TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
MATEMAATIKA INSTITUUT
MATEMAATIKA ERIALA

Teele Laas

Rändkaupleja ülesande lahendamine sipelgaalgoritmiga

Bakalaureusetöö

Juhendaja: dots. Peep Miidla

TARTU 2013

Sisukord

Sissejuhatus

Meid ümbritsevat loodust jälgides on inimesed teinud mitmeid tähelepanekuid looduses ja looduslikes ning füüsikalistes protsessides toimuva kohta. Need tähelepanekud on inspireerinud teadlasi kasutama arusaamu loodusest heuristiliste optimiseerimisalgoritmide konstrueerimisel. Heuristilisi optimiseerimisalgoritme kasutatakse selliste optimiseerimisülesannete lahendamiseks, kus on ette näha arvutustöö ülisuur maht ja kus lahendi optimaalsuse hindamise võimalikkus on kaheldav. Üheks näiteks sellistest meetoditest on sipelgaalgoritm. Sipelgate uurimine innustas arvutiteadlasi ja matemaatikuid arendama algoritme, mis kasutaksid sipelgate võimet leida lühim teekond oma pesa ja toiduallikate vahel.

Käesoleva bakalaureusetöö eesmärgiks on kirjeldada sipelgaalgoritmi kui ühte metaheuristilist matemaatilise optimiseerimise meetodit. Seda tutvustakse tuntuima kombinatoorse optimiseerimisülesande – Rändkaupleja ülesande lahendamisel. Töös vaadeldakse rändkaupleja ülesande kasutusvõimalusi ja rakendatakse sipelgaalgoritmi rändkaupleja ülesande lahendamiseks.

Bakalaureusetöö esimeses peatükis kirjeldatakse sipelgate käitumise eripärasid ja oskusi toiduotsingutel. Selles peatükis selgitatakse optimiseerimisülesande olemust ja erinevaid optimiseerimisülesande algoritmilisi lahendamisviise. Põhjalik kirjeldus on antud sipelgaalgoritmi kohta: selgitatakse erinevaid tähtsamaid termineid ja protsesse, mis on olulised ülesande lahendi konstrueerimisel tehissipelgate poolt.

Rändkaupleja ülesandele ja selle lahendamisele on pühendatud töö teine peatükk. Selles peatükis tutvustatakse rändkaupleja ülesannet ja erinevaid võimalusi selle ülesande lahendamiseks. Täpsem kirjeldus on antud sipelgaalgoritmi rakendamisest rändkaupleja ülesandele ning on toodud näide varasematest tulemustest. Samuti vaadeldakse rändkaupleja ülesande erinevaid rakendusi.

Antud töö kolmandas peatükis lahendatakse sipelgaalgoritmiga konkreetne rändkaupleja ülesanne. Selleks kasutatakse rakenduspaketti Matlab. Selles peatükis kirjeldatakse ülesande lahendamiseks vajalikke algandmeid, analüüsitakse erinevate sipelgaalgoritmi mõjutavate parameetrite mõju ülesande tulemusele ning analüüsitakse ülesande lahendamisel saadud tulemusi. Peatükis 3 lahendatud ülesande algandmed, lahendeid illustreerivad joonised ja ülesande lahendamiseks kasutatud programmikood on esitatud lisades.

Autor tänab dotsent Peep Miidlat arvukate paranduste ja täienduste ning mitmete huvitavate ideede eest.

1. Sipelgaalgoritm

Loodus on inspireerinud mitmeid meetodeid ja tehnikaid erinevate optimiseerimisülesannete lahendamiseks. Üheks selliseks meetodiks on sipelgaalgoritm, mis on suhteliselt uus, kuid edukas optimiseerimisülesannete lahendamise võte. Sipelgaalgoritm sai alguse 1992. aastal Milano Polütehnikumis Marco Dorigo poolt kaitstud doktoriväitekirjast [1]. Sipelgaalgoritmi puhul võime kohata ka nime sipelgasüsteem, mis tuleneb sipelgaalgoritmi inglisekeelsest nimetusest *Ant System.* Alates üheksakümnendate aastate algusest, kui loodi esimene sipelgaalgoritm, on see meetod äratanud teadlaste seas suurt tähelepanu ja praeguseks on välja töötatud mitmeid modifikatsioone nagu näiteks sipelgakoloonia optimiseerimine, max-min sipelgasüsteem, rivipõhine sipelgasüsteem jt.

Mitmetes valdkondades nagu majanduses, kaubanduses, tehnika valdkonnas, tööstuses ja meditsiinis on esile kerkinud keerukad kombinatoorsed optimiseerimisülesanded [3]. Sellised ülesanded on tihti väga raskesti lahendatavad. Arvutiteaduses on püütud leida meetodeid keerukate optimiseerimisülesannete lahendamiseks, kuid siiski nõuab nende lahendamine tohutult paljude keerukate arvutuste tegemist. Optimiseerimisülesannete lahendamiseks on välja pakutud erinevaid algoritmilisi lähenemisviise. Üheks võimaluseks on lahendada optimiseerimisülesanne täpse algoritmiga (*exact algorithm*) [3]. Täpne algoritm garanteerib optimiseerimisülesande lahendamisel optimaalse lahendi leidmise ja tõestab, et leitud lahend ka on optimaalne. Kuna täpse algoritmiga optimiseerimisülesande lahendamise aeg sõltub ülesande mahust ja suuremamahuliste ülesannete puhul on see väga suur, siis on loodud ka ligikaudseid algoritme (*approximate algorithms*) [3]. Ligikaudsed algoritmid, mida nimetatakse ka heuristilisteks algoritmideks, on optimiseerimisülesannete lahendamiseks küllaltki efektiivsed. Kuigi heuristilised algoritmid ei suuda tagada lahendite optimaalsust, on nad võimelised leidma optimaalsele lahendile väga lähedase lahendi väikese ajakuluga. Selleks, et heuristiliste algoritmidega saadud lahendite kvaliteeti

veelgi parandada, on viimase kahekümne aasta jooksul pööratud tähelepanu heuristiliste algoritmide alustehnika täiustamisele, nn. metaheuristika kavandamisele, et juhtida ülesandespetsiifilisi konstruktsioone ja parameetreid. Metaheuristiliste algoritmide puhul on loodusliku protsessi matemaatilist analoogi püütud algoritmi tasandilt, st metatasandilt juhtida. Sellist tehnikat nimetatakse metaheuristikaks. Teisiti öeldes on metaheuristika üldotstarbeline algoritmiline meetod, mida saab suhteliselt väheste muudatustega rakendada konkreetsetele optimiseerimisülesannetele. Metaheuristiliste optimiseerimismeetodite näideteks on sipelgaalgoritm, simuleeritud lõõmutamine, iteratiivne lokaalne otsing, adaptiivne otsing, mesilaste algoritm, käo algoritm jt. Metaheuristilised optimiseerimismeetodid laenavad ideid erinevatest valdkondadest nagu geneetika, bioloogia, füüsika, tehisintellekt ja matemaatika. [2, 3]

1.1 Inspiratsioon sipelgatest

Sipelgaalgoritm on inspireeritud sipelgate käitumisest toiduotsingutel. Inimesed on vaadelnud sipelgaid ja püüdnud mõista nende maailma. Prantsuse etnoloog Pierre-Paul Grass'e on märkinud, et sipelgatevaheline suhtlus toimub läbi *stigmergia* [2]. See on kaudne ning sümboliteta suhtlusvorm, mis toimub läbi suhtlevaid subjekte ümbritseva keskkonna muutmise. Sellise suhtlusvormi eripäraks on ka see, et infot selles keskkonnas saavad ainult need, kes infoga vahetult kokku puutuvad. Näitena stigmergiast võib tuua sipelgakoloonia**.** Sipelgad eritavad maa peale keemilist ainet – feromooni, et märkida ära radu pesast toiduallikani ja tagasi. Kuigi sipelgate nägemine on vähearenenud ja osal sipelgaliikidel see isegi puudub, on nende uurijad märganud putukate võimet orienteeruda ümbritsevas keskkonnas toidu leidmiseks. Seda seletab asjaolu, et sipelgad tajuvad feromooni olemasolu ning nad eelistavad liikuda mööda neid radu, millele feromooni eritatud on. Mida rohkem sipelgaid ühte rada mööda liigub, seda rohkem eritatakse sinna feromooni ning rada muutub eelistatumaks. Seega tõusevad enamkasutatavad rajad esile ja teised jäetakse maha. Selle mehhanismi abil on sipelgad võimelised transportima toitu tõhusat teed mööda. Sipelgaalgoritm kasutab analoogilist mehhanismi optimiseerimisülesannete lahendamiseks. [2, 3]

Huvitavaks asjaoluks sipelgate käitumise juures on ka see, et nad on võimelised leidma lühima tee pesa ja toiduallika vahel. Teadlased on põhjalikult uurinud feromooni eritamist ja sipelgate edasist käitumist. Ühes uurimuses, tuntud kui "Kahe silla eksperiment" ("*Double bridge experiment*"), selgus, et sipelgad valivad kahe erineva, kuid sama pikkusega teekonna puhul selle, millel on kõrgem feromooni kontsentratsioon. Selles uuringus jälgiti Argentiina sipelgaid, kelle pesast oli toiduallikani võimalik minna kahte erinevat teed mööda, mis olid ühepikkused. Oma teekonnal toiduallikani ja tagasi eritasid sipelgad feromooni. Esialgu valis iga sipelgas juhuslikult ühe tee kahest, kuna feromooni nendel radadel ei olnud. Mõne aja möödudes mängis rolli juhuslikkuse kõikumine ning üks teedest oli saanud rohkem valituks. Suurema feromoonikontsentratsiooniga tee muutus atraktiivsemaks ja meelitas sellele teele rohkem sipelgaid, tuues kaasa täiendava hulga feromooni ning mõne aja möödudes liikus terve koloonia mööda seda teed. Sellist terve koloonia käitumist nimetatakse positiivse tagasiside mehhanismiks, mis aitab leida lühima tee pesa ja toidu vahel. [2]

Uuringus tehti ka teistsugune eksperiment, kus üks teedest oli märkimisväärselt pikem kui teine. Sel juhul juhuslikkuse kõikumise mõju oli väiksem, kuid olulist rolli mängis teine mehhanism. Uuringutes selgus, et kuna lühemate teede läbimiseks kulub vähem aega, siis neid valinud sipelgad jõuavad pessa tagasi varem, kui pika tee valinud sipelgad. Niimoodi saavad lühemad teed feromooni varem ja muutuvad aja möödudes sipelgate jaoks atraktiivsemaks. Vaadeldav protsess kestab seni, kuni sipelgad valivad toiduallikani jõudmiseks lühima tee. [2]

1.2 Optimiseerimisülesanne

Optimiseerimisülesanneteks nimetatakse ülesandeid, kus on vaja leida selline muutuja väärtus, nn lubatavate lahendite hulgast, mis annab etteantud funktsioonile, nn sihifunktsioonile (*objective function*) ekstremaalse väärtuse. Sihifunktsioon sõltub ülesande eesmärgist ja on seega iga ülesande puhul erinev. Eesmärgiks on leida sellised muutujate väärtused, et sihifunktsiooni väärtus oleks minimaalne, kui tegemist on minimeerimisülesandega, või maksimaalne, kui tegemist on maksimeerimisülesandega.

Muutujate lubatavaid väärtusi, mis realiseerivad sihifunktsiooni maksimumi (miinimumi), nimetatakse optimaalseiks lahendeiks. Sellist sihifunktsiooni väärtuse leidmist nimetatakse optimiseerimiseks. Optimiseerimisülesanded on tihti seotud kombinatoorikaga. Sel juhul lubatavate lahendite hulgast muutujate väärtusi otsides tuleb sihifunktsiooni ekstremaalse väärtuse leidmiseks uurida kõiki erinevaid etteantud väärtuste kombineerimise võimalusi, st kõigi objektide hulgast tuleb leida ainult teatud tingimustele vastavad objektid. Sellist ülesannet nimetatakse kombinatoorseks optimiseerimisülesandeks.

Optimiseerimisülesannete üldise püstituse puhul on antud sihifunktsioon

$$f: X \to \mathbb{R},$$

kus X on lahendite ruum ja $\mathbb{R}$ on reaalarvude hulk. Ülesandeks on leida $x^* \in \Omega$ c X selline, et

$$f(x^*) = min\, f(x), x \in \Omega,$$

kui tahetakse leida sihifunktsiooni minimaalset väärtust või

$$f(x^*) = max\, f(x), x \in \Omega,$$

kui tahetakse leida sihifunktsiooni maksimaalset väärtust. Sümboliga Ω tähistatakse lubatavate lahendite hulka.

Sipelgaalgoritmi kasutatakse kombinatoorsete optimiseerimisülesannete lahendamiseks. Sipelgaalgoritm põhineb tehissipelgate koloonial. Tehissipelgad on lihtsad arvutuslikud agendid, kes suhtlevad kaudselt feromoonijälgede kaudu. Nad töötavad ühise eesmärgi nimel leida keerulisele optimiseerimisülesandele lahend.

1.3 Sipelgaalgoritmi kirjeldus

Järgnevalt kirjeldame sipelgaalgoritmi tööd. Olgu meil tehissipelgate koloonia, kuhu kuulub m sipelgat. Sipelgaalgoritm on oma olemuselt konstrueerimisalgoritm, kus igal algoritmi iteratsioonil iga sipelgas konstrueerib ülesandele lahendi, milleks on teed läbi graafi [2]. Tehissipelgad liiguvad mööda konstruktsioonigraafi *G = (C,E)*, mida kasutatakse probleemi esitamiseks. *G* on graaf, kus *C* on graafi tippude hulk ja *E* on graafi servade hulk. Liikumine toimub graafis tõenäosusreegli põhjal, mis arvestab lahendi konstrueerimisel kättesaadavat heuristilist informatsiooni ja feromoonijälgede väärtusi. Igal iteratsioonisammul, kus kõik sipelgad on valmis saanud ühe lahendi, uuendatakse feromooni väärtuseid lahendile kuuluvatel graafi komponentidel, kusjuures feromooni kogus, mida väljastatakse, sõltub leitud lahendi kvaliteedist.

Sipelgaalgoritmi algust võime ette kujutada nii, et *m* sipelgat lastakse graafile lahendeid konstrueerima, kusjuures iga sipelgas valib algtipu juhuslikult. Sellest tipust saab alguse lahendi konstrueerimine. Enne algoritmi käivitamist on kõikidele graafi servadele lisatud suur kogus feromooni. Edaspidi hakkavad m sipelgat konstrueerima lahendeid, st teid läbi graafi ning igal iteratsioonisammul leiab iga sipelgas uue lahendi. Lahendite leidmine sipelgaalgoritmis toimub nii, et liikudes mööda graafi valib igal sammul tipus c_i asuv sipelgas *k* järgmise tipu c_j oma naabrusest N_i^k tõenäosusega p_{ij}^k [1, 3].

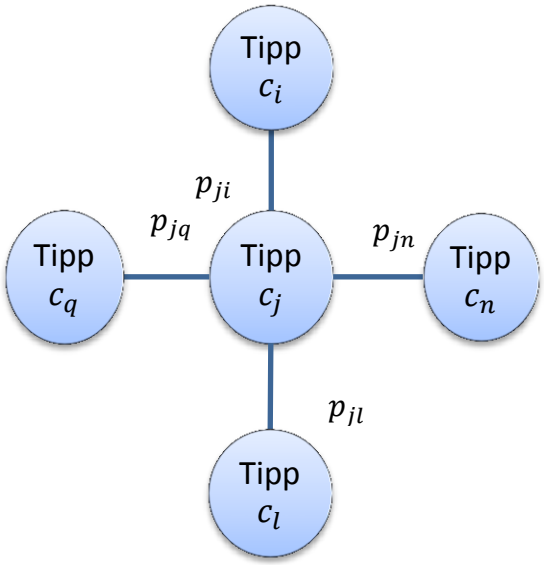
Naabruse ehk lähedaste tippude hulgaga N_i^k on seotud sipelga *k* mälu. Erinevalt pärissipelgatest on tehissipelgad mõnevõrra arenenumad. Eeldatakse, et neil on mälu. Igal sipelgal *k* on mälu M_k juba külastatud tippude nimekirja hoidmiseks. Seda on vaja, et sipelgas läbiks graafi nii, et igat tippu läbib ta ainult ühe korra. See on oluline sellepärast, et kui graafis on rohkem kui 2 tippu, võivad sipelgad genereerida selles graafis tsükleid ning tsüklisse kuuluvad servad saavad hiljem rohkem feromooni kui teised. Seetõttu lühema tee leidmine muutub ebaefektiivsemaks. Lisaks on mälu vajalik naabruse N_i^k koostamiseks tipus c_i, et teada, millistesse graafi tippudesse on võimalik sipelgal *k* edasi liikuda. [5]

Sipelgad liiguvad mööda graafi tõenäosusreegli põhjal. Tõenäosus p_{ij}^k, et sipelgas *k* liigub tipust c_i tippu c_j, on antud valemiga:

$$p_{ij}^k = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{il \in N_j^k} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, \tag{1.1}$$

kus sümboliga η_{il} on tähistatud heuristiline parameeter serval e_{il} ja sümboliga τ_{il} on tähistatud feromoonijälgede väärtus serval e_{il}.

Järgnevalt selgitatakse tehissipelgate liikumist sipelgaalgoritmis 5-tipulisel graafil. Seda illustreerib joonis 1.1.



Joonis 1.1. Sipelgaalgoritm 5-tipulisel graafil

Olgu Joonisel 1.1 kujutatud viietipuline graaf mingi suurema täisgraafi alamgraaf ja olgu sipelgas *k* jõudnud lahendi konstrueerimisega tippu c_j. Eeldades, et sipelgas *k* pole varasemalt ühtegi tippudest c_i, c_n, c_l ja c_q külastanud, on sipelgal ka järgmise tipu valimiseks 4 võimalust. Sel juhul sipelga *k* naabruse tipus c_j, tähistatakse N_j^k, moodustavad tipud c_i, c_n, c_l ja c_q. Joonisel 1.1 on näha ka tõenäosused sipelga *k* liikumiseks järgmisesse

tippu: p_{ji}, p_{jq}, p_{jl}, p_{jn}, mis tähistavad vastavalt tõenäosust, et sipelgas k liigub tipust c_j tippu c_i, tipust c_j tippu c_q, tipust c_j tippu c_l ja tipust c_j tippu c_n. Nende tõenäosuste arvutamisel on võetud arvesse nii feromoonikoguseid kui ka heuristilise parameetri suuruseid konkreetsetel servadel. Heuristilise parameetri väärtused sõltuvad ülesandest. Kui sipelgas k otsustab tõenäosusreegli põhjal liikuda järgmisena näiteks tippu c_n, siis tipp c_n enam tema naabrusesse ei kuulu, vaid tipp c_n ja c_j kuuluvad sipelga k külastatud tippude hulka, mida tähistatakse sümboliga M_k . Selline liikumine graafil kestab seni, kuni sipelgas on läbinud kõik graafi tipud täpselt ühe korra.

Feromoon on sipelgaalgoritmis väga tähtsal kohal. See matkib tegelikku feromooni, mida pärissipelgad eritavad. Feromooni kogused τ_{ij} muutuvad algoritmi jooksul. Kui m sipelgat on ühel iteratsioonisammul oma lahendid konstrueerinud, siis hakatakse saadud lahendeid omavahel võrdlema. See, millistele tingimustele peab üks hea lahend vastama, sõltub ülesandest. Vastavalt vajadusele selgitatakse võrdluse käigus parim lahend või näiteks kolm paremat lahendit.

Peale lahendite omavahelist võrdlemist leiab aset feromooni aurustumine. See on protsess, mille käigus feromooni kogust τ_{ij} konstruktsioonigraafi igal serval vähendatakse mingi kindla koguse ulatuses. Päriselus on selline protsess keerulisem, kuna feromoon aurustub pika aja möödudes. Feromoonirajad on väga püsivad ja sõltuvalt sipelgate liigist ning pinnase tüübist püsib feromoon rajal mõnest tunnist mõne kuuni. Selleks, et lahendada optimiseerimisülesandeid tõhusalt, on sipelgaalgoritmis aurustumisaeg muudetud lühemaks. Feromooni aurustumist on vaja selleks, et eristada paremaid teid kehvematest.

Feromooniaurustumine on seotud parameetriga ρ, mis kuulub vahemikku [0,1] ja mida nimetatakse feromooni aurustumisnäitajaks. Aurustumisvalem on järgmine:

$$\tau_{ij}(\text{järgmisel etapil}) = (1-\rho) \cdot \tau_{ij}(\text{eelmisel etapil}) \qquad (1.2)$$

Feromooni aurustumine on vajalik ka otsingu kiire koondumise vältimiseks optimaalse lahendi ümber. Põhjus, miks algoritmi alguses graafi igale servale lisati suur kogus

feromooni, on selles, et kui mingil serval oleks kohe alguses feromoonikogus väike, siis aurustumise tõttu võib sellel serval feromooni kogus muutuda nulliks ning tõenäosus selle serva valimiseks oleks samuti võrdne nulliga.

Peale aurustumist lisatakse feromooni nendesse lahenditesse kuuluvatele servadele, mis võrdluse käigus osutusid headeks. Feromooni kogus parimasse lahendisse kuuluvatel servadel muutub vastavalt reeglile

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}, \tag{1.3}$$

kus lisatav feromooni kogus $\Delta\tau_{ij}$ sõltub lahendi kvaliteedist.

Vaadates tõenäosuse arvutamise valemit (1.1), näeme, et see sõltub nii feromooni kogusest τ_{ij} tippude c_i ja c_j vahel kui ka nn. heuristilisest parameetrist η_{ij}.

Heuristilise parameetri väärtus arvutatakse valemiga:

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{1.4}$$

kus d_{ij}on tippude c_i ja c_j vaheline kaugus. Heuristilist informatsiooni sipelgate poolt ülesande jooksul ei muudeta, kuid see sõltub ülesandest, st iga ülesande puhul võib see olla erinev.

Valemis 1.1 näeme ka parameetreid α ja β, mis kontrollivad vastavalt feromoonijälgede τ_{ij} ja heuristilise informatsiooni η_{ij} osatähtsust. Kui näiteks $\alpha = 0$, siis sipelgas otsustab, millisesse tippu järgmisena minna ainult heuristilise informatsiooni põhjal. Juhul, kui $\beta = 0$, siis otsustab feromooni koguste põhjal.

Igal iteratsioonisammul otsitakse parimaid lahendeid. Kui konkreetsel iteratsioonisammul saadud parim lahend on parem kui eelnevatel iteratsioonisammudel saadud parim lahend, siis uue parima lahendina jäetakse meelde antud iteratsioonisammul saadud lahend. Teisisõnu, igal iteratsioonisammul saadud parimat lahendit võrreldakse seni saadud parima

lahendiga. Protsess kestab seni, kuni kõik iteratsioonisammud on tehtud ning parim lahend leitud.

2. Rändkaupleja ülesanne

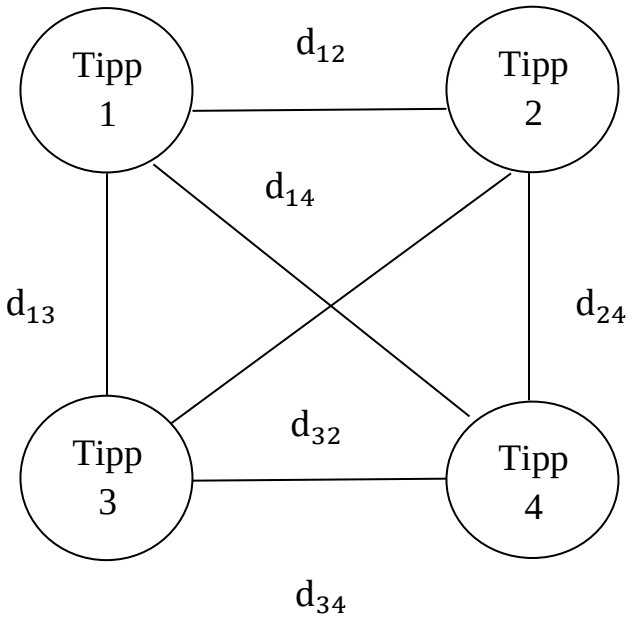
2.1 Rändkaupleja ülesande formuleerimine

Rändkaupleja ülesanne (*Travelling Salesman Problem*) on tuntud kombinatoorne optimiseerimisülesanne, mis formuleeritakse järgnevalt: on antud *n* linna ja nende vahelised kaugused ning on vaja leida lühim teekond, mis läbib kõiki *n* linna täpselt ühe korra, alustades ja lõpetades lahendi konstrueerimist ühest ja samast linnast, mis valitakse juhuslikult. [4]

Rändkaupleja ülesannet on lihtne kirjeldada konstruktsioonigraafi abil. Graafi teooria defineerib rändkaupleja ülesande kui Hamiltoni tsükli leidmist kaalutud täisgraafil [1]. Hamiltoni tee on tee, mis läbib graafi igat tippu parajasti ühe korra ning Hamiltoni tee, mille algustipp ja lõpptipp langevad kokku, on Hamiltoni tsükkel.

Rändkaupleja ülesannet saame kujutada täisgraafil *G = (C,E)*, kus *C* on graafi *G* tippude hulk, mis tähistab ülesandes antud linnade hulka ning *E* on graafi *G* servade hulk, mis tähistab linnadevaheliste ühendusteede hulka. Olgu *C* selline tippude hulk, kuhu kuulub *n* linna, st on antud linnad $c_1, c_2, \dots, c_i, c_j, \dots, c_n \in \mathrm{C}$ ning iga kahe linna vahelist ühendusteed tähistame sümboliga $e_{ij} \in \mathrm{E}$, mis antud juhul tähendab linnade c_i ja c_j vahelist ühendusteed. Graafi *G* kaaludeks on ülesandes antud linnadevaheliste ühendusteede pikkused. Linnade c_i ja c_j vahelist ühendustee pikkust tähistame sümboliga $\mathrm{d_{ij}}$.

Järgnev joonis illustreerib rändkaupleja ülesannet 4 linna korral.



Joonis 2.1. Rändkaupleja ülesande kujutus graafil 4 linna korral.

Jooniselt 2.1 näeme, et on antud 4 linna, mis on tähistatud vastavalt tipp 1, tipp 2, tipp 3 ja tipp 4. Graafi servad tähistavad ühendusteid linnade vahel: tipu 1 ja tipu 2 vaheline serv tähistatakse sümboliga e_{12}. Sümboliga d_{ij} on antud kõigi linnadevaheliste ühendusteede pikkused , mis on linna 1 ja linna 2 korral tähistatud sümboliga d_{12}, linna 2 ja linna 4 puhul vastavalt sümboliga d_{24} jne. Rändkaupleja ülesande puhul on ülesandeks leida lühim teekond, mis läbib kõiki nelja linna ühe korra.

2.2 Rändkaupleja ülesande lahendamine

Rändkaupleja ülesanne on üks tuntumaid NP-keerukusega (*NP-complexity*) ülesandeid [4]. Rändkaupleja ülesande puhul ei ole teada algoritmi, mille puhul ei peaks optimaalse lahendi leidmiseks vaatama läbi kõiki võimalikke teekondi. Rändkaupleja ülesande teeb keeruliseks see, et etteantud linnade arvu kasvades suureneb võimalike marsruutide arv faktoriaalselt ning nende läbivaatamine nõuab väga suuri ressursse.

Üheks võimaluseks, kuidas Rändkaupleja ülesannet lahendada, on vaadelda läbi kõik *n* linna permutatsioonid ehk kõik *n* linna erinevad järjestused ning valida nende seast lühim [5]. Seega võimalike teekondade arv on võrdne arvuga *n!*. See tähendab, et *n* linna korral on meil teekonna moodustamisel esimese linna valikuks *n* võimalust, teise linna valikuks *n-1* võimalust, kuna 1 linn on juba ära valitud, kolmanda linna valikuks *n-2* võimalust ja nii samamoodi jätkates on meil *n-1* linna valikuks 2 võimalust ja viimase, n-nda linna valikuks 1 võimalus. Neid võimalusi kokku korrutades saamegi, et erinevate teekondade moodustamiseks on meil *n!* võimalust.

Kui rändkaupleja ülesandes on antud näiteks 5 linna, siis võimalikke erinevaid teekondi on:

$$Teekondade\ arv = 5! = 120.$$

Nende läbivaatamine ei tundugi kõige raskem. Kuid ülesande puhul, kus on antud 40 linna, on permutatsioonide arv väga suur:

$$Teekondade\ arv = 40! \approx 8.1591 * 10^{47}.$$

2.3 Rändkaupleja ülesande lahendamine sipelgaalgoritmiga

Teadlased, eriti just matemaatikud ja infotehnoloogid, on 1950. aastatest alates püüdnud leida efektiivsemaid lahendamismeetodeid selliste keeruliste ülesannete lahendamiseks nagu on rändkaupleja ülesanne. Praeguseks on teada mitmeid heuristilisi algoritme, mis võimaldavad leida optimaalsele lahendile lähedase lahendi palju väiksema aja jooksul. Üheks tuntud algoritmiks, mida on kasutatud rändkaupleja ülesande lahendamiseks, on eelpool kirjeldatud sipelgaalgoritm. Sipelgaalgoritmiga on saadud optimaalsele lahendile väga lähedasi tulemusi.

Rändkaupleja ülesande lahendamisel sipelgaalgoritmiga on lahendi komponentideks tehissipelgate liikumised linnade vahel, st liikumine linnast c_i linna c_j on lahendi komponent e_{ij}. Olgu meil ülesande lahendamiseks *m* sipelgat. Algoritmi iga iteratsioonisammu alguses valivad kõik *m* sipelgat endale juhuslikult algustipu, mis on

ühtlasi ka lõpptipp. Kõik m sipelgat konstrueerivad lahendeid liikudes ühest linnast teise ehk ühest graafi tipust teise tõenäosusreegli järgi. Tõenäosus, et sipelgas k liigub tipust c_i tippu c_j on antud valemiga:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{il \in N_j^k} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}, \tag{2.1}$$

kus sümboliga η_{il}on tähistatud heuristiline parameeter serval e_{il} ja sümboliga τ_{il} on tähistatud feromoonikogus serval e_{il}. Feromoonikoguste astendaja α on parameeter, mis kontrollib feromooni τ_{ij} osatähtsust tõenäosuse arvutamisel ning β on parameeter, mis kontrollib heuristilise parameetri η_{ij} osatähtsust tõenäosuse arvutamisel. Heuristilise parameetri η_{ij} väärtus ülesande jooksul ei muutu. Tõenäosusreegel võtab arvesse nii feromooni koguse väärtust kahe linna vahelisel ühendusteel kui ka heuristilise parameetri väärtust, milleks rändkaupleja ülesande puhul on tihti võetud linnadevaheliste ühendusteede pikkused d_{ij}. Parameetritel α ja β on väga tähtis roll ülesande lahendamisel ning nende väärtuste muutmisel võib ülesande tulemus väga palju muutuda. Selleks, et leida head lahendit, tuleb leida õiged väärtused nendele parameetritele. Parameetrite α ja β väärtusi on raske määrata ja tihti saadakse nende parameetrite parimad väärtused proovimise teel. [4]

Tehissipelgad eelistavad liikuda linnadesse, kus ühendusteedel on rohkem feromooni ja mis asuvad lähemal. Siit tulenebki üks tehissipelgate eripära võrreldes päris sipelgatega. Rändkaupleja ülesande lahendamiseks on tehissipelgad võimelised kindlaks määrama, kui kaugel on linnad üksteisest ja neil on mälu selleks, et mäletada juba külastatud linnu. Iga lahendi konstrueerimise protsessi alguses ehk iga iteratsioonisammu lõpus tehissipelgate mälu tühjendatakse ja täiendatakse uuesti järgmisel iteratsioonisammul igal liikumisel ühest linnast teise. Tehissipelgad valivad igas linnas järgmise linna, kuhu liikuda, nende hulgast, mida nad veel külastanud ei ole. Nende kõrvalasuvate ja külastamata linnade hulka nimetatakse naabruseks. Tipus c_i asuva sipelga k naabrus tähistatakse sümboliga N_i^k ning juba külastatud tippude arv tähistatakse sümboliga M_k. Vaadates joonist 2.1 näeme, et juhul, kui sipelgas alustab teekonda tipust 1, siis tema naabruseks N_1^k on sel hetkel kõik

ülejäänud tipud ehk tipp2, tipp3 ja tipp 4. Sel hetkel külastatud tippude arv on 1 ehk tipp 1. Kui sipelgas liigub tipust 1 tippu 3, siis külastatud tippude hulka kuuluvad tipud 1 ja 3 ning naabrusesse N_3^k kuuluvad tipud 2 ja 4.

Tõenäosuse arvutamisel vajaliku heuristilise parameetri väärtuse arvutamiseks kasutatakse valemit:

$$\eta_{ij} = \frac{1}{d_{ij}}, \tag{2.2}$$

kus d_{ij}on tippude c_i ja c_j vaheline kaugus.

Peale seda, kui tehissipelgad on mingil kindlal iteratsioonisammul saanud valmis lahendid, hakatakse neid omavahel võrdlema. Parim lahend ehk lühim teekond, mis läbis kõiki linnu täpselt ühe korra, saab kõige rohkem feromooni, st feromoonikogus, mis lisatakse, on pöördvõrdeline teekonna pikkusega. Mida lühem teekond, seda rohkem feromooni lisatakse. Enne seda, kui paremuse alusel lahenditele feromooni lisatakse, aurustatakse kõigilt graafi servadelt teatud kogus feromooni. Aurustamine toimub sellepärast, et vältida väga heade servade valimist kõigi sipelgate poolt. Aurustumine toimub järgmise reegli järgi:

$$\tau_{ij}(\text{järgmisel etapil}) = (1-\rho)\cdot\tau_{ij}(\text{eelmisel etapil}). \tag{2.3}$$

Feromooni aurustumine on seotud parameetriga ρ, mis kuulub vahemikku [0,1] ja mida nimetatakse feromooni aurustumisnäitajaks.

Peale aurustumist lisatakse headesse lahenditesse kuuluvatele servadele feromooni. Seda tehakse järgneva reegli põhjal:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}, \tag{2.4}$$

kus lisatav feromoonikogus $\Delta\tau_{ij}$ sõltub lahendi kvaliteedist.

Parim teekond jäetakse meelde ja seejärel alustatakse järgmist iteratsioonisammu. Kui järgneval iteratsioonisammul saadud parim teekond on parem, kui eelnev meelde jäetud parim teekond, siis see asendatakse ja nii iga iteratsioonisammu korral, kuni saadakse optimaalsele lahendile lähim lahend.

2.4 Näide rändkaupleja ülesande lahendamisest sipelgaalgoritmiga

Järgnevalt on toodud üks näide, kus sipelgaalgoritmiga lahendati rändkauplejaülesanne, mille puhul taheti leida lühim tee 32 Slovakkia linna vahel. Esmalt leiti optimaalne lahend matemaatilise optimiseerimisprogrammiga Gams. Lahend leiti 17498 iteratsioonisammuga ning optimaalseks teekonna pikkuseks saadi 1453 km.

Teiseks püüti leida sama ülesande lahend, kasutades selleks sipelgaalgoritmi. Tõenäosusi arvutades võeti parameetrite α ja β väärtusteks $\alpha = 1$ ja $\beta = 5$. Iteratsioonisammude arvuks võeti 1000. Algoritm käivitati neljal korral, kus iga kord sipelgate arv oli erinev. Esimesel korral oli 100 sipelgat. Lahendiks saadi teekond pikkusega 1713 km, mis erines optimaalsest lahendist rohkem kui 17%. Edasi katsetati 1000 sipelgaga ja teekonna pikkuseks saadi 34. iteratsioonisammul 1621 km. See tulemus erines optimaalsest 11.56%. Seejärel suurendati sipelgate arvu 5000 ja juba 21. iteratsioonisammul saadi tulemuseks teekond pikkusega 1532 km. Erinevus optimaalsest lahendist oli 5.44%. Viimasel korral võeti sipelgate arvuks 10 000 ning 242 iteratsioonisammuga saadi teepikkuseks 1465 km, mis erines optimaalsest lahendist kõigest 0.83%. [4]

Kokkuvõttes selgus sipelgaalgoritmiga rändkaupleja ülesande lahendamisel, et lahendi kvaliteet sõltus väga palju sipelgate arvust. Sipelgaalgoritm andis väga hea tulemuse võrdlemisi väikese arvu iteratsioonisammudega ja oli võimeline leidma optimaalsele lahendile väga lähedase lahendi suhteliselt lühikese ajaga.

2.5 Rändkaupleja ülesande rakendused

Vaadates enda ümber, ei teagi me tegelikult, et igapäevaelus puutume kokku väga paljude ülesannetega, mis sisult ongi rändkaupleja ülesanded. Rändkaupleja ülesandel on väga palju rakendusi erinevates valdkondades. Rändkaupleja ülesanne on populaarne logistiliste ja planeerimisülesannete lahendamisel. Samuti on selle kasutamine levinud tehnoloogilistes rakendustes. Rändkaupleja ülesannet kasutatakse riistvara seadmete disainimisel, seoses sidega ka raadiote elektroonilistes seadmetes ja arvutivõrkude ülesehitustel. Lisaks on rändkaupleja ülesannet kasutatud mõnede probleemide formuleerimisel tööstuses, näiteks tootmisliinidel ning masinate sõiduplaanide koostamisel. Rändkaupleja ülesanne on paljudes valdkondades ka kui alamülesanne, näiteks DNA järjestuse korral, kus linnadeks on DNA fragmendid ja vahemaadeks linnade vahel on fragmentide sarnasused. On ka ülesandeid, kus kasutatakse täiendavaid piiranguid, näiteks ressursside piiratud kogus või ajapiirangud. Need muudavad rändkaupleja ülesande lahendamise tunduvalt raskemaks. Järgnevalt on kirjeldatud mitmeid rändkaupleja ülesande rakendusi lähemalt.

1. Trükkplaatide puurimine

Üheks ülesandeks, mille lahendamisel saab kasutada rändkaupleja ülesannet, on trükkplaatide (*printed circuit boards*) puurimine. Trükkplaat on montaažiplaat, mida kasutatakse elektroonikas. Trükkplaadile on võimalik paigaldada elektroonikakomponendid ja need elektriliselt ühendada. Üheks viimaseks toiminguks trükkplaatide tootmisel on aukude puurimine plaadile. Need augud on vajalikud erinevate komponentide, näiteks transistorite ja vooluringi kinnitamiseks plaadile või ühenduste loomiseks erinevate plaadikihtide vahel. Tavaliselt on need puuritavad augud erineva läbimõõduga. Selleks, et puurida kahte erineva läbimõõduga auku järjestikku, peab vahepeal ära vahetama puuri otsiku. Selline iga järgneva augu korral puuri otsiku vahetamine on üsna aeganõudev tegevus. Sellepärast on mõttekas ära puurida kõik ühesuurused augud ja seejärel vahetada puuri otsik. Seega on selge, et tuleb valida mingi läbimõõduga auk, valida selle puurimiseks sobiv otsik ja puurida sellega kõik augud, mis on sama läbimõõduga. Seejärel tuleb vahetada puuri otsik ja puurida järgmise läbimõõduga augud jne. Sellist puurimise

ülesannet võib vaadelda kui rändkaupleja ülesande rakendamist ühesuurustele aukudele, kus ühesuguse diameetriga augud moodustavad linnade hulga, mida ühe puuri otsaga puuritakse. Need aukudevahelised kaugused, mis antakse puuri otsiku vahetamise ajal, vastavad ajale, mis kulub puuri liikumiseks ühe augu juurest teise juurde. Eesmärgiks on leida ühesuuruste aukude puurimiseks lühim tee, mis kokkuvõttes aitab vähendada puurimiseks kuluvat aega. [6]

2. Röntgenkristallograafia

Kristallide struktuuri analüüs on samuti üks rändkaupleja ülesande rakendustest. Kristallide struktuuri analüüsimisel kasutatakse röntgendifraktomeetrit, et saada teavet kristalsete ainete struktuuri kohta. Selleks mõõdab detektor, liikudes mööda kristalli, rõntgenkiire peegelduste intensiivsust kristallilt. Kuigi mõõtmistulemus ise võib olla saavutatud üsna kiiresti, on masina ümberpositsioneerimine kristallil väga aeganõudev. Ülesandeks on kaardistada kristalli struktuur nii, et ümberpositsioneerimise teekond kristallil oleks minimaalne. See aitab muuta kristalsete ainete struktuuri analüüsimise kiiremaks. [6]

3. Sõiduki marsruutimine

Sõiduki marsruutimise ülesanne on väga levinud mitmetes valdkondades nagu näiteks transpordis, turustamises ja logistikas. Sageli on ülesande sisuks leida lühim marsruut kõikide klientide tellimuste jagamiseks, kasutades selleks kindlat sõidukite arvu. Lahend peab olema selline, mis tagab, et kõik kliendid on teenindatud tegevuspiiranguid arvestades, milleks võivad olla näiteks sõidukite arv, sõiduki võimsus ja juhi maksimaalne tööaeg. Eesmärgiks on leida selline marsruut, et selle läbimise kulud kauba jagamisel oleksid minimaalsed, st eesmärgiks on minimiseerida marsruudi pikkust või selle läbimiseks kuluvat aega. Näitena võib tuua olukorra, kus n klienti vajavad teatud koguses tarbekaupu ja tarnijad peavad vastama kõikidele nõudmistele teatud arvu veoautodega. Ülesandeks on määrata, milline veoauto milliseid kliente teenindab ja tarnimise ajakava igale veoautole nii, et ühegi veoauto mahutavus ei ole ületatud ja kogu teekonna pikkus on minimaalne. Teiseks võib tuua näite, kus oletatakse, et ühe linna n postkasti tuleb tühjendada iga päev teatud aja jooksul, näiteks 1 tunni jooksul. Ülesandeks on leida minimaalne veoautode arv

postkastide tühjendamiseks ja teostuseks vajalik lühim aeg selle arvu veokitega. Mitmed variandid nendest kahest ülesandest, kus aeg ja mahupiirangud on seotud, on levinud paljudes päriselu probleemides. See probleem on lahendatav rändkaupleja ülesandena, kui seal ei ole aja- ega mahupiiranguid ning kui veoautode arv on fikseeritud. [6, 8]

4. Kauba toimetamine ladudest klientideni

Tellimuste ettevalmistamise operatsioon on ladudes üks logistiline protsess. See seisneb selles, et kogutakse kindel kogus tarbekaupu kokku enne nende toimetamist tarbijateni. See on põhiline ladustamise protsess ja sellel on oluline mõju tarneahela tootlikkusele. Tellimuste ettevalmistamine seisneb selles, et ladudes on igale tootele määratud oma koht ning ühe kliendi tellimuse kokkupanemiseks on vaja lao erinevatest paikadest tuua kaup kokku. Sellist tegevust peetakse kõige töömahukamaks ja kulukamaks tegevuseks iga lao jaoks, kus kulud selleks on hinnanguliselt 55% kogu lao tegevuskuludest. Kuna see hõlmab suuri kulutusi ja võib mõjutada klientide rahulolu, pööratakse suurt tähelepanu selle protsessi kiiremaks ja odavamaks toimimiseks. [7]

See ülesanne on seotud materjalide kogustega laos. Oletame, et teatud kogus kaubaartikleid ladustatakse laos. Sõidukid peavad koguma tellimuses esinevad kaubad kokku, et saata need kliendile. Seos rändkaupleja ülesandega on kohe näha. Laoruumide asukohad vastavad graafi tippudele. Vahemaa kahe tipu vahel on antud aeg, mis on vajalik sõiduki liikumiseks ühest kohast teise. Ülesandeks on leida lühim sõiduki liikumise marsruut. See aitab hoida kokku masina liikumiseks vajalikke kulusid, aega ja parandab klientide rahulolu. [6]

5. Koolibussi transport

Sõiduplaani eesmärk on leida selline busside liikumine, et marsruutide arv on minimaalne ja kogudistants, mida bussid kokku läbivad, on samuti minimaalne. Seejuures on oluline, et ükski buss ei ole ülekoormatud ja aeg, mis kulub iga marsruudi läbimiseks, ei ületa lubatud maksimumi. [6]

6. Sularaha vedu pangas

Rändkaupleja ülesannet on hea rakendada ka panga transpordikulude vähendamiseks. Kuna pankades saab teha erinevaid sularahatehinguid, on vajalik ka raha transportimine panga erinevatest kontoritest peakontorisse. Selle ülesande puhul on kõik ühe kindla panga kontorid rändkaupleja ülesandes nn linnad, mida tuleb külastada ühe korra. Teekond algab ja lõpeb peakontoris. Ülesandeks on leida lühim teekond raha toimetamiseks peakontorisse. [6]

7. Missiooni planeerimise ülesanne

Missioonide kavandamise puhul on samuti võimalik kasutada rändkaupleja ülesannet. Missioonide planeerimise ülesanne seisneb selles, kuidas määrata igale armeeliikmele optimaalset liikumisteekonda nii, et missiooni eesmärk saavutatakse minimaalse ajaga. Missiooni planeerija saab kasutada rändkaupleja ülesannet, kus on antud n armeeliiget, m sihtkohta, mis peavad olema külastatud n liikme poolt ning antud on ka kindel sihtkoht, kust armeeliikmed peavad liikumist alustama ja kuhu peavad ka missiooni lõpuks jõudma. Ülesandeks on leida lühim teekond, mis vastab eelnevatele tingimustele. Rändkaupleja ülesande kasutamine aitab vähendada missioonikulusid. [6]

3. Teekond Eesti maakonnakeskuste vahel

Käesolevas peatükis on lahendatud ülesanne, milles on vaja leida lühim teekond, mis läbib kõiki Eesti 15 maakonnakeskust täpselt ühe korra. Nendeks keskusteks, mida teekond pidi läbima, olid järgnevad linnad: Tallinn, Haapsalu, Kärdla, Kuressaare, Pärnu, Rapla, Paide, Rakvere, Jõhvi, Jõgeva, Tartu, Viljandi, Põlva, Võru ja Valga. Ülesanne lahendati sipelgaalgoritmiga programmis Matlab. Linnadevahelised kaugused on leitud linnulennuliselt, kasutades interaktiivset veebivahendit [10]. Arvutatud kaugused on näha lisas 1 esitatud tabelis. Ülesande lahendamisel saadud lahendeid illustreerivate jooniste tegemisel on kasutatud Eesti kaarti, mis on saadud allikast [11] ning jooniste täiustamiseks on kasutatud programmi Jing. Sipelgaalgoritmiga rändkaupleja ülesande lahendamiseks vajalik programm on võetud allikast [9].

Järgnevalt kirjeldatakse ülesande algandmeid. On leitud Eesti 15 linna vahelised kaugused. Sipelgaalgoritmiga ülesande lahendamise tulemus sõltub palju tema tööd mõjutavatest viie parameetri väärtustest. Nendeks viieks parameetriks on sipelgate arv, iteratsioonisammude arv, aurustumisparameeter ρ, feromoonikoguse astendaja α ja heuristilise parameetri astendaja β. Parameetrite algväärtused ülesande lahendamisel on esitatud järgnevas tabelis:

Parameeter	Sipelgate arv	Iteratsioonisammude arv	ρ	α	β
Väärtus	10	1000	0.1	1	2

Tabel 3.1. Parameetrite algväärtused lühima teekonna leidmisel 15 Eesti linna vahel

Kasutades antud parameetrite väärtusi tabelis 3.1 saadi ülesande lahendiks teekond pikkusega 948.79 km. Linn, kust teekond alguse sai, valiti juhuslikult ja selleks osutus Kuressaare. Teekond nägi välja järgmine:

Kuressaare – Pärnu – Viljandi – Valga – Võru – Põlva – Tartu – Jõgeva – Jõhvi – Rakvere – Paide – Rapla – Tallinn – Haapsalu – Kärdla – Kuressaare

Antud ülesande puhul osutus selline teekond ka parameetrite muutes parimaks. Seda teekonda illustreerib joonis, mille leiab lisast 3.

Järgnevalt on selgitatud erinevate parameetrite mõju sipelgaalgoritmile ja nende olulisus antud ülesande lahendamisel. Sipelgaalgoritmi tööd saab juhtida viie parameetri abil:

- Sipelgate arv

Üheks parameetriks, mis sipelgaalgoritmi tööd mõjutab, on sipelgate arv. Mida rohkem sipelgaid ülesandele lahendust otsivad, seda rohkem aega kulub iteratsioonisammu tegemiseks ja seda rohkem aega kulub seega ka algoritmi töö tegemiseks. Sipelgaalgoritmiga ülesandeid lahendades on märgitud, et mida suurem on sipelgate arv, seda kvaliteetsemad on iteratsioonisammude lahendid ja seda väiksema arvu iteratsioonisammudega on võimalik leida sobiv lahend. Antud ülesande puhul, kus leidsime lühima teekonna Eesti maakonnakeskuste vahel, valisime algseks sipelgate arvuks 10. Programmi mitmeid kordi käivitades püüdsime uurida sipelgate arvu mõju tulemusele. Sipelgate arve katsetasime erinevaid: 10, 15, 20, 30 ja 60 sipelgat. Teiste parameetrite samaks jäädes tõdesime, et antud ülesande puhul sipelgate arv ülesande lahendi kvaliteeti ei mõjutanud. Saime kõigi erinevate sipelgate arvu väärtuste korral sama tulemuse, teekonna pikkuseks 948.79 km. Kuid erinevalt teekonna pikkusest mõjutas sipelgate arv algoritmi töötamise aega. Kui lahendi leidmiseks oli võetud 10 sipelgat, siis algoritm tegi oma töö ära keskmiselt 38.5 sekundiga. Kui sipelgaid oli 15, siis oli tööajaks u 60 sekundit, 20 sipelgaga 70 sekundit, 30 sipelgaga pisut üle saja sekundi ning 60 sipelga puhul kulus algoritmi töö tegemiseks u 230 sekundit. Seega antud ülesande puhul kehtis seos, mida suurem sipelgate arv, seda kauem töötas algoritm, kuid ülesande tulemus jäi samaks.

- Iteratsioonisammude arv

Mida rohkem iteratsioonisamme ülesande lahendamiseks tehakse, seda rohkem lahendeid sipelgad konstrueerivad. Esialgseks iteratsioonisammude väärtuseks oli 1000 sammu. Selle puhul leiti ülesande tulemus u 38 sekundiga. Iteratsioonisammude suurenedes leidis algoritm lahendiks ikkagi varem leitud parima lahendi, mille pikkuseks oli 948.79 km. Iteratsioonisammude suurendamine mõjutas aga algoritmi tööaega. Iteratsioonisammude arvuks võtsime 1100, 1200, 1500, 2000, kuid proovisime ka hästi väikeseid väärtusi nagu näiteks 50, 75 ja 100 iteratsioonisammu. Kui iteratsioonisammude arvuks oli suur arv, üle tuhande sammu, siis algoritmi tööaeg suurenes 2000 iteratsioonisammu korral kuni 80 sekundini. Kuigi tööaeg pikenes, ei mõjutanud suurem iteratsioonisammude arv algoritmi tööpikkust nii palju, kui sipelgate arvu suurenemine mõjutas. Kui iteratsioonisammude arv oli 100 või 75, siis seni parim lahend 948.79 km leiti ja seda väga väikese ajaga 3.84 ja 1.91 sekundiga. Kui võtsime iteratsioonisammude arvuks 50 sammu, siis parima tulemuseni ei jõutud ning teekonnapikkusteks saadi 997.41 km, mis on üle 50 km pikem kui varem leitud parim tulemus. Leitud teekond oli järgmine:

Kuressaare – Kärdla – Haapsalu – Rapla – Tallinn – Rakvere – Jõhvi – Jõgeva – Tartu – Põlva – Võru – Valga – Viljandi – Paide – Pärnu – Kuressaare

Seda teekonda illustreerib joonis lisas 3.

- Aurustumisparameeter ρ

Aurustumisparameeter ρ määrab, millise kiirusega vähenevad feromooni kogused konstruktsioonigraafi servadel igal iteratsioonil enne feromooni väljastamist. Parameetri ρ väärtused kuuluvad lõiku [0,1]. Kui $\rho = 0$, siis feromooni ei aurustata ning kui $\rho = 1$, siis kogu feromoonikogus servadel aurustatakse. Kui ρ väärtused on suuremad, siis otsing ei koondu väga kiiresti. Erinevaid sipelgaalgoritmiga lahendatud ülesandeid vaadates on ρ väärtuseks võetud väike väärtus, näiteks 0.1. Antud ülesande parameetreid katsetades valisime aurustumisteguri ρ väärtusteks 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, ja 0.9. Väärtusega $\rho = 0.1$ andis algoritm parimaks tulemuseks 948.79 km. Suurendades ρ

väärtusi 0.3, 0.4, 0.5, ei andnud algoritm enam nii häid tulemusi. Enamikel kordadel leiti tulemuseks 948.81 km. Teekond pikkusega 948.81 km nägi välja selline:

Kuressaare – Kärdla – Haapsalu – Tallinn – Rapla – Paide – Rakvere – Jõhvi – Jõgeva – Tartu – Põlva – Võru – Valga – Viljandi – Pärnu – Kuressaare

Seda teekonda illustreerib joonis lisas 4.

Kui ρ väärtuseks oli 0.6 või 0.7, siis saime tulemuseks teekonna pikkusega 948.79 km. Väärtuste 0.8 ja 0.9 korral oli algoritmi lahendi kvaliteet halvem, teekond pikenes mitmekümne kilomeetri võrra. Näidetena võib tuua teekonnad pikkusega 965.26 km, 975.71 km, 994.32 km, kuid mõnel üksikul korral jõuti ka seni parima lahendini, 948.79 km. Algoritmi tööajale ρ väärtus erilist mõju ei avaldanud. Kõigil kordadel jäi tööaeg 38 ja 39 sekundi vahele.

- Alfa

Sümboliga α tähistasime feromoonikoguste astendaja, mis kontrollib feromooni τ_{ij} osatähtsust tõenäosuse arvutamisel. Ülesannet lahendades nägime, et andes alfale väärtused 1 või 2, siis algoritm leidis lahendiks teekonna pikkusega 948.79 km. Suuremate väärtuste 3, 4 ja 5 korral saime teepikkuseks rohkem kui 1350 km. Seega annab algoritm parema tulemuse, kui alfa väärtus on väike.

- Beeta

β on parameeter, mis kontrollib heuristilise parameetri η_{ij} osatähtsust tõenäosuse arvutamisel. Heuristilise parameetri η_{ij} väärtus ülesande jooksul ei muutu. Selle ülesande lahendamisel osutus ka β selliseks parameetriks, mille muutmine ülesande lahendit ja ülesande lahendamisele kuluvat aega ei mõjutanud. Katsetasime erinevaid väärtusi lõigust [1,8], kuid algoritmi tööaeg oli ikkagi u 38 sekundit ja teekonna pikkuseks saadi 948.79 km.

- Mitme parameetri muutmine korraga.

Antud ülesande puhul uurisime ka mitme parameetri muutmise mõju ülesande tulemusele. Märkimisväärseid muutusi ülesande lahendile mitme parameetri muutmine korraga kaasa ei toonud. Pigem oli muutus iga parameetri puhul selline nagu nende muutmine ühe kaupa. Ühe näitena võib tuua olukorra, kus sipelgate arvuks oli 10, iteratsioonisammude arv 100, $\alpha = 1$, $\beta = 4$ ja $\rho = 0.1$. Tulemus osutus selliste parameetrite väärtuste korral küll üsna kehvaks, 981.29 km, kuid tulemus leiti märkimisväärselt väikese ajaga, 0.5 sekundiga.

Teekonnaks oli:

Kuressaare - Pärnu – Viljandi – Valga – Põlva – Võru – Tartu – Jõgeva – Jõhvi – Rakvere – Paide – Rapla – Tallinn – Haapsalu – Kärdla – Kuressaare

Antud teekonda illustreerib joonis lisas 5.

Kokkuvõtvalt võib öelda, et antud ülesande parimaks lahendiks osutus teekond pikkusega 948.79 km. Selline teekond saavutati ka algandmetega. Kiireim aeg, millega selle lahendini jõuti, oli 1.91 sekundit. Sel korral olid parameetrite väärtused järgmised: sipelgate arvuks oli 10, iteratsioonisammude arvuks 75, $\alpha = 1$, $\beta = 2$ ja $\rho = 0.1$.

Ant Algorithm for Travelling Salesman Problem

Bachelor's thesis

Abstract

Teele Laas

Complex combinatorial optimization problems have arised in many different fields. However, often this kind of problems are very hard to solve in practice, scientists have worked out many algorithms for solving combinatorial optimization problems. Ant Algorithm is a recent metaheuristic method that is one of the applicable algorithms for solving optimization problems. The first chapter of this bachelor's thesis gives an overview of Ant Algorithm. Ant Algorithm was first introduced by Dorigo and his colleagues in early 1990s and it is inspired by the behavior of real ants. People have explored the nature and have tried to understand different kinds of processes around us. A very interesting aspect of the behavior of several ant species is their ability to find shortest paths between the ant's nest and the food sources. In the Ant Algorithm there are several artificial ants' capabilities used to find solutions to the problem. Although the Ant Algorithm is quite new method, it is a well defined and good performing method that is more and more often applied to solve a variety of complex combinatorial problems.

One of the Ant Algorithm's successful applications is Travelling Salesman Problem, which was the first famous combinatorial problem solved by Ant Algorithm. Ant Algorithm is one of the most efficient algorithms for Travelling Salesman Problem. The second chapter of this bachelor's thesis is dedicated to introducing the Travelling Salesman Problem. Travelling Salesman Problem is easy to describe but it is so difficult to solve. Travelling Salesman Problem is a well known and extensively studied problem and it has wide application background. We can't imagine how many problems of our real life can be solved as Travelling Salesman Problems. In this chapter an overview of current applications is given.

The last chapter of this thesis is a practical part of it. A certain Travelling Salesman Problem is solved with the Ant Algorithm there. The task was to find the shortest route between 15 different counties' centers of Estonia. In this chapter the description and importance of different parameters in Ant Algorithm is given and the relationship between Ant Algorithm's parameters is analyzed. Ant Algorithm is led by five parameters. The result of the task was a 948.79 km long tour through all these 15 cities which was found in 1.91 seconds. It was the fastest time to find the solution. Ant Algorithm showed good performance for solving this problem. Computings were realized in MATLAB environment.

This thesis is interesting to read for those who would like to get some knowledge about mathematical optimization problems. It is interesting to know the practical side of math. In addition, this thesis is useful for those who want to solve different kinds of optimization problems using Ant Algorithm.

Kasutatud kirjandus

[1] F.T. Gonzalez. Handbook of Approximation Algorithms and Metaheuristics, M. Dorigo, K. Socha. peatükk 26 Ant Colony Optimization, Chapman & Hall, 2007.

[2] M. Dorigo, M. Birattari, T. Stützle. Ant Colony Optimization, IRIDIA, 2006.

[3] O. Cordon, F. Herrera, T. Stüzle. A Review on the Ant Colony Optimization Metaheuristic: Basis, Models and New Trends, Mathware & Soft Computing, 2002.

[4] I. Brezina, Z. Čičková. Solving the Travelling Salesman Problem Using the Ant Colony Optimization, 2010.

[5] Zar Chi Su Su Hlaing, May Aye Khine, Solving Traveling Salesman Problem by using Improved Ant Colony Optimization Algorithm, International Journal of Information and Education Technology, Vol. 1, No. 5, 2011.

[6] R. Matai, S.P. Singh, M.L. Mittal. Traveling Salesman Problem: An overview of Applications, Formulations, and Solution Approaches

[7] M. Martin. Order Picking In The Warehouses [http://logistics.about.com/od/operationalsupplychain/a/order_pick.htm] mai 2013

[8] A.E. Rizzoli, F. Oliverio, R. ontemanni, L.M. Gambardella. Ant Colony Optimization for vehicle routind problems: from theory to applications. IDSIA [http://www.idsia.ch/idsiareport/IDSIA-15-04.pdf] aprill, mai 2013

[9] Georigios Sarigiannidis, 2004 [http://www.mathworks.se/matlabcentral/newsreader/view_thread/63740] mai 2013

[10] Distance between [http://distancebetween.net/cities.html]

[11] Tallinna Ülikooli Riigiteaduste Instituut, 2011

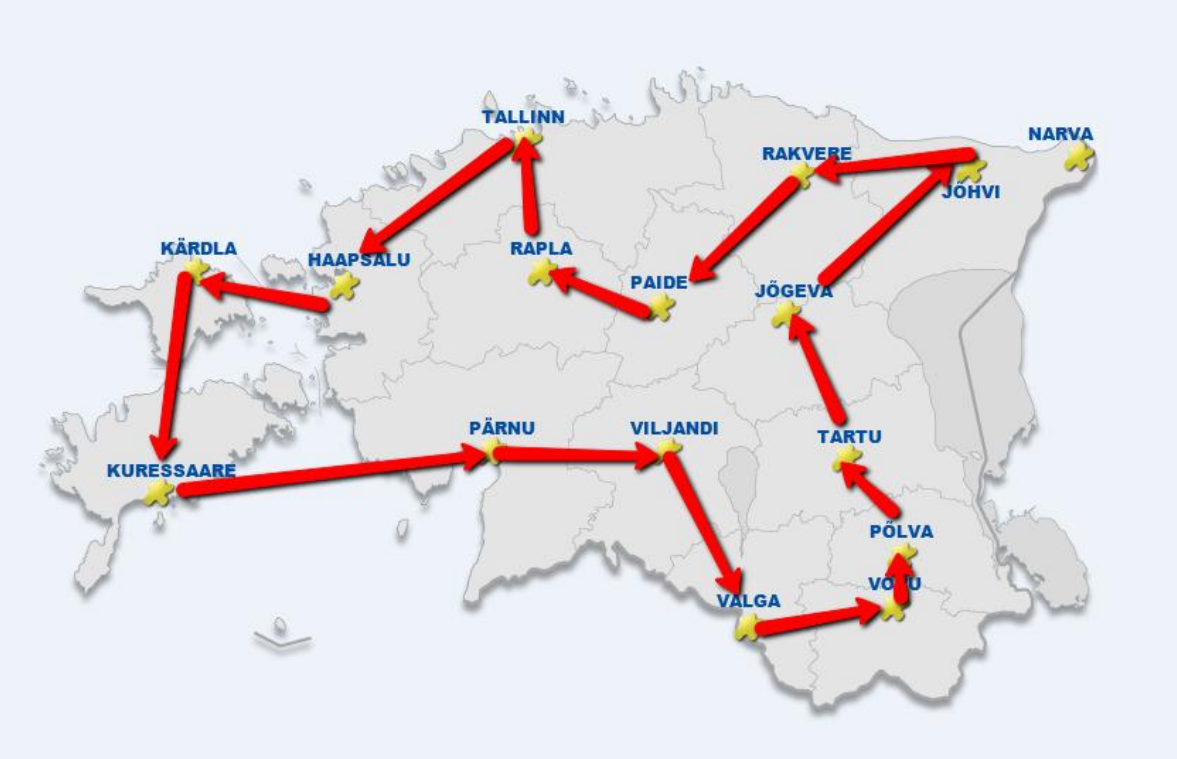
[http://riigiteadused.tlu.ee/elteemapaev/index.php]

Lisad

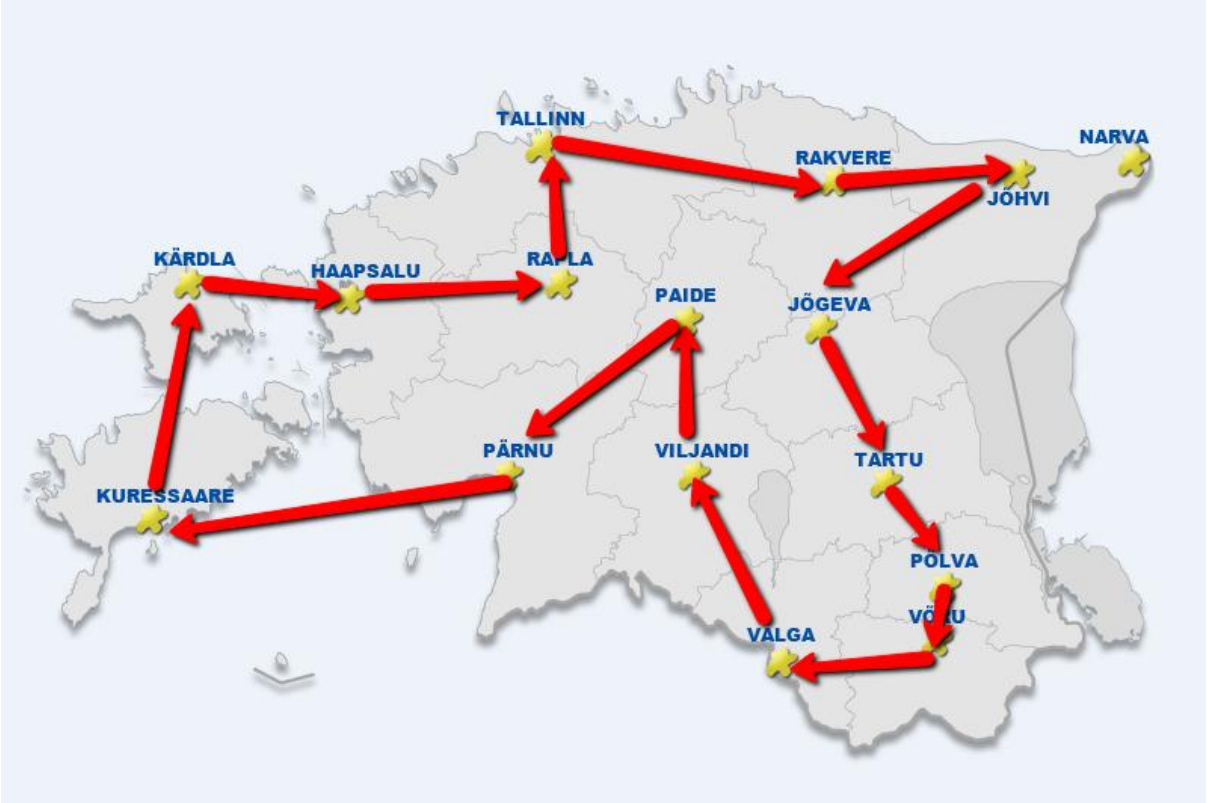
Lisa 1. Eesti linnade vahemaade tabel (km)

Eesti linnad	Tallinn	Haapsalu	Kärdla	Kuressaare	Pärnu	Rapla	Paide	Rakvere	Jõhvi	Jõgeva	Tartu	Viljandi	Põlva	Võru	Valga
Tallinn	0	86.87	122.6	183.72	118.42	47.6	77.15	92.67	152.66	122.03	165.52	128.87	203.87	221.87	199.67
Haapsalu	86.87	0	45.81	97.9	84.59	72.07	115.9	166.63	225.95	165.52	195.58	134.86	227.21	237.23	195.47
Kärdla	122.6	45.81	0	84.56	123.27	117.02	161.55	209.1	269.12	211.32	240.85	178.66	271.52	280.27	235.35
Kuressaare	183.72	97.9	84.56	0	118.22	157.08	190.69	253.22	308.88	232.67	247.53	180.72	268.61	269.66	215.12
Pärnu	118.42	84.59	123.27	118.22	0	72.15	82.92	151.29	199.77	116.63	129.59	62.78	153.69	159.01	112.1
Rapla	47.6	72.07	117.02	157.08	72.15	0	45.88	96.7	154.73	96.47	133	85.08	168.94	183.91	155.29
Paide	77.15	115.9	161.55	190.69	89.92	45.88	0	68.57	118.68	50.59	59.41	58.04	127.08	144.73	126.8
Rakvere	92.67	166.63	209.1	253.22	151.29	96.7	68.57	0	60.4	66.72	111.17	117.78	148.82	172.51	175.51
Jõhvi	152.66	225.95	269.12	308.88	199.77	154.73	118.68	60.4	0	89.93	117.23	152.74	145.84	171.19	193.42
Jõgeva	122.03	165.52	211.32	232.67	116.63	96.47	50.59	66.72	89.93	0	46.74	63.13	85.87	107.88	110.02
Tartu	165.52	195.58	240.85	247.53	129.59	133	59.41	111.17	117.23	46.72	0	66.84	39.21	61.47	77.32
Viljandi	128.87	134.86	178.66	180.72	62.78	85.08	58.04	117.78	152.74	53.13	66.84	0	92.99	102.6	70.8
Põlva	203.87	227.21	271.52	268.61	153.69	168.94	127.08	148.84	145.84	85.87	39.21	92.99	0	23.35	68.33
Võru	221.87	237.23	280.27	269.66	159.01	183.91	144.73	172.51	171.19	107.88	61.47	102.6	23.35	0	58.09
Valga	199.67	195.47	235.35	215.12	112.1	155.29	126.8	175.51	193.42	110.02	77.32	70.8	68.33	58.09	0

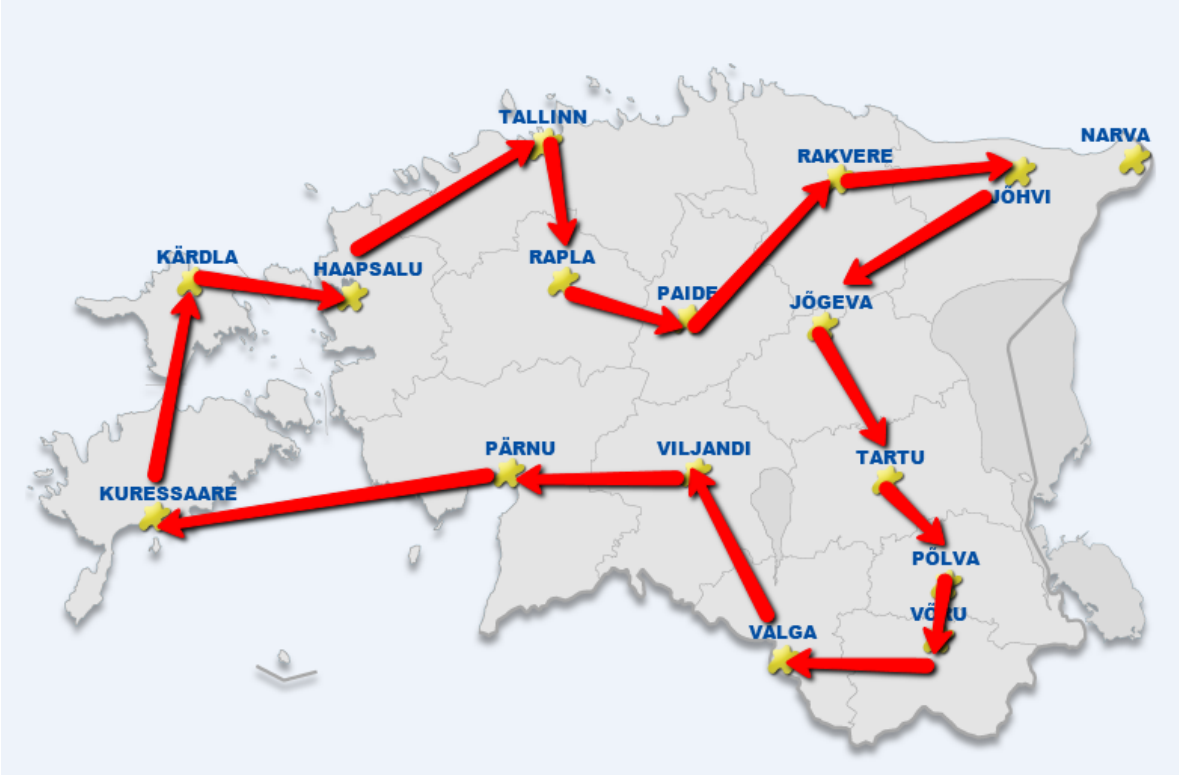
Lisa 2. Eesti linnadevaheline teekond pikkusega 948.79 km



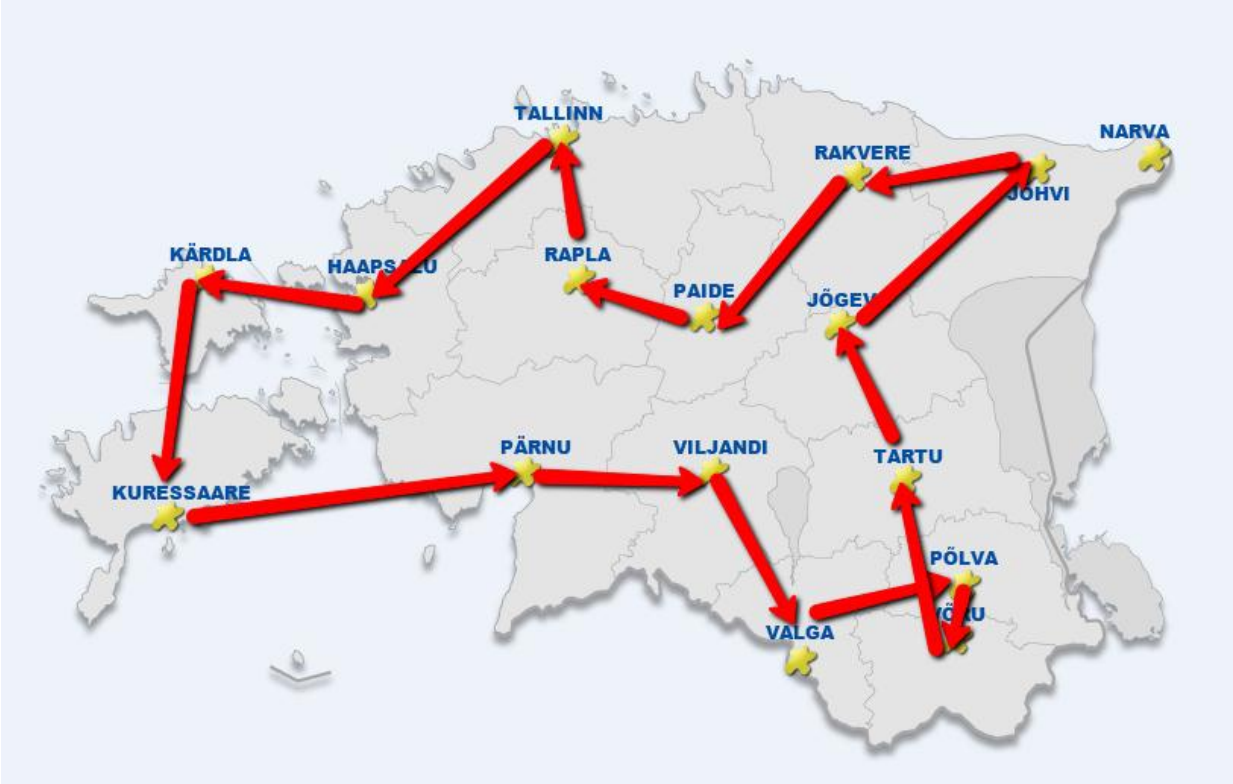
Lisa 3. Eesti linnadevaheline teekond pikkusega 997.41 km



Lisa 4. Eesti linnadevaheline teekond pikkusega 948.81 km



Lisa 5. Eesti linnadevaheline teekond pikkusega 981.29 km



Lisa 6. Eesti linnadevahelise lühima teekonna leidmiseks kasutatud programmikood

```
function [BestTourLength, BestTour] = ACSTSP(distances_matrix,...
number_of_ants, MaxIterations, alpha, beta, rho, target_length)

% Viide
% http://www.mathworks.se/matlabcentral/newsreader/view_thread/63740


%  Andmete sisselugemine:
%  Maatriksi "Eestilinnadevahemaad" sisselugemine:
%  a=xlsread('Eestilinnadevahemaad.xlsx');

%  Pöördumine: väljastab teekonna pikkuse, konkreetse teekonna linnade
%  läbimise järjekorra ja aja, mis kulus lahendi leidmiseks
%  tic;[BestTourLength, BestTour] = ACSTSP(a,10, 1000, 1, 2, 0.1,
35);toc;

% Ant Colony System for the TSP
%
% ACSTSP(distances_matrix, number_of_ants, MaxIterations, alpha, beta,
% rho, target_length)
%
% distances_matrix: symmeric real (NrOfNodes x NrOfNodes)-matrix
%  containing distances
% between all nodes: d[i,i] = 0, d[i,j]=d[j,i]
% MaxIterations: maximum number of iterations to run
% target_length: if a tour is found with less than
% target_length, the function returns
% stops and returns this tour.
%
% parameter standard values:
% number_of_ants = NrOfNodes
% alpha = 1
% beta = 9
% rho = 0.9
%
% returns [BestTourLength, BestTour]

clc;
d = distances_matrix; %define d
n = max(size(d));

m = number_of_ants;
t_max = MaxIterations;
L_target = target_length;

% [L_nn, P_nn] = NearestNeighborTSP(d);
%[L_nn,P_nn]=NearestNeighborTSP(d);
```

```
L_best = inf;
T_best = 0;

% Iteratsioonisammu algus
% ============================================================

% Feromoonijäljed
% c = 1 / (n * L_nn);
% c=1/(n*L_nn);
c=10^-6; % muudan kordaja
tau = ones(n,n) * c; % muudan maatriksi elemendid

% m sipelga paigutamine n linna
ant_tours = zeros(m, n+1);
ant_tours(:,1) = randi(m,1,[1,25]);

t = 1;
while ((t <= t_max) && (L_target < L_best))

% Teekondade konstrueerimine
% ============================================================

for s = 2 : n
for k = 1 : m
            current_node = ant_tours(k,s-1);
            visited = ant_tours(k,:);
            to_visit = setdiff(1:n,visited);
            c_tv = length(to_visit);
            p = zeros(1,c_tv);
            for i = 1 : c_tv
                p(i) = (tau(current_node,to_visit(i)))^alpha *...
(1/d(current_node,to_visit(i)))^beta;
            end
            sum_p = sum(p);
            p = p / sum_p;
            for i = 2 : c_tv
                p(i) = p(i) + p(i-1);
            end
            r = rand;
            select = to_visit(c_tv);
for i = 1 : c_tv
                if (r <= p(i))
                    select = to_visit(i);
                    break;
                end
end
            city_to_visit = select;
            ant_tours(k,s) = city_to_visit;
            tau(current_node,city_to_visit) = (1 - rho) *...
tau(current_node,city_to_visit) + c;
end
end
```

```
% Uuendamine
% ======================================================================

ant_tours(:,n+1) = ant_tours(:,1);
L_T = zeros(1,m);
best_ant = 1;
for k = 1 : m
        P = ant_tours(k,:);
        L = 0;
        for i = 1 : n
            L = L + d(P(i),P(i+1));
        end
        L_T(k) = L;
        if (L_T(k) < L_T(best_ant))
            best_ant = k;
        end
end
L_min = min(L_T);
T_min = ant_tours(best_ant,:);

% Feromoonijälgede uuendamine;
for i = 1 : n
        tau(T_min(i),T_min(i+1)) = (1 - rho) *...
tau(T_min(i),T_min(i+1)) + rho / L_min;
end

% Lõpetamine
% ======================================================================
clc;
t = t + 1;  % Lisatud semikoolon.
current_cities = ant_tours(:,n);
ant_tours = zeros(m, n+1);
ant_tours(:,1) = current_cities;
if (L_min < L_best)
        L_best = L_min;
        T_best = T_min;
end
L_best; %#ok<*VUNUS>

end % ends while

clc;
t;
BestTourLength = L_best;
BestTour = T_best;
```

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina **Teele Laas**

(sünnikuupäev: **12.03.1991**)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Rändkaupleja ülesande lahendamine sipelgaalgoritmiga

mille juhendaja on **dotsent Peep Miidla**,

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus/Tallinnas/Narvas/Pärnus/Viljandis, 13.06.2013