

Kur $d_{X_i Y_j}$, yra atstumas tarp taško „i" ir mazgo „j". Funkcija $\phi(d, k)$, yra Gauso funkcija: $\exp(-d^2/2k^2)$, kur parametras „k" yra mažinamas kiekvienos iteracijos metu.

Toks neuroninio tinklo veikimo būdas užtikrina gerą rezultato radimą, bet kartu reikalauja nemažai skaičiavimo resursų, kas padaro ji nelabai efektyviu.

2.1.4.2 Savarankiškai organizuojantis ypatybių žemėlapis

Pagrindinė modelio idėja yra sudėlioti kuo arčiau panašius objektus, atsižvelgiant į jų pradinius atributus. Modelis kilo iš žinduolių smegenų sandaros, kur neuronai atsakingi už tam tikrus veiksmus yra sudėlioti arti vienas kito, lyg suskirstyti į grupes. Tokio modelio panaudojimas leidžia modeliuoti problemas, artimas realiam gyvenimui. Kad sukurti toki neuroninį tinklą, kiekvienas neuronas mokosi atskirai konkurencingai. Tokio tipo neuroniniai tinklai yra dažnai naudojami klasifikavimo ir klasterizavimo uždaviniuose, tai pat konkurencingas mokymasis leidžia mums pasieki optimalaus sprendimo radimą (Hertz) 1991 metai". Kiekvienas naujas objektas patekęs į tinklą, keliaudamas iš vieno neurono į kitą, randa sau geriausia, ir tokiu būdu sudaromas optimalus sprendimas. Kiekvienas naujas objektas, patekęs į tinklą, daro įtaką kiekvienam tinklo mazgui, kadangi atsižvelgiant į naują objektą yra keičiami ir mazgų, per kuriuos jis keliauja, svoriai. Veikimo pradžioje tinklas yra iniciuojamas su atsitiktiniais svoriais, o kiekvienos iteracijos metu tinklas vis keičia savo topologiją. Kai objektas yra priskiriamas kažkokiam vienam iš neuronų, jis daugiau nedaro jokios įtakos kitiems tinklo mazgams.

3. Uždavinio sprendimas genetinio algoritmo pagalba

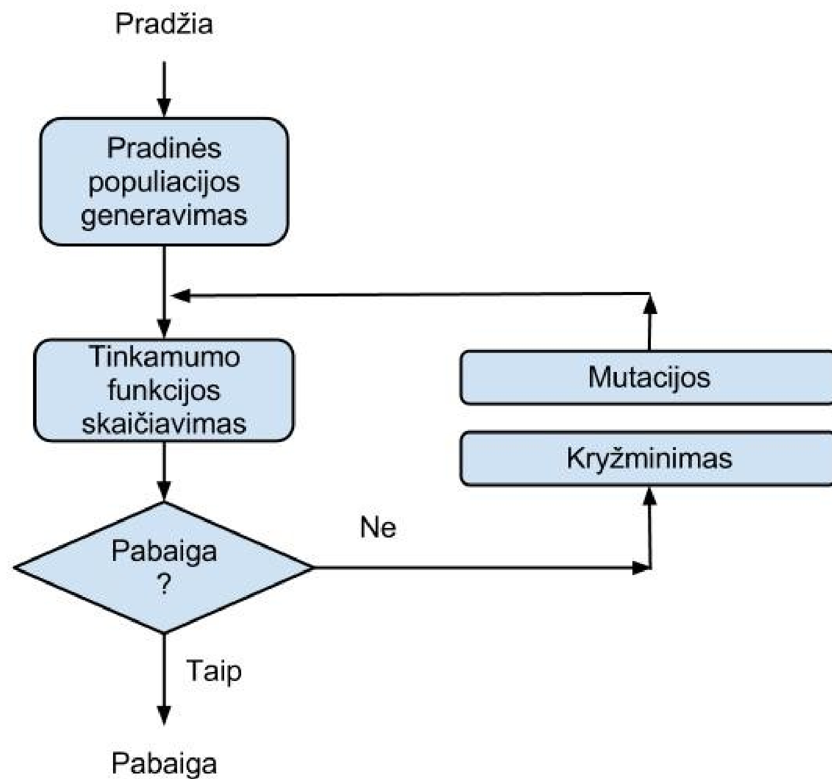
Pagrindiniam uždavinio sprendimui bus naudojamas genetinis algoritmas. Genetiniai algoritmai gerai sprendžia uždavinius kuriems yra svarbus rūšiavimas, uždavinyje svarbų yra atitinkamai išrūšiuoti pradinius taškus taip kad bendra kelionės kaina butu ko mažesnė. Uždavinio sprendimui naudoti genetinius algoritmus yra praktiška kadangi galima naudoti kitus euristinius algoritmus kaip jo operatorius. Algoritmas gali būti praktiškai be galo pildomas vis naujai sugalvotais euristiniais operatoriais ir taip gauti vis geresnius sprendimus. Naudojant genetinius algoritmus yra sprendžiama lokalaus minimumo problema. Naudojant genetinius algoritmus ne sudėtinga yra pridėti papildomų apribojimų prie uždavinio papildant tinkamumo funkciją, ir pritaikant genetinius operatorius prie jų.

3.1 Pagrindinė genetinio algoritmo schema

Genetiniai algoritmai yra pasiskolinti iš gamtos, ir jų veikimo principas yra lygiai toks pats kaip ir vyksta gamtoje. Vyksta ląstelių atranka, kryžminamasis, mutacijos ir populiacijos palaikymas. Populiacija negali plėstis iki begalybės – kai kurios iš chromosomų turi palikti ląstelę. Programuojant MTSP uždavinį genetinių algoritmų pagalba turi būti įgyvendintos tos pačios funkcijos, kurios vyksta gamtoje:

1. Sugalvoti efektyvią chromosomos architektūrą.
2. Algoritmo sustojimo sąlygos parinkimas.
3. Sudaryti pradinę populiaciją.
4. Efektyvumo funkcija.
5. Lokalus optimizavimas.
6. Genetinius operatorius, kurie atliks:
    6.1. Chromosomos pasirinkimą.
    6.2. Chromosomų kryžminimą.
    6.3. Chromosomos mutaciją.

Bendras genetinio algoritmo pseudo kodas yra pateiktas 10 paveiksliuke.



10. Bendra genetinio algoritmo schema.

3.2 Chromosomos sandara

Pagrindinis tikslas kuriant genetinius algoritmus – kaip sukurti chromosomos struktūrą taip, kad būtų kuo mažiau pasikartojančių chromosomų. Pasikartojančios chromosomos žymiai padidina paieškos aibę, taip kartu padidindamos ir laiką, reikalingą mūsų užsibrėžtam atsakymui rasti. Gerai suprojektuota chromosoma neturi turėti pasikartojimo galimybės, taip kad iteracinis evoliucinis procesas, vykdantis paiešką, kuo greičiau pasiektų tikslą. Darbe bus naudojamas 2.1.3.3 skyriuje aptartas dviejų dalių chromosomos modelis. Pasirinktas modelis turi mažiausią paieškos aibę tarp ankščiau nagrinėtų modelių.

3.3 Algoritmo veikimo nutraukimo sąlyga

Genetinio algoritmo veikimas gali būti praktiškai begalinis arba kol bus išbandytos visos įmanomos kombinacijos, o toks jo veikimo principas prilygtų grubiam perrinkimo algoritmui. Genetiniuose algoritmuose, kad taip neatsitiktų, visada yra sugalvojami specialūs saugikliai ir kai jie viršijami, algoritmo darbas sustabdomas ir pateikiamas geriausias rastas rezultatas. Algoritmui nereikia išbandyti visų įmanomų variantų, kad būtų galima pateikti ganėtinai tikslų atsakymą, kuris dažniausiai būna nutolęs nuo tikslaus žinomo sprendimo per kelis procentus, bet vykdymo laikas yra nepalyginimai trumpesnis, kai su tiksliais algoritmais

net ir po mėnesio veikimo galime nesulaukti rezultato. Aptarsime kelis dažniausiai naudojamus saugiklius, o vieną iš jų naudosime savo darbe.

3.3.1 Viršyto laiko saugiklis

Pradėjus atlikinėti skaičiavimo darbus yra užfiksuojamas laikas, ir turime iš pradžių apsibrėžę, kiek laiko gali vykdytis algoritmas. Kiekvienoje iteracijoje tikrinama, ar nėra viršytas leistinas laiko limitas. Viršijus skirtą laiką, algoritmas priverstinai stabdomas ir pateikiamas geriausias per tą laiką rastas rezultatas. Tokio saugiklio kaip vieną iš didžiausių minusų galime įžvelgti tai, kad jis negali pats spręsti, kiek jam laiko reikia, kad gauti priimtiną sprendinį. Laikas turi būti nustatomas žmogaus, priklausomai nuo kokio dydžio uždavinys yra sprendžiamas, arba gali būti paskaičiuojamas bendrą uždavinio dydį padauginus iš konstantos.

3.3.2 Viršyto iteracijų skaičiaus saugiklis

Algoritmas veikia iteracini principu, tai galime matyti paveikslėlyje numeris 1. Dažnai kaip sustojimo saugiklis yra naudojamas iteracijų skaičius. Toks sprendimas turi lygiai tokius pačius minusus kaip ir punkte 3.2.1 aprašytas laiko saugiklis, kadangi reikalauja, kad algoritmo pradžioje butu apsispręsta, kiek iteracijų leista jam įvykdyti.

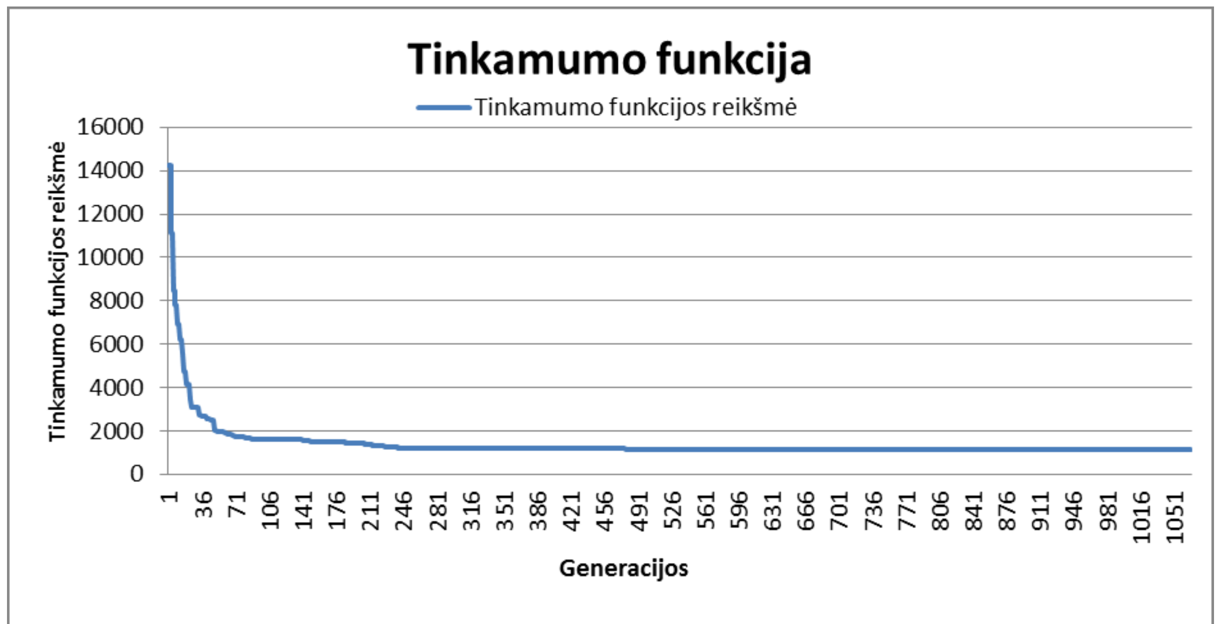
3.3.3 Tinkamumo funkcijos įvertinimas

Kaip sustojimo sąlygą galima apsibrėžti tam tikrą tinkamumo funkcijos reikšmę, ir kai vykdymo eigoje funkcija maždaug pasiekia apsibrėžtą skaičių, algoritmo veikla yra sustabdoma. Tokia sustojimo sąlyga – dažniausiai nelabai efektyvi, kadangi sunku iš anksto pasakyti koks maždaug turi gautis sprendimas, bet ji yra efektyvi tada, kai mes atliekame skaičiavimus daugelį kartų, o pradiniai duomenys skiriasi nežymiai, arba kai algoritmo paskirtis yra surasti iš anksto apibrėžtą minimumą.

3.3.4 Pasirinkta sustojimo sąlyga

Sprendžiamam uždaviniui kaip sustojimo sąlyga buvo pasirinktas kombinuotas saugiklis. Jis susidės iš iteracijų skaičiaus ir efektyvumo funkcijos generuojamo rezultato. Algoritmas vykdysis kol jo efektyvu funkcijos reikšmė netaps pastovi, o kai reikšmė pasieks savo pastovią reikšmę, tai yra generuojant visas naujas chromosomų populiacijas, nebus randamas geresnis sprendimas, o kai reikšmė nusistovės bus bandoma dar „n“ kartų gauti geresni rezultatą. Skaičius, kiek kartų dar vykdyti naujų sprendinių paiešką, algoritmo veikimo pradžioje bus mažesnis, o ilgėjant algoritmo veikimui – skaičius didės, jis bus

priklausomas nuo įvykdytų iteracijų skaičiaus. Toks sprendimas buvo priimtas dėl efektyvumo funkcijos kitimo grafiko, kuris yra pateiktas paveikslėlyje numeris 11.



11. Efektyvumo funkcijos priklausomybė nuo iteracijų skaičiaus.

Iš 11 paveikslėlyje pateikto grafiko gerai matosi, kad efektyvumo funkcija per pirmas iteracijas ženkliai mažėja, užsibūna pastovios reikšmės labai trumpą laiką ir sparčiai mažėja, o iteracijų skaičiui padidėjus lieka ilgesnį laiką pastovi, dėl to ir iteracijų skaičius reikalingas sustoti bus didinamas, kad negauti per anksti nevisiškai optimalaus sprendimo.

3.4 Pradinės populiacijos sudarymas

Projektuojant genetinį algoritmą reikia nepamiršti apie pradinės populiacijos sudarymo procedūrą. Jos pagalba yra gaunami pradiniai populiacijos individai, nuo jų priklauso kaip greitai bus konverguojama prie optimalaus sprendimo, kadangi iš pradinių chromosomų bus sudaromi nauji palikuoniai ir atliekamos mutacijos. Reikia nepamiršti, kad negalima sudaryti idealios pradinės populiacijos, kadangi tai privestų algoritmą prie optimalaus minimumo ir yra labai didelė tikimybė, kad tai nebus optimalus sprendinys.

3.4.1 Atsitiktinis populiacijos sudarymas

Populiaciją galima sudaryti atsitiktinai išdėliojant genus chromosomoje. Kiekviena chromosoma turės stochastiškai išdėstytus miestus ir pasitaikys viršytu leistiną efektyvumo funkcijos ribą, taip gauta populiacija bus labai įvairi, turės kaip ir labai prastų taip ir labai gerų pradinių palikuonių. Galima uždėti papildomą saugiklį, kad palikuoniai, kurie viršijo nustatytos efektyvumo funkcijos ribą, nepatektų į galutinę populiaciją, bet taip kartu bus

sumažinta paieškos sritis, kadangi tokia viena chromosoma po vienos mutacijos arba kryžminimo operacijos gali tapti dominuojanti.

3.4.2 Populiacijos sudarymas kitais grubiais algoritmais

Vienas iš efektyviausių populiacijos sudarymo būdų yra pasinaudoti kitais euristiniais algoritmais ir taip gauta populiacija bus daug artimesnė galutiniam sprendiniui. Kad sudaryti pradinę populiaciją MTSP arba VRP uždaviniui mes turime sukombinuoti kelis euristinius algoritmus. Aptarsime vieną iš efektyviausių, kai miestai keliautojams yra priskiriami pasinaudojant k-centrų metodų ankščiau aprašytame skyriuje 2.1.2.1 , o jau patys miestai kelionėje yra išrikiuojami pasinaudojant paprasčiausiu artimiausio kaimyno algoritmu 2.1.2.2 skyrius.

3.4.3 Pasirinktas pradinės populiacijos generavimo modelis

Šiame skyriuje nagrinėti 2 populiacijos generavimo modeliai yra savotiškai labai patogūs sugeneruoti pradinę populiaciją, kadangi maišydami miestus atsitiktine tvarka yra gaunami iš pradžių nelabai optimalūs sprendimai, bet tokių chromosomų egzistavimas leidžia mums nepapulti i lokalaus minimumo tašką, kadangi iš pradžių labai netinkamu atrodžiusi chromosoma po kelių kryžminimų arba mutacijų gali priartėti prie optimalaus sprendimo. Gerai sutvarkytos pradinės chromosomos tai pat padeda labai greitai judėti link optimalaus sprendimo, sukryžminus kelias geriausias chromosomas arba atlikus mutacijas galima greitai pasiekti optimalų sprendimą. Kad algoritmas būtų kuo labiau panašus į gamtoje egzistuojantį analogą, kuriame yra vien geriausios chromosomos, o kartu ir labai prastos chromosomos, kurios padeda nepapulti į lokalius minimumo taškus, nuspręsta pradinę populiaciją sudarinėti abiem ankščiau aprašytais būdais, į pradinę populiaciją papuls ir stochastiškai sugeneruotos chromosomos ir pasinaudojant kitais euristiniais algoritmais sudarytos chromosomos. Tvarkingai sudarytų chromosomų bus daugiau, jos sudarys 65% pradinės populiacijos, o likusius 35% sudarys stochastiškai sugeneruoti individai. Tai yra pradiniai autoriaus sugalvoti suskirstymo procentais koeficientai.

3.4 Tinkamumo funkcija

Tam, kad galėtume atrinkti chromosomas kryžminimui yra būtina jas kažkaip įvertinti. Kiekviena chromosoma turi turėti reikšmę, nurodančią jos tinkamumą, kaip jos saugomas sprendinys atrodo prieš kitus individus. Tai leidžia mums palyginti individus tarpusavyje. Paprasčiausia tinkamumo funkcija yra parodyta 12 paveikslėlyje.

$$f_s = \sum_r cost_{s,r}.$$

12. Tinkamumo funkcija.

Aukščiau pateiktame paveikslėlyje sprendimo tinkamumas yra skaičiuojamas paprasčiausiai sumuojant kiekvienos keliones „r“ sprendime „s“ kainas. Kaina „$cost_{s,r}$“ yra skaičiuojama sumuojant atstumus tarp pristatymo taškų. Panaudojant taip suprojektuotą tinkamumo funkciją susiduriama su pirklio svorio viršijimo problema, kadangi ji visai neatsižvelgia į tai, ar buvo viršyta pirklio keliamoji galia. Tam, kad tokie variantai nepapultų į galutinį atsakymą, reikia į efektyvumo funkciją pridėti saugiklį ir kai jis yra viršijamas, pridėti papildomą skaičių, kuris priklausytų ant kiek sprendime kiekviename rastame kelyje yra viršytas svoris pakeltas kvadratu. Paveiksliuke numeris 13 yra pateikta papildomo pridėjimo sąlyga, kuri skaičiuoja, kiek reikia pridėti papildomai prie pagrindinės tinkamumo funkcijos, kur „$totdem_{s,r}$“ yra kelionės „r“ sprendime „s“ užkrovimas, o „cap“ žymi kokia yra pirklio keliamoji galia.

$$\sum_{r \in s}(\max(0, totdem_{s,r} - \mathrm{cap}))^2.$$

13. Papildomo svorio pridėjimas kai viršyta leistina keliamoji pirklio galia.

Panaudojus papildomo pridėjimo sąlygą, mes gauname, kad funkcija visada gražina reikšmę, net jei sprendinys nepapuola į mus dominančių sprendinių aibę. Tai leidžia mums atlikti tokių chromosomų kryžminimus su daug geresniais individais ir taip gauti vis naujus sprendinius. Taip pat leidžia mums spręsti lokalaus minimumo problemą, kadangi nėra operuojama visada tik su geriausiais, o kartais yra pasirenkama iš prastesnių, o tokia efektyvumo funkcija leidžia neatmesti sprendimų, kurie iš pradžių atrodo labai netinkami.

3.5 Lokalus optimizavimas

Atliekant genetines operacijas su individu, tokias kaip kryžminimas arba mutacija, genų išsirikiavimas chromosomoje labai pasikeičia. Kas nulemia, kad bendra kelionė labai pailgėja, kadangi genai yra stochastiškai išsimetę. Norėdami, kad algoritmas kuo greičiau konverguotu prie optimalaus sprendimo, turime išrikiuoti genus taip, kad jie sudarytų kuo trumpiausią kelionę kiekvienam pirkliui. Lokali chromosomos optimizacija bus vykdoma tik vienai kelionei. Vykdant optimizaciją vienai kelionei yra sumažinamas laikas, reikalingas optimizuoti kelionei. Kiekvienos kelionės optimizacijai bus naudojami paprastesni euristiniai algoritmai. Pirmiausia kelionė bus optimizuota 2.1.2.2 skyriuje apžvelgtu artimiausio

kaimyno algoritmu, o sekančiame žingsnyje bus pritaikytas dviejų briaunų apkeitimo algoritmas, kuris pagerins keliones. Dviejų briaunų apkeitimo algoritmas buvo ankščiau aptartas skyriuje 2.1.2.4.

Lokaliai paieškai galima taikyti ir daugiau briaunų palyginimo ir apkeitimo operatorių, bet tai labai sulėtina algoritmo veikimą. Apkeitimo skaičiui artėjant prie turimų taškų, skaičiaus algoritmas tampa tikslusis, kadangi bus atliktos visos įmanomos operacijos, kas ir yra tikslaus algoritmo pagrindas – perrinkti visas įmanomas kombinacijas.

3.6 Genetiniai operatoriai

Genetiniai operatoriai sudaro branduolį kiekvieno genetinio algoritmo, kadangi jų pagalba yra reguliuojama populiacija, populiacijos vystymasis. Kiekvienas genetinis algoritmas turi įgyvendinti 3 pagrindinius operatorius:

1. Pasirikimo
2. Kryžminimo
3. Mutacijos

3.6.1 Pasirinkimo operatorius

Pasirinkimo operatorius genetiniuose algoritmuose labai tampriai susijęs su populiacijos įvairove. Jo pagalba yra išrenkami individai, kurie formuos naują chromosomą. Jei išrinkimo funkcija bus labai primityvi, tai prives prie populiacijos homogeniškumo. Tokiu atveju tik mutacijų pagalba bus įmanoma sukurti unikalią naują chromosomą ir nepatekti į lokalų minimumą, kas yra labai svarbu bet kokiame algoritme. Pasirikimo operatorius atsako už populiacijos išlaikymą. Chromosomų populiacija negali tik didėti, ji visada turi būti arba apsibrėžto pradžioje dydžio arba kisti priklausomai nuo atliktų iteracijų skaičiaus. Operatorius turi spręsti, ar naujai sukurtas individas turi papulti į populiaciją, ar turi būti iškart sunaikintas, ar kažkoks kitas silpnesnis arba stipresnis individas turi palikti populiaciją.

3.6.1.1 Reitingavimo metodas

Tai yra pats paprasčiausias individų išrinkimo būdas. Žinant kiekvienos chromosomos efektyvumo funkcijos reikšmę, galime išreitinguoti pagal tai, kiek efektyvumo funkcijos rezultatas yra geras. Iš taip išrūšiuotų individų atrenkam pačius geriausius kryžminimui. Įmanoma pasirinkinėti ne būtinai tik geriausius, nes tokiu atveju populiacija greitai taptų homogeniška, bet pasirinkinėti tarp gero ir prastesnį rezultatą turinčio individo. Toks sprendimo metodas nėra labai geras, kadangi dažniausiai vis tiek bus operuojama tik su geriausiomis chromosomom.

3.6.1.2 Ruletės metodas

Metodas yra labai panašus į ankščiau aprašytą reitingavimo metodą. Jo sudarymui mums reikės kiekvieno individo efektyvumo funkcijos reikšmės. Kiekvieno individo reikšmė sudarys tikimybę, su kokia individas gali būti pasiriktas. Geriausią rezultatą parodžiusi chromosoma turės didžiausią tikimybe būti išrinkta kryžminimui, bet nebūtinai bus išrinkta. Chromosoma pagal jos tikimybe būti išrinktai stochastiniu būdu yra išrenkama iš populiacijos, bet kiekvieno išrinkimo operacijos metu yra galimybė ir mažiausią reikšmę turinčiai chromosomai būti išrinktai. Toks išrinkimo metodas padidina populiacijos įvairovę.

3.6.1.3 Turnyro metodas

Pradžioje visos chromosomos yra suskirstomos į pasirinktą skaičių grupių. Tai, kad į kiekvieną grupę patektų po lygiai individų ir kad jų kiekvienoje grupėje būtų kaip gerų taip ir blogesnį rezultatą parodžiusių individų. Geriausiu atveju jų vidurkis turi būti labai panašus. Sekančiame žingsnyje iš kiekvienos grupės pasinaudojant ruletės metodu yra išrenkama po vieną individą. O paskutiniame žingsnyje iš kiekvienos grupės išrinktas individas konkuruoja su kitais išrinktais individais ir geriausią rezultatą parodęs individas būna atrinktas ir išsaugomas. Išrinkti individai yra pašalinami iš grupių. Ciklas kartojamas tiek kartų, kiek individų mums reikia operacijoms atlikti, mūsų atveju tai bus 2 individai, kadangi kryžminimas bus atliekamas tik su 2.

3.6.1.4 Pasirinktas išrinkimo metodas

Darbe bus naudojamas turnyro metodas. Turnyro metodas geriausiai tinka MTSP uždavinio sprendimui kadangi mažinant arba didinant turnyro dydi galima keisti tikimybe kad blogesni arba geresni individai butu išrinkti tolimesnėms operacijoms.

3.6.2 Kryžminimo operatorius

Darbe nagrinėsime chromosomų kryžminimo operatorius, kuriems reikia dviejų tėvinių chromosomų, iš kurių bus gaunamas naujas vaikinis individas. Vienas iš tėvinių individų visada turės geresnę efektyvumo funkcijos reikšmę.

3.6.2.1 Paprastas atsitiktinis kryžminimas

Pasirinkus 2 tėvines chromosomas yra išrenkama viena, kuri yra geriausia. Jos visi genai nukopijuojami į vaikinę chromosomą. Atsitiktinai pasirenkamas genas iš antrosios chromosomos ir nukopijuojamas į vaikinę chromosomą. Taip operacija kartojama apsibrėžtą iteracijų skaičių. Toks kryžminimo operatorius yra labai primityvus, kadangi jis neatsižvelgia

į mūsų uždavinį, jis gali būti taikomas bet kokiame genetiniame algoritme. Atsitiktinį kryžminimą MTSP uždavinių galima pritaikyti atsižvelgiant į tai kad mes saugome atskiras keliones kiekvienam pirkliui, mes galime kopijuoti ne po vieną geną, o atsitiktine tvarka pasirinkę tam tikrą maršrutą vienam pirkliui ir nukopijuoti jį visą. Darbe bus naudojamas ne pilnai atsitiktinis kryžminimas, o atsižvelgiant į ta, kokį maršrutą ji sudaro.

3.6.3 Mutacijos operatorius

Mutacijos į bendrą algoritmo veikimą įneša sprendimų, kurių nesugeneruoja nei pradinė populiacija, nei kryžminimas. Mutacijų pagalba yra gaunami labai įvairūs sprendimai. Kad mutacijų operacijos atneštų į populiaciją įvairesnių sprendimų, jos bus įgyvendintos paprasčiausiu atsitiktiniu būdu. Individas mutacijai irgi yra pasirenkamas atsitiktine tvarka. Pasirinkus individą, kuriam bus atliekama mutacijos operacija, jo keli genai yra apkeičiami atsitiktine tvarka ir taip gaunamas naujas genas patalpinamas į populiaciją.

4. Genetinių algoritmų tyrimas panaudojant "Symphony" duomenų rinkinį

Įgyvendinant 3 skyriuje aprašytą genetinį algoritmą buvo pasirinkta „Java“ programavimo kalba. Papildomai buvo naudojamos kelios viešai prieinamos bibliotekos: „log4j“ kaupti sisteminius įrašus, „watchmaker“ genetinis karkasas ir „uncommons-maths“ biblioteka matematinėms skaičiavimams. Pasinaudojant „watchmaker“ karkasu įgyvendinimas tapo lengvesnis, kadangi nereikėjo rūpintis pagrindine algoritmo veikimo schema, jis turi įgyvendintas kelias genetinių algoritmų schemas. Buvo pasirinktas tradicinis evoliucionuojantis modelis. Naudojant karkasą yra vienas didelis privalumas, kad galima susikaupti ties sprendžiama užduotimi, o ne bendra algoritmo veikimo schema.

4.1 Pradiniai duomenys

Pradiniai duomenys, su kuriais buvo bandomas algoritmo veikimas, buvo atsisiųsti iš http://branchandcut.org/?611de4f0?bc79c800 puslapio, kur yra pateikiami būtent VRP uždavinio testavimo duomenys. Kiekviename testiniame faile yra pateikiamos kiekvieno taško, į kurį reikia aplankyti euklido koordinatės ir kiek į tą tašką reikia pristatyti krovinio. Papildomai yra pateikiamos pradinio taško koordinatės ir su kiek transporto priemonių pagalba buvo rastas optimalus sprendimas. Kartu pateikiamos ir rastos optimalios kelionės, tai yra kokia tvarka kiekvienai transporto priemonei reikia aplankyti taškus. Visi testiniai failai, kurie buvo naudojami testuojant algoritmą turėjo surastą optimalią kelionę. Nurodyta optimali kelionė buvo naudojama palyginimams, kiek tiksliai veikia algoritmas, vienas ar kitas operatorius.

4.2 Chromosomos modelis

Sudarant chromosomą buvo pasirinktas dviejų dalių chromosomos modelis, kuris yra aprašytas skyriuje 2.1.3.3. Toks modelis leidžia mums nesudėtingai paskaičiuoti tinkamumo funkciją ir atlikinėti genetines operacijas su chromosoma. Chromosoma buvo realizuota pasinaudojant dviem sąrašais. Vienas sąrašas savyje saugo informaciją apie visus pristatymo taškus, o antras saugo informaciją, kiek kiekviena transporto priemonė turi aplankyti taškų. Kiekvienas pristatymo taškas moka paskaičiuoti atstumą iki kokio nors kito taško. Skaičiavimams buvo naudojama paprasčiausia euklido atstumo funkcija:

$$d(p,q) = \sqrt{((p_1 - q_1) + (p_2 - q_2))}.$$

Realiu atveju toks variantas nėra geras, kadangi euklido atstumas tarp dviejų taškų ant žemėlapių nėra lygus realiam atstumui keliaujant keliais, nors jei taikyti uždavinį lėktuvams tai visai įmanoma.

4.3 Pradinės populiacijos sudarymas

Pradinės populiacijos sudarymui buvo vadovaujamasi ankščiau skyriuje 3.4 aptartu metodu, kai dalis populiacijos yra sudaroma atsitiktine tvarka, o likusi dalis generuojama pritaikant paprasčiausius euristinius algoritmus. Kiek individų imtyje sudarys euristiniu algoritmu sudarytų chromosomų buvo paliktas laisvas parametras ir vartotojas ji gali keisti.

4.3.1Atsitiktinis individų generavimas

Atsitiktinai sumaišyti pristatymo taškus nėra sudėtinga, kadangi mums tokią galimybę suteikia pasirinktas chromosomos modelis, kur visi pristatymo taškai yra saugomi sąraše. Tai mums tiesiog reikia surašyti visus taškus iš eilės į sąrašą ir kviesti programavimo aplinkos numatytąją funkciją, kad sąrašas butu išmaišytas. Funkciją galima kviesti tiek su numatytu pagal nutylėjimą atsitiktinių skaičių generatorium, tiek paduodant savo.

Didžiausia problema sudarant atsitiktinai chromosomas iškilo tada, kai buvo norima irgi atsitiktine tvarka užpildyti sąrašą, kiek kiekviena transporto priemonė turi aplankyti taškų taip, kad kiekviena transporto priemonė galėtų nuvažiuoti bent į vieną tašką ir bendras visų transporto priemonių aplankomų taškų skaičius neviršytų pristatymo taškų skaičiaus.

Pirmas variantas, kuris buvo bandomas, tai sukurti sąrašą visų įmanomų derinių. Idėja puikiai veikdavo su nedideliais testiniais skaičiais, bet kai reikėjo atliktį veiksmą su realiais duomenimis, paprasčiausiai buvo gautas pranešimas apie klaidą, kad išnaudojame visą mums išduotą atminties kiekį. Reikia pažymėti, kad klaida atsirado bandant mažiausią failą su 32 pristatymo taškais ir 5 transporto priemonėmis.

Sprendimas buvo rastas pasinaudojant atsitiktinį generavimų skaičių. Skaičiai buvo generuojami atsižvelgiant į tai, kiek mes turim pristatymo taškų ir kiek transporto priemonių turi jas aplankyti. Kaip generuojami atsitiktiniai deriniai parodyta žemiau esančiame pseudo kode.

```
def get_random_permutation(destinations_size,vehicles_count)
while(true){
    sum = 0;
    i=0;
    list[];
    while(i<vehicles_count){
        seed = (destinations_size)-sum;
        if(seed <=0 ){
            break;
        }
        int number = rnd.nextInt(seed)+1;
        sum = sum + number;
        list[i] = number;
        i++;
    }
    if(sum == destinations_size && list.size == vehicles_count){
        return list;
    }
}
```

14. Atsitiktinio derinio generavimo funkcija.

4.3.2 Individų generavimas euristiniu algoritmu

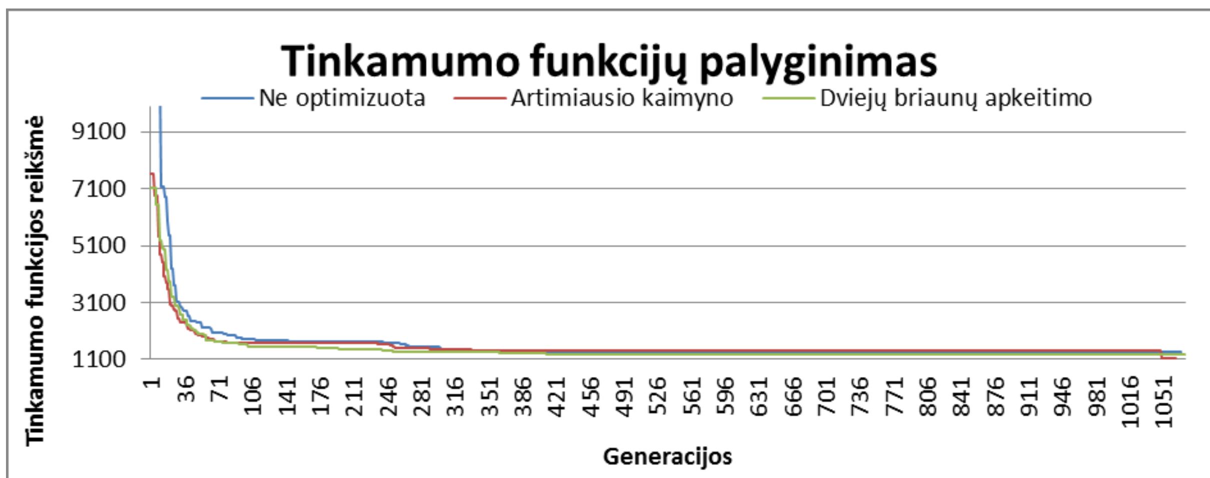
Norint, kad chromosomos nebūtų sudarytos vien tik atsitiktine tvarka ir atsakymas greičiau konverguotu į optimalų, pradinę populiaciją turime papildyti individais, kurie bus sudaryti pasinaudojant klasterizavimo algoritmu. Taškų klasterizavimui buvo pasirinktas 2.1.2.1 skyriuje apžvelgtas k- centrų algoritmas. Algoritmas sugrupuoja taškus taip, kad kiekvieno klasterio atstumų tarp taškų suma būtų panaši. Pasinaudojant standartiniu k-centrų algoritmo įgyvendinimu visiems individams gautume tokius pačius klasterius, kas mūsų atveju netinka. Mes norime gauti kuo daugiau skirtingų sprendimų, nors jie bus ir blogesni, o gal ir geresni. Kad paįvairint algoritmą atsitiktinumo dėsniu buvo sugalvota, kad kiekvieno klasterio centras bus nustatomas stochastiškai. Pradžioje yra turime rasti visų taškų „x“ ir „y“ ašies didžiausias ir mažiausias reikšmes, tokiu būdu apibrėžiame stačiakampį, kuriame galime generuoti kiekvieno centrinio taško koordinates. Pasinaudojant atsitiktinumais mes gauname daug įvairiausių sprendimų, kurių negautume jei naudotume standartiškai k – centrų algoritmą.

4.4 Tinkamumo funkcija

Tinkamumo funkcijos modelis buvo pateiktas skyriuje 3.4. Pasirinktam chromosomos modeliui pritaikyti tinkamumo funkcija buvo nesudėtinga. Pirmiausiai reikia pasiimti iš transporto priemonių sąrašo pirmą reikšmę, toliau pagal gautą reikšmę reikia nuskaityti tiek

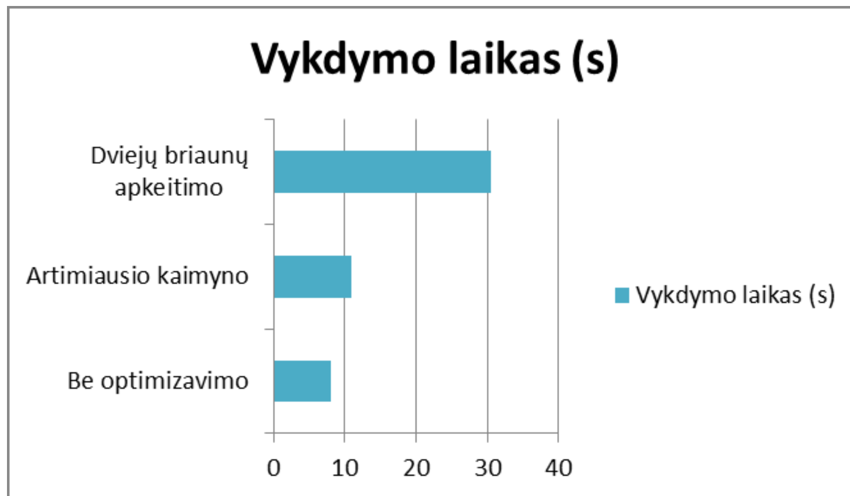
elementų iš pristatymo taškų sąrašo ir palikti nuorodą kur baigėm skaityti. Sekančiame žingsnyje yra apskaičiuojama einamos kelionės kaina. Kaina skaičiuojama iš eilės sudėjus visus atstumus tarp taškų ir pridėjus jei yra reikalingas baudos narį jei einamos transporto priemonės keliamoji galia buvo viršyta. Procesas kartojamas kol visų kelionių kainos nėra žinomos. Sudėjus visas kelionių kainas yra gaunama mus dominančios tinkamumo funkcijos reikšmė.

Taip aprašyta tinkamumo funkcija moka greitai neatliekant sudėtingu operacijų gauti chromosomos tinkamumą. Sprendžiant uždavinį iškilo problema, kad rezultatai lėtai artėja prie optimalaus sprendimo. Spręsti iškilusią problemą teko papildant tinkamumo funkciją kiekvienos kelionės optimizavimo algoritmų. Prieš skaičiuojant kiekvieno atskiros kelionės kainą taškai yra apdorojami euristiniais optimizavimo algoritmais. Rezultatus, gautus panaudojus optimizavimo algoritmus galima pamatyti paveiksliuke numeris 15.



15. Tinkamumo funkcijų palyginimas.

Aukščiau pateiktame paveiksliuke yra pateiktas tinkamumo funkcijų palyginimas kai yra nenaudojamas kelionių optimizavimo algoritmas ir kai yra naudojamas artimiausio kaimyno aprašyto skyriuje 2.1.2.2 ir dviejų briaunų apkeitimo 2.1.2.4 optimizavimo algoritmas. Naudojant bet kokį optimizavimo algoritmą yra stebimas žymiai geresnis funkcijos rezultatas atliekant pirmines generacijas. Dviejų briaunų apkeitimo algoritmas savo pradiniams skaičiavimams naudoja artimiausio kaimyno algoritmą. Panaudojus aukščiau paminėtus optimizavimo algoritmas buvo pastebėta, kad programos vykdymo laikas pailgėjo, tai gerai atspindi žemiau pateiktas paveiksliukas numeris 16.



16. Vykdymo laikas naudojant optimizavimo algoritmus.

Pastebima, kad naudojant dviejų briaunų apkeitimo algoritmas pradėjo veikti 6 kartus lėčiau lyginant kaip buvo be optimizavimo algoritmo, tai atsitinka dėl to, kad yra bandoma apkeisti visas briaunas ir taip užduodama nemažai operacijų, bet verta paminėti kad ir iš 15 paveiksliuko yra matoma kad rezultatai naudojant dviejų briaunų apkeitimo algoritmą žymiai geriau artėja prie minimumo. Artimiausio kaimyno algoritmas užtrunka nežymiai ilgiau kaip ir be jo. Programos darbo laikas naudojant artimiausio kaimyno algoritmą pailgėjo 25% lyginant su kai su dviejų briaunų apkeitimo algoritmų 600%. Darbe vėliau buvo naudojamas tik artimiausio kaimyno optimizavimo algoritmas kadangi jis optimizuoja kelionę pakankamai gerai ir jo veikimas neužtrunka taip ilgai kaip dviejų briaunų apkeitimo.

4.5 Mutacijų operatoriai

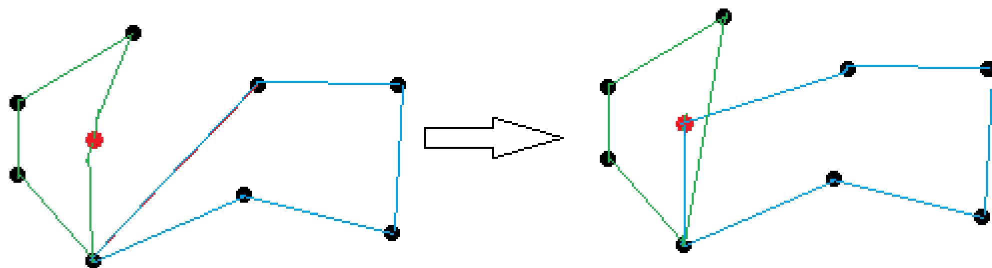
Pirmiausiai buvo įgyvendintas 3.6.3 skyriuje aprašytas mutacijos operatorius. Atrinktam individui buvo atsitiktine tvarka pasirenkama viena alėja, kuri turėjo mutoti ir toliau buvo surandama su kuria alėja ją apkeisti, visi veiksmai vykdavo pilnai atsitiktine tvarka. Tai pat buvo daroma ir su sąrašu automobilių surandamas vienas atsitiktinis kandidatas ir apkeičiamas su kitu atsitiktiniu. Kaip veikdavo toks operatorius yra pateikta paveiksliuke numeris 17.

17. Atsitiktinės mutacijos.

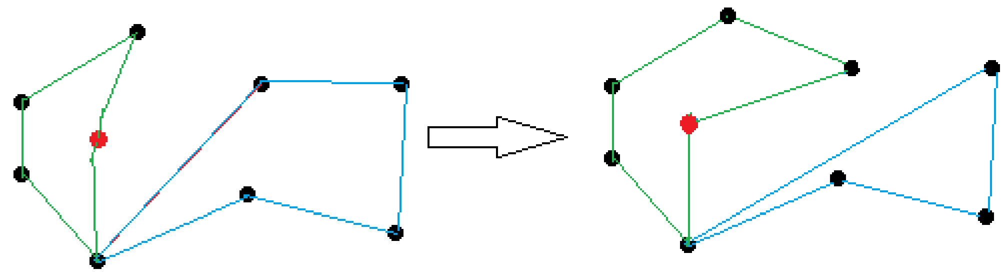
Iš aukščiau pateikto grafiko puikiai matosi, kad VRP uždavinio sprendimui atsitiktinės mutacijos visiškai netinka. Jos generuoja žymiai blogesnius atsakymus ir tinkamumo funkcija labai silpnai konverguoja.

Sprendimą iškilusiai situacijai pasiūlė darbo vadovas doc. Algirdas Bastys. Buvo siūloma atsitiktines kiekvieno pristatymo taško mutacijas pakeisti į atsitiktines taško perkėlimo į kitą kelionę operatorius. Pirmiausia atsitiktinai parenkamas taškas iš individo, sekančiame žingsnyje yra surandamas kitas artimiausias taškas iš kitų kelionių. Paskutiniame žingsnyje ir vėl atsitiktinai yra parenkamas vienas iš 3 operatorių:

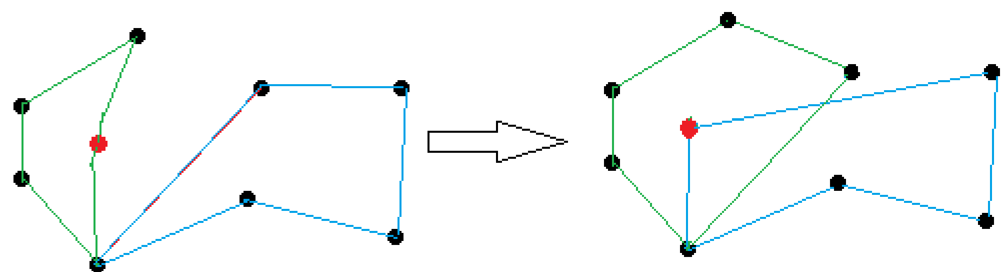
- Perkelt atsitiktinį tašką prie kaimyninio taško paveikslėlis 18.
- Perkelti kaimyninį tašką prie atsitiktinio taško paveikslėlis 19.
- Apkeisti taškus vietomis paveikslėlis 20.



18. Pasirinkto taško perkėlimas prie kaimyninio.



19. Kaimyninio taško perkėlimas prie pasirinkto.



20. Taškų apkeitimas.

Toks taškų apkeitimas yra daug lokiškesnis kaip tiesiog atsitiktine tvarka perkėlinėjant, operatorius lieka priklausomas nuo atsitiktinumų bet tie atsitiktinumai yra labiau prognozuojami palyginus su tiesiog atsitiktinių taškų apkeitimų. Tai dar geriau matoma iš tinkamumo funkcijos grafiko, kuris yra pateiktas 21 paveikslėlyje. Funkcija yra tolydi ir greitai artėjanti prie savo minimalios reikšmės.

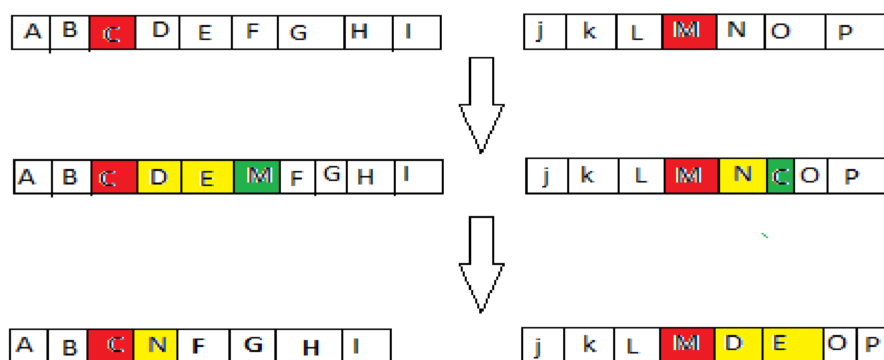
21. Artimiausiu miestų mutacijos.

4.6 Kryžminimo operatoriai

Ankščiau darbe buvo nagrinėjamas kryžminimo operatorius, kai iš dviejų tėvinių individų yra sukuriamas naujas vaikinis. Tą galima atlikti paprasčiausia atsitiktinai perkėlinėjant atsitiktinius taškus iš vienos chromosomos į kitą. Toks sprendimo variantas net nebuvo bandomas realizuoti, kadangi grubus atsitiktinis apkeitimas jau save parodė realizuojant mutacijas kaip labai neefektyvus. Kaip galimą sprendimo variantą buvo siūloma kopijuoti kelionėmis, bet pabandžius tai realizuoti praktiškai tai pasirodė nelabai įmanoma. Kelionės skirtinguose individuose yra skirtingo ilgio, pristatymo taškai yra išbarstyti nepriklausomai ir nėra galimybės kažkokiu būdu logiškai apjungti chromosomas. Darbo vadovo buvo pasiūlyta į uždavinį pažiūrėti kitu kampu. Kryžminimą atlikinėti ne tarp skirtingų dviejų individų, o su vienu. Kryžminimas būtų atliekamas su atskiromis kelionėmis. Kryžminimui atsitiktinai yra pasirenkama viena iš tėvinių chromosomų. Pritaikomas genetinis operatorius ir modifikuotas palikuonis gražinamas į populiaciją kartų su nepasirinktų tėvinių kandidatų.

4.6.1 Persikertančių miestų apkeitimas

Geresnis individas yra nukopijuojamas pilnai į vaikinę chromosomą. Iš vaikinės chromosomos yra išrenkamos dvi atsitiktinės kelionės. Kelionės negali būti vienodos, jos turi būtinai skirtis. Sekančiame žingsnyje iš kiekvienos kelionės yra išrenkama po vieną kelionės tašką. Išrinkti kelionės taškai yra įdėdami į priešingas keliones. Kelionės yra optimizuojamos, taip, kad visi taškai būtų aplankomi kuo trumpesnių maršrutų. Iš kiekvienos išrikiuotos kelionės pasiimami taškai, kurie atsidūrė tarp pasirinkto atsitiktinai taško ir įterpto iš kitos kelionės. Tokiu būdu yra gaunamos dvi tarpinės kelionės. Tarpinės kelionės yra apkeičiamos vietomis ir ištrinami įdėti iš kitos kelionės taškai. Trumpai algoritmo veikimą nupasakoja žemiau pateiktas paveiksliukas.

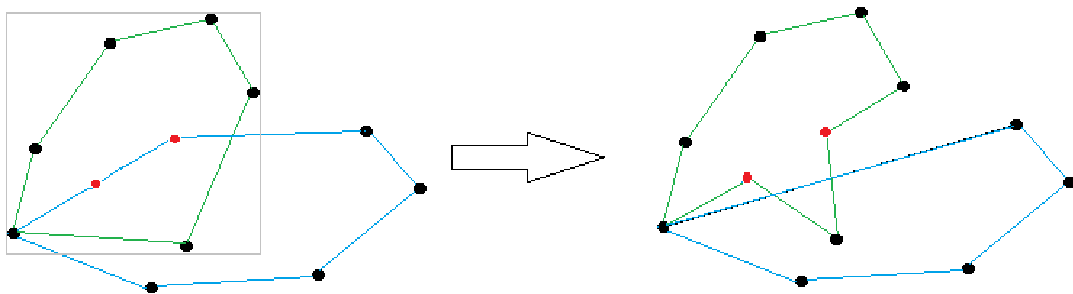


22. Persikertančių miestų apkeitimas.

Aukščiau pateiktas paveikslėlis kai per 3 žingsnius yra atliekamas miestų apkeitimas. Pirmame žingsnyje yra surandami atsitiktiniai miestai abiejuose kelionėse, jie pažymėti raudonai. Sekančiame žingsnyje miestai yra įterpiami į priešingus maršrutus ir maršrutai yra optimizuojami. Žaliai pažymėti miestai įterpti iš kito maršruto, o geltonai mes pažymim miestus, kurie atsirado tarp jų. Paskutiniame žingsnyje rasti persikertantys mažesni maršrutai yra perkeliami ir ištrinami dirbtinai pridėti taškai. Ištrinti taškai antrame žingsnyje buvo pažymėti žaliai, o taškai, kurie buvo perkelti – geltonai.

4.6.2Uždaros dėžutės

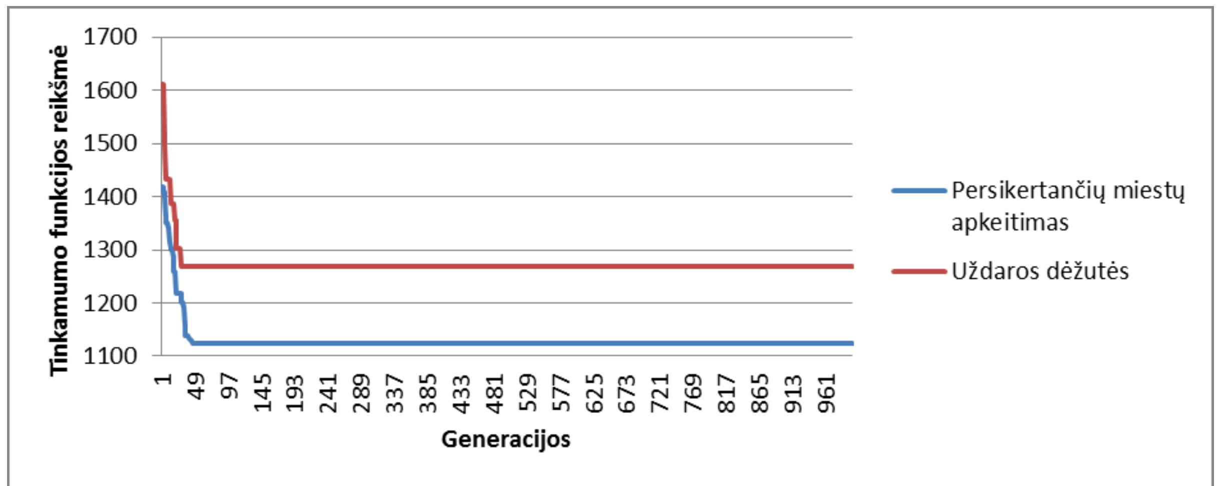
Kryžminimas kaip ankščiau aprašytu metodu yra atliekamas tik su viena chromosoma. Iš dviejų tėvinių yra pasirenkama, kuri turi geresnę tinkamumo funkcija ir nukopijuojama į vaikinę. Iš visų kelionių pasirenkama viena atsitiktine tvarka. Yra nustatomos kelionės ribos. Tai bus stačiakampis, į kuri įsipaišo kelionė, nustatomos yra minimalios ir maksimalios „x“ ir „y“ reikšmės. Einant iš eilės per visas likusias kelionės bandoma rasti taškus, kurie patenka į mūsų ankščiau apibrėžta stačiakampį. Jei tokių taškų yra surandama su tam tikra tikimybe taškai būna perkeliami į pradine kelione. Algoritmo veikimą vizualizuoja žemiau pateiktas paveiksliukas.



23. Uždaros dėžutės perkėlimas.

Aplink žalią kelionę yra apibrėžiamas stačiakampis žymintis kelionės ribas. Raudonai yra pažymėti taškai iš kitos kelionės, kurie patenka į apibrėžtą stačiakampį. Galiausiai taškai yra perkeliami į pradinę kelionę. Kelionė nėra optimizuojama kaip tai yra daroma tada, kai yra naudojamas persikertančių miestų operatoriuje, dėl to optimizavimas nors ir netobulas yra reikalingas tinkamumo funkcijos skaičiavimuose.

Dviejų aukščiau aprašytų operatorių palyginimui yra pateikiamas grafikas.



24. Kryžminimo operatorių palyginimas.

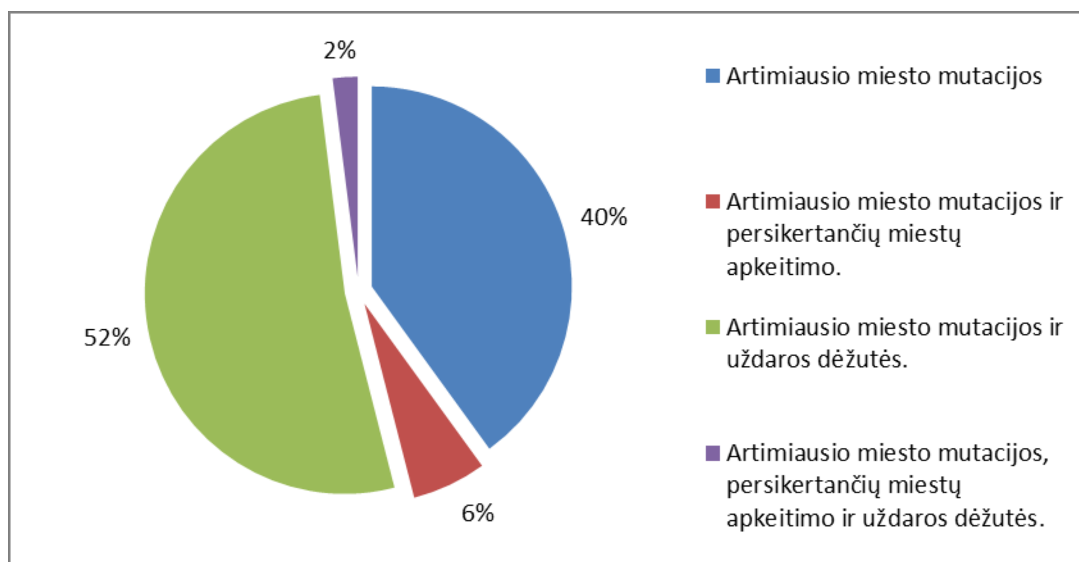
Iš grafiko matoma, kad abu operatoriai konverguoja. Verta pastebėti, kad persikertančių miestų operatorius geriau susitvarko su užduotim, bet neverta nurašyti ir uždaros dėžutės operatoriaus, kadangi jis generuoja savotiškai įdomius sprendimo variantus. Darbe bus naudojami abu aukščiau išvardinti operatoriai.

5. Rezultatai

Atlikus programavimo ir derinimo darbus buvo atliekami programos veikimo testavimo darbai. Testavimas buvo atliekamas su 50 uždavinių. Kiekvienas gentinis operatorius buvo testuojamas atskirai. Kiekvienam operatoriui buvo atliekami 5 testiniai paleidimai. Operatoriai buvo bandomi apjunginėti tarpusavyje. Taip mes gavome 6 skirtingus operatorių testavimo variantus:

- Artimiausio miesto mutacijos.
- Persikertančių miestų apkeitimas.
- Uždaros dėžutės.
- Artimiausio miesto mutacijos ir persikertančių miestų apkeitimo.
- Artimiausio miesto mutacijos ir uždaros dėžutės.
- Artimiausio miesto mutacijos, persikertančių miestų apkeitimo ir uždaros dėžutės.

Viso buvo atlikta 1500 programos paleidimu su skirtingai operatoriais. Kiekvienam uždaviniui buvo atrinktas geriausias gautas sprendimas ir kokio operatoriaus dėka buvo gautas sprendimas. Žemiau pateiktoje diagramoje yra matoma kaip pasiskirstė operatoriai kurie gavo geriausius sprendimus kiekvienam uždaviniui.



25. Operatorių pasiskirstymas gaunant geriausia sprendimą.

Iš visų operatorių dominuojantis yra artimiausio miesto mutacijos operatorius kartu su uždaros dėžutės operatorium, jo dėka buvo rastas geriausias sprendimas 52% spręstų uždavinių. Antras pagal efektyvumą buvo tiesiog artimiausio miesto mutacijos operatorius jo dėka buvo gauta 40% sprendinių. Verta pastebėti kad programose naudoti vien tik

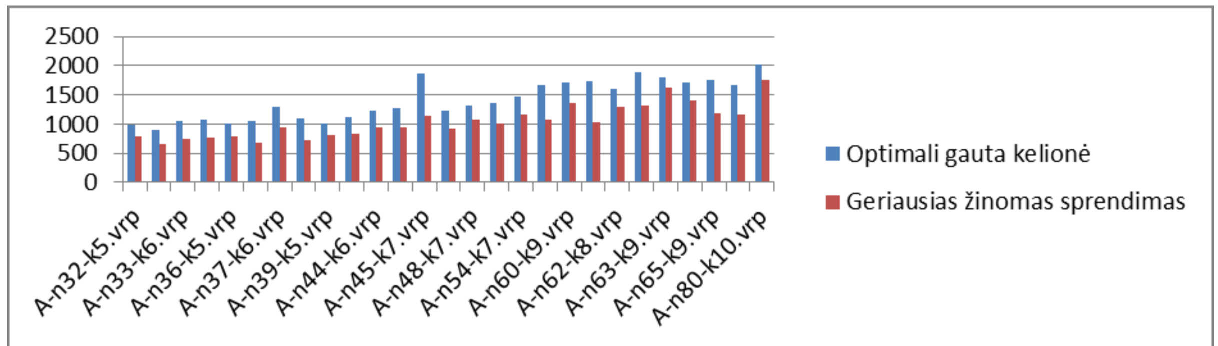
kryžminimo operatorių nėra prasminga. Sprendžiant uždavinius kai buvo naudojamas tik vienas kažkuris iš kryžminimo operatorių geriausio sprendimo nebuvo rasta. Vertinant genetinius algoritmus verta atkeipti dėmesį ir į programos vykdymosi laiką. Nuo vykdymosi laiko priklauso ar programa yra tinkama naudoti, nes praktiškai nėra veiksminga naudoti programas kurios skaičiuotu rezultatą valandomis. Žemiau yra pateiktas grafikas kuris vaizduoja vidutinių programos veikimo laiką naudojant skirtingus operatorius bei jų kombinacijas.

26. Vidutinis programos vykdymosi laikas su skirtingais operatoriais.

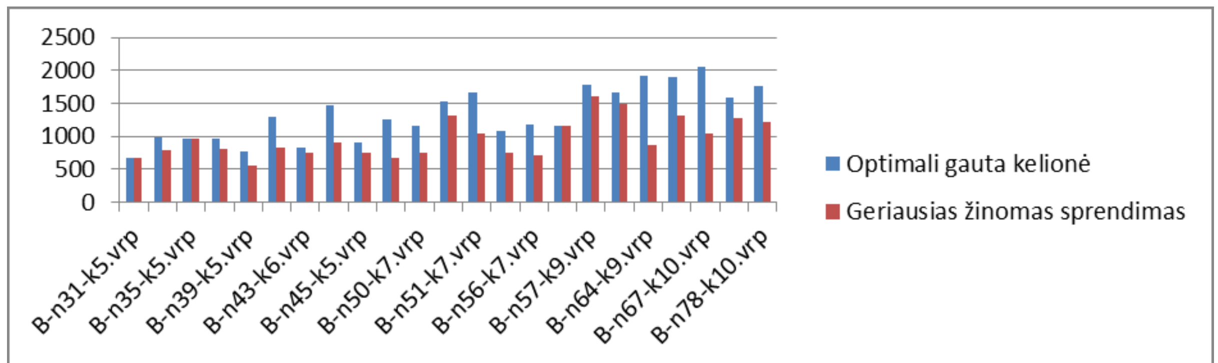
Greičiausiai programa veikdavo kai naudojo vien tik uždaros dėžutės kryžminimo operatorių. Antroje vietoje liko persikertančių miestų kryžminimo operatorius, ne toli nuo jo atsiliko ir artimiausio miesto mutacijos operatorius, o toliau jau seka kombinuotų operatorių rezultatai kaip ir reikėjo tikėtis jų veikimo laikas yra ilgesnis. Vykdymosi laikai buvo skaičiuojami nano sekundžių tikslumų.

Pagrindinis darbo rezultatas yra algoritmo tinkamumo funkcijos įvertinimas ir jos nukrypimas nuo optimalaus rezultato. Vidutiniškai gautas programos rezultatas nuo

optimalaus nukrypdavo 38.06 %. Žemiau pateiktuose diagramose yra pateikta kiek nuo žinomo optimalaus sprendimo atsiliko algoritmo gautas rezultatas.



27. Gauto sprendimo palyginimas su žinomu optimaliu (1).



28. Gauto sprendimo palyginimas su žinomu optimaliu (2).

Geresnius rezultatus algoritmas parodė spręsdamas „B“ bibliotekos testinius uždavinius. Geriausiai buvo išspręstas uždavinys „B-n57-k7.vrp“ nuokrypis nuo optimalaus sprendimo sudarė 0.91%. Blogiausias rezultatas buvo gautas sprendžiant „B-n64-k9.vrp“ uždavinį, nuokrypis nuo optimalaus sudarė 121,97%.

6. Išvados

Pagrindinė darbo išvada – genetinių algoritmų pagalba galima sukurti efektyvų algoritmą sugebanti spręsti MTSP uždavinį su papildomais apribojimais.

Sukūrus genetinį algoritmą uždavinio sprendimui tai pat nustatyta:

- Pradinė populiacija sudaryta vien stochastinių principų yra ne efektyvi. Pradinė populiacija turi būti sudaroma atsižvelgiant į uždavinį. Darbe pradinė populiacija papildomai buvo generuojama pasinaudojant k – centrų algoritmų.
- Turi būti naudojamas ko efektyvesnis chromosomos modelis, atsižvelgiant į uždavinio struktūra, paieškos aibė turi būti kaip įmanoma sumažinta. Darbe pateikiama dviejų dalių chromosomos technika.
- Sprendžiant uždavinius, norint pagreitint atsakymo konvergavimą prie optimalaus, siūloma naudotis lokalaus optimizavimo algoritmais. Algoritmas turi būti greitas kadangi yra kviečiamas daugeli kartų. Darbe naudojamas artimiausio kaimyno algoritmas.
- Vengti naudoti visiškai stochastinių operatorių. Jų dėka yra gaunami unikalus sprendimai, bet labai lietai konverguojama prie optimalaus sprendimo.
- Kuriant gentinius operatorius yra būtina atsižvelgti į uždavinio specifiką. Kiekvienam uždaviniui turi būti kuriami jam tinkami genetiniai operatoriai. Darbo metų buvo sukurti artimiausio miesto mutacijos, persikertančių miestų kryžminimo ir uždaros dėžutės kryžminimo operatoriai tinkantis MTSP uždavinio sprendimui.
- Kuriant operatorius ir lokalaus optimizavimo algoritmus būtina atsižvelgti į jų veikimo greiti. Algoritmo veikimo metų jei yra naudojami daugeli kartų. Darbo metų buvo atsisakyta dviejų briaunų apkeitimo lokalaus optimizavimo algoritmu dėl jo prasto veikimo greičio.

Darbo tobulinimui yra siūloma papildyti genetinius operatorius papildomom funkcijom kuriuos leistų atsižvelgti į kiekvienos kelionės sudaryta bendra svorį ir į tai kiek pirklys dar gali panešti ir perkėlinėti taškus jau atsižvelgus į šita papildoma informacija. Papildyti chromosomą papildoma informacija apie tai ar buvo jis keistas einamojoje iteracijoje ir saugoti tinkamumo funkcijos reikšme. Tai leistų sumažint nereikalingus tinkamumo funkcijos skaičiavimus kiekvienoje iteracijoje.

7. Šaltinių sąrašas

[Bek05] Tolga Bektas. The Multiple traveling salesman problem: an overview of Formulations and solution procedures, The International Journal of Management Science, Omega 34, p. 209-219, 2006. Prieiga per internetą: http://han-4.tem.nctu.edu.tw/students/thesis/097/1/Literature/Time%20Windows%20Discretization/The%20multiple%20traveling%20salesman%20problem-an%20overview%20of%20formulations%20and%20solution%20procedures.pdf [žiūrėta 2011-05-16].

[HaW79] J. A Hartigan, M. A. Wong, Algorithm AS 136: A K-Means Clustering Algorithm, Journal of the Royal Statistical Society, Vol. 28, No. 1, p. 100-108, 1979. Prieiga per internetą: http://www.jstor.org/discover/10.2307/2346830?uid=3738480&uid=2129&uid=2&uid=70&uid=4&sid=47699043378347 [žiūrėta 2011-05-16].

[Che03] Sofiya Cherni, Nearest neighbord method, 2003. Prieiga per internetą: http://www.mcs.sdsmt.edu/rwjohnso/html/sofiya.pdf [žiūrėta 2011-06-01].

[Mer99] Stephan Mertens , TSP Algorithms in action, 1999. Prieiga per internetą: http://www-e.uni-magdeburg.de/mertens/TSP/ [žiūrėta 2011-07-23].

[LiK73] Lin Shen, B.W. Kernighan, An Effective Heuristic Algorithm for the Traveling-Salesman Problem, Vol. 21, No. 2, p. 498-516, 1973. Prieiga per internetą: http://www.jstor.org/discover/10.2307/169020?uid=3738480&uid=2129&uid=2&uid=70&uid=4&sid=47699043378347 [žiūrėta 2011-05-25].

[NDDP09] R. Nallusamy, K.Duraiswamy, R.Dhanalaksmi, P. Parthiban, Optimization of Non-Linear Multiple Traveling Salesman Problem Using K-Means Clustering, Shrink Wrap Algorithm and Meta-Heuristics, International Journal of Nonlinear Science, Vol.8 No. 4, p.480-487, 2009. Prieiga per internetą:http://www.worldacademicunion.com/journal/1749-3889-3897IJNS/IJNSVol08No4Paper13.pdf [žiūrėta 2012-02-30].

[Hus89] Ahmad Husban, An Exact Solution Method for the MTSP, The Journal of the Operational Research Society, Vol. 40, No. 5,p. 461-469, 1989. Prieiga per internetą:http://www.jstor.org/discover/10.2307/2583618?uid=3738480&uid=2129&uid=2&uid=70&uid=4&sid=47698974064027 [žiūrėta 2012-03-04].

[Koh06] S.P. Koh , Design and Performance Optimization of a Multi-TSP (Traveling Salesman Problem) Algorithm, ICGST International Journal on Artificial Intelligence and Machine Learning, Volume 06 - Issue (III),p. 29-33, 2006. Prieiga per internetą: http://www.icgst.com/aiml/Volume6/Issue3/P1120632001.html [žiūrėta 2011-04-25].

[ChK01] Chandra Chekuri, Amit Kumar, Maximum Coverage Problem with Group Budget Constraints and Applications, 2001. Prieiga per internetą: http://www.cs.uiuc.edu/~chekuri/papers/coverage.pdf [žiūrėta 2012-01-24].

[MSE99] A. Modares, S. Somhom, T. Enkawa. A self-organizing neural network approach for Multiple traveling salesman and vehicle routing problems, International Transactions in Operational Research, Volume 6, Issue 6, p. 591–606, 1999. Prieiga per internetą:
http://onlinelibrary.wiley.com/doi/10.1111/j.1475-3995.1999.tb00175.x/abstract
[žiūrėta 2011-07-13].

[FrS05] Gereon Frahling, Christian Sohler, A fast k-means implementation using coresets, 2005. Prieiga per internetą:
http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en//pubs/archive/33320.pdf [žiūrėta 2012-01-12].

[Tan00] Lixin Tang, A multiple traveling salesman problem model for hot rolling scheduling in Shanghai Baoshan Iron & Steel Complex, European Journal of Operational Research, Volume 124, Issue 2,p. 267–282, 2000. Prieiga per internetą:
http://www.sciencedirect.com/science/article/pii/S037722179900380X [žiūrėta 2011-06-08].

[EnK06] L. Engebretse, M. Karpinski, TSP with bounded metrics, Journal of Computer and System Sciences, Volume 72, Issue 4,p. 509–546. Prieiga per internetą: http://www.sciencedirect.com/science/article/pii/S0022000005001285 [žiūrėta 2010-03-27].

[MiK02] Snezana Mitrovic-Minic, Ramesh Krishnamurti, The Multiple Traveling Salesman Problem with Time Windows: Bounds for the Minimum Number of Vehicles, Operations Research Letters , Volume 34, Issue 1, p. 111–120, 2002. Prieiga per internetą: ftp://fas.sfu.ca/pub/cs/TR/2002/CMPT2002-11.pdf [žiūrėta 2012-02-12].

[Bja04] Aslaug Soley Bjarnabottir, Solving the Vehicle Routing Problem with Genetic Algorithms, Ieee, Volume: 2, p. 126-131, 2004. Prieiga per internetą: http://www.mendeley.com/research/solving-vehicle-routing-problems-with-genetic-algorithms/ [žiūrėta 2012-03-14].

[CaR05] Arthur E. Carter, Cliff T. Ragsdale, A new approach to solving the multiple traveling salesperson problem using genetic Algorithms, European Journal of Operational Research, Volume 175, Issue 1, p. 246–257 2005. Prieiga per internetą: http://www.sciencedirect.com/science/article/pii/S0377221705004236 [žiūrėta 2011-05-19].

8. Santrumpos

1. TSP (angl. Traveling salesman problem) – keliaujančio pirklio problema.
2. MTSP (angl. Multi traveling salesman problem) – keliaujančių pirklių problema.
3. VRP (angl. Vehicle routing problem) - transporto priemonių maršrutų optimizavimo problema.

9. Priedai

1. Programos gautų rezultatų lentelė. Duomenis gauti atlikus kiekvieno užduoties failo 5 paleidimus su skirtingais genetinių algoritmų operatorių variacijom. Rezultate pateikiamas geriausias gautas sprendimas.

Uždavinio failas	Geriausias gautas rezultatas	Geriausias žinomas sprendimas	Nuokrypis nuo geriausio žinomo sprendimo (%)
A-n32-k5.vrp	979.4	784	24.9
A-n33-k5.vrp	906.1	661	37
A-n33-k6.vrp	1059.4	742	42.7
A-n34-k5.vrp	1075.5	778	38.2
A-n36-k5.vrp	1006.5	799	25.9
A-n37-k5.vrp	1059.1	669	58.3
A-n37-k6.vrp	1285.5	949	35.4
A-n38-k5.vrp	1087.2	730	48.9
A-n39-k5.vrp	1015.7	822	23.5
A-n39-k6.vrp	1125.6	831	35.4
A-n44-k6.vrp	1239.1	937	32.2
A-n45-k6.vrp	1279.92643	944	35.5
A-n45-k7.vrp	1854.9	1146	61.8
A-n46-k7.vrp	1235.0	914	35.1
A-n48-k7.vrp	1309.7	1073	22.0
A-n53-k7.vrp	1369.8	1010	35.6
A-n54-k7.vrp	1462.9	1167	25.3
A-n55-k9.vrp	1662.2	1073	54.9
A-n60-k9.vrp	1717.5	1354	26.8
A-n61-k9.vrp	1736.4	1034	67.9
A-n62-k8.vrp	1598.9	1288	24.1
A-n63-k10.vrp	1893.5	1314	44.1
A-n63-k9.vrp	1791.5	1616	10.8
A-n64-k9.vrp	1719.1	1401	22.7

A-n65-k9.vrp	1745.2	1174	48.6
A-n69-k9.vrp	1661.3	1159	43.3
A-n80-k10.vrp	2022.7	1763	14.7
B-n31-k5.vrp	679.5	672	1.1
B-n34-k5.vrp	976.7	788	23.9
B-n35-k5.vrp	966.8	955	1.2
B-n38-k6.vrp	967.9	805	20.2
B-n39-k5.vrp	765.3	549	39.4
B-n41-k6.vrp	1300.6	829	56.8
B-n43-k6.vrp	817.9	742	10.2
B-n44-k7.vrp	1475	909	62.2
B-n45-k5.vrp	911.9	751	21.4
B-n45-k6.vrp	1263.8	678	86.4
B-n50-k7.vrp	1166.2	741	57.3
B-n50-k8.vrp	1524.7	1312	16.2
B-n51-k7.vrp	1659.8	1032	60.8
B-n52-k7.vrp	1079.6	747	44.5
B-n56-k7.vrp	1173.4	707	65.9
B-n57-k7.vrp	1163.5	1153	0.9
B-n57-k9.vrp	1790	1598	12
B-n63-k10.vrp	1668.5	1496	11.5
B-n64-k9.vrp	1911.2	861	121.9
B-n66-k9.vrp	1901.9	1316	44.5
B-n67-k10.vrp	2050.4	1032	98.6
B-n68-k9.vrp	1584.7	1272	24.5
B-n78-k10.vrp	1754.9	1221	43.7

2. Pradinių kelionių sudarytų klasterizavimo algoritmų pagerinimas genetiniais operatoriais. Duomenis gauti atlikus bandymus su kiekvienu užduoties failų, parinkus kad veiksmą atlikinėtu persikertančių miestų kryžminimo operatorius, uždaros dėžutės kryžminimo operatorius ir artimiausių miestų mutacijų operatorius.

Uždavinio failas	Tinkamumo funkcijos reikšmė atlikus klasterizavimą	Tinkamumo funkcijos reikšmė pritaikius genetinius operatorius	Tinkamumo funkcijos pagerinimas (%)
A-n32-k5.vrp	1698.4	1116.8	34.2
A-n33-k5.vrp	3409	1045.8	69.3
A-n33-k6.vrp	4278.7	1233.6	71.1
A-n34-k5.vrp	3634.2	1176.8	67.6
A-n36-k5.vrp	2080.7	1112.7	46.5
A-n37-k5.vrp	1805.4	1152.2	36.1
A-n37-k6.vrp	5021.7	1460.1	70.9
A-n38-k5.vrp	4105.8	1263.3	69.2
A-n39-k5.vrp	2348.8	1261.7	46.2
A-n39-k6.vrp	2696.8	1266.2	53
A-n44-k6.vrp	4916.9	1468.5	70.1
A-n45-k6.vrp	9294.5	1644.4	82.3
A-n45-k7.vrp	3140.6	1918.2	38.9
A-n46-k7.vrp	6063.4	1436.5	76.3
A-n48-k7.vrp	6137.4	1569.4	74.4
A-n53-k7.vrp	11349.5	1866.6	83.5
A-n54-k7.vrp	7781.6	1850.2	76.2
A-n55-k9.vrp	11934.8	2204.1	81.5
A-n60-k9.vrp	16811.6	2218.9	86.8
A-n61-k9.vrp	21721.8	2362.9	89.1
A-n62-k8.vrp	10009.8	1969.9	80.3
A-n63-k10.vrp	12956.3	2515.6	80.5
A-n63-k9.vrp	22302.5	2371	89.3
A-n64-k9.vrp	18064	2393.6	86.7
A-n65-k9.vrp	13200.9	2509.9	80.9
A-n69-k9.vrp	18609.4	2218	88
A-n80-k10.vrp	14318.2	2562.1	82.1
B-n31-k5.vrp	1322.9	611.5	53.7
B-n34-k5.vrp	4133.7	996.1	75.9

B-n35-k5.vrp	3117.3	961	69.1
B-n38-k6.vrp	2971.3	1117.5	62.3
B-n39-k5.vrp	1503	804.1	46.4
B-n41-k6.vrp	4721.2	1391.1	70.5
B-n43-k6.vrp	4828.9	956.5	80.1
B-n44-k7.vrp	5193.5	1718.9	66.9
B-n45-k5.vrp	2864	1122.0	60.8
B-n45-k6.vrp	8276.7	1465.9	82.2
B-n50-k7.vrp	6927.1	1323.2	80.8
B-n50-k8.vrp	12319.1	1751.2	85.7
B-n51-k7.vrp	10818.6	1884.4	82.5
B-n52-k7.vrp	6205.5	1223.3	80.2
B-n56-k7.vrp	2438.4	1361.9	44.1
B-n57-k7.vrp	11831	1757.9	85.1
B-n57-k9.vrp	9470.1	2283.7	75.8
B-n63-k10.vrp	6574.9	2086.3	68.2
B-n64-k9.vrp	11020.1	2491.9	77.3
B-n66-k9.vrp	10410	2573.2	75.2
B-n67-k10.vrp	16828.4	2491.6	85.1
B-n68-k9.vrp	8009.9	2046.8	74.4
B-n78-k10.vrp	9790.7	2320.0	76.3