KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
PRAKTINĖS INFORMATIKOS KATEDRA

Laurynas Tamošiūnas

# Euristinių paieškos algoritmų tyrimas ir taikymas atviro kodo geografinėse informacinėse sistemose

Magistro darbas

Darbo vadovas
akad. J. Mockus
2011-05-

Kaunas, 2011

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS

PRAKTINĖS INFORMATIKOS KATEDRA

Laurynas Tamošiūnas

# Euristinių paieškos algoritmų tyrimas ir taikymas atviro kodo geografinėse informacinėse sistemose

Magistro darbas

Recenzentas
dr. G. Paulikas
2011-05-

Darbo vadovas
akad. J. Mockus
2011-05-

Atliko
IFM-9/1gr. Stud.
Laurynas Tamošiūnas
2011-05-20

Kaunas, 2011

# Turinys


1 Įvadas....1
1.1 Tyrimų tikslas ir uždaviniai....2
2 Poreikių tyrimas....3
3 Sukurtų produktų analizė....4
3.1 Navit navigatorius....5
3.2 ModRana navigatorius....6
3.3 Roadnav navigatorius....7
3.4 Tango GPS navigatorius....8
3.5 OpenStreetMap traveling-salesman navigatorius....9
3.6 GpsDrive navigatorius....10
3.7 Ištirtų produktų lyginamoji analizė....12
4 Euristiniai paieškos metodai....12
5 Algoritmų apžvalga....14
5.1 Keliaujančio pirklio uždavinys....14
5.1.1 Keliaujančio pirklio uždavinio apibrėžimas....15
5.1.2 Keliaujančio pirklio uždavinio pateikimas....15
5.2 Artimiausio kaimyno metodas....16
5.3 Modeliuotojo grūdinimo metodas....16
5.4 Tabu metodas....18
5.5 Skruzdėlių kolonijų optimizavimo metodas....21
6 Algoritmų tyrimas....23
6.1 Algoritmų efektyvumo priklausomybės nuo skaičiavimų trukmės tyrimas....23
6.1.1 Algoritmų veikimo laiko ir paklaidos vertinimas „lin105“ testu....24
6.1.2 Algoritmų veikimo laiko ir paklaidos vertinimas „berlin52“ testu....25
6.1.3 Algoritmų veikimo laiko ir paklaidos vertinimas „ch150“ testu....26
6.1.4 Algoritmų veikimo laiko ir paklaidos vertinimas „kroA100“ testu....27
6.1.5 Algoritmų veikimo laiko ir paklaidos vertinimas „kroD100“ testu....28
6.1.6 Algoritmų veikimo laiko ir paklaidos vertinimas ch130 testu....29
6.1.7 Algoritmų veikimo laiko ir paklaidos vertinimas „a280“ testu....30
6.1.8 Apibendrintas rezultatų įvertinimas....31
7 Išvados....32

8 Literatūra.....33
9 Paveiksliukų ir lentelių sąrašas.....34
9.1 Paveiksliukų sąrašas.....34
9.2 Lentelių sąrašas.....34
10 Terminų žodynėlis ir santrumpos.....35
11 Summary.....36
12 Priedas A. Apklausos rezultatai.....37

# 1 Įvadas

Žmonės, norėdami rasti patogiausią ar trumpiausią maršrutą, nuo seno naudojasi žemėlapiais. Didėjant gyvenimo tempui dažnas iš mūsų negali įsivaizduoti gyvenimo be žemėlapio. Neretai keliautojai savo gyvenime sunkiai išsiverčia ir be kompiuterinių navigacinių sistemų, padedančių rasti trumpiausią kelią iki tikslo ir nuolat prižiūrinčių, kad iš to kelio neišklystume. Navigacinės sistemos tikslas – išanalizuoti kiek įmanoma daugiau kelių kombinacijų, taip parenkant patį greičiausią, trumpiausią, pigiausią ar vaizdingiausią maršrutą. Šis uždavinys apibrėžiamas kaip keliaujančio pirklio (arba komivojažieriaus) problema.

Keliaujančio pirklio problema yra klasikinis grafų teorijos uždavinys, iškylantis daugelyje įvairių kelionių organizavimo atvejų. Pagrindinis šio uždavinio tikslas yra apskaičiuoti optimalų maršrutą svoriniame grafe, kai kelionė prasideda vienoje viršūnėje ir aplankius visas viršūnes po vieną kartą, sugrįžtama atgal į pirmą viršūnę. Ši problema sutinkama daugybėje realaus pasaulio sričių. Autobusų maršrutų planavimas, siuntų gabenimo planavimas ar kompiuterinių tinklų projektavimas taip pat gali būti sprendžiama kaip keliaujančio pirklio uždavinys. Netgi automatinėse gręžimo staklėse galima pritaikyti keliaujančio pirklio algoritmą darbui optimizuoti. Šiuo atveju miestai būtų skylės, o kelionės kaina būtų laiko trukmė, reikalinga grąžtui nukeliauti nuo vienos skylės prie kitos[6, 7].

Keliaujančio pirklio uždavinys yra standartinis kombinatorinis uždavinys, priklausantis NP sunkių algoritmų klasei[6, 8, 9], taigi sprendimui žinomi tikslaus rezultato algoritmai yra eksponentinio sudėtingumo ir labai greitai pasiekiamas toks uždavinio dydis (taškų skaičius), kai procesoriaus resursų jau nepakanka norint išanalizuoti turimus duomenis ir pateikti optimalų rezultatą per naudotoją tenkinantį laiką[7]. Didžiausias išspręstas uždavinys 1954 metais buvo 49 miestai (Dantzig et al., 1954) [6, 14, 15], o 2006 metais jau buvo išspręstas uždavinys su 85900 (Applegate et al., 2006)[13] miestų. Nors skaičiai ir įspūdingi, paskutinieji rezultatai pareikalavo tokio didelio kiekio skaičiuojamosios kompiuterių galios, jog uždavinio sprendimui vienu procesoriumi prireiktų 137 metų[6]. Taigi, norint rezultatus gauti greičiau, reikia naudoti spartesnius euristinius algoritmus, net jei juos naudojant nukenčia paieškos rezultatų tikslumas[9]. Panašiems uždaviniams spręsti yra sukurta nemažai euristinių algoritmų, kurie skaičiuoja optimaliausią įmanomą per tam tikrą laiką gauti rezultatą.

Euristiniai algoritmai yra naudojami optimizavimo uždaviniuose, ir jie padeda rasti aukštos kokybės, bet nebūtinai optimalų sprendinį per norimą skaičiavimo laiką. Euristiniai algoritmai

negarantuoja gautų sprendinių optimalumo, o surasti sprendiniai paprastai yra tik lokaliai optimalūs duotos aplinkos atžvilgiu. Tuo euristiniai metodai skiriasi tiek nuo tiksliųjų algoritmų (angl. „exact algorithms“), kurie garantuoja optimalaus sprendinio suradimą, tiek nuo apytikslių (aproksimacinių) algoritmų (angl. „approximate algorithms“), kurie užtikrina, kad gauto sprendinio kokybė skirsis nuo optimalaus sprendinio kokybės ne daugiau kaip iš anksto fiksuota paklaida, kuri yra didesnė už nulį.

Šiuo metu sprendžiant keliaujančio pirklio problemą, tiksliam rezultatui pasiekti reikia tokių skaičiavimo resursų, kurie dažnai paprastam vartotojui yra neprieinami. Tuo tarpu vartotojo poreikiai dažnai apsiriboja neypatingai dideliu duomenų kiekiu ir jį tenkina optimaliausias rezultatas, kurį galima pasiekti su turimais resursais. Todėl ir šiais laikais navigacinėse sistemose išlieka aktualus uždavinys – pateikti sprendinį turint ribotus skaičiavimo resursus.

## *1.1 Tyrimų tikslas ir uždaviniai*

Atsižvelgiant į tai, jog su keliaujančio pirklio funkcionalumu sukurtų produktų nėra daug, galima daryti išvadas, jog šioje srityje dar yra ką atrasti ar patobulinti. Taigi, yra keliamas tikslas išsiaiškinti įvairių algoritmų galimybes ir nuspręsti, kokį efektyviausią iš šiuo metu žinomų algoritmų būtų galima taikyti keliaujančio pirklio uždaviniui spręsti GPS navigatoriuose.

Navigaciniuose įtaisuose atminties ir skaičiavimų resursai yra labai riboti, o taikant euristinius algoritmus, keliaujančio pirklio problemos sprendimo rezultatų kokybė priklauso nuo turimų resursų. Todėl, reikia parinkti tokį algoritmą, kuris optimaliai išnaudodamas turimus resursus, gautų geriausią rezultatą.

Taigi užsibrėžiamas tikslas yra išanalizuoti žinomus algoritmus keliaujančio pirklio problemai spręsti ir parinkti tinkamiausią algoritmą bei optimizuoti jo parametrus pagal turimus atminties ir skaičiavimo resursus bei problemos sudėtingumą.

Keliami šie uždaviniai:

- ištirti vartotojų poreikius bei reikalavimus keliaujančio pirklio funkcionalumui;
- pasirinkti programinius paketus ir ištirti jų funkcines galimybes keliaujančio pirklio problemos realizacijai;
- nustatyti kurį algoritmą tikslingiausia naudoti navigacinėse sistemose.

Viena iš šio darbo sąlygų – algoritmų tyrimui nagrinėjamos galimybės panaudoti tik atviro kodo sistemas, tuo siekiant prisidėti prie nemokamų ir visiems prieinamų sistemų populiarinimo. Programa

gali būti laikoma atviro kodo tik tada, kai jos išeitinį kodą galima laisvai studijuoti, tobulinti, kopijuoti ar platinti. Tai sudaro galimybę bet kuriam visuomenės nariui prisidėti prie produkto vystymo, modifikavimo, ar pritaikymo savo reikmėms.

# 2 Poreikių tyrimas

Buvo apžvelgti rinkoje esantys produktai, siekiant nustatyti, ar yra navigatorių, su keliaujančio pirklio funkcija. Po įvairių navigacinių sistemų apžvalgos paaiškėjo, jog keletas Garmin firmos navigatorių gali pasiūlyti keliaujančio pirklio funkcionalumą. Tuo tarpu nerasta nei vieno nemokamo analogo. Taigi, buvo nuspręsta ištirti, kiek reikalingas toks nemokamas įrankis.

Norint nustatyti kiek dažnai žmonės susiduria su situacijomis, kai reikia nuvykti į tokį kiekį adresų, jog be kompiuterio pagalbos sunku nusistatyti optimalų maršrutą, buvo atlikta nedidelė apklausa (žiūrėti priedą A). Analizuojant apklausos rezultatus galime įžvelgti, jog žmonės arba nelabai dažnai susiduria su šia problema, arba ji nėra tokia svarbi. Todėl jie nėra linkę išleisti navigatoriui kelis šimtus litų. Tuo tarpu nemaža dalis apklaustųjų nurodė, jog naudotųsi šia funkcija, jei galėtų ją nemokamai turėti savo telefone.

Norint realizuoti tokį funkcionalumą mobiliame įtaise, reiktų išsiaiškinti kiek efektyvūs keliaujančio pirklio algoritmai, esant ribotus skaičiavimų, atminties ir laiko resursus.

Apžvelgus ankstesnių tyrimų rezultatus nepavyko rasti detalių lyginamųjų tyrimų, kaip skirtingų algoritmų efektyvumas priklauso nuo skaičiavimo resursų. Norint išrinkti tinkamiausią algoritmą šiai problemai spręsti, tokie tyrimai yra būtini.

Taip pat reiktų apžvelgti, kiek realu sukurti tokį įrankį ir pateikti jį vartotojams nemokamai. Pilnaverčio navigatoriaus projektavimas ir sukūrimas iš pagrindų užtruktų nemažai laiko ir pareikalautų didelių resursų. Reikia ieškoti pigesnės alternatyvos. Viena iš išeičių yra sukurti įskiepį, kuris spręstų šią problemą ir jį suintegruoti su jau esamu navigatoriumi.

# 3 Sukurtų produktų analizė

Analizei pasirinkti nemokami produktai, nes vienas iš tikslų yra ištirti galimybę nemokamo įrankio sukūrimui, vartotojams, kurie su keliaujančio pirklio problema susiduria tik retkarčiais ir nelinkę į šios problemos sprendimą investuoti pinigų.

Sekantis iškeltas reikalavimas – pasirinkti atviro kodo, laisvai modifikuojami produktai, nes integracija su uždaro kodo produktais, kai jie nėra pritaikyti įskiepiams, beveik neįmanoma dėl didelių produkto perdarymo ir pritaikymo kaštų, arba teisiškai negalima dėl licencinių apribojimų.

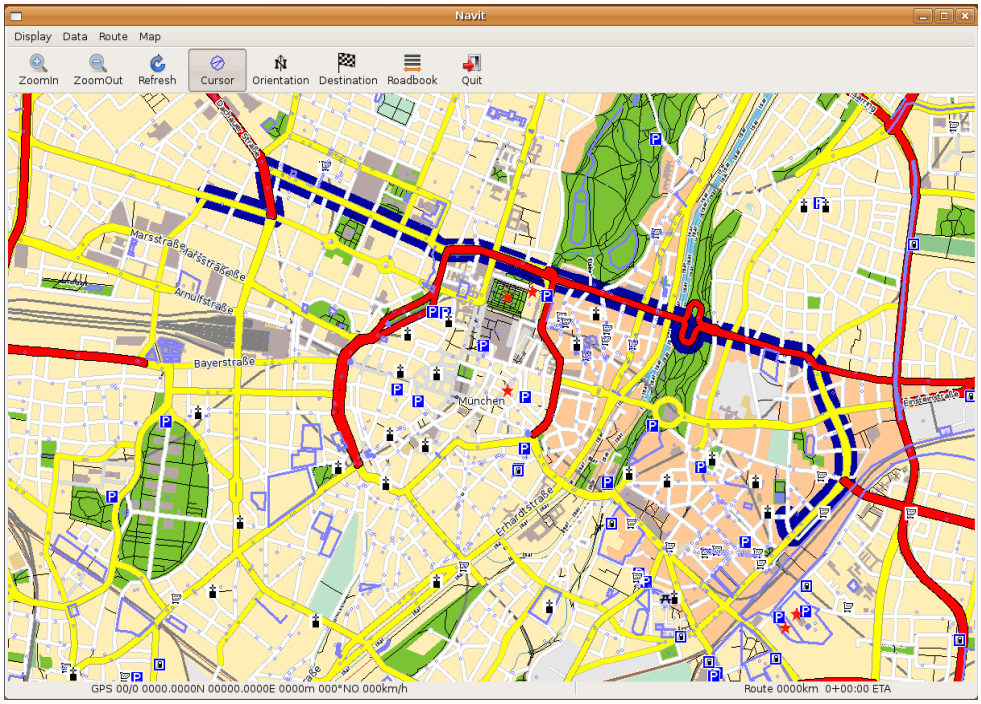
Kadangi keliaujančio pirklio uždavinio vienas iš pradinių duomenų yra atstumų tarp visų taškų vektorius, būtina šį vektorių apskaičiuoti pačiame navigatoriuje su tikslu naudoti keliaujančio pirklio įskiepyje, kaip keliaujančio pirklio uždavinio vieną iš pradinių duomenų. Dėl to reikia ištirti galimybę gauti iš navigatoriaus šiuos duomenis per priimtiną laiką. Taigi būtina atlikti atstumo tarp dviejų žemėlapio taškų skaičiavimo trukmės įvertinimą. Išsiaiškinti, ar tokiam funkcionalumui nėra būtina perdaryti paties navigatoriaus paieškos funkcijų, kadangi keliaujančio pirklio uždaviniui spręsti reikalingas pilnas kelionės tikslų atstumų grafas.

## 3.1 Navit navigatorius

Navit yra aktyviai kuriamas projektas, kol kas gan sudėtingai konfigūruojamas per XML konfigūracinį failą, tačiau toks konfigūravimo būdas suteikia labai daug galimybių, tereikia tik įsigilinti, kaip viskas veikia. Kaip ir daugelis kitų navigatorių, naudoja gpsd demoną informacijai apie esamą poziciją nustatyti.

Navit navigatorius parašytas C programavimo kalba, naudoja vektorinius žemėlapius ir trumpiausio maršruto paieškai naudoja Deikstros algoritmą. Atlikus bandymus Vilniaus mieste, atstumo tarp dviejų taškų vidutinė skaičiavimo trukmė buvo 1,4 sekundės.

Pritaikytas naudoti OpenStreetMap, Garmin, Grosser Reiseplanner ar Routenplanner žemėlapius. Įprastai naudojantis su Lietuvos žemėlapiu programa reikalauja apie 13 MB operatyviosios atminties.




*1 pav. Navit programos ekrano vaizdas.*

Taigi, pagrindinės programos funkcijos (privalumai):

- keliavimo būdo pasirinkimas (pėsčiomis, dviračiu, automobiliu, autobusu, traukiniu) ;
- atstumų tarp pasirinktų taškų apskaičiavimas;
- yra integruotas kompasas;

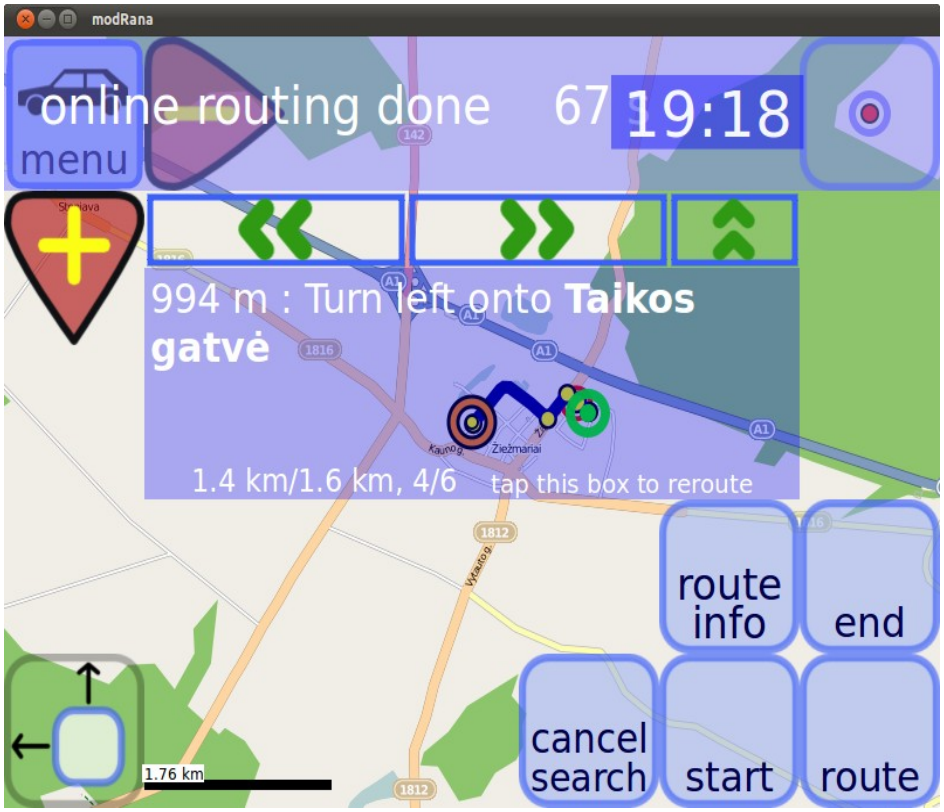
- automatiškai pasukamas žemėlapis pagal važiavimo kryptį;
- nukeliauto maršruto saugojimas žurnale;
- automatinis greičiausio kelio skaičiavimas;
- skaičiuojama kiek laiko truks kelionė ir kurią valandą bus pasiektas tikslas;
- sąsaja pritaikyta lietimui jautriems ekranams ir valdymui pirštais;
- 2d/3d režimai.

Trūkumai:

- sunkus instaliacijos, konfigūravimo procesas.

## 3.2 ModRana navigatorius

Šis Python programavimo kalba sukurtas navigatorius yra patogios grafinės vartotojo sąsajos ir turi palyginti daug funkcijų. Žemėlapiai yra parsiunčiami iš OpenStreetMap žemėlapių duomenų bazės. Navigatorius lengvai pritaikomas mobiliems telefonams su Linux branduoliu pagrindu veikiančiomis operacinėmis sistemomis.




*2 pav. ModRana programos ekrano vaizdas.*

Taip pat navigatoriumi galima patogiai dirbti ir delniniais bei nešiojamaisiais kompiuteriais, turinčiais liečiamuosius ekranus, bei Linux operacinę sistemą. Navigatorius dirba su vektoriniais žemėlapiais ir turi maršrutų skaičiavimo galimybes.

Pagrindinės navigatoriaus funkcijos (privalumai):

- keliavimo būdo pasirinkimas (pėsčiomis, dviračiu, automobiliu, autobusu, traukiniu) ;
- gana išsamios informacijos apie maršrutą suteikimas (pilni adresai, kelionės atstumas, trukmė, koordinatės);
- pasirinkimas iš kur pradėti kelionę – ar iš taško artimiausio dabartiniam, ar iš sekančio nurodyto maršrute;
- galimybė naudoti „turn – by – turn“ navigaciją – kaskart ties kitu maršrute esančiu punktu atsiranda žymė, padedanti pastebėti posūkius ir kitas svarbias kelionei gaires;
- yra galimybė gauti netgi su eismo saugumu ir kelio ženklinimu susijusią informaciją;
- įgyvendinta galimybė išsisaugoti kelionės maršrutą, jau įveikto kelio duomenis ir t.t.;

Trūkumas – vėluojantis palaikymas. Nors seno gpsd protokolo palaikymas jau nutrauktas, navigatorius vis dar nepalaiko naujos protokolo versijos.

## *3.3 Roadnav navigatorius*

Aktyvus navigatoriaus vystymas nutrūko apie 2008 metus, todėl šis navigatorius nepalaiko naujos gpsd protokolo versijos, kas apsunkina navigatoriaus naudojimą. Be to, yra problemų su Europos žemėlapių įdiegimu. Dėl šių priežasčių nepavyko detaliau išsianalizuoti navigatoriaus savybių.

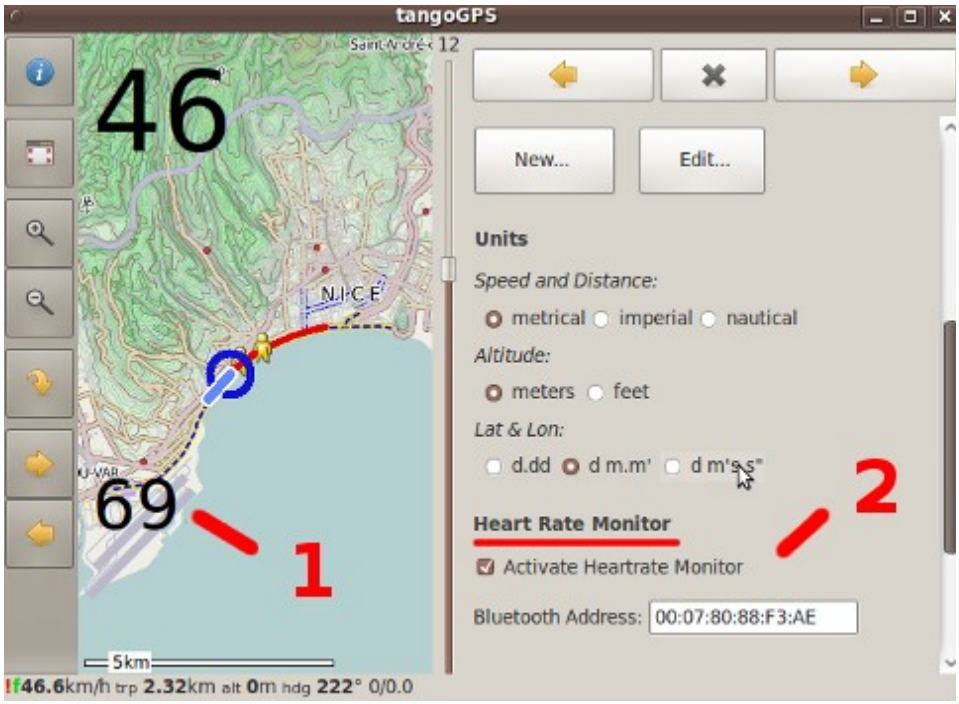
Čia pateikiamos kūrėjų svetainėje skelbiamas navigatoriaus funkcijos:

- generuoja gatvių lygio žemėlapius Jungtinių Amerikos valstijų teritorijai;
- realiu laiku rodo dabartinę buvimo vietą;
- balsu nurodo keliavimo kryptį į pasirinktą vietą žemėlapyje, automatiškai perskaičiuoja; nurodymus jeigu pravažiuojamas posūkis;
- palaiko liečiamųjų ekranų technologiją;
- trimatis žemėlapio vaizdas iš vairuotojo perspektyvos;
- dienos ir nakties žemėlapio spalvų deriniai;
- automatinis dienos/nakties režimų perjungimas;
- parodo informaciją apie netoliese esančius įstabaus kraštovaizdžio objektus ir lankytinas vietas;

- gali veikti neprisijungęs prie interneto;
- palaiko kelias operacines sistemas, tarp kurių yra ir Windows, Linux bei MAC OS X.;
- palaiko ekrano temų naudojimą;
- per LCD procesorių gali išvesti būsenos informaciją į LCD ekranus turinčius įrenginius;
- palaiko GPX ir OpenStreetMap (ši funkcija yra eksperimentinėje stadijoje).

## 3.4 Tango GPS navigatorius

TangoGPS yra labai paprasta programa, naudojanti žemėlapio paveiksliukus, vadinamus *tiles*. Programa puikiai tinka peržiūrėti žemėlapiams dideliu masteliu. Navigatorius parašytas naudojant C programavimo kalbą. Kaip ir daugelis kitų GPS navigatorių naudoja gpsd demoną, informacijai apie einamąją poziciją nustatyti. Būtent dėl to, kad programa naudoja žemėlapio paveiksliukus, o ne vektorinius žemėlapius, programa neturi galimybių ieškoti maršruto, be to reikalauja nemažo interneto srauto, kadangi visus žemėlapius siunčiasi iš OpenStreetMap, Google ar kitų, tokias paslaugas tiekiančių serverių.




*3 pav. TangoGPS programos ekrano vaizdas.*

Palaikantys įrenginiai:

- Neo FreeRunner;

- Telefonai su Maemo operacine sistema.

Pagrindinės programos funkcijos (privalumai):

- paprasta instaliacija ir sklandus veikimas;
- judėjimo greičio vaizdavimas;
- nukeliauto maršruto saugojimas žurnale;
- širdies ritmo saugojimas susiejant jį su geografinėmis koordinatėmis ir laiku;
- pozicijos dalinimosi funkcija;
- galimybė keisti žemėlapių šaltinį.

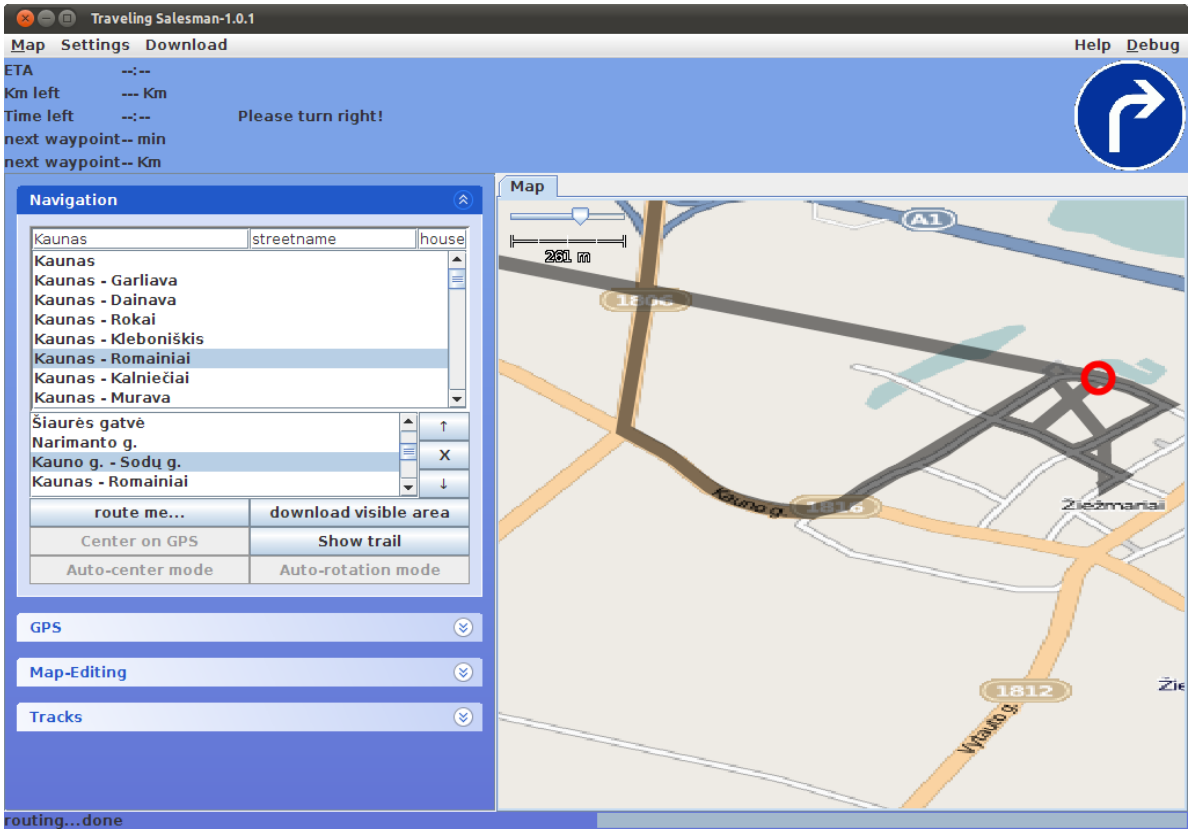
Trūkumai:

- nėra galimybės skaičiuoti maršrutą iki adreso ir nukreipti vartotoją artimiausiu keliu.

## *3.5 OpenStreetMap traveling-salesman navigatorius*

Traveling-salesman navigatorius naudoja vektorinius žemėlapius iš OpenStreetMap. Navigatorius parašytas Java programavimo kalba.

Atlikus bandymus Vilniaus mieste, parenkant skirtingus atstumo skaičiavimo algoritmus, atstumo tarp dviejų taškų vidutinė skaičiavimo trukmė buvo 2,2 sekundės.




*4 pav. traveling-salesman programos ekrano vaizdas.*

Navigatorius turi maršruto skaičiavimo funkcionalumą, bei galimybę vienu metu įvesti keletą norimų aplankyti adresų. Bet ši savybė labai daug vertės navigatoriui neprideda, kadangi įvestus taškus navigatorius siūlo aplankyti ta tvarka, kurią įveda vartotojas ir neturi maršruto optimizavimo funkcionalumo.

Pagrindinės programos funkcijos (privalumai):

- maršruto skaičiavimas.

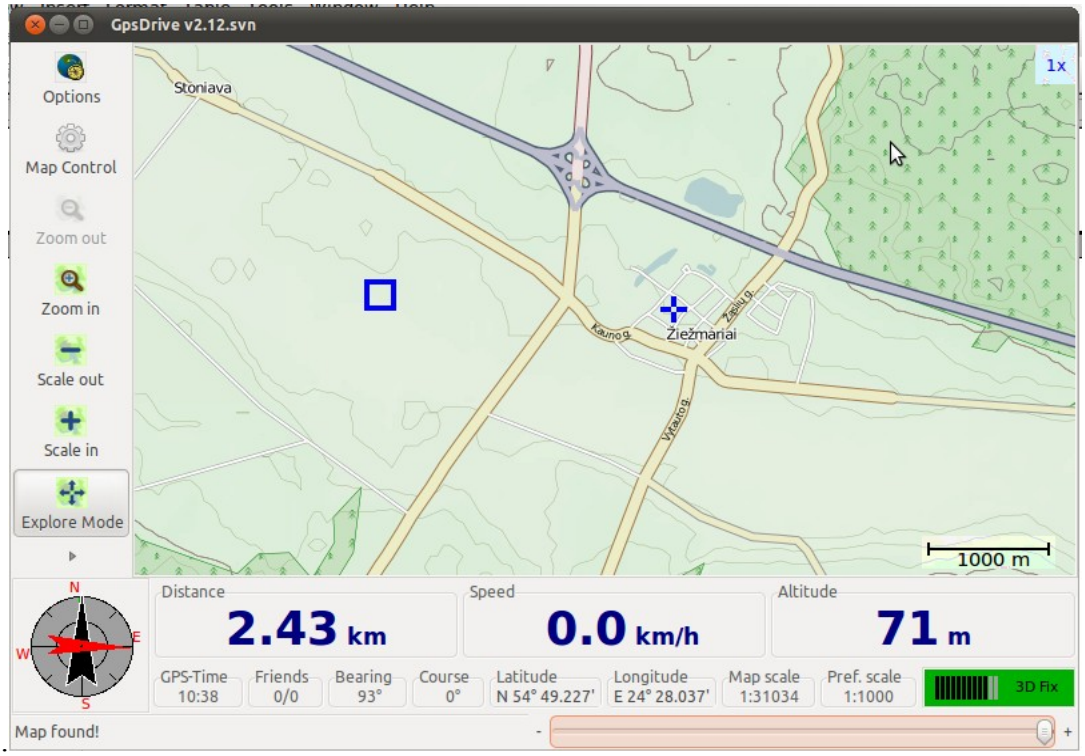
Trūkumai:

- sudėtingas instaliacijos, konfigūracijos procesas;
- produktas dar tik kūrimo stadijoje, taigi besinaudojant dažnai rodomi klaidų pranešimai;
- nepatogi vartotojo sąsaja.

## 3.6 GpsDrive navigatorius

GpsDrive, kaip ir TangoGPS, naudoja žemėlapio paveiksliukus (tiles), kuriuos galima ne tik atsisiųsti iš OpenStreetMap ar Google Maps, bet ir įsidėti pačiam png, jpeg, gif, tiff ar kito plačiai

paplitusio paveiksliukų formato. Ši navigatoriaus savybė taip pat labai riboja navigatoriaus galimybes, kadangi neturint vektorinio žemėlapio navigatorius neskaičiuoja maršruto, ir tinkamas naudoti daugiau žemėlapių peržiūrai, nei kaip navigatorius keliaujant




*5 pav. GpsDrive programos ekrano vaizdas.*

Pagrindinės programos funkcijos (privalumai):

- judėjimo greičio, atstumo iki tikslo vaizdavimas;
- integruotas kompasas;
- nukeliauto maršruto saugojimas žurnale;
- pozicijos dalinimosi funkcija;
- galimybė keisti žemėlapių šaltinį;

Trūkumai:

- nėra galimybės skaičiuoti maršrutą iki adreso ir nukreipti vartotoją artimiausiu keliu;
- sudėtingas instaliacijos ir konfigūracijos procesas;
- dažni programos nulūžimai (pavyzdžiui „segmentation fault“ klaidos).

### 3.7 Ištirtų produktų lyginamoji analizė

Atlikus nemokamų navigacinių sistemų funkcijų analizę paaiškėjo, jog nei vienas iš jų neturi keliaujančio pirklio funkcionalumo. Taigi imtasi nagrinėti keliaujančio pirklio funkcionalumo pritaikymo galimybes vienai iš tiriamų nemokamų navigacinių sistemų.

Atlikus esamų produktų analizę aiškiai, pagal naudojamus žemėlapius, visus navigatorius galime skirstyti į dvi grupes – tai žemėlapių paveiksliukus naudojančius ir vektorinius žemėlapius naudojančius navigatorius. Būtent vektorinius žemėlapius naudojantys navigatoriai ir buvo panagrinėti plačiau, nes vektorinių žemėlapių neturintys navigatoriai neturi galimybių skaičiuoti maršrutų.

Atsižvelgiant į tai, jog traveling–salesman navigatorius vienintelis iš analizuotų navigatorių turi įgyvendintą maršrutų paiešką per kelis taškus, paskatino į tai atkreipti daugiau dėmesio. Paanalizavus detaliau paaiškėjo, jog funkcija nėra išbaigta, nes neskaičiuoja optimalaus maršruto visų taškų pasiekimui, o tiesiog veda per pasirinktus taškus tokia tvarka, kuria įveda naudotojas, kas realiai nesprendžia keliaujančio pirklio problemos. Taigi, kilo idėja patobulinti navigatorių taip, kad ieškotų optimalaus maršruto per pasirinktu taškus. Be to, atlikus detalesnę analizę, pasirodė, kad pats navigatorius reikalauja labai daug kompiuterio resursų, dėl ko sunkiai pritaikomas mobiliems įrenginiams.

Atlikus bandymus su Vilniaus miesto žemėlapiu, paaiškėjo, jog geriausius rezultatus rodęs navigatorius Navit, šiam atstumui paskaičiuoti užtrunka vidutiniškai 1,4 sekundės. Šis parametras nagrinėjamas dėl to, kad vienas iš keliaujančio pirklio pradinių duomenų yra atstumų tarp visų taškų matrica. Atstumų matricos paieška navigatoriuje užtruktų $k * (n - 1)^2$ laiko vienetų, kai n – taškų skaičius, o k – vidutinė atstumo tarp dviejų taškų skaičiavimo trukmė. Taigi nesunkiai galime paskaičiuoti, jog turint 11 taškų rezultatų reiktų laukti apie 140 sekundžių, o jau turint 101 tašką gali tekti laukti ilgiau nei 3 valandas. Taigi galime prieiti prie išvadų, jog su naudojamais navigacinių įtaisų algoritmais surasti visas grafo briaunas pareikalautų labai didelių skaičiavimo resursų, dėl keliaujančio pirklio funkcionalumui būtina pritaikyti esamą kelio paieškos algoritmą efektyvesniam skaičiavimui.

# 4 Euristiniai paieškos metodai

Euristiniai paieškos metodai tapo labai svarbūs moksliniu požiūriu tuomet, kai atsirado sričių, kur standartiniai kombinatoriniai skaičiavimo algoritmai dėl didelės duomenų imties tapo netinkami arba tiksliau netinkami naudojimui. Netinkami, nes didėjant uždavinio duomenų imčiai, ilgėja skaičiavimų

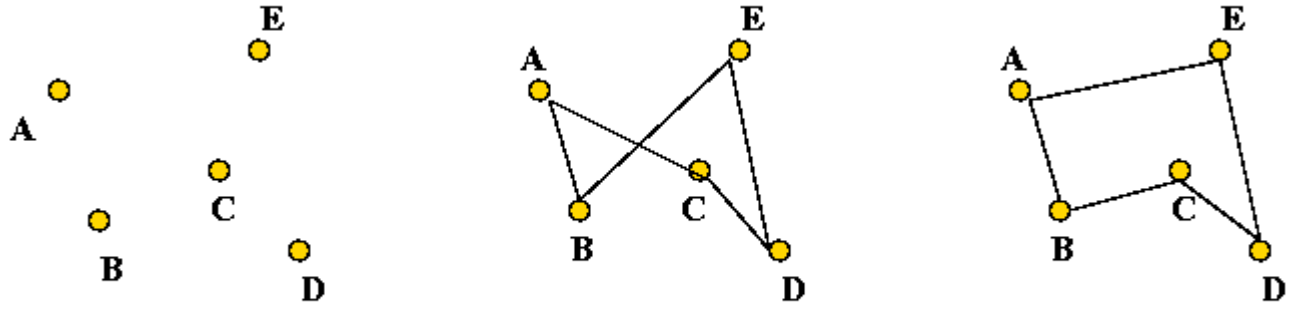
trukmė, o kombinatorinių perrinkimo (angl. „brute force“) algoritmų sudėtingumas dažnai būna eksponentiškas ir labai greitai pasiekiama ta riba, kai su turimais resursais uždavinio sprendimo laikas mūsų nebetenkina. Taigi didelio sudėtingumo optimizavimo uždaviniams spręsti naudojami euristiniai paieškos metodai. Euristiniai metodai dažnai negali pasiūlyti optimaliausio sprendimo, bet praktiškai sprendimas gaunamas artimas optimaliam.

Skirtingai nei nuoseklaus perrinkimo algoritmai, kurie peržiūri visas įmanomas kombinacijas ir atrenka pačią optimaliausią, euristiniai algoritmai nuosekliai bando privesti sprendimą prie optimalaus iš pradžių pasirinkę pirmą pasitaikiusį.

# 5 Algoritmų apžvalga

## *5.1 Keliaujančio pirklio uždavinys*

Keliaujančio pirklio problema buvo pradėta plačiai tyrinėti apie 1930-uosius metus, kai ja ypač susidomėjo Princtono universiteto profesoriai Bob Singleton ir Hassler Whitney. Ši problema buvo aktuali jau ilgą laiką, nuo pat išvežiojamosios prekybos atsiradimo.




*6 pav. Keliaujančio pirklio problemos sprendimas. Iš kairės 1 – pradiniai duomenys, 2 – neoptimalus sprendimas, 3 – optimalus sprendimas.*

Keliaujančio pirklio (komivojažieriaus) uždavinys formuluojamas taip: turint tam tikrą miestų kiekį ir kelionės iš vieno miesto į kitą kainas, reikia rasti pigiausią maršrutą, kad aplankius kiekvieną miestą po vieną kartą maršrutas baigtųsi pradiniame mieste. Grafų teorijoje galima uždavinį performuluoti – kaip rasti mažiausio svorio Hamiltono ciklą grafe su svoriais.

### 5.1.1 Keliaujančio pirklio uždavinio apibrėžimas

Duota miestų aibė $n_\pi$ ir tikslas rasti mažiausią įmanomą Hamiltono maršrutą. Pažymėkim $\pi$ – uždavinio sprendinio miestų kombinacija $\{1, \ldots, n_\pi\}$, kur $\pi(i)$ yra nuoroda į i-tąjį aplankytą miestą. Tuomet $\Pi(n_\pi)$ yra visų įmanomų $\{1, \ldots, n_\pi\}$ kombinacijų aibė arba paieškos erdvė. Formaliai keliaujančio pirklio problema aprašoma kaip optimali kombinacija $\pi^*$, kur

$$\pi^* = arg \min_{\pi \in \Pi(n_\pi)} f(\pi) \quad (1)$$

kur

$$f(\pi) = \sum_{i,j=1}^{n_\pi} d_{ij} \quad (2)$$

yra tikslo funkcija ir $d_{ij}$ – atstumas tarp taškų i ir j. Pažymėkime $D = [d_{ij}]_{n_\pi \times n_\pi}$ – atstumų matrica.

Dvi keliaujančio pirklio uždavinio versijos yra apibrėžiamos atstumų matricos charakteristikomis:

Jei $d_{ij} = d_{ji}$, kai $i, j = 1, \ldots, n_\pi$ – problema apibrėžiama kaip simetrinis keliaujančio pirklio uždavinys

Jei $d_{ij} \neq d_{ji}$, kaip $i, j = 1, \ldots, n_\pi$ – asimetrinis keliaujančio pirklio uždavinys[10].

### 5.1.2 Keliaujančio pirklio uždavinio pateikimas

Problema susideda iš trijų elementų, G = (V, E, D), kur V – taškų aibė, kai kiekvienas taškas atvaizduoja miestą. E – jungtys tarp miestų ir D – atstumų matrica, kur priskiria svorį (atstumą) kiekvienai grafo jungčiai $(i, j) \in E$

Sprendimas pateikiamas kaip aibė elementų $\pi = (1, 2, \ldots, n_\pi)$, kurie nurodo miestų apkeliavimo eiliškumą.[10]

## 5.2 Artimiausio kaimyno metodas

Artimiausio kaimyno metodas (angl. „nearest neighbor“) yra vienas iš paprasčiausių „greedy“ keliaujančio pirklio problemai spręsti. Algoritmas labai paprastas ir jį galime aprašyti šiais žingsniais:

1. Pasirenkame pirmą viršūnę ir ją įrašome į eilę ir pažymime ją kaip einamąją.
2. Išrenkame artimiausią, einamajai ir dar neaplankytą viršūnę, įrašome ją į eilę ir pažymime kaip einamąją.
3. Jei dar yra neaplankytų viršūnių, keliaujame prie (2) punkto.
4. Įrašome pirmą viršūnę į eilę.

Algoritmo sudėtingumas $O(n^2)$, kai n – taškų skaičius. Algoritmas labai paprastas, nors rezultatai kartais gali būti nuviliantys, nes egzistuoja situacijų, kai šis algoritmas pateikia blogiausią įmanomą sprendimą. Taigi, algoritmas yra labai spartus šio tyrimo kontekste ir rezultatus vartotojui gali pateikti per labai trumpą laiką, bet dėl rezultatų neefektyvumo nebus detaliau nagrinėjamas.

## 5.3 Modeliuotojo grūdinimo metodas

Šį metodą nepriklausomai vienas nuo kito apibūdino Scott Kirkpatrick, C. Daniel Gelatt ir Mario P. Vecchi 1983 metais ir Vlado Černý 1985 metais.

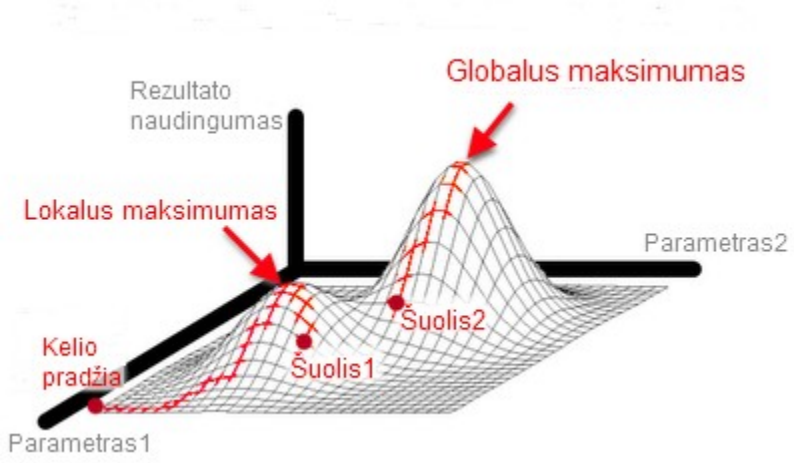
Modeliuotojo grūdinimo metodas yra skirtas įvairaus tipo globalaus optimizavimo uždaviniams spręsti. Jo tikslas – rasti duotosios funkcijos, turinčios didelę galimų sprendinių aibę, sprendinį, pakankamai artimą globaliam optimaliam sprendiniui (įprastai tai yra kažkokios funkcijos minimali arba maksimali reikšmė). Pagrindinis algoritmo apribojimas yra laikas, kuris yra skirtas pakankamai tikslaus sprendinio radimui. Algoritmui skyrus pakankamai daug laiko, įmanoma rasti net tikslią globalaus optimumo reikšmę, tačiau įprastai tai yra nepraktiška, nes šis laikas dažniausiai ilgesnis už laiką, kurio reiktų pereiti visus galimus sprendinius pilno perrinkimo metodu.

Tiek algoritmo pavadinimas (iš anglų kalbos – Modeliuotojo grūdinimo), tiek veikimo principas yra paremti natūraliais procesais, vykstančias grūdinant metalus. Atliekant grūdinimą, tam tikras metalas įkaitinamas iki aukštesnės nei kristalizavimosi temperatūros ir paliekamas vėsti. Jei vėsimo procesas trunka pakankamai lėtai, metalas tampa patvaresnis ir turintis mažiau defektų (metalas kristalizuojasi į stambius kristalus), nes įkaitę metalo atomai turi daugiau galimybių išsidėstyti taip, kad būtų sumažinta bendra vidinė medžiagos energija. Jei vėsimas vyktų greitai, atomai neturėtų pakankamai laiko pereiti į

globalaus optimumo būsenas vidinės energijos atžvilgiu, o liktų tik lokalaus minimumo zonose, kurios dažnai nesutampa su globaliu minimumu.

Analogiškai fiziniam grūdinimui, modeliuotojo grūdinimo metu kiekvienoje iteracijoje atsitiktine tvarka pasirenkamas naujas „gretimas“ sprendinys, pamažu artėjant prie globalaus optimumo reikšmės. Tikimybė, jog bus pasirinktas tam tikras naujas sprendinys, priklauso nuo dabartinio ir naujo sprendinio reikšmių skirtumo bei nuo globalaus parametro, vadinamo temperatūra, kuris kiekvienoje naujoje iteracijoje vis mažėja, kol galiausiai pasiekia nulį. Kol temperatūros parametras yra aukštas, yra ganėtinai didelė tikimybė pasirinkti mažiau optimalius sprendinius, taip siekiant išvengti strigimo tam tikrame lokaliame optimume. Mažėjant temperatūros parametro reikšmei, algoritmo elgesys panašėja į paprastą godųjį algoritmą, sprendinys pradeda artėti prie lokalaus optimumo ir nusistovi.

Kiekvienas duotosios funkcijos, kuriai taikomas Modeliuotojo grūdinimo metodas, sprendinių erdvės taškas vadinamas būsena, o optimizuojančioji funkcija prilyginama vidiniai fizinės sistemos energijai. Metodo tikslas – priversti sistemą pereiti iš pradinės būsenos į būseną su mažiausia galima vidine energija.




*7 pav. Lokalaus bei globalaus optimumo iliustracija.*

„Būsenos kaimynais“ vadinamos naujos būsenos, kurios gaunamos sistemą paveikus tam tikru būdu. Pavyzdžiui, keliaujančio pirklio problemos atveju, kiekviena būsena apibūdinama kaip tam tikras maršrutas per visus miestus. Kaimyniniai maršrutai (būsenos) gaunami apkeitus tam tikrą gretimų miestų porą. Veiksmas, atliekamas siekiant rasti kaimynes būsenas, vadinamas „ėjimu“, ir skirtingi ėjimai perveda sistemą į skirtingas kaimynines būsenas. Įprastai šie ėjimai sistemą pakeičia minimaliai,

taip padėdami algoritmui optimizuoti sprendimą kiek įmanoma labiau, tuo pačiu išlaikant optimalias sprendimo dalis nepaliestas arba mažai paliestas.

Modeliuotojo grūdinimo metodo pseudokodas:

```
Pradiniai duomenys: ProblemosDydis, Temp_maksimali
Rezultatai: S_geriausias
S_einamaisis ← SukonstruotiPradinįSąrašą()
S_geriausias ← S_einamaisis
while (¬Baigimo sąlyga)
   S_i← SurastiArtimąSprendinį(S_einamasis)
   Temp_einama← PaskaičiuotiTemperatura(i, Temp_maksimali)
   if(ilgis(S_i) <= ilgis(S_einamas))
      S_einamas ← S_i
      if(ilgis(S_i) <= ilgis(S_geriausias))
         S_geriausias ← S_i
      end
   end
   elseif (Exp( (ilgis(S_einamas) − ilgis(S_i)) / Temp_einama > Rand()))
      S_einamas ← S_i
   end
end
```

Perėjimo į naują būseną *s′* iš pradinės būsenos *s* tikimybę nusako funkcija *P(e, e′, T)*, priklausanti nuo būsenų energijas nusakančių funkcijų *e = E(s)* ir *e′ = E(s′)* bei nuo kintančio laike parametro *T*, vadinamu temperatūra. Vienas iš esminių reikalavimų, keliamų funkcijai *P* yra tai, kad jos reikšmė turi būti nelygi nuliui tuomet, kai *e′ > e*, t.y. turėtų egzistuoti tikimybę, kad bus pereita prie prastesnio sprendinio, taip užtikrinant galimybę išeiti iš lokalaus minimumo, prastesnio už globalųjį, zonos. Kita vertus, kai *T* parametro reikšmė yra 0, *P* funkcijos reikšmė turėtų būti lyg 0, jei *e′ > e,* ir teigiama, jei *e′ < e*. Taip pat svarbi funkcijos *P* savybė yra ta, jog jos reikšmė mažėja, didėjant skirtumui *e′ – e*, t.y. perėjimai labiau tikėtini į „gretimas“ sistemos energijos būsenas, o didesni šuoliai pasitaiko rečiau. Žinoma, kai kurios funkcijos *P* savybės gali kisti priklausomai nuo konkrečios algoritmo realizacijos variacijos.

## 5.4 Tabu metodas

Tabu paieška yra meta-euristinis, globalaus optimizavimo algoritmas, taikomas daugybėje uždavinių. Tokių kaip tvarkaraščių optimizavimo, kuprinės uždavinio, keliaujančio pirklio uždavinio[1, 2]. Metodas pirmą kartą pristatytas 1986 metais Glover[3] ir dėka didelio metodo efektyvumo labai

greit išpopuliarėjo ir buvo imtas taikyti daugelyje didelių kombinatorinių uždavinių sprendimui. Paaiškėjo, jog tabu metodu sprendžiamų problemų rezultatai labai artimi optimaliems arba kai kuriais atvejais algoritmas yra pas efektyviausias galimas[3]. Būtent dėl metodo efektyvumo jis taip ir išpopuliarėjo, buvo plačiai nagrinėjamas, dėl ko atsirado nemažai metodo pritaikymo galimybių bei modifikacijų[4].

Kaip ir kitų nagrinėjamų metodų, taip ir tabu metodo sprendinių paieškos laukas yra visų galimų Hamiltono ciklų aibė. Algoritmas tęsią paiešką ir tuomet, kai lokalus optimumas (7 pav.) jau rastas ir sprendinio aplinkoje jau nebeįmanoma rasti geresnio sprendimo. [5] Siekiant surasti globalų minimumą metodas leidžia atlikti pakeitimus net ir tuomet, kai gauti rezultatai yra prastesni už turimus. Tokiu būdu pereidinėjant nuo vieno lokalaus optimumo prie kito ieškoma globalaus optimumo.

Šio metodo išskirtinumas prieš kitus nagrinėtus metodus yra tai, jog metodas įsimena nenaudingus perėjimus, taip padedant išvengti užsiciklinimo, kai vis grįžtama prie jau nagrinėtų, nenaudingų, sprendinių. Visi nagrinėti ir nenaudingi sprendiniai įtraukiami į tabu sąrašą iš ko ir kilo toks metodo pavadinimas.

Tabu metodo pseudokodas:

```
Pradiniai duomenys: TabuSąrašas_dydis
Rezultatai: S_geriausias
S_geriausias ← SukonstruotiPradinįSąrašą()
TabuSąrašas ← []
while (¬Baigimo sąlyga)
   KandidatuSąrašas  ← []
   for(S_kandidatas  ∈ SGeriausio_aplika)
      if (¬TuriSąvybių(S_kandidatas, TabuSąrašas)
         KandidatuSąrašas ← S_kandidatas
      end
   end
   S_kandidatas ← SurastiGeriausiąKandidatą(KandidatuSąrašas)
   if(ilgis(Skandidatas) <= ilgis(Sgeriausias))
      S_geriausias ← S_kandidatas
      TabuSarasas ← SavybiuSkirtumai(S_kandidatas, S_geriausias)
      while(TabuSąrašas_dydis > TabuSąrašas_dydis)
         TrintiĮrašą(TabuSąrašas)
      end
   end
end
```

Pagrindinis tabu metodo parametras yra maksimalus tabu sąrašo ilgis. Jį svarbu parinkti tinkamo dydžio. Jei sąrašas per trumpas, galime prieiti prie situacijos, kai metodas užsiciklina ties nenaudingais sprendiniais, kadangi neužtenka sąrašo ilgio visiems nenaudingiems sprendiniams sutalpinti. Tuo tarpu didėjant tabu sąrašui, algoritmas tampa lėtesnis ir nenaudinga sąrašą turėti per ilgą.

## 5.5 Skruzdėlių kolonijų optimizavimo metodas

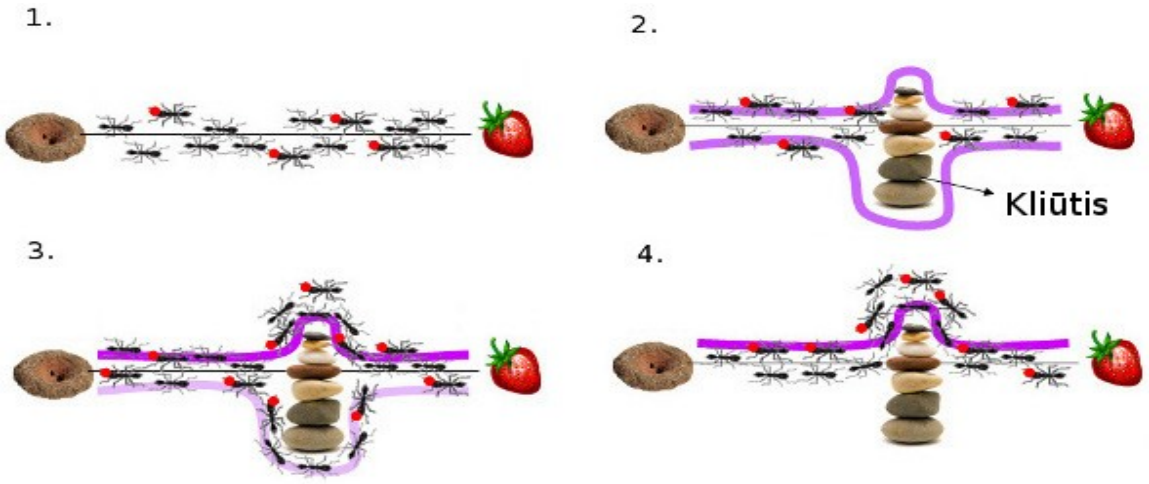
Vabzdžiai gyvendami kolonijose specializuojasi atlikti skirtingas užduotis, skirtas visos kolonijos individų sėkmingam bendram egzistavimui. Skruzdėlių kolonijoms būdingas paskirstytųjų sistemų elgesys, kur kiekviena posistemė veikia lygiagrečiai sąveikaudama su kitomis posistemėmis. Vieną skruzdėlių koloniją sudaro nuo keliasdešimt iki keliasdešimt milijonų skruzdėlių. Visa kolonja funkcionuoja be centrinio valdymo ir tvarkai palaikyti užtenka dviejų svarbių savybių, tai specializacija ir gebėjimas komunikuoti.

Skruzdėlių kolonijose būna nuo kelių iki keliolikos specializuotų skruzdėlių grupių, kurios kiekvienai būdingas funkcijas:

- reprodukcijos
- gynybos
- maisto rinkimo
- lizdo valymo
- lizdo statymo ir priežiūros

Skirtingas funkcijas atliekančioms skruzdėlėms būdingos skirtingos savybės, pavyzdžiui kareiviai yra didelės skruzdėlės, o darbininkės yra mažos.

Toks paskirstytasis elgesys, kai nėra centrinio koordinatoriaus ir komunikacija vyksta tik lokaliai tarp individų pasireiškiant teigiamam grįžtamajam ryšiui, tikslaus lietuviško pavadinimo neturi (angl. simergy) [11].




*8 pav. Pavyzdys iš gamtos. Skruzdės ieško artimiausio kelio.*

Skruzdėlių modelio elgesio modeliavimui buvo sukurtas skruzdėlių optimizavimo metodas, kurio esmė paremta dirbtine tarpusavyje lokaliai kontaktuojančių agentų sistemos imitacija.

Naudodamos feromonus skruzdėlės sprendžia trumpiausio kelio radimo problemą. Pasiekusios kiekvieną atsišakojimą skruzdės remdamosi feromonais sprendžia kuriuo keliu pasukti. Pradinėje būsenoje visi keliai iki maisto būna vienodi, bet vėliau artimesni keliai dėl to, jog yra dažniau naudojami(skruzdės per trumpesnį laiką įveikia tą patį kelią daugiau kartų), vėliau šiais keliais pradeda eiti daugiau skruzdėlių. Trumpesniuose keliuose stipriau jaučiasi feromonų pėdsakai ir skruzdės eidamos tais keliais, kurių feromonų pėdsakai stipresni, natūraliai pradeda eiti tais keliais, kurie yra optimaliausi.

Matematiškai skruzdėlių kolonijų optimizavimo uždavinius nėra sudėtinga ir keliaujančio pirklio algoritmą galime aprašyti tokiu pseudokodu[11]:

1. Atsitiktiniu būdu inicializuojamas mažas feromonų kiekis atkarpoje $\tau_{ij}$ tarp miestų i ir j.
2. Visi agentai (skruzdėlės) $k \in 1,\dots,m$ „sudedami“ pradiniame taške (miesto iš kurio pradedama paieška)
3. Tegu $T^+$ žymi trumpiausią kelią, o $L^+$ pasirinkto kelio ilgį.
4. Nuo t = 1 iki $t_{max}$, kai t – iteracijos numeris ir $t_{max}$– maksimalus iteracijų skaičius, kartojame šiuos žingsnius:
   a) kiekvienam agentui su tikimybe $F_{ij,k}(t)$ parenkamas kelias $T_k(t)$ į kitą miestą; operacija kartojama n - 1 kartų, kai n – miestų skaičius.
   b) kiekvienam agentui apskaičiuojamas kelio ilgis $L_k(t)$;
   c) jeigu randamas geresnis kelias, atnaujinama $T^+$ reikšmė;
   d) kiekvienai atkarpai atnaujinamas feromonų kiekis $\tau_{ij}$, naudojant šias formules:
   e) $$\tau_{ij}(t+1)=(1-\rho)\tau_{ij}(t)+\Delta\tau_{ij}(t), \quad (3)$$

   kur $$\Delta\tau_{ij}=\sum_{k=1}^{m}\Delta\tau_{ij,k}(t), \quad (3)$$

   tai kiekvienos skruzdėlės paliekamų feromonų suma, kur

   $$\Delta\tau_{ij,k}=\begin{cases}Q/L_k(t), kai(i,j)\in T_k(t),\\ 0, kai(i,j)\notin T_k(t)\end{cases}, \quad (4)$$

   Q – optimalaus kelio ilgis; $L_k(t)$ – k–tosios skruzdėlės nukeliautas atstumas.
5. Taip atrenkamas trumpiausias maršrutas $T^+$.

# 6 Algoritmų tyrimas

Algoritmų tyrimas buvo atliktas eksperimentuojant su skirtingomis duomenų imtimis, siekiant įvertinti kiekvieno jų rezultatų apskaičiavimo trukmę. Skaičiavimams naudotas nešiojamas kompiuteris su *Intel(R) Pentium(R) M, 1.60GHz* dažnio procesoriumi, turinčiu 2048 KB podėlio. Ubuntu Linux operacinė sistema su 2.6.35-28 branduoliu bei Ruby 1.8.7 versijos interpretatorius.

Siekiant rezultatus palyginti su optimaliais galimais, skaičiavimuose buvo naudoti duomenys iš TSPLIB[14] bibliotekos, kuriems jau yra žinomi optimalūs sprendiniai. Skaičius testo pavadinime reiškia miestų kiekį (pvz.: „lin105“ – testas su 105 miestais).

Tyrimo palengvinimui algoritmų išeities kodai paimti iš literatūros šaltinio[12] ir modifikuoti įdiegiant šias papildomas funkcijas:

- duomenų skaitymą iš TSPLIB[14] failų;
- galimybę nutraukti skaičiavimus viršijus laiko limitą;
- statistinių skaičiavimo rezultatų išsisaugojimas rezultatų faile.

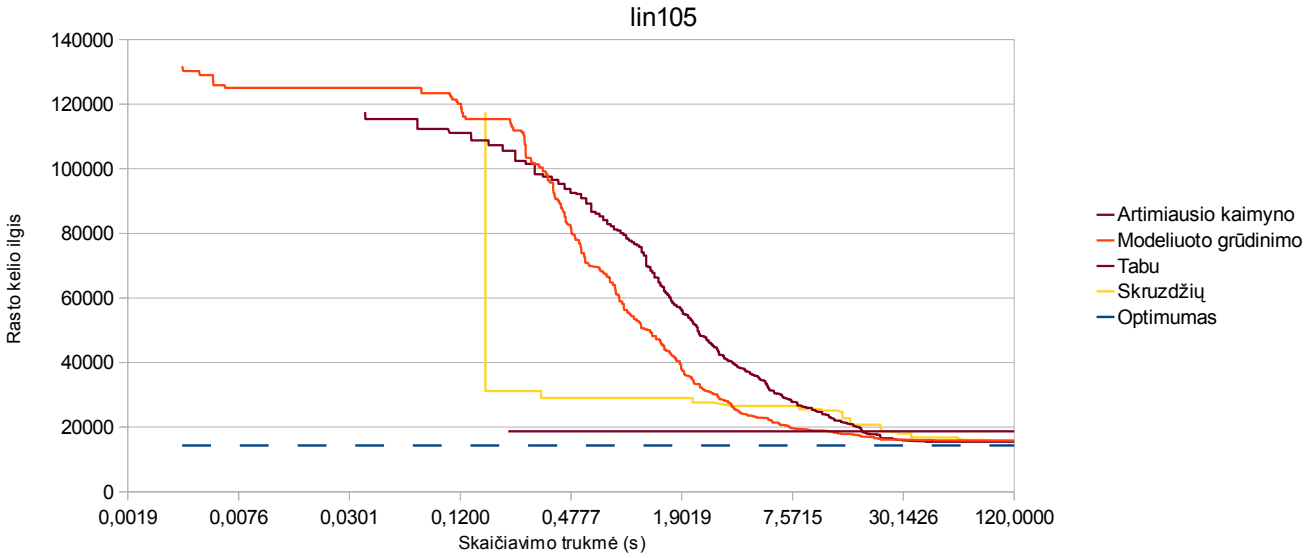
Tyrimo tikslas yra išsiaiškinti, kuris algoritmas efektyviausiai sprendžia keliaujančio pirklio uždavinį, kai turime ribotus laiko/skaičiavimų resursus.

## *6.1 Algoritmų efektyvumo priklausomybės nuo skaičiavimų trukmės tyrimas*

Šios tyrimo dalies tikslas yra išnagrinėti visų algoritmų apskaičiuojamų rezultatų kokybės priklausomybę nuo skaičiavimų trukmės. Stengtis įžvelgti dėsningumą bei išskirti vieną ar kelis algoritmus, išsiskiriančius iš kitų algoritmų kuriomis nors gerosiomis savo savybėmis.

Kiekvienas algoritmas testuojamas užduodant šaltinyje[12] rekomenduojamus parametrus. Testai atliekami su skirtingomis duomenų imtimis, siekiant įvertinti kaip tarpusavyje konkuruoja šie algoritmai esant skirtingai duomenų imčiai.

### 6.1.1 Algoritmų veikimo laiko ir paklaidos vertinimas „lin105“ testu




*9 pav. Eksperimentų rezultatai „lin105“ testui*

Iš grafiko (9 pav.) akivaizdžiai matome, jog nėra ryškaus favorito ir skirtingais laiko momentais skirtingi algoritmai duoda geriausią, tam laiko momentui, rezultatą. Detaliau pažvelgus į (1) lentelę matome, jog visi trys algoritmai tam tikrais laiko momentais pateikia geriausią rezultatą.

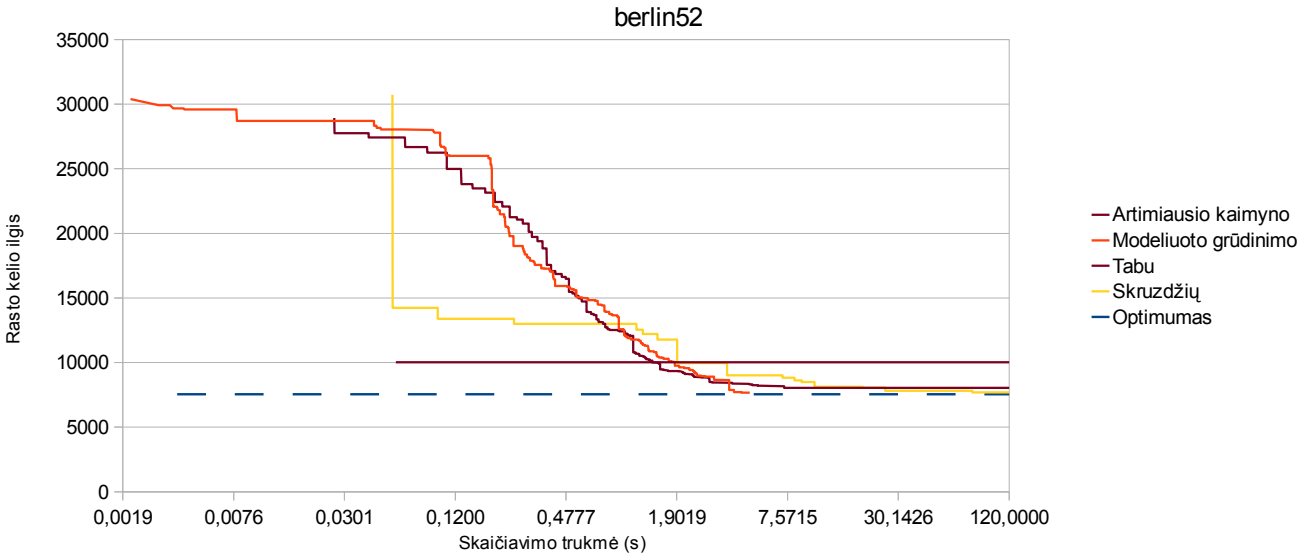
*1 lentelė. Detalūs algoritmų rezultatai „lin105“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,5 | 18699 | 79520 | 92543 | 29090 |
| 3,55 | | 26313 | 40396 | 26566 |
| 10 | | 18956 | 25240 | 25583 |
| 27 | | 16180 | 16308 | 18449 |
| 120 | | 15835 | 15482 | 15969 |

Optimalus testo lin105 sprendinys – 14379

Tiksliausi rezultatai nuo optimumo skiriasi 7,7%. Tabu algoritmas

### 6.1.2 Algoritmų veikimo laiko ir paklaidos vertinimas „berlin52“ testu




*10 pav. Eksperimentų rezultatai „berlin52“ testui*

Grafike (10 pav.) matomas, panašus išsidėstymas kaip ir „lin105“ testo atveju.

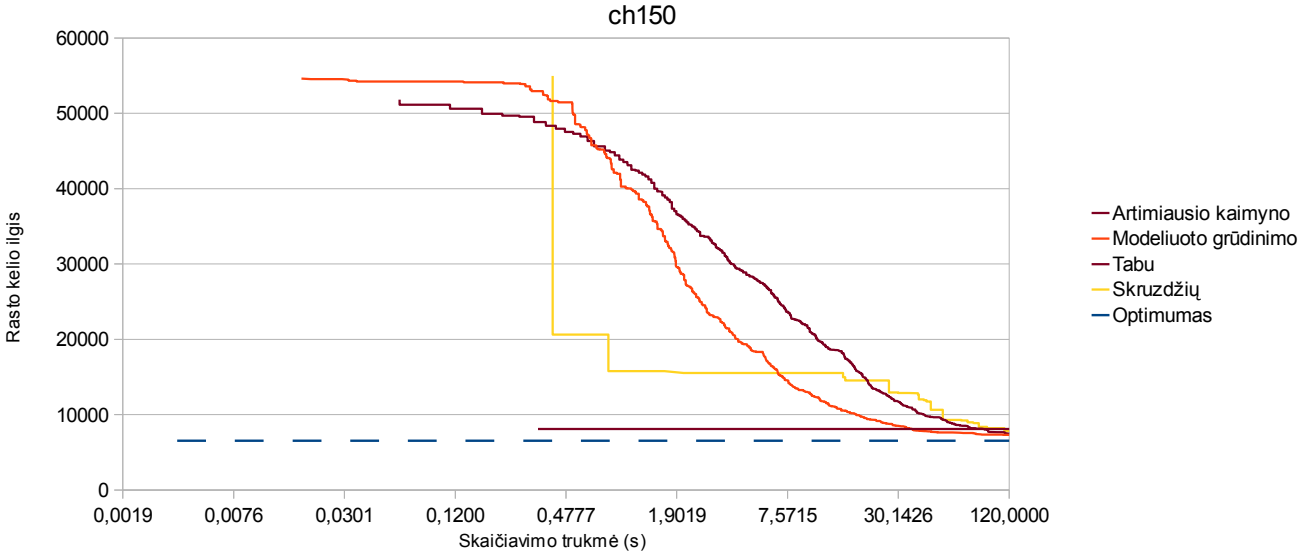
*2 lentelė. Detalūs algoritmų rezultatai „berlin52“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,06 | 10000 | 28040 | 27424 | 14239 |
| 0,11 | | 26617 | 26236 | 13384 |
| 0,22 | | 21473 | 22082 | 13384 |
| 0,82 | | 13764 | 12519 | 12985 |
| 1,9 | | 9766 | 9341 | 9954 |

Optimalus testo „berlin52“ sprendinys – 7542

Tiksliausi rezultatai – tabu algoritmo, kurie nuo optimumo skiriasi per 23,9%.

### 6.1.3 Algoritmų veikimo laiko ir paklaidos vertinimas „ch150“ testu




*11 pav. Eksperimentų rezultatai „ch150“ testui*

Iš grafiko (11 pav.) matome, jog padidėjus duomenų imčiai funkcijos reikšmės nusistovi tik skaičiavimų pabaigoje.

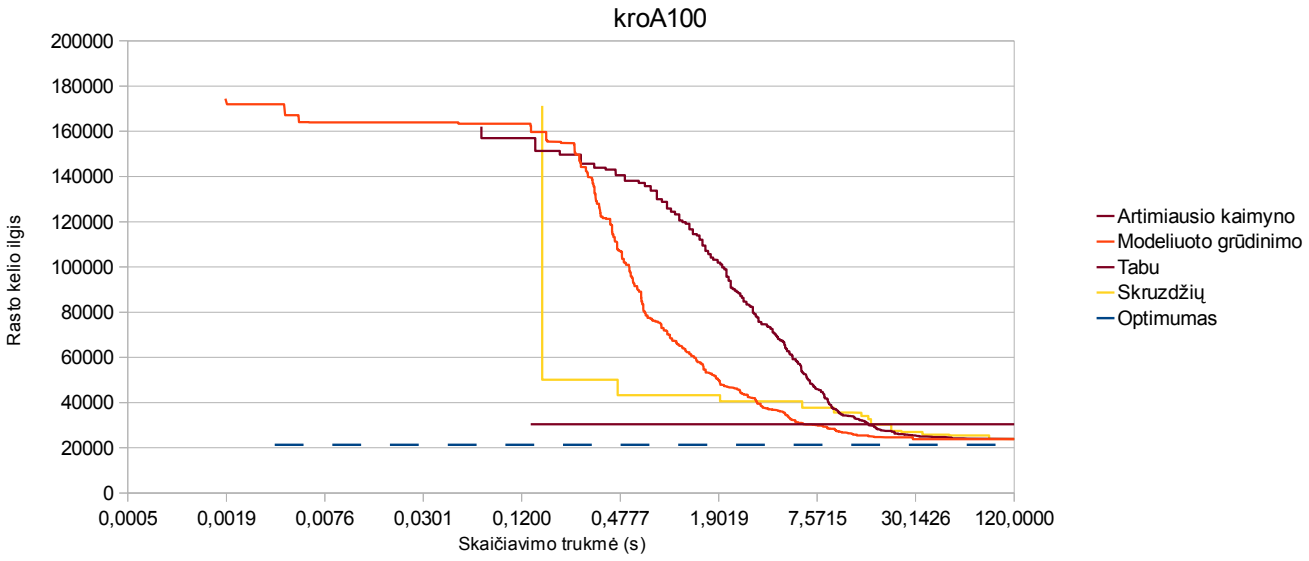
*3 lentelė. Detalūs algoritmų rezultatai „ch150“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,4 | 8084 | 54307 | 48325 | 20614 |
| 0,64 | | 46713 | 46332 | 9314 |
| 7,05 | | 15053 | 45603 | 15508 |
| 20,5 | | 9406 | 14460 | 14526 |
| 120 | | 7317 | 7452 | 7859 |

Optimalus testo „ch150“ sprendinys – 6528

Tiksliausi rezultatai – modeliuoto grūdinimo algoritmas, kurie nuo optimumo skiriasi per 12,5%.

### 6.1.4 Algoritmų veikimo laiko ir paklaidos vertinimas „kroA100“ testu




*12 pav. Eksperimentų rezultatai „kroA100“ testui*

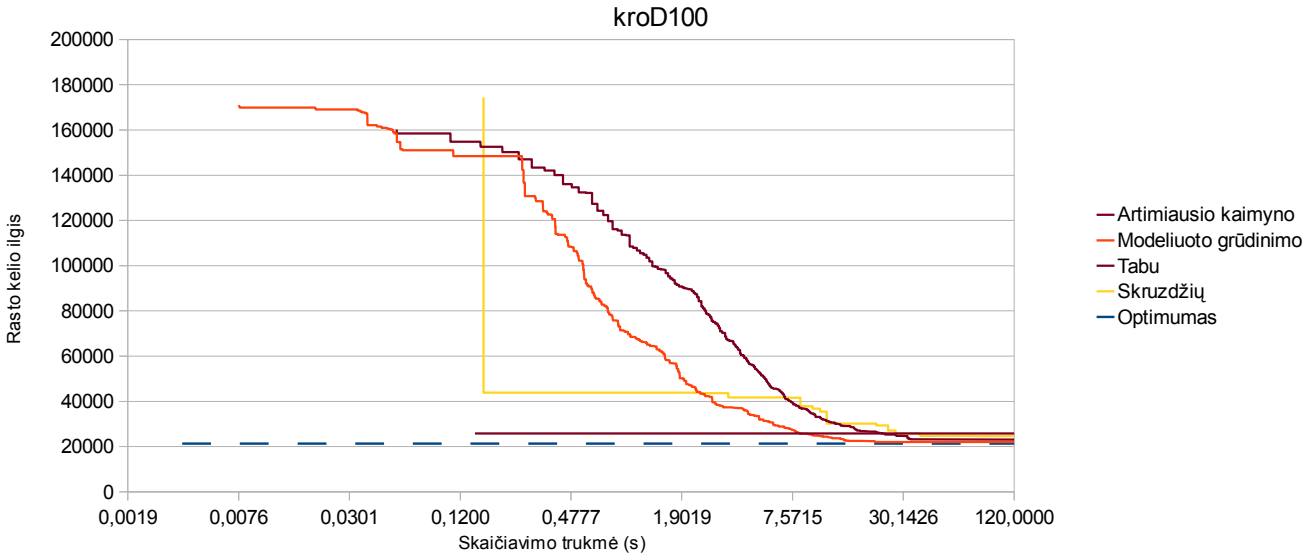
*4 lentelė. Detalūs algoritmų rezultatai „kroA100“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,16 | 30368 | 159698 | 151294 | 50067 |
| 0,25 | | 150524 | 149701 | 50067 |
| 3,26 | | 40016 | 77562 | 40518 |
| 16,6 | | 25151 | 30461 | 30253 |
| 90 | | 23736 | 23968 | 23916 |

Optimalus testo „kroA100“ sprendinys – 21282

Tiksliausi rezultatai – modeliuoto grūdinimo algoritmas, kurie nuo optimumo skiriasi per 11,5%.

### 6.1.5 Algoritmų veikimo laiko ir paklaidos vertinimas „kroD100“ testu




*13 pav. Eksperimentų rezultatai „kroD100“ testui*

Remiantis 4, 5 lentelių duomenimis galime daryti išvadas – griežtai išskirti, kuris algoritmas, Modeliuotojo grūdinimo ar Tabu, pateikia statistiškai geresnį rezultatą negalime.

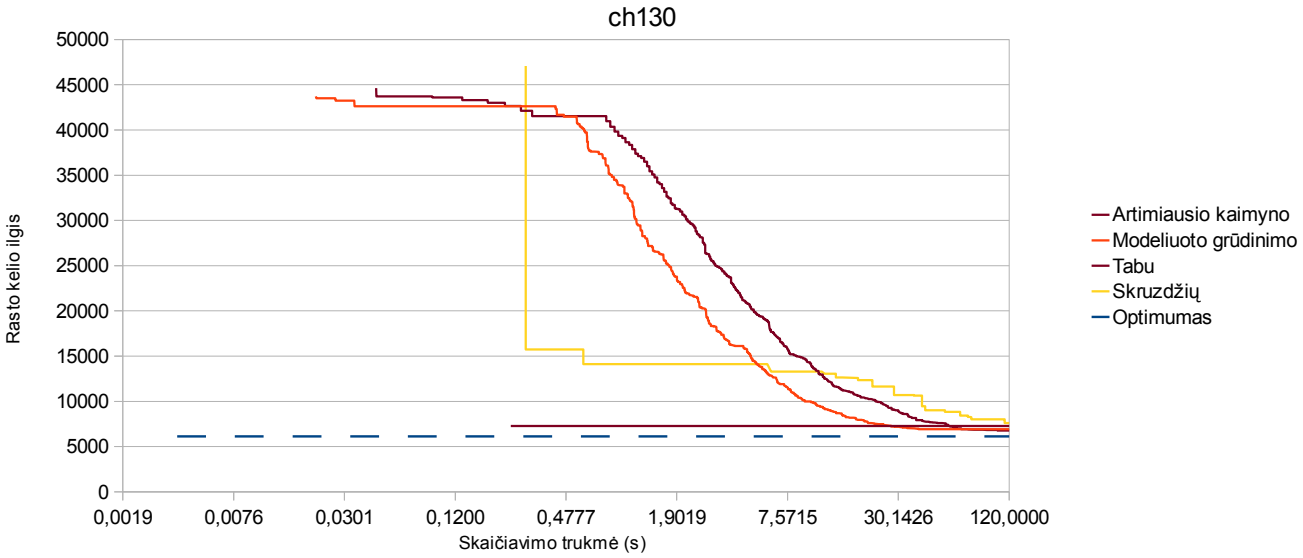
*5 lentelė. Detalūs algoritmų rezultatai „kroD100“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,25 | 25816 | 169062 | 147044 | 43877 |
| 0,35 | | 124020 | 142012 | 43877 |
| 3,5 | | 37338 | 66745 | 41743 |
| 10 | | 24796 | 34116 | 36739 |
| 120 | | 22062 | 23129 | 24589 |

Optimalus testo „kroD100“ sprendinys – 21294

Tiksliausi rezultatai nuo optimumo skiriasi 3,6%. Tabu algoritmas

### 6.1.6 Algoritmų veikimo laiko ir paklaidos vertinimas ch130 testu




*14 pav. Eksperimentų rezultatai „ch130“ testui*

Šiame grafike (14 pav.) taip pat naujų tendencijų nepastebime.

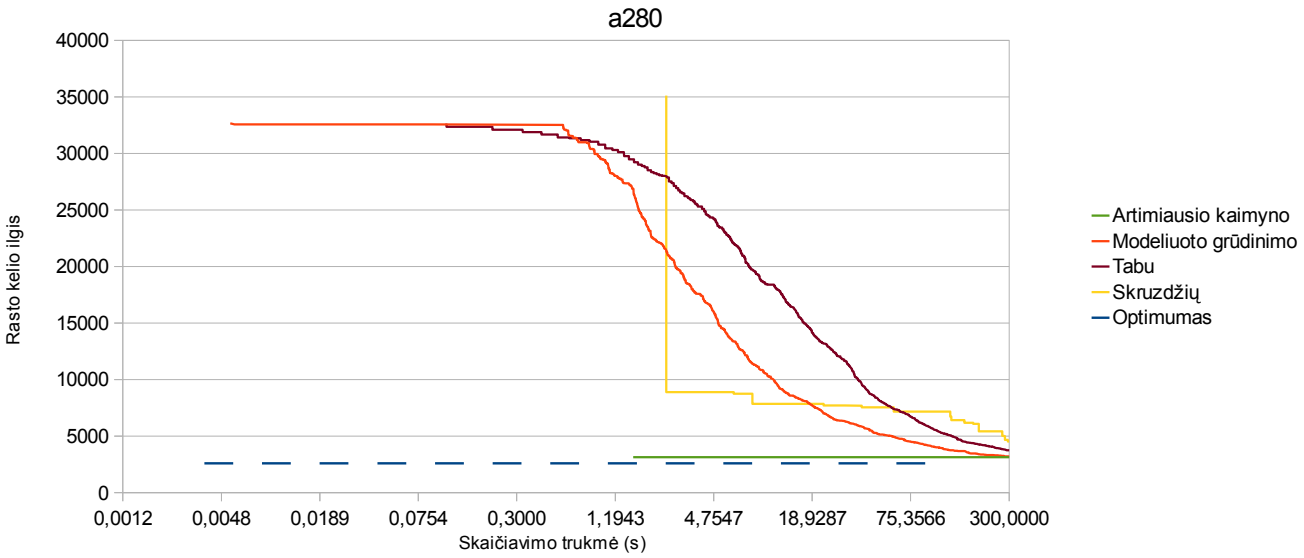
*6 lentelė. Detalūs algoritmų rezultatai „ch130“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 0,4 | 7271 | 42624 | 41537 | 15724 |
| 0,7 | | 37590 | 41537 | 14122 |
| 5 | | 14461 | 19923 | 14122 |
| 20 | | 7772 | 10347 | 12356 |
| 120 | | 6914 | 6766 | 7601 |

Optimalus testo „ch130“ sprendinys – 6110

Tiksliausi rezultatai nuo optimumo skiriasi per 10,7%. Tabu algoritmas

### 6.1.7 Algoritmų veikimo laiko ir paklaidos vertinimas „a280“ testu




*15 pav. Eksperimentų rezultatai „a280“ testui*

Grafike (15 pav.) matome, jog artimiausio kaimyno algoritmas rezultatai geriausi. Taip yra dėl to, jog esant dideliai duomenų imčiai, iteracinio gerinimo algoritmai nesuspėja per eksperimento laiką rasti optimalesnio sprendinio. Detaliau pažvelgus ir į (7) lentelę, galime spėti, jog su didele tikimybe, modeliuoto grūdinimo algoritmas pateiktų geriausią rezultatą, jei padidintumėm eksperimento trukmę.

*7 lentelė. Detalūs algoritmų rezultatai „a280“ testui.*

| Skaičiavimų trukmė (s) | Artimiausio kaimyno | Modeliuotojo grūdinimo | Tabu | Ant System |
|---|---|---|---|---|
| 4 | 3140 | 17424 | 25306 | 8878 |
| 35 | | 6036 | 10173 | 7692 |
| 80 | | 4424 | 6535 | 7161 |
| 270 | | 3260 | 3866 | 5413 |
| 300 | | 3196 | 3741 | 4510 |

Optimalus testo „a280“ sprendinys – 2579

Tiksliausi rezultatai nuo optimumo skiriasi per 21,7%. Artimiausio kaimyno algoritmas.

### 6.1.8 Apibendrintas rezultatų įvertinimas

Apibendrinant tyrimo rezultatus galima teigti, jog egzistuoja tam tikras dėsningumas, kuriuo remiantis galima būtų optimaliai parinkti kelio paieškos algoritmą atsižvelgiant į turimus resursus ir duomenų imtį bei laukiamų rezultatų kokybę.

Artimiausio kaimyno metodas yra geriausias sprendimas, kai reikia apytiksliai apskaičiuoti trumpiausią kelią per visus taškus ir neturime didelių skaičiavimo resursų. Kiti nagrinėti algoritmai išsiskiria tuo, jog iš pradžių pateikia labai prastus rezultatus ir juos nuolat gerina. Akivaizdžiai matomas dėsningumas, bei kiekvieno algoritmo išskirtinis „charakteris“.

Skruzdžių algoritmas apskaičiuoja santykinai gerą maršrutą per trumpesnį laiką nei Modeliuoto grūdinimo, bet esant ilgesnei skaičiavimų trukmei, Modeliuoto grūdinimo algoritmas pateikia geresnius rezultatus.

Tabu metodą tikslingiausia naudoti tuomet, kai turime daug skaičiavimo resursų ir rezultatų optimalumas yra didelio prioriteto.

# 7 Išvados

Atliekant sukurtų priemonių analizę nustatyta, jog nepakanka sukurti tik įskiepį keliaujančio pirklio algoritmui spręsti. Taip pat reikia modifikuoti kelio paieškos algoritmą, kadangi ištyrus navigatoriaus Navit atstumo tarp taškų skaičiavimų funkcijas paaiškėjo, jog navigatoriaus naudojamo Dejkstros algoritmo modifikacija nepritaikyta apskaičiuoti atstumus tarp keleto taškų ir naudojant keliaujančio pirklio algoritmą reiktų modifikuoti šį algoritmą, kad atstumai tarp visų taškų keliaujančio pirklio algoritmui būtų pateikti per prieinamą laiką.

Atliekant keturių euristinių algoritmų, keliaujančio pirklio problemai spręsti tyrimą, nustatyta, jog tirti algoritmai konkuruoja tarpusavyje ir negalime išskirti nei vieno algoritmo kaip turinčio didelį pranašumą prieš kitus.

Analizuojant tyrimo duomenis nustatyta, kad yra tikslinga parinkti algoritmą pagal turimus skaičiavimų resursus ir problemos duomenų imtį. Nustatyta, jog turint mažus skaičiavimo resursus, kai rezultatą reikia pateikti per trumpą laiką yra tikslinga naudoti artimiausio kaimyno metodą. Bet šis algoritmas nusileidžia kitiems tirtiems algoritmas, kai skaičiavimams turime santykinai daug laiko.

Tyrinėjant iteracinio optimizavimo algoritmus buvo nustatytas tam tikras dėsningumas arba algoritmų „charakteriai“. Paaiškėjo, jog iš šių trijų algoritmų, greičiausiai santykinai tikslius rezultatus pateikia skruzdėlių kolonijos optimizavimo uždavinys, o tiek modeliuoto grūdinimo, tiek tabu algoritmai, nors skaičiavimų pradžioje ir atsilieka, ilgėjant skaičiavimų trukmei pateikia geresnius rezultatus nei skruzdžių algoritmas.

Nustatyta, jog yra tikslinga navigatoriuje realizuoti kelis algoritmus ir problemą spręsti pasirinktu būdu, įvertinant problemos dydį, skaičiavimo resursus, bei rezultatų pateikimo laiką arba norimą rezultatų tikslumą.

# 8 Literatūra


1. TOSHIHIDE IBARAKI, KOJI NONOBE, MUTSUNORI YAGIURA, Metaheuristics: progress as real problem solvers, p. 344
2. FRED GLOVER, MANUEL LAGUNA, Tabu search, Volume 1, Springer, 1998, p. 2
3. GENDREAU MICHEL, An Introduction to Tabu Search, Kluwer Academic Publishers, 2002 p. 2-5
4. EDMUND BURKE (PH. D.), EDMUND K. BURKE, GRAHAM KEND, Search methodologies: introductory tutorials in optimization and decision, Springer, 2005 p. 171-175
5. JOHANNES JOSEF SCHNEIDER, SCOTT KIRKPATRICK, Stochastic optimization, Springer, 2005 p. 209-231
6. MICHAEL NEGNEVITSKY, Artificial Intelligence, A Guide to Intelligent Systems, Second Edition, 2005, p. 336-340
7. JOSEPH C. GIARRATANO, GARY D. RILEY, Expert Systems Principles and Programming, 2005, p. 52-53
8. Algorithms and Complexity , HERBERT S. WILF, Philadelphia, 1994, p. 104-105
9. STUART RUSSEL, PETER NORVIG, Artificial intelligence A Modern Approach, p. 68, 88
10. ANDRIES P. ENGELBRECHT Computational Intelligence 406 p.
11. DORIGO MARCO (1999) Artificial life: the swarm intelligence approach. Tutorial TD1, congress of evolutionary computing, Washington DC.
12. JASON BROWNLEE, Clever Algorithms, Nature-Inspired Programming Recipes, Melbourne, 2011, p. 73-79, 169 -174, 238-244
13. APPLEGATE, D. L.; BIXBY, R. M.; CHVÁTAL, V.; COOK, W. J. (2006), The Traveling Salesman Problem
14. Keliaujančio pirklio uždavinio pavyzdinių duomenų katalogas. <http://www.tsp.gatech.edu/data/index.html> [žiūrėta 2011-04-21]
15. The traveling salesman problem <http://www.tsp.gatech.edu/history/pictorial/pictorial.html> [žiūrėta 2011-04-11]
16. Lietuvių kalbos žinynas. Kaunas: Šviesa, 1998.
17. Valstybinės lietuvių kalbos komisijos interneto svetainė <http://www.vlkk.lt> [žiūrėta 2011-05-26]

# 9 Paveiksliukų ir lentelių sąrašas

## 9.1 Paveiksliukų sąrašas

1 pav. Navit programos ekrano vaizdas.....5
2 pav. ModRana programos ekrano vaizdas.....6
3 pav. TangoGPS programos ekrano vaizdas.....8
4 pav. traveling-salesman programos ekrano vaizdas.....10
5 pav. GpsDrive programos ekrano vaizdas.....11
6 pav. Keliaujančio pirklio problemos sprendimas. Iš kairės 1 – pradiniai duomenys, 2 – neoptimalus sprendimas, 3 – optimalus sprendimas.....14
7 pav. Lokalaus bei globalaus optimumo iliustracija.....17
8 pav. Pavyzdys iš gamtos. Skruzdės ieško artimiausio kelio.....21
9 pav. Eksperimentų rezultatai „lin105“ testui.....24
10 pav. Eksperimentų rezultatai „berlin52“ testui.....25
11 pav. Eksperimentų rezultatai „ch150“ testui.....26
12 pav. Eksperimentų rezultatai „kroA100“ testui.....27
13 pav. Eksperimentų rezultatai „kroD100“ testui.....28
14 pav. Eksperimentų rezultatai „ch130“ testui.....29
15 pav. Eksperimentų rezultatai „a280“ testui.....30

## 9.2 Lentelių sąrašas

1 lentelė. Detalūs algoritmų rezultatai „lin105“ testui.....24
2 lentelė. Detalūs algoritmų rezultatai „berlin52“ testui.....25
3 lentelė. Detalūs algoritmų rezultatai „ch150“ testui.....26
4 lentelė. Detalūs algoritmų rezultatai „kroA100“ testui.....27
5 lentelė. Detalūs algoritmų rezultatai „kroD100“ testui.....28
6 lentelė. Detalūs algoritmų rezultatai „ch130“ testui.....29
7 lentelė. Detalūs algoritmų rezultatai „a280“ testui.....30

# 10 Terminų žodynėlis ir santrumpos

**GPX formatas** – duomenų failo formatas, naudojamas maršrutams saugoti. Tekstinis, XML sintaksės failas.

**OpenStreetMap –** projektas skirtas atvirų ir nemokamų kelių bei gatvių žemėlapių kūrimui bei dalinimuisi.

**Maršruto saugojimas žurnale –** nemaža dalis GPS navigatorių turi galimybę einamuoju momentu maršrutą išsaugoti failuose, suteikiant galimybę vėliau peržiūrėti judėjimo koordinates ir panaudoti jas tokiems tikslams, kaip žemėlapių braižymas, nuotraukų GeoTag'inimas ar kelionės išlaidų skaičiavimas.

**Pozicijos dalinimosi funkcija** – galimybė perduoti einamąsias koordinates į serverį ir taip dalintis esamomis koordinatėmis. Ši funkcija leidžia jums perduoti draugams jūsų dabartines koordinates ir matyti kur šiuo metu yra jūsų draugai, jei jie to nori.

**Hamiltono ciklas** – maršrutas grafe, kuriuo galima aplankyti visas grafo viršūnes po vieną kartą ir sugrįžti į pirmą viršūnę.

**Feromonas** – bet koks cheminis junginys naudojamas komunikacijai tarp dviejų tos pačios rūšies organizmų.

**NP sunkus** – uždavinių sunkumo klasė. Šiai klasei uždavinys priklauso tada ir tik tada, kai yra įrodyta, kad visi egzistuojantys tikslūs uždavinio sprendimo algoritmai yra polinominio sudėtingumo.

**Įskiepis** – programinis paketas, naudojamas kaip kito programinio paketo dalis ir skirtas atlikti tam tikras užduotis.

# 11 Summary

Years after the Global Positioning System has become operational, it is now being used by millions of people around the world. The possibilities and actual cases of application of the system are endless: navigation, Geo-tagging, map-making, tracking, emergency services, etc.

This master's thesis covers an investigation of a specific type of problem that is closely related to GPS navigation systems – the traveling salesman problem (TSP). The core idea of it is as follows: given a list of cities with specific coordinates, the goal is to find a shortest possible route that visits all points exactly once.

The original definition of the problem predates the GPS by several centuries and has been the subject of research for many years. It has been proven to belong to NP-hard class of problems and thus there is no simple method for calculating exact solutions to the problem within a reasonable time period even with relatively small amount of points. However, many meta-heuristic algorithms have been applied to the problem giving relatively good result with limited time budgets.

The investigation had a list of objectives: analyze the capabilities and resources of a range of chosen GPS navigation devices; analyze the needs and requirements of traveling salesman related GPS navigator functions for regular users; analyze what types of TSP algorithms are used in existing navigation software products; analyze the capabilities of various TSP algorithms with regard to used resources and speed of calculations; determine which algorithms are optimal for a range of specific situations.

Research of different algorithms led to a conclusion that there is no single algorithm that is always better than the rest. Under different circumstances, different algorithms showed different results. Some were clearly optimal in some situations, while others competed with each other in other situations. The key element to success of an algorithm was how much time it got to do it's calculations. The amount of the input data changed the duration of the calculations but the algorithm function declination rate remained mostly the same with different sets of input data.

Keywords: Global Positioning System, GPS, Navigation, Traveling Salesman, Meta-heuristic, Open-Source System.

# 12 Priedas A. Apklausos rezultatai

**1. Ar dažnai susiduriate su situacijomis kai reikia aplankyti kelis taškus ir ieškote optimalaus maršruto?**

**a.** Beveik kiekvieną dieną 86 %

**b.** Kartą ar kelis per mėnesį 9 %

**c.** Kartą ar kelis per metus 5 %

**2. Ar naudojatės kokia nors optimalaus kelio radimo programine įranga?**

**a.** Taip 5 %

**b.** Ne 95 %

**3. Jei tokia įranga dar nesinaudojate ar pirktumėte tokią?**

**a.** Taip 10 %

**b.** Ne 70 %

**c.** Nežinau 20 %

**4. Ar naudotumėte optimalaus maršruto radimo programinę įrangą, jei ją galėtumėte nemokamai įsidiegti į savo telefoną?**

**a.** Taip 63,4%

**b.** Ne 18,6%

**c.** Nežinau 18%

Apklausoje dalyvavo 40 žmonių.