KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

Algoritmų sudarymas ir analizė (P170B400)

Laboratorinių darbų ataskaita

Atliko:
IFF-1/6 gr. studentas
Lukas Kuzmickas
2023 m. gegužės 17 d.
Priėmė:
Doc. Pilkauskas Vytautas

KAUNAS 2023

TURINYS

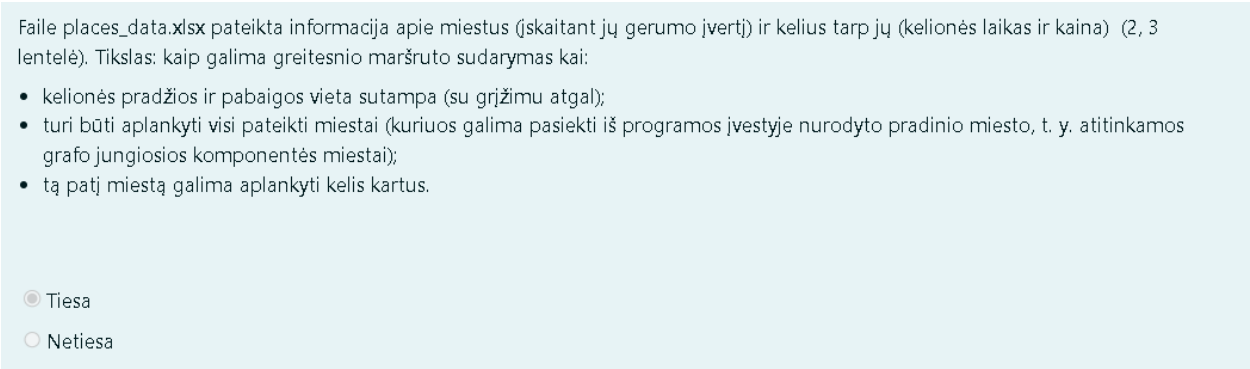
Užduoties aprašymas

1. **Pirma dalis.** (~3 balai). Realizuoti programą, kuri individualiai problemai pateiktų optimalų sprendinį. Nustatyti, prie kokios duomenų apimties sprendinį pavyksta rasti jei programos vykdymo laikas negali būti ilgesnis nei 10 sek.

2. **Antra dalis.** (~2 balai). Realizuoti programą, kuri individualios problemos rezultatą pateiktų priimant „lokaliai geriausią“ sprendinį (pvz. esant keliems pasirinkimams keliauti į skirtingas viršūnes, visuomet renkamasi: pigiausias ar trumpiausias ar ... kelias)

3. **Trečia dalis.** (~5 balai). Realizuoti programą, kuri pateiktų sprendinį taikant Genetinio Optimizavimo metodą. Programa pateikti rezultatą turi ne ilgiau, nei per 60 sek.

- apskaičiuotas asimptotinis programos vykdymo laiko sudėtingumas (esant rekursiniams metodams, turi būti suformuojama rekurentinė lygtis);
- realizuotų programų (sudaryto algoritmo konkrečiam uždaviniui) abstraktus aprašas („pseudo“ kodas ar „workflow“ diagrama);
- skirtingų metodų rezultatų analizė uždavinio gerumo ir vykdymo laiko prasme esant kelioms skirtingoms pradinėms sąlygoms.
- pateikti rekomenduojamus genetinio optimizavimo parametrus užduoties sprendimui (populiacijos dydis, elito kiekis, ...). Pateikti analizės rezultatus, kaip buvo parinkti rekomendaciniai parametrai, t. y. analizės rezultatuose turi pasimatyti, jog per maksimalų programos vykdymo laiką, tikėtina, jog tikslo funkcija įgaus didžiausią reikšmę.
- marštutai, sudaryti analizuojant atksirus atvejus atvaizduojami grafiškai.



1 pav. Inžinerinio projekto individuali užduotis

Pirma dalis užduoties sprendimas

Pirma dalis. (~3 balai). Realizuoti programą, kuri individualiai problemai pateiktų optimalų sprendinį. Nustatyti, prie kokios duomenų apimties sprendinį pavyksta rasti jei programos vykdymo laikas negali būti ilgesnis nei 10 sek.

Pirmos užduoties apskaičiuotas asimptotinis programos vykdymo laiko sudėtingumas

public static List<PirmasisTable> SudarytiMaršruta(List<AntrasisTable> keliones, string pradinisMiestas)	Kaina	Kiekis
{		
List<PirmasisTable> marsrutas = new List<PirmasisTable>();	c1	1
string dabartinisMiestas = pradinisMiestas;	c2	1
foreach (AntrasisTable kelione in keliones)	c3;c4;c5	1;n;n
{		
if (kelione.miestasIs == dabartinisMiestas)	(1-x)c6	n
{		
PirmasisTable dabartinisMarsrutas = new PirmasisTable();	(1-x)c7	n
dabartinisMarsrutas.MiestoPav = dabartinisMiestas;	(1-x)c8	n
marsrutas.Add(dabartinisMarsrutas);	(1-x)c9	n
dabartinisMiestas = kelione.miestasI;	(1-x)c10	n
}		
else if (kelione.miestasI == dabartinisMiestas)	(1-y)c11	n
{		
PirmasisTable dabartinisMarsrutas = new PirmasisTable();	(1-y)c12	n
dabartinisMarsrutas.MiestoPav = dabartinisMiestas;	(1-y)c13	n
marsrutas.Add(dabartinisMarsrutas);	(1-y)c14	n
dabartinisMiestas = kelione.miestasIs;	(1-y)c15	n
}		
}		
PirmasisTable paskutinisMarsrutas = new PirmasisTable();	c16	1
paskutinisMarsrutas.MiestoPav = dabartinisMiestas;	c17	1
marsrutas.Add(paskutinisMarsrutas);	c18	1
}		

Šio metodo asimptotinis sudėtingumas priklauso nuo duomenų kiekio, kuris yra Pirmame Table ir Antrame Table. Priklauso nuo (pirmos lentelės) ir (antros lentelės) duomenų kiekio.

$$c_i - konstanta\ O(1).$$

$T(n) = c_1 + c_2 + c_3 + c_4$*n + c_5*n + (1-x)* c_6*n + (1-x)* c_7*n + (1-x)* c_8*n + (1-x)* c_9*n + (1-x)* c_{10}*n + (1-y)* c_{11}*n + (1-y)* c_{12}*n + (1-y)* c_{13}*n + (1-y)* c_{14}*n + (1-y)* c_{15}*n;

Kai mūsų x = 1arba y = 1, praeinamos sąlygos mūsų metodo asimptotinis sudėtingumas yra:

$T(n) = (n + m);$

n – pirmos Excel lentelės duomenų kiekis.

m – antros Excel lentelės duomenų kiekis.

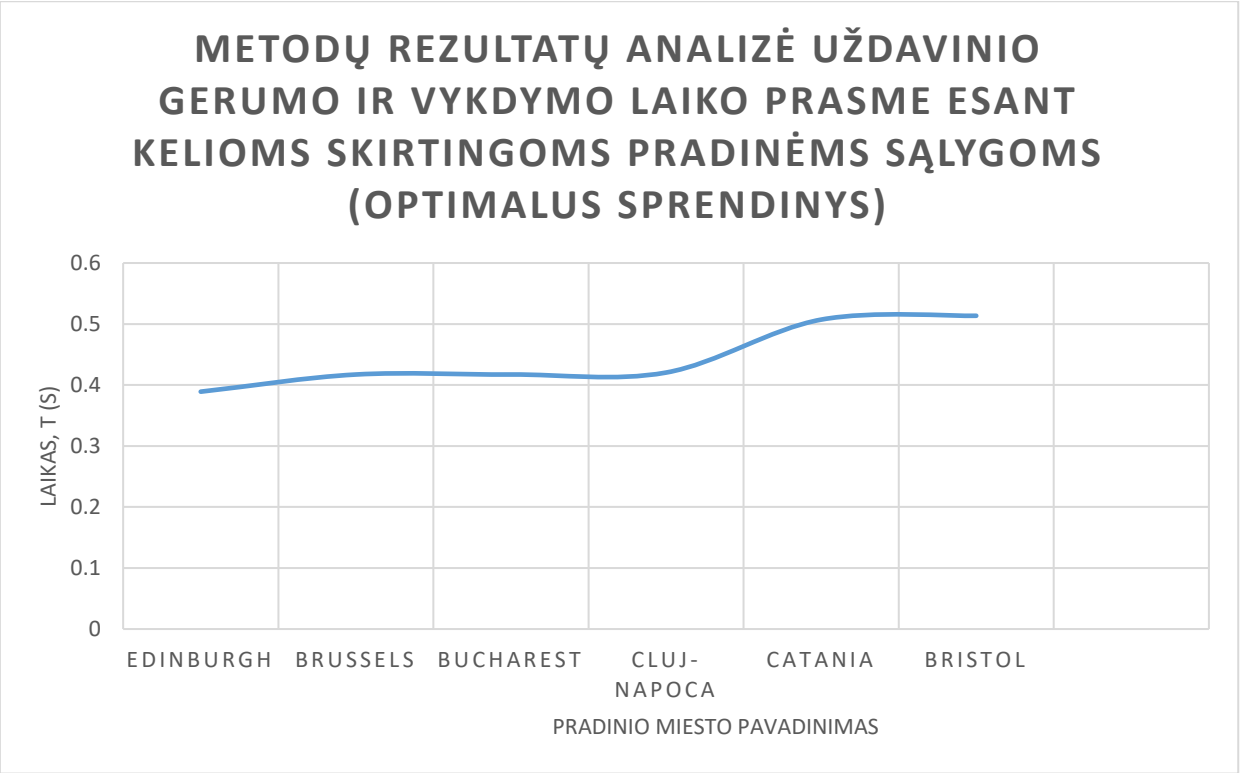
Realizuotos pirmos užduoties abstraktus aprašas „pseudo“ kodas

Veikimo paaiškinimas žingsniais:

1. Sukuriamas tuščias sąrašas `maršrutas`, kuriame bus laikomi maršruto taškai.
2. Pradinis miestas `pradinisMiestas` priskiriamas kintamajam `dabartinisMiestas`.
3. Pradedamas ciklas per kiekvieną kelionę `kelione` iš sąrašo `keliones`.
4. Tikrinama, ar kelionės `kelione` pradžios miestas (`kelione.miestasIs`) yra lygus dabartiniu miestu `dabartinisMiestas`.
    - Jei taip, tai reiškia, kad dabartinis miestas yra pradinis miestas kelionėje, ir tai yra naujas maršruto taškas.
    - Sukuriamas naujas objektas `dabartinisMarsrutas` iš klasės `PirmasisTable` ir jam priskiriamas dabartinis miestas `dabartinisMiestas`.
    - `dabartinisMarsrutas` pridedamas prie `marsrutas` sąrašo.
    - Dabartinis miestas atnaujinamas į kelionės pabaigos miestą (`kelione.miestasI`).
5. Tikrinama, ar kelionės `kelione` pabaigos miestas (`kelione.miestasI`) yra lygus dabartiniu miestu `dabartinisMiestas`.
    - Jei taip, tai reiškia, kad dabartinis miestas yra kelionės pabaigos miestas, ir tai yra naujas maršruto taškas.
    - Sukuriamas naujas objektas `dabartinisMarsrutas` iš klasės `PirmasisTable` ir jam priskiriamas dabartinis miestas `dabartinisMiestas`.
    - `dabartinisMarsrutas` pridedamas prie `marsrutas` sąrašo.
    - Dabartinis miestas atnaujinamas į kelionės pradžios miestą (`kelione.miestasIs`).
6. Ciklas kartojamas su kitomis kelionėmis sąraše `keliones`.
7. Sukuriamas paskutinis maršruto taškas `paskutinisMarsrutas` su dabartiniu miestu `dabartinisMiestas` ir pridedamas prie `marsrutas` sąrašo.
8. Grazinamas pilnas maršrutas `marsrutas`.

Šis kodas sukūrią maršrutą pagal pateiktas keliones ir pradinį miestą, laikydamasis nurodytų taisyklių, kurios leidžia pasiekti kelionės tikslą.

Pirmos užduoties skirtingų metodų rezultatų analizė uždavinio gerumo ir vykdymo laiko prasme esant kelioms skirtingoms pradinėms sąlygoms



Raskime duomenų rinkinį, kuri galime įvykdyti greičiau, nei per 10sekundžių

Mūsų Edinburgh pradinis miestas turi 14 maršrutų iš jo, jeigu padidinsime miestų dydį dvigubai, pailginame programos laiką.

Sprendžiame paprastą lygtį:

14 maršrutų – 0,347s
x maršrutų - 10s

$$\boldsymbol{x * 0.347 = 140}$$

$$\boldsymbol{x = \frac{140}{0.347} \approx 403}$$

Tai apytiksliai mums reikia apie 400 maršrutų duomenų imties, kad mūsų algoritmas gautų sprendinį greičiau nei per 10 sek.

Antra užduoties dalis sprendimas

Antra dalis. (~2 balai). Realizuoti programą, kuri individualios problemos rezultatą pateiktų priimant „lokaliai geriausią“ sprendinį (pvz. esant keliems pasirinkimams keliauti į skirtingas viršūnes, visuomet renkamasi: pigiausias ar trumpiausias ar ... kelias)

Naudojame Greedy algoritmo metodiką šitam uždaviniui.

Antros užduoties apskaičiuotas asimptotinis programos vykdymo laiko sudėtingumas

public static List<PirmasisTable> SudarytiMaršrutaGreedy(List<AntrasisTable> keliones, string pradinisMiestas)	Kaina	Kiekis
{		
List<PirmasisTable> marsrutas = new List<PirmasisTable>();	c1	1
string dabartinisMiestas = pradinisMiestas;	c2	1
while (keliones.Count > 0)	c3(1-x)	n
{		
bool rastaKelione = false;	c4(1-x)	n
for (int i = 0; i < keliones.Count; i++)	c5(1-x);c6(1-x);c7(1-x)	n;n^2+1;n^2
{		
AntrasisTable kelione = keliones[i];	c8(1-x)	n^2
if (kelione.miestasIs == dabartinisMiestas)	c9(1-y)	n^2
{	c10(1-y)	n^2
PirmasisTable dabartinisMarsrutas = new PirmasisTable();	c11(1-y)	n^2
dabartinisMarsrutas.MiestoPav = dabartinisMiestas;	c12(1-y)	n^2
marsrutas.Add(dabartinisMarsrutas);	c13(1-y)	n^2
dabartinisMiestas = kelione.miestasI;	c14(1-y)	n^2
keliones.RemoveAt(i);	c15(1-y)	n^2
rastaKelione = true;	c16(1-y)	n^2
break;	c17(1-y)	n^2
}		
else if (kelione.miestasI == dabartinisMiestas){	c18(1-z)	n^2
PirmasisTable dabartinisMarsrutas = new PirmasisTable();	c19(1-z)	n^2
dabartinisMarsrutas.MiestoPav = dabartinisMiestas;	c20(1-z)	n^2
marsrutas.Add(dabartinisMarsrutas);	c21(1-z)	n^2
dabartinisMiestas = kelione.miestasIs;	c22(1-z)	n^2
keliones.RemoveAt(i);	c23(1-z)	n^2
rastaKelione = true;	c24(1-z)	n^2
break;	c25(1-z)	n^2
}}		
if (!rastaKelione)	c26(1-l)	n^2
break;	c27(1-l)	n^2

}		
PirmasisTable paskutinisMarsrutas = new PirmasisTable();	c28	1
paskutinisMarsrutas.MiestoPav = dabartinisMiestas;	c29	1
marsrutas.Add(paskutinisMarsrutas);	c30	1
return marsrutas;}	c31	1

$$c_i - konstanta\ O(1).$$

T(n) = c_1 + c_2 + c_3 (1-x) + c_4 (1-x)*n + c_5 (1-x)*n + (1-x)* c_6*n^2 + (1-x)* c_7*n^2 + (1-x)* c_8*n^2 + (1-y)* c_9*n^2 + (1-y)* c_{10}*n^2+ (1-y)* c_{11}*n^2 + (1-y)* c_{12}*n^2 + (1-y)* c_{13}*n^2 + (1-y)* c_{14}*n^2 + (1-y)* c_{15}*n^2+ $c_{16}(1-y) * n^2 + c_{16}$ (1-y)*n^2 + c_{17} (1-y)*n^2 + c_{18}*n^2(1-z) + c_{19}*n^2(1-z) + (1-z)* c_{20}*n^2(1-z) + (1-z)* c_{21}*n^2 + (1-z)* c_{22}*n^2 + (1-z)* c_{23}*n^2 + (1-z)* c_{24}*n^2 + (1-z)* c_{25}*n^2 + (1-z)* c_{26}*n^2 + (1-z)* c_{27}*n^2 + c_{28} + c_{29} + c_{30} + c_{31};

Suprastiname konstantas ir gauname, kad mūsų metodo asimptotinis sudėtingumas yra,

kai x=0, y=0, z=0, mūsų Greedy algoritmas parenka geriausius pasirinkimus, šitam naudojame Boolean reikšmę.

Greedy algoritmo asimptotinis sudėtingumas yra:

$$T(n) = n^2;$$

Realizuotos antros užduoties abstraktus aprašas „pseudo“ kodas

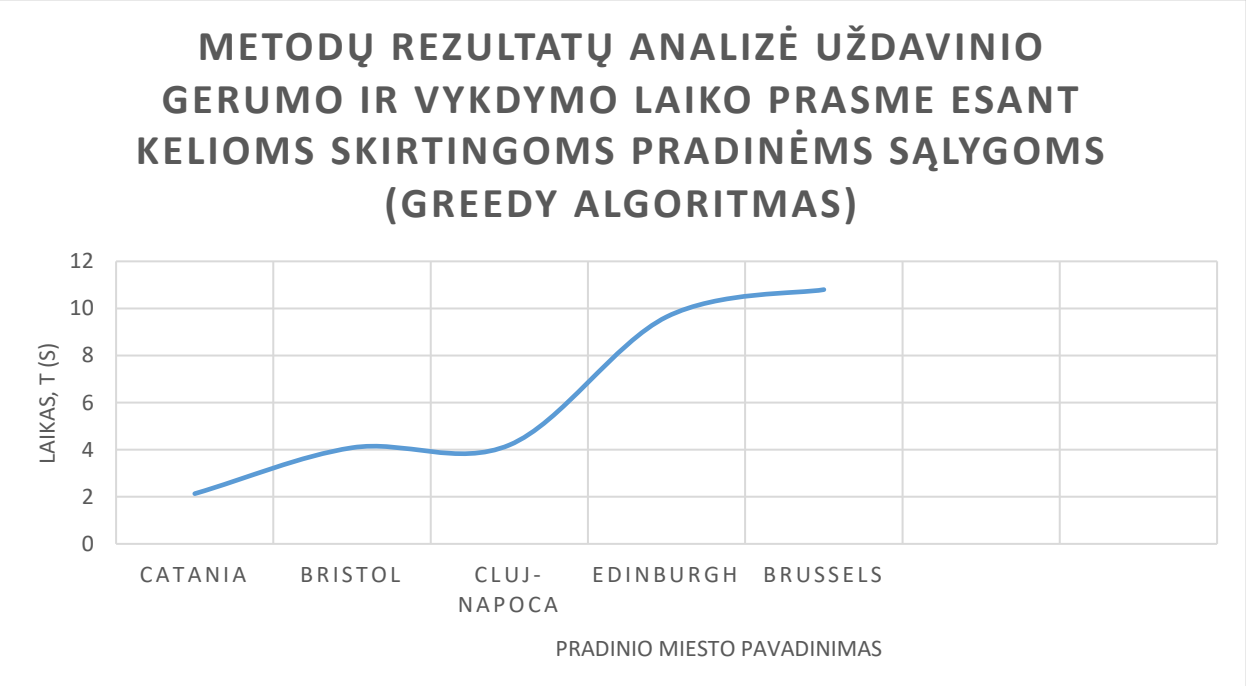
Šis metodas `SudarytiMaršrutaGreedy` naudoja Greedy (stipriai paprastintą) algoritmą siekiant sudaryti maršrutą pagal pateiktas keliones ir pradinį miestą.

1. Sukuriamas tuščias `marsrutas` sąrašas, kuriame saugosime sudarytą maršrutą.
2. Nustatomas `dabartinisMiestas` kaip pradinis miestas.
3. Kol yra likę kelionės, kartojame šias žingsnių sekcijas:
    - Patikriname kiekvieną kelionę `kelione` iš `keliones` sąrašo.
    - Jei kelionės `miestasIs` sutampa su `dabartinisMiestas`, tai reiškia, kad galime vykti į kelionės `miestasI`.
        - Sukuriame naują `dabartinisMarsrutas` objektą ir priskiriame `dabartinisMiestas` kaip `MiestoPav`.
        - Pridedame `dabartinisMarsrutas` į `marsrutas` sąrašą.
        - Atnaujiname `dabartinisMiestas` su kelionės `miestasI`.
        - Pašaliname šią kelionę iš `keliones` sąrašo.
        - Pažymime, kad rasta kelionė (`rastaKelione = true`) ir nutraukiame keliavimą per `for` ciklą.

- Kitu atveju, jei kelionės `miestasI` sutampa su `dabartinisMiestas`, tai reiškia, kad galime vykti į kelionės `miestasIs`.
    - Taikome panašų veiksmų seka, kaip anksčiau aprašytai situacijai.
- Jei niekas nebuvo rasta (nėra tinkamų kelionių), nutraukiame keliavimą per `for` ciklą.

4. Sukuriame `paskutinisMarsrutas` objektą ir priskiriame `dabartinisMiestas` kaip `MiestoPav`.
5. Pridedame `paskutinisMarsrutas` į `marsrutas` sąrašą.
6. Grąžiname `marsrutas` sąrašą, kuriame yra sudarytas maršrutas, pradedant nuo pradinio miesto ir vykstant per tinkamas keliones.

Antros užduoties skirtingų metodų rezultatų analizė uždavinio gerumo ir vykdymo laiko prasme esant kelioms skirtingoms pradinėms sąlygoms



Trečia užduoties dalis sprendimas

Trečia dalis. (~5 balai). Realizuoti programą, kuri pateiktų sprendinį taikant Genetinio Optimizavimo metodą. Programa pateikti rezultatą turi ne ilgiau, nei per 60 sek.

Trečios užduoties apskaičiuotas asimptotinis programos vykdymo laiko sudėtingumas

	Kaina	Kiekis
`public static List<PirmasisTable> SudarytiMaršrutaGenetinis1(List<AntrasisTable> keliones, string pradinisMiestas) – m(populiacijos dydis)`		
`{// Parametrai genetiniam algoritmui`		
`int populiacijosDydis = 50;`	c1	1
`double kryzminimoTikimybe = 0.8;`	c2	1
`double mutacijosTikimybe = 0.2;`	c3	1
`int maksimalusIteracijuSkaicius = 1000;`	c4	1
`Stopwatch stopwatch = new Stopwatch();`	c5	1
`stopwatch.Start();`	c6	1
`// Sukuriama pradinė populiacija`		
`List<List<PirmasisTable>> populiacija = GeneruotiPradiniusSprendinius(populiacijosDydis, keliones, pradinisMiestas);`	c7	m*n
`// Genetinio algoritmo iteracijos`		
`int iteracija = 0;`	c8	1
`while (iteracija < maksimalusIteracijuSkaicius){`	c9(1-x)	n
`// Įvertinami individai populiacijoje pagal tikslo funkciją`		
`List<double> tinkamumoReikšmės = ĮvertintiPopuliacija(populiacija, keliones);`	c10	m*n
`// Atrinkti geriausius individus`		
`List<List<PirmasisTable>> atrinktiIndividai = AtlrinktiIndividus(populiacija, tinkamumoReikšmės);`	c11	n*log(n)
`// Tikrinti ar pasiekėme geriausią rezultatą`		
`if (PatikrintiTerminavimoSąlygą(stopwatch))`	c12(1-y)	1
`{`		
`return atrinktiIndividai[0]; // Grąžiname geriausią rastą sprendimą`	c13(1-y)	1
`}`		
`// Sudaryti naują populiaciją kryžminant ir mutuojant individus`		
`populiacija = KryžminimasIrMutacija(atrinktiIndividai, kryzminimoTikimybe, mutacijosTikimybe);`	c14	n*m
`// Įvertinami individai populiacijoje pagal tikslo funkciją`		
`List<double> tinkamumoReikšmės = ĮvertintiPopuliacija(populiacija, keliones);`	c15	n*m
`// Atrinkti geriausius individus`		
`List<List<PirmasisTable>> atrinktiIndividai = AtlrinktiIndividus(populiacija, tinkamumoReikšmės);`	c16	n*m
`// Tikrinti ar pasiekėme geriausią rezultatą`		
`if (PatikrintiTerminavimoSąlygą(stopwatch))`	c17(1-z)	1
`{`		
`return atrinktiIndividai[0]; // Grąžiname geriausią rastą sprendimą`	c18	1
`}`		
`populiacija = KryžminimasIrMutacija(atrinktiIndividai, kryzminimoTikimybe, mutacijosTikimybe);`	c19	n*m

`iteracija++;}`	c20	n
`return populiacija[0]; // Grąžiname geriausią rastą sprendimą`	c21	1
}		

$$T(n) = c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 * (m * n) + c_8 + c_9(1 - x) * n + c_{11} * (n * \log(n)) + c_{10}(m * n) + c_{12}(1 - y) + c_{13}(1 - y) + c_{14}(n * m) + c_{15}(n * m) + c_{16}(n * m) + c_{17}(1 - z) + c_{18} + c_{19}(n * m) + c_{20}(n) + c_{21}$$

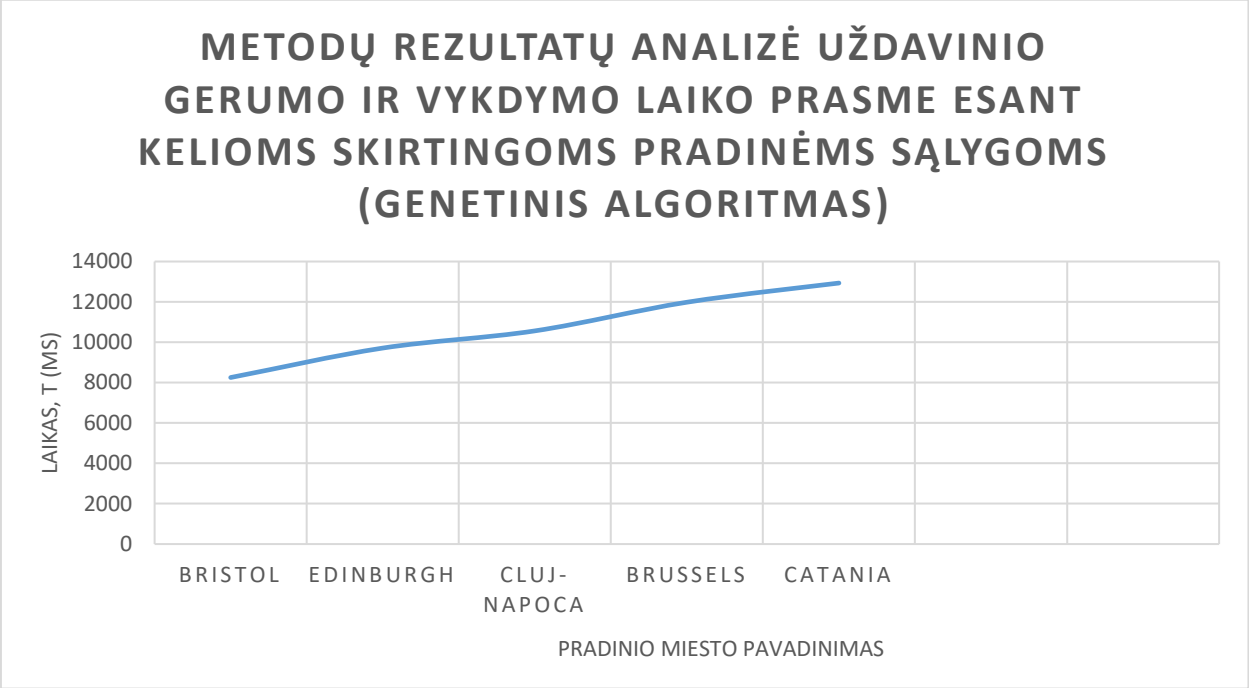
Viską suprastinus gauname, kad mūsų metodo asimptotinis sudėtingumas priklauso nuo iteracijų skaičiaus ir populiacijos dydžio:

$T(n) = n * m$, *kur n yra iteracijų skaičius, m-populiacijos dydis.*

Realizuotos trečios užduoties abstraktus aprašas „pseudo“ kodas

1. Nustatomos genetinio algoritmo parametrų reikšmės:
    - `populiacijosDydis` - populiacijos dydis (pirmųjų sprendinių skaičius)
    - `kryzminimoTikimybe` - tikimybė, kad vyks kryžminimas (crossover)
    - `mutacijosTikimybe` - tikimybė, kad vyks mutacija (mutation)
    - `maksimalusIteracijuSkaicius` - maksimalus iteracijų skaičius, po kurio baigsime vykdyti genetinį algoritmą
2. Pradedama laikyti laikas vykdymo trukmei su `Stopwatch` objektu.
3. Sukuriama pradinė populiacija naudojant `GeneruotiPradiniusSprendinius` metodą, kuris sugeneruoja pradinius sprendinius su atsitiktinai sudarytais maršrutais.
4. Pradedamos genetinio algoritmo iteracijos. Ciklas tęsiasi, kol nepasiekiamas maksimalus iteracijų skaičius.
5. Įvertinami individai populiacijoje pagal tikslo funkciją naudojant `ĮvertintiPopuliacija` metodą, kuris skaičiuoja tinkamumo reikšmes (fitness values) kiekvienam individui.
6. Atlrinkti geriausi individai naudojant `AtlrinktiIndividus` metodą, kuris išrenka geriausius individus pagal tinkamumo reikšmes.
7. Patikrinama, ar pasiektas terminavimo sąlyga. Šiuo atveju, naudojama `PatikrintiTerminavimoSąlygą` funkcija, kuri tikrina, ar vykdymo trukmė (laikas, kurį programa jau veikia) viršija tam tikrą ribą, pavyzdžiui, 60 sekundžių.
8. Jei pasiekiama terminavimo sąlyga, grąžinamas geriausias rastas sprendimas (maršrutas) iš atrinktų individų.
9. Jei terminavimo sąlyga nepasiekiama, vykdoma kryžminimas ir mutacija individams populiacijoje naudojant `KryžminimasIrMutacija` metodą, kuris parenka tėvus, kryžmina juos ir mutuoja gautus vaikus.
10. Vykdymo iteracijos skaičius padidinamas ir grįžtama į 5 žingsnį.
11. Jei pasiekiama maksimalus iteracijų skaičius, grąžinamas geriaus

Trečios užduoties skirtingų metodų rezultatų analizė uždavinio gerumo ir vykdymo laiko prasme esant kelioms skirtingoms pradinėms sąlygoms



Trečios užduoties pateikti rekomenduojami genetinio optimizavimo parametrai, užduoties analizė

```
// Parametrai genetiniam algoritmui – jeigu mūsų algoritmo vykdymo laikas ilgesnis
negu 60s. vykdymas sustabdomas.
int populiacijosDydis = 50;
double kryzminimoTikimybe = 0.8;
double mutacijosTikimybe = 0.2;
int maksimalusIteracijuSkaicius = 1000;
```

Šaltiniai

https://moodle.ktu.edu/course/view.php?id=2470