TARTU ÜLIKOOL
MATEMAATIKA-INFORMAATIKATEADUSKOND
Matemaatika instituut
Matemaatika eriala

Indrek Loolaid

Ülevaade metaheuristilistest meetoditest ja rändkaupmehe ülesande lahendamine GRASP meetodiga

Bakalaureusetöö (6 EAP)

Juhendaja: dotsent Peep Miidla

Tartu 2013

Sisukord

Sissejuhatus

Eksisteerib hulk optimiseerimisülesandeid, mille jaoks tõestatult optimaalse lahendi leidmine pole praktiliselt võimalik. Selliste ülesannete jaoks on võimalik kasutada metaheuristilisi meetodeid, mis otsivad iteratiivselt antud ülesande lahenditeruumist võimalikult head lahendit. Selleks kasutatakse lahendiruumi kohta eelnevatel iteratsioonidel saadud infot ja teatud määral juhuslikkust.

Käesolevas töös antakse ülevaade mitmetest metaheuristilistest algoritmidest. Kuna metaheuristilised meetodid on oma olemuselt väga üldised, siis ei ole antud kirjeldused absoluutne tõde. Enamike algoritmide kohta võib leida kirjandusest veidi erinevaid kirjeldusi ja algoritmidest on mitmeid edasiarendusi.

Vaatluse all olevad metaheuristikad on jagatud kaheks selle põhjal, mitut lahendikandidaati nad korraga vaatlevad. Kui korraga on vaatluse all ainult üks lahend, siis on tegu trajektoormeetoditega, kui üle ühe, siis populatsioonimeetoditega.

Töö esimeses peatükis antakse ülevaade metaheuristiliste meetodite olemusest ja ühistest omadustest. Teises peatükis vaatleme trajektoormeetodeid ja kolmandas peatükis populatsioonimeetodeid. Neljandas peatükis anname kiire ülevaate kriitikast mõningate "uuemate" meetodite suunal, mis algoritmi kirjeldamiseks võtavad kasutusele mõnel metafooril põhineva uue terminoloogia, kuid sisuliselt uusi ideid ei paku.

Viiendas peatükis on antud tulemused ühe metaheuristilise meetodit rakendamisest rändkaupmehe ülesande lahendamiseks. Rändkaupmehe ülesanne tuntud ja põhjalikult uuritud ülesanne, mida sageli kasutatakse metaheuristikate proovikivina. Valitud algoritmiks on GRASP ehk *Greedy Randomized Adaptive Search Procedure*. Kuna "ahne juhuslik kohanev otsinguprotseduur" ei ole kõige suupärasem algoritmi nimi, siis jätame kasutusele nimetuse GRASP. Lisades on välja toodud rändkaupmehe ülesande kirjeldamiseks vajalikud andmed ja lahendamiseks kasutatud programmi kood.

1 Metaheuristika mõistest

Definition 1. Ülesandeks nimetame käesoleva töö kontekstis mistahes (optimiseerimis)ülesannet, mille täpse lahendi leidmiseks pole teada ühtegi algoritmi või on need praktikas rakendamiseks liiga suure ajalise keerukusega. Näited selliste ülesannete kohta on

- lühima Hamiltoni tsükli leidmine täisgraafis
- tunniplaani koostamine
- roboti jalgpalli mängima õpetamine
- jne

Definition 2. Lahend või lahendikandidaat S on miski, mida me saame mõnele konkreetsele ülesandele vastuseks pakkuda. Lahendi esitusviisiks võib olla reaalarvude vektor, graaf, programm, funktsioon vms kuju, mis on vastava ülesande jaoks sobilik, kusjuures iga kahe lahendi korral peab olema võimalik hinnata, kumb on parema kvaliteediga (optimaalsem) antud ülesande jaoks.

Definition 3. Ülesande lahendiruumiks nimetame hulka, mis sisaldab kõiki võimalikke lahendikandidaate antud ülesande jaoks.

Definition 4. Olgu X mingi ülesande lahendiruum ja $S \in X$. Funktsiooni $Q : X \to \mathbb{R}$ nimetame antud ülesande sobivusfunktsiooniks. $Q(S)$ annab lahendile vastava hinnangu selle kvaliteedi kohta. Edaspidi, kui pole vastupidist väidetud, siis eeldame, et mida suurem on $Q(S)$ väärtus, seda parem on lahendi S kvaliteet.

Metaheuristika on termin, millele erinevad allikad annavad veidi erineva definitsiooni, aga need võib kokku võtta järgmiselt: metaheuristilised meetodid (metaheuristikad) on üldised (konkreetsest õlesandest sõltumatud) optimiseerimisalgoritmid, mis kasutavad teataval määral juhuslikkust, et iteratiivselt leida võimalikult hea lahend, kasutades selleks sageli eelneval sammul leitud lahendit.[1, 2, 3]

Enamasti alustavad metaheuristikat tööd mõnest juhuslikult valitud lahendist (või lahenditest), seega peab olema antud ülesande jaoks võimalik juhuslikku lahendit välja pakkuda. Näiteks, kui ülesandeks on lühima Hamiltoni tsükli leidmine täisgraafis, siis juhuslikuks lahendiks sobib iga Hamiltoni tsükkel selles graafis.

Kui esialgne lahend on olemas, hakatakse ülesande lahenditeruumis iteratiivselt teostama otsingut, et leida võimalikult hea lahend. Igal iteratsioonil leitakse uus juhuslik lahendikandidaat, kusjuures uue lahendi leidmisel võetakse aluseks eelnev(ad) lahend(id). Uue lahendi leidmiseks kasutatakse mõnda ülesandespetsiifilist võtet või heuristikat. Metaheuristika ülesanne on otsustada, kas võtta vastu uus lahend ja jätkata otsingut järgmisest asukohast lahenditeruumis või jääda vana lahendi juurde ja selle naabrusest edasi otsida.

Itereeritakse kuni jõutakse mõne eelnevalt defineeritud peatumiskriteeriumini (näiteks leitud lahend on piisavalt hea, maksimaalne iteratsioonide arv sai tehtud, lubatud aeg sai otsa vms). Töö lõppedes väljastab algoritmi kõikidest vaadeldud lahenditest parima.

Otsingu koondamine ja hajutamine. Laias laastus on kõik metaheuristikad kombinatsioon kahest vastandlikust protsessist: otsingu koondamine ja hajutamine (ingl *intensification/diversification* [3] või *exploitation/exploration* [1]). Hajutamine on protsess, mille eesmärgiks on suunata otsingut lahendiruumi võimalikult erinevatesse piirkondadesse ning peab eksisteerima võimalus liikuda iga lahendini lahendiruumis. Koondamise eesmärk on keskenduda otsingule mingis konkreetses lahendiruumi piirkonnas, et leida sealt parim lahend (lokaalne optimum).

Üldjuhul on metaheuristikatel seadistatavad parameetrid, mis määravad kuivõrd hajutatud või koondatud on parasjagu otsing. Metaheuristikate efektiivsus on suuresti sõltuv nende parameetrite valikust. Enamasti saab ainult katsetuste tulemusel öelda, millised parameetrid annavad kõige parema tulemuse, kuid on ka algoritme, kus parameetrite väärtusi kohandatakse jooksvalt algoritmi töö käigus. Kui esituselt sarnased lahendid on sarnase kvaliteediga, siis viib koondatud otsing kiiremini parema tulemuseni. Vastasel juhul on vajalik hajutatum otsing.

Metaheuristikate vajalikkusest. Metaheuristikate üks oluline omadus on nende üldisus, mis võimaldab sama algoritmi rakendada erinevate ülesannete lahendamiseks. Loomulikult on igal ülesandel oma eripärad, võtted ja lahendi esitusviisid, mida on võimalik (vaja) kasutada, aga kuna metaheuristikate poolt pakutavaid lahendusskeeme saab kasutada väga erinevatel juhtudel, ei ole vaja iga ülesande lahendamiseks täiesti uut algoritmi luua.

Teine metaheuristikate omadus on see, et lahendatava ülesande sobivusfunktsioon ei pea olema diferentseeritav, ei eeldata ka midagi muud, kui see, mis on kirjas definitsioonis 4. Seega on metaheuristikad rakendatavad ka siis, kui traditsioonilisemaid analüüsimeetodeid (näiteks sobivusfunktsiooni tuletise võtmine optimumi leidmiseks) ei ole võimalik kasutada.

2 Trajektoormeetodid

Viise metaheuristikate liigitamiseks võib leida mitmeid, kuid käesolevas töös liigitame vaadeldavad meetodid kaheks selle põhjal, mitme lahendiga vastav meetod igal iteratsioonil korraga tegeleb. Trajektoormeetoditeks (ingl *trajectory methods* [3], *single-state methods* [1]) nimetame metaheuristikaid, mis käsitlevad korraga ühte lahendit. Vastasel juhul on tegemist populatsioonimeetoditega.

Definition 5. Olgu X mingi ülesande lahendiruum. Siis $Muuda : X \to X$, on funktsioon, mis teeb sisendlahendile väikese juhusliku muudatuse ja tagastab selle. $Muuda$ rakendamisel saadud lahend on kujult (ja ideaalis ka kvaliteedilt) sarnane sisendile.

Definition 6. Olgu X mingi ülesande lahendiruum ja $S \in X$. Hulka $N(S)$ nimetame lahendi S naabruseks, kui $N(S)$ sisaldab neid ja ainult neid lahendeid, mida on võimalik saada funktsiooni $Muuda$ rakendamisel elemendile S.

2.1 Mägironimine

Mägironimine (ingl *hill climbing*) on lihtne optimiseerimisalgoritm lokaalse optimumi leidmiseks. Algoritm töötab järgmiselt: alustatakse suvalisest lahendist, lahendist tehakse koopia, millele tehakse väike juhuslik muudatus. Kui sel viisil saadud lahend on parem esialgsest, siis visatakse eelnev lahend kõrvale ja asendatakse uuega. Lahendite muutmist/asendamist jätkatakse, kuni enam paremat lahendit ei leita s.t on jõutud kohaliku optimumini.

Mägironimist võib teha ka nii, et ühe juhusliku muudatuse tegemise asemel vaadeldakse mitut lahendit, mis jäävad olemasoleva lahendi naabrusesse ja valitakse nende hulgast välja parima kvaliteediga lahend.

Mitmed metaherustikad põhinevad mägironimisel ja lisavad sellele mõne mehhanismi, et vältida otsingu peatumist mõnes kohalikus optimumis või sadulpunktis.

Algoritm 1 Mägironimine

$S \leftarrow$ esialgne lahend
repeat
  $R \leftarrow Muuda(Koopia(S))$
  if $Q(R) > Q(S)$ **then**
    $S \leftarrow R$
until jõuti peatumiskriteeriumini
return S

2.2 Simuleeritud lõõmutamine

Lõõmutamine on metalli töötlemise protsess, kus metalli kuumutatakse ja lastakse aeglaselt jahtuda. Seeläbi ühtlustub osakeste struktuur metallis ja sisepinged vähenevad.[1] Simuleeritud lõõmutamine on sellest protsessist inspiratsiooni saanud algoritm. Algoritmi töö on sarnane mägironimisele, kuid selle erinevusega, et kui kandidaatlahend juhtub olema kehvem jooksvast lahendist, siis võetakse see teatud tõenäosusega siiski omaks ja heidetakse eelnev lahend kõrvale.

Tõenäosus kehvema lahendi vastu võtmiseks sõltub algoritmile ette antud parameetrist $t > 0$ (tähistab temperatuuri) ja uue ning vana lahendi kvaliteetide vahest. See tõenäosus esitub valemiga $e^{\frac{Q(\text{uus lahend})-Q(\text{vana lahend})}{t}}$. Paneme tähele, et selle avaldise väärtus arvutatakse ainult juhul, kui uue lahendi kvaliteet on kehvem jooksva lahendi kvaliteedist, seega on astendaja negatiivne. Väärtus t on esialgu suur ja seda vähendatakse iga iteratsiooniga. Sellest tulenevalt on tõenäosus võtta vastu kehvema kvaliteediga lahend alguses suur ja see tõenäosus väheneb järk-järgult. Kui t saab juba nullilähedaseks, siis teeb algoritm sisuliselt mägironimist.[1]

[1] http://opiobjektid.tptlive.ee/Materjaliopetus/termottlemine.html

Algoritm 2 Simuleeritud lõõmutamine

$t \leftarrow$ temperatuur, esialgu suure väärtusega
$S \leftarrow$ esialgne lahend
$Parim \leftarrow S$
repeat
  $R \leftarrow Muuda(Koopia(S))$
  if $Q(R) > Q(S)$ **or** juhuslik arv $r \in [0; 1] < e^{\frac{Q(R)-Q(S)}{t}}$ **then**
    $S \leftarrow R$
  if $Q(S) > Q(Parim)$ **then**
    $Parim \leftarrow S$
until jõuti peatumiskriteeriumini
return $Parim$

2.3 Tabu otsing

Tabu otsing (ingl *tabu search*) kasutab lokaalsest optimumist välja saamiseks mälu, et vältida lahendeid, mida lähiminevikus on külastatud. Mälu kasutamine seisneb selles, et iga vastuvõetud lahend pannakse tabu nimekirja. Tabu nimekiri on järjekord (*First In First Out*), mille maksimaalne pikkus on algoritmile etteantav parameeter. Kui uue lahendi lisamisel tabu nimekirja saab nimekiri pikemaks lubatust, siis eemaldatakse sealt vanim lahend. Uue lahendi valimiseks valitakse käesoleva lahendi naabrusest teatud arv kandidaatlahendeid ning vastu võetakse neist parim, mis ei ole parasjagu tabu.[4, 1] Algoritm 3 kirjeldab tabu otsingut tema lihtsamail kujul.

Tabu nimekirjas lihtsalt lahendite hoidmine sobib ainult ülesannete jaoks, mille lahendiruum on diskreetne, vastasel juhul on väga ebatõenäoline, et täpselt sama lahendi otsa rohkem kui üks kord satutakse. Üks variant on pidada tabuks lahendeid, mis on nimekirjas olevatele mingi kriteeriumi järgi piisavalt sarnased.

Isegi diskreetse ruumi korral ei pruugi lahendite hoidmine olla parim variant. Seda näiteks juhul, kui lahendiruumi mõõtmelisus on väga suur ja tabu nimekirja ei ole võimalik mahutada piisavalt palju lahendeid, et sundida otsingut lahkuma kohaliku optimumi ümbrusest. Sel juhul oleks alternatiivideks näiteks hoida tabu nimekirjas lahendite omadusi/atribuute või muudatusi teatud suunas. Lahendite omaduste tabuks pidamine võib muuta tabude lahendite hulga väga suureks,

mistõttu oleks kasulik algoritmi lisada tingimus (näiteks, kui lahendi kvaliteet ületab teatud lävendi), mis lubaks lahendi siiski vastu võtta isegi, kui oma omaduste poolest peaks lahend olema tabu.

Algoritm 3 Tabu otsing

$l \leftarrow$ tabu nimekirja maksimaalne pikkus
$n \leftarrow$ lahendi ümbruses vaadeldava valimi suurus
$S \leftarrow$ esialgne lahend
$Parim \leftarrow S$
$Tabu \leftarrow \varnothing$
Lisa S $Tabu$ nimekirja
repeat
  if $|Tabu| > l$ **then**
    Eemalda vanim lahend $Tabu$ nimekirjast
  $N'(S) \leftarrow n$ lahendit saadud operatsiooniga $Muuda(Koopia(S))$
  $S \leftarrow R_i : R_i \in N'(S),\ Q(R_i) \geqslant Q(R_j)\ \forall i, j \in (1 \ldots n),\ R_i, R_j \notin Tabu$
  Lisa S $Tabu$ nimekirja
  if $Q(S) > Q(Parim)$ **then**
    $Parim \leftarrow S$
until jõuti peatumiskriteeriumini
return $Parim$

2.4 Itereeritud kohalik otsing

Üks viis vältida mägironimsalgoritmi kinni jäämist kohalikku optimumi on iga teatud aja möödudes otsingut juhuslikust kohast uuesti alustada. Itereeritud otsing teeb midagi sarnast, aga kohta, kust uut otsingut alustada ei valita täiesti juhuslikult.

Algoritmi eesmärk on teha otsingut lahendiruumi lokaalsete optimumide ruumis. Lisaks vaadeldavale lahendile S kasutab algoritm ka ühte lahendit S^* lokaalsete optimumide ruumist, mille ümbrusest proovitakse leida teisi lokaalseid optimume. Selleks rakendatakse lahendile S lokaalset otsingut, et leida mõni kohalik optimum. Seejärel otsustatakse, kas asendada vana lokaalne optimum S^* uuega või mitte. Algoritmis 4 on protseduur $UusOtsinguLähe(S^*, S)$ see, mis otsustab, kas võtta leitud lahend uueks lähteks või mitte. Kui otsingu lähteks valida alati parema kvaliteediga lahend, siis on algoritm samaväärne kaheastmelise lokaalse otsinguga

(üks lokaalne otsing on teise alamprotseduur). Sellisel juhul pole algoritm võimeline otsingut iga lahendiruumi elemendini viima. Et iga lahendini oleks võimalik jõuda võib uue lähte valikul kasutada näiteks simuleeritud lõõmutamist või tabu otsingut.

Protseduur $MuudaTugevalt(S^*)$ on iseloomult sarnane $Muuda(S)$ protseduurile, kuid lahendile S^* tehtud muudatus on suurusjärgu võrra suurem. Tugeva häirituse (*perturbation*) eesmärk on leida lahend, mis oleks olemasolevale lähtepunktile suhteliselt lähedal, kuid mille puhul mägironimise rakendamine annab tulemuseks antud lähtepunktist erineva lokaalse optimumi.[5, 1, 3]

Algoritmis 4 (ja ka mujal) tähendab avaldis $LokaalneOtsing(S)$, et lahendile S rakendatakse mõnda ülesandespetsiifilist lokaalse otsingu algoritmi või üldisemat mägironimisalgoritmi.

Algoritm 4 Itereeritud kohalik otsing

$S \leftarrow$ esialgne lahend
$S^* \leftarrow S$
$Parim \leftarrow S$
repeat
  $S \leftarrow LokaalneOtsing(S)$
  if $Q(S) > Q(Parim)$ **then**
    $Parim \leftarrow S$
  $S^* \leftarrow UusOtsinguLähe(S^*, S)$
  $S \leftarrow MuudaTugevalt(S^*)$
until jõuti peatumiskriteeriumini
return $Parim$

2.5 VNS

Seni vaadeldud algoritmid otsivad uut lahendit olemasoleva ühest kindlast naabrusest. VNS (*Variable Neighborhood Search*) seevastu kasutab mitut erinevat naabrust N_k, $k = 1, \ldots, k_{max}$. Iga naabruse jaoks on funktioon $Muuda_k(S)$, mis tagastab S naabrusest numbriga k uue lahendi. Otsingut alustatakse jällegi mingist juhuslikust lahendist. Otsingu teostamisel toimitakse iga naabruse korral nii, et valitakse sellest naabrusest juhuslikult uus lahend ja rakendatakse lahendile lokaalset otsingut. Kui sel viisi saadud lahend on parem olemasolevast, siis võetakse

see vastu ja alustatakse otsingut uuesti esimesest naabrusest. Vastasel juhul minnakse edasi järgmise naabruse juurde. Ning kui kõik naabrused on läbi käidud, siis minnakse tagasi esimese juurde, kui algoritmi peatumiskriteerium ei ole täidetud.

Kohalikku optimumi kinnijäämist aitab vältida tõsiasi, et kohalik optimum ühes naabruses ei pruugi seda olla teises. Seevastu globaalne optimum on lokaalne optimum igas naabruses. Lisaks on paljude ülesannete korral ühe naabruse kohalik optimum sarnane teise naabruse kohalikule optimumile. See tähendab, et kohalik optimum vihjab sageli milline on globaalne optimum.[6]

Algoritm 5 VNS

$S \leftarrow$ esialgne lahend
$N_k \leftarrow$ naabrused, kust otsingut teostatakse $(k = 1 \dots k_{max})$
repeat
  $k \leftarrow 1$
  repeat
    $R \leftarrow Muuda_k(Koopia(S))$ $(R \in N_k(S))$
    $R \leftarrow LokaalneOtsing(R)$
    if $Q(R) > Q(S)$ **then**
      $S \leftarrow R$
      $k \leftarrow 1$
    else
      $k \leftarrow k + 1$
  until $k = k_{max}$
until jõuti peatumiskriteeriumini
return S

2.6 GRASP

Punktis 1 sai mainitud, et metaheuristikad parandavad iteratiivselt mingit olemasolevat lahendit või olemasolevaid lahendeid. GRASP e. *Greedy Randomized Adaptive Search Procedure* on selles suhtes veidi erandlik, sest igal iteratsioonil konstrueeritakse uus lahend, mis ei sõltu eelnevast.

GRASP sobib eelkõige kombinatoorika ülesannete lahendamiseks, kus lahendiks on alamhulk mingist lõplikust etteantud komponentide hulgast C. Algoritm koosneb kahest etapist: lahendi konstrueerimine ja kohalik otsing. Igal iteratsioonil konstrueeritakse poolahnelt uus lahend. Alustatakse lahendist, mis ei sisalda ühtegi

komponenti. Iga komponendi jaoks arvutatakse, kui palju antud komponendi lisamisel lahendi kvaliteet muutub. Seejärel koostatakse nimekiri parimatest komponentidest, mille lisamine lahendile kvaliteeti kõige rohkem tõstab või kõige vähem vähendab (olenevalt konkreetsest ülesandest) ja valitakse sellest nimekirjast juhuslikult üks komponent, mis lisatakse lahendile. Kui lisataks alati parim komponent, siis oleks algoritm täielikult ahne, mitte poolahne. Seda jada alates komponentide mõjus arvutamisest kvaliteedile kuni komponendi lahendile lisamiseni korratakse kuni konstrueeritud lahend on täielik s.t lahend kuulub antud ülesande lahendiruumi. Näiteks Hamiltoni tsükli konstrueerimisel alustame tühjast hulgast ja lisame servi, kuni lahendisse lisatud servade hulk moodustab Hamiltoni tsükli.

Iga järgneva komponendi valimine mingi hulga parimate komponentide seast muudab algoritmi ahneks (**G**RASP). Juhuslikkuse annab algoritmile lahendi konstrueerimisel järgneva komponendi juhuslik valik parimate hulgast (G**R**ASP). Ning algoritm on kohanev, kuna enne iga järgneva komponendi lahendisse valimist arvutatakse kõigi komponentide mõju lahendi üldisele kvaliteedile uuesti, mis tähendab, et juba lisatud komponentide mõju järgnevale on arvesse võetud (GR**A**SP).

Kuna miski ei garanteeri, et konstrueeritud lahend oleks lokaalne optimum, siis rakendatakse saadud lahendile lokaalset otsingut. Lõpuks tagastatakse parim leitud lahend.[7]

Algoritm 6 GRASP

repeat
  $Parim \leftarrow \varnothing$
  $S \leftarrow KonstrueeriAhneltLahend()$ (kirjeldatud täpsemalt algoritmis 7)
  $S \leftarrow LokaalneOtsing(S)$
  if $Parim = \varnothing$ **or** $Q(S) > Q(Parim)$ **then**
    $Parim \leftarrow S$
until jõuti peatumiskriteeriumini
return $Parim$

Algoritm 7 KonstrueeriAhneltLahend

$S \leftarrow \varnothing$
$r \leftarrow$ lahendi konstrueerimisel valitavate komponentide arv
$C \leftarrow$ kõik võimalikud komponendid
repeat
  $C' \leftarrow$ lubatud komponendid hulgast C
  Arvuta iga lubatava komponendi jaoks tema mõju lahendi kvaliteedile
  $C^* \leftarrow r$ parimat komponenti hulgast C'
  $c \leftarrow$ juhslik komponent hulgast C^*
  $S \leftarrow S \cup \{c\}$
until S on täielik lahend
return S

3 Populatsioonimeetodid

Populatsioonimeetodid on sageli inspiratsiooni saanud loodusest või looduslikest protsessidest ja algoritmi kirjeldamiseks on kasutusele võetud ka vastav sõnavara. Sõltuvalt konreetsest metaheuristilisest meetodist tähendab populatsioon hulka lahendikandidaate või agente, mille ülesanne on lahendeid konstrueerida või manipuleerida.

3.1 Sipelgameetod

Sarnaselt GRASP algoritmile on ka sipelgameetod (*Ant Colony Optimization*) mõeldud kombinatoorika ülesannete jaoks, kus on ette antud komponentide hulk C ja lahend S kuulub lubatavate lahendite hulka $X \subseteq 2^C$. Sipelgameetodis moodustavad populatisoooni tehislikud sipelgad - agendid, mis algoritmi igal iteratsioonil konstrueerivad uue lahendi (üks lahend iga sipelga kohta). Iga komponendiga $c_i \in C$ on seotud väärtus p_i, mida nimetame feromooniks. Feromoon on väärtus, mis sisaldab infot nende lahendite kvaliteedi kohta, milles feromoonile vastavat komponenti on kasutatud.

Iga sipelgas alustab lahendi koostamist tühjast hulgast ja hakkab sellele ükshaaval komponente lisama, kuni lahend on täielik. Iga komponendi valituks osutumise tõenäosus on sõltuv selle kvaliteedist ja vastava feromooni väärtusest. Komponentidel,

mis kuuluvad sagedamini kõrge kvaliteediga lahendite hulka, on suurem tõenäosus saada lahendi ehitamisel valituks.[1, 8]

Pärast lahendite konstrueerimist vähendatakse kõigi feromoonide väärtust (feromoonide "aurustumine") ja seejärel uuendab iga sipelgas enda konstrueeritud lahendi komponentidele vastavaid feromoonide väärtusi proportsioonis antud lahendi kvaliteediga (sipelgas jätab enda käidud rajale feromoonide jälje). Feromoonide aurustumine on vajalik selleks, et feromoonide väärtused piiramatult ei kasvaks.

Sipelgameetodi nimetuse taga peitub tegelikul mitu veidi erinevat, aga sama üldist ideed kasutavat algoritmi (*Ant System, Ant Colony System, ANTS*), mis muudavad feromoonide väärtusi erinevate strateegiate järgi. Vaatleme ühte võimalikku viisi feromoonide kasutamiseks ja nende väärtuste uuendamiseks. Olgu q_i komponendi c_i väärtus vastavalt sobivusfunktsioonile Q ja C' nende komponentide hulk, mille lisamine antud poolikusse lahendisse on lubatud. Siis tõenäosus komponendi c_i valimiseks lahendi hulka esitub valemiga (1) ja feromoonide uuendamiseks kasutatakse valemit (2).

$$P(c_i) = \begin{cases} \frac{p_i^\alpha + q_i^\beta}{\sum_{c_j \in C'} p_j^\alpha + q_j^\beta} & \text{, kui } c_i \in C' \\ 0 & \text{vastasel juhul} \end{cases} \tag{1}$$

$$p_i(t+1) = \rho * p_i(t) + \Delta p_i \tag{2}$$

$$\Delta p_i = \begin{cases} \sum_{k=1}^{\mu} \epsilon * Q(S_k) & \text{, kui } c_i \in S_k \\ 0 & \text{vastasel juhul} \end{cases} \tag{3}$$

Valemis (1) on α ja β ($0 \leqslant \alpha, \beta \leqslant 1$) valitavad parameetrid, mis määravad kui suur mõju komponendi valikul on antud komponendi kvaliteedil ja vastaval feromoonil. Valemis (2) on ρ ($0 \leqslant \rho \leqslant 1$) valitav parameeter, mis määrab, kui kiiresti feromoonid "aurustuvad", t on vastava iteratsiooni number ja valemis (3) on λ sipelgate arv, S_k on siplega k poolt konstrueeritud lahend ja ϵ on valitav parameeter, mis määrab, kui suurt mõju avaldab lahendi kvaliteet feromoonide uuendamisel.

Algoritm 8 Sipelgameetod

$C \leftarrow$ komponendid $\{c_1, \ldots, c_n\}$
$\lambda \leftarrow$ populatsiooni suurus (sipelgate arv)
$p \leftarrow \{p_1, \ldots, p_n\}$ (igale komponendile vastav feromoon, esialgu kõigil sama väärtus)
$Parim \leftarrow \varnothing$
repeat
  $P \leftarrow \varnothing$
  for $i \leftarrow 1$ **to** λ **do**
    $P_i \leftarrow KonstrueeriLahend()$ (arvestades iga komponendi valikul valemiga (1) antud tõenäosust)
    $P \leftarrow P \cup P_i$
  for $P_i \in P$ **do**
    if $Parim = \varnothing$ **or** $Q(P_i) > Q(Parim)$ **then**
      $Parim \leftarrow P_i$
  $p \leftarrow UuendaFeromoone()$ (vastavalt valemile (2))
until jõuti peatumiskriteeriumini
return Osakese $Parim$ asukoht

3.2 Evolutsioonistrateegiad

Metaheuristilistes meetodites, mis on saanud inspiratsiooni bioloogilisest evolutsioonist (evolutsioonistrateegiad, geneetiline algoritm), tähendab termin populatsioon lahendikandidaatide hulka. Indiviidiks nimetatakse ühte konkreetset lahendikandidaati populatsioonis ja algoritmi ühte iteratsiooni nimetatakse populatsiooni põlvkonnaks. Funktsiooni $Muuda$ rakendamist indiviidile nimetatakse sageli indiviidi muteerumiseks ning indiviidi kvaliteeti nimetatakse tugevuseks (tugevamate ellujäämine).

On kahte tüüpi evolutsioonistrateegiaid: (μ, λ) ja $(\mu + \lambda)$, kus väärtused μ ja λ on algoritmile etteantavad parameetrid. Mõlemad algoritmid alustavad mingist esialgsest lahendite hulgast - populatsioonist. Populatsioonis on λ indiviidi e. lahendit, mille hulgast leitakse igal iteratsioonil μ parimat. Igast indiviidist (vanemast) μ parima hulgast luuakse $Muuda(S)$ operatsiooni abil $\frac{\lambda}{\mu}$ uut indiviidi (last), ning nõutud on, et parameeter λ jaguks parameetriga μ.

Erinevus strateegiates on selles, mida tehakse vanematega. (μ, λ) strateegia puhul

heidetakse vanemad kõrvale ja λ last moodustavad uue põlvkonna. $(\mu + \lambda)$ strateegia korral jäävad vanemad järgmiseks põlvkonnaks ellu ja jäävad laste kõrvale konkureerima.[1]

Algoritm 9 (μ, λ) evolutsioonistrateegia

$P \leftarrow$ esialgne populatsioon (λ juhuslikku lahendit)
$Parim \leftarrow$ juhuslik lahend populatsioonist P
repeat
  for $P_i \in P$ **do**
    if $Q(P_i) > Q(Parim)$ **then**
      $Parim \leftarrow P_i$
  $P^* \leftarrow \mu$ parimat lahendit populatsioonist
  $P \leftarrow \varnothing$ ($(\mu + \lambda)$ strateegia korral jääksid μ vanemat hulka P)
  for $P_i^* \in P^*$ **do**
    for $j \leftarrow 1$ **to** λ/μ **do**
      $P \leftarrow P \cup \{Muuda(Koopia(P_i^*))\}$
until jõuti peatumiskriteeriumini
return $Parim$

3.3 Geneetiline algoritm

Geneetilise algoritmi ülesehitus on võrdlemisi sarnane evolutsioonistrateegiatele. Oluline erinevus seisneb selles, kuidas sünnib uus põlvkond. Kui evolutsioonistrateegiate puhul on igal lapsel üks vanem, siis geneetilise algoritmi korral on igal lapsel kaks vanemat. Iga uue indiviidi saamiseks valitakse juhuslikult kaks vanemat. Tõenäosus valituks osutuda on proportsioonis indiviidi kvaliteediga. See tähendab, et parematel kandidaatlahenditel on suurem tõenäosus saada valituks.

Kui kaks vanemat on valitud, siis nendest saadakse ristamise teel kaks järglast. Olgu meil kaks lahendikandidaati S ja T, mis koosnevad vastavalt komponentidest $s_1, \ldots, s_n$ ja $t_1, \ldots, t_n$, kus n on komponentide arv lubatud lahendis. Ristamine on protseduur, mille käigus toimub kahe lahendi komponentide vahetus teatud viisil. Näiteks võib ristamise käigus lahendites S ja T omavahel ära vahetada vastavalt komponendid $s_k, \ldots, s_l$ komponentidega $t_k, \ldots, t_l$, $1 \leqslant k < l \leqslant n$, tõenäosusega $1/n$ komponendid s_i ja t_i, $i = 1, \ldots, n$ omavahel ära vahetada vms.

Ristamise teel saadud lahendeid ka muteeritakse populatsiooni mitmekesistamiseks, sest ainult ristamise teel ei pruugi olla võimalik jõuda iga lahendiruumi elemendini. Sel viisil luuakse kahekaupa uusi indiviide, kuni lubatud populatsiooni suurus on saavutatud, asendatakse eelnev põlvkond uuega. Loomulikult hinnatakse ka iga iteratsiooni alguses kõikide isendite kvaliteeti, et tuvastada ja hoida meeles kõigist vaadeldud lahenditest parim.

Tüüpiline (kuid mitte ainus) viis, kuidas geneetilise algoritmi jaoks lahendit esitada, on tõeväärtuste massiivina. Kui näiteks lahendi S iga komponent $s_i \in \{0, 1\}$, $i = 1, \ldots, n$, siis saab lahendile S omistada 2^n erinevat väärtust. Tuleb arvestada sellega, et lahendiruumi võib kuuluda vähem elemente kui 2^n ja ristamise tulemusel tekkivad lahendid ei pruugi olla lubatud antud ülesande jaoks. Kui sobimatute lahendite arv kõigist võimalikest on suhteliselt väike, siis on võimalik neid algoritmi töö käigus neid lihtsalt eirata, kui nad esile kerkivad. Vastasel juhul ei ole geneetilise algoritmi kasutamine otstarbekas, kuna tühja töö peale kulub liiga palju aega.[1, 9]

Algoritm 10 Geneetiline algoritm

$\lambda \leftarrow$ populatsiooni suurus
$P \leftarrow$ esialgne populatsioon (λ juhuslikku lahendit)
$Parim \leftarrow$ juhuslik lahend populatsioonist P
repeat
  for $P_i \in P$ **do**
    if $Q(P_i) > Q(Parim)$ **then**
      $Parim \leftarrow P_i$
  $P' \leftarrow \varnothing$
  for $i \leftarrow 1$ **to** $\lambda/2$ **do**
    $P_a \leftarrow$ juhuslik lahend hulgast P
    $P_b \leftarrow$ juhuslik lahend hulgast P
    $C_a, C_b \leftarrow Rista(Koopia(P_a), Koopia(P_b))$
    $P' \leftarrow P' \cup \{Muuda(C_a), Muuda(C_b)\}$
  $P \leftarrow P'$
until jõuti peatumiskriteeriumini
return $Parim$

3.4 Differentsiaalevolutsioon

Sarnaselt teistele evloutsioonilistele meetoditele moodustub ka differentsiaalevolutsiooni korral populatsioon lahendikandidaatidest ning kasutades indiviide olemasolevas populatsioonis luuakse igal iteratsioonil järgmine põlvkond. Kuna järglaste loomisel kasutatakse tehteid vektoritega (vektorite liitmine ja korrutamine skalaariga), siis on vajalik, et lahend esituks vektorina hulgas $X \subseteq \mathbb{R}^n$.

Järglaste loomine uue põlvkonna jaoks toimub kasutades kahe vanema ristamist nagu seda tehti geneetilise algoritmi korral. Üheks vanemaks on indiviid populatsioonist nii, et igat indiviidi kasutatakse otsese vanemana üks kord. Teine vanem konstrueeritakse kasutades kolme erinevat indiviidi. Populatsioonis peab olema järelikult vähemalt neli indiviidi, et saaks luua järglase - üks otsene vanem ja kolm indiviidi, kellest konstrueeritakse teine vanem. Konstrueeritava vanema loomiseks kasutatavad lahendid valitakse täiesti juhuslikult, sõltumata antud lahendite kvaliteedist.

Olgu meil valitud kolm lahendit $\vec{S}_a, \vec{S}_b, \vec{S}_c$, siis nendest konstrueeritakse teine vanem $\vec{S}_d$ järgmiselt: $\vec{S}_d = \vec{S}_a + \alpha * (\vec{S}_b - \vec{S}_c)$, kus α (mutatsiooni määr) on konstant vahemikus $[0; 2]$. Kuigi ristamise tulemusel tekib kaks järglast, on neist vaja ainult ühte ja teine visatakse kõrvale. Kui uus indiviid on kehvema kvaliteediga, kui tema otsene vanem populatsioonis, siis visatakse see lahend kõrvale ja vanem jääb populatsiooni alles. Vastasel juhul asendatakse vanem lapsega.

Algoritm 11 Differentsiaalevolutsioon

$\alpha \leftarrow$ mutatsiooni määr
$\lambda \leftarrow$ populatsiooni suurus
$P \leftarrow$ esialgne populatsioon (λ juhuslikku lahendit)
$P' \leftarrow \varnothing$ (Vanemad. Igale vanemale P'_i vastab laps P_i)
$Parim \leftarrow$ juhuslik lahend populatsioonist P
repeat
  for $P_i \in P$ **do**
    if $P' \neq \varnothing$ **and** $Q(Q_i) > Q(P_i)$ **then**
      $P_i \leftarrow P'_i$
    if $Q(P_i) > Q(Parim)$ **then**
      $Parim \leftarrow P_i$
  $P' \leftarrow P$
  for $P'_i \in P'$ **do**
    $\vec{a} \leftarrow$ juhuslik lahend hulgast P'
    $\vec{b} \leftarrow$ juhuslik lahend hulgast P'
    $\vec{c} \leftarrow$ juhuslik lahend hulgast P'
    ($\vec{a}$, $\vec{b}$, $\vec{c}$, P'_i on paarikaupa erinevad)
    $\vec{d} \leftarrow \vec{a} + \alpha(\vec{b} - \vec{c})$
    $P_i \leftarrow$ üks laps $Rista(\vec{d}, Koopia(P'_i))$ tulemusest
until jõuti peatumiskriteeriumini
return $Parim$

3.5 Osakestepilve optimiseerimine

Osakestepilve optimiseerimisel koosneb populatsioon osakestest, mis imiteerivad loomade parvekäitumist. Iga osake koosneb kolmest komponendist: osakese asukoht, kiirus ja antud osakese poolt parim leitud asukoht. Asukoht tähendab koordinaate lahendiruumis $X \subseteq \mathbb{R}^n$. Igal iteratsioonil muutub vastava osakese kiirus ja osakese asukoht vastavalt kiirusele. Kiirus on lihtsalt vektor $\vec{v} \in \mathbb{R}^n$, mis liidetakse osakese asukohavektorile, et liikuda uude asukohta.

Kiiruse muutmisel võetakse arvesse antud osakese poolt leitud parimat asukohta, osakese informaatorite poolt parimat leitud asukohta ja kõigi osakeste poolt leitud seni parimat asukohta. Informaatoriteks on mingi hulk teisi osakesi, mis võivad olla juhuslikult valitud või mingil teataval viisil defineeritud naabrid (näiteks asukoha poolest lähimad osakesed).

Algoritm 12 Osakestepilve optimiseerimine

$\lambda \leftarrow$ populatsiooni (osakestepilve) suurus
$P \leftarrow$ esialgne populatsioon (λ lahendit juhusliku positsiooni ja kiirusega)
$Parim \leftarrow$ juhusliku lahendi asukoht populatsioonist P
$\alpha \leftarrow$ kiiruse säilumise proportsioon
$\beta \leftarrow$ osakese parima asukoha arvestamise proportsioon
$\gamma \leftarrow$ osakese informaatorite parima asukoha arvestamise proportsioon
$\delta \leftarrow$ parima vaadeldud asukoha arvestamise proportsioon
repeat
  for $P_i \in P$ **do**
    $\vec{x}_i \leftarrow$ osakese P_i asukoht
    if $Q(\vec{x}_i) > Q(Parim)$ **then**
      $Parim \leftarrow \vec{x}_i$
  for $P_i \in P$ **do**
    $\vec{v}_i \leftarrow$ osakese P_i kiirus
    $\vec{x}_i \leftarrow$ osakese P_i asukoht
    $\vec{x}_i^* \leftarrow$ osakese P_i seni parim asukoht
    $\vec{x}_i^+ \leftarrow$ osakese P_i informaatorite seni parim asukoht
    $\vec{x}^* \leftarrow$ kõigi osakeste poolt seni parim avastatud asukoht
    $b \leftarrow$ juhuslik arv vahemikus $[0; \beta]$
    $c \leftarrow$ juhuslik arv vahemikus $[0; \gamma]$
    $d \leftarrow$ juhuslik arv vahemikus $[0; \delta]$
    $\vec{v}_i \leftarrow \alpha\vec{v}_i + b(\vec{x}_i^* - \vec{x}_i) + c(\vec{x}_i^+ - \vec{x}_i) + d(\vec{x}^* - \vec{x}_i)$
    $\vec{x}_i \leftarrow \vec{x}_i + \vec{v}_i$
until jõuti peatumiskriteeriumini
return $Parim$

4 Kriitika "uudsete" meetodite suunal

Enamik antud töös kirjeldatud metaheuristilisi meetodeid põhineb mõnel loodusest laenatud metafooril. Sageli aitab see intuitiivselt aru saada algoritmi põhimõtetest, kuid erinevate metafooridega tuleb enamasti ka kaasa vastav sõnavara. Artikkel [2] annab kriitilise hinnangu mitmetele metaheuristikatele, mis võtavad kasutusele täiesti uue terminoloogia, kuid sisuliselt midagi uut ei paku. Viimastel aastatel on ilmunud mitmeid artikleid algoritmide kohta, mis kasutavad oma metafoorina erinevate putukate (mesilased, kärbsed, termiidid, jaanimardikad) käitumist, kusjuures sisuliselt erinevad need algoritmid üksteisest vaid marginaalselt ja ei paku

peale sõnavara midagi uut. Lisaks putukatele on oma tee algoritmidesse leidnud ka käod, nahkhiired, intelligentsed veepiisad, impeeriumid oma kolooniatega ja galaktikad.

Põhjalikuma vaatluse alla on artiklis [2] võetud harmooniaotsing (*Harmony search*), mis imiteerib jazzmuusikut, kes improviseerides otsib parema kõlaga harmooniaid, mida hoitakse harmooniamälus. Kuid raske on näha, mis võiks olla harmooniamälu mõistele vaste päris elus. Põhjalikuma analüüsi tulemusel selgub, et harmooniaotsing on tegelikult $(\mu + 1)$ evolutsioonistrateegia, kus igal järgneval põlvkonnal asendatakse populatsioonis kõige kehvem lahend uuega. Tuleb vaid sõnavara ümber vahetada (harmoonia oleks lahend/isend, harmooniamälu oleks populatsioon jne). Kuid sellest hoolimata, et harmooniaotsing midagi uut ei paku on selle põhjal sündinud suur hulk teaduslikke artikleid.[10]

Kõige suurema probleemina näeb [2] "uudsete" meetodite juures seda, et need juhivad tähelepanu kõrvale tõsiselt uurimistöölt ja põhjustavad oma uue sõnavara ja vanade ideedega taandarengu metaheuristikate uurimisvallas, kuna sisuliste uuenduste asemel tegeletakse lihtsalt uute metafooride leidmisega.

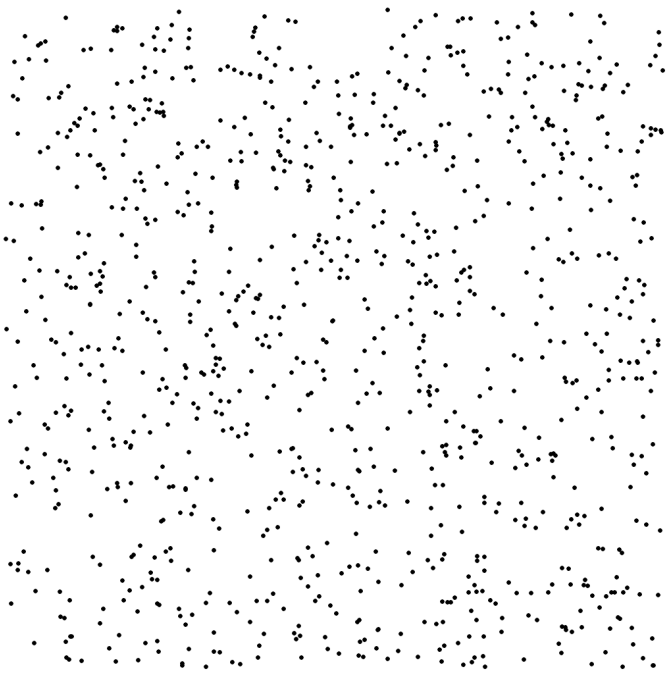
5 Rändkaupmehe ülesande lahendamine kasutades GRASP meetodit

5.1 Ülesande püstitus

Rändkaupmehe ülesanne (ingl *Travelling Salesman problem*) on tuntud ja põhjalikult uuritud ülesanne, mille püstitus on järgmine. Olgu antud n linna nii, et iga linna vahel on tee. Rändkaupmees peab külastama igat linna täpselt ühe korra ja tema teekond peab lõppema samas linnas, kust see algas. Ülesandeks on leida lühim selline marsruut. Teisisõnu on vaja leida lühim Hamiltoni tsükkel kaalutud täisgraafis. Loomulikult leidub sellel ülesandel mitmeid variatsioone, kuid antud töös neid ei vaadelda.

Olgu linnadeks lisas A toodud xy-koordinaatidega määratud punktid ning kahe linna v_1 ja v_2 vastavalt koordinaatidega (x_1, y_1) ja (x_2, y_2) vahelise kauguse d arvutame valemiga $d(v_1, v_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Andmestik on võetud aine

Algoritmika (MTAT.03.238) kodutööde hulgast, antud linnad on kujutatud joonisel 1. Ülesandeks on võetud leida algoritmi GRASP abil võimalikult lühike Hamiltoni tsükkel antud linnade läbimiseks. Täpsemalt öeldes on kasutatud algoritmi GRASP variatsiooni nimega reageeriv GRASP (ingl *Reactive GRASP*).



Joonis 1: 1000 ”linna”

Tavalisel GRASP algoritmil on võrreldes teiste metaheuristikatega üks oluline puudus. Nimelt on algoritmi iteratsioonid üksteisest sõltumatud ja igasugune informatsioon lahendiruumi kohta ei levi edasi ühest iteratsioonist järgmisesse. Teised metaheuristikad üldjuhul annavad igal sammul edasi mingisugust teavet, et otsingut edasi suunata lahendiruumi soodsamatesse piirkondadesse. Näiteks tabu otsing keelab hiljuti vaadeldud lahendite vastu võtmise jooksvaks lahendiks, sipelga algoritmis annab feromoonide kasutamine infot, millised komponendid on andnud parimaid lahendeid jne.

Muutmata kujul GRASP midagi analoogset ei tee. Iga kord, kui poolahnet lahendit konstrueeritakse, tehakse seda eelnevast tööst sõltumatult. Kuid on ka algoritmi GRASP selliseid variatsioone, mis kasutavad ühel või teisel kujul mälu.[7]

Üks sellistest variatsioonidest on reageeriv GRASP, mille korral ei ole lahendi

konstrueerimisel valitavate komponentide arv r lõplikult fikseeritud vaid selle parameetri väärtus valitakse enne iga uue lahendi konstrueerimist ette antud väärtuste $r_1, \ldots, r_k$ seast ning olgu A_i kõigi selliste lahendite keskmine väärtus, mis on saadud kasutades parameetrit r_i. Esialgu on r_i, $i = 1, \ldots, k$ valimise tõenäosus $p_i = 1/k$ ning teatud perioodilisusega arvutatakse see tõenäosus valemiga (4) uuesti.

$$p_i = \frac{q_i}{\sum_{j=1}^{k} q_j} \tag{4}$$

$$q_i = \frac{Parim}{A_i} \tag{5}$$

5.2 Praktiline teostus

Püstitatud ülesande lahendamiseks kirjutatud programmi kood on toodud lisas B, kood on kirjutatud programmeerimiskeeles Python 3.3 [11]. Järgneb programmi töö kirjeldus.

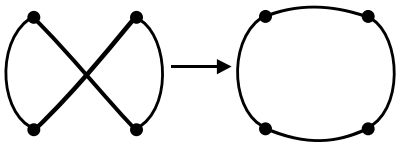
Enne kui algoritmi põhiline töö saab alata, tuleb vajalikud parameetrid eelnevalt lähtestada, mis on valitud järgnevalt:

- peatumiskriteerium - programm lõpetab töö, kui on tehtud 1000 iteratsiooni
- ahnuse määra r võimalikud väärtused - $\{1, 2, 3, 4, 5\}$, tõenäosus iga väärtuse valimiseks esialgu 1/5.
- võimalike r väärtuse valimise tõenäosuste ümberarvutamise sagedus - iga 100 iteratsiooni tagant

Lisaks on igale linnale vastavusse seatud massiiv teistest linnadest, mis on sorditud linnadevahelise kauguse kasvamise järjekorras ehk igale linnale vastava massivi esimene element on antud linnale lähim linn, teine element on antud linnale kauguselt teine linn jne. See tabel on kasutusel uue lahendi konstrueerimise käigus.

Lahendi poolahne konstrueerimine. Enne lahendi konstrueerima asumist valitakse juhuslikult parameetri r väärtus, võttes arvesse iga väärtuse valituks saamise tõenäosust (tõenäosused arvutatakse määratud perioodi tagant ümber, kasutades valemit (4)). Seejärel valitakse juhuslik linn alguspunktiks ja hakatakse poolikule lahendile lisama linnu kuni on konstrueeritud Hamiltoni tsükkel. Linnade tähistamiseks on kasutusel numbrid $0, 1, \ldots, 999$ ning lahendikandidaadiks võib olla näiteks jada linnadest $0, 1, \ldots, 999, 0$. Iga järgneva linna valikul kasutatakse ära varasemalt loodud kauguste järgi sorditud linnade tabelit - viimati lahendisse lisatud linnale vastavast massiivist võetakse r esimest linna, mis ei ole juba lahendis ja neist üks juhuslikult valitud linn lisatakse lahendisse. Kui kõik linnad on lahendis olemas, siis lisatakse lõppu veel linn, millest alustati, ja konstrueerimine on lõppenud.

Lokaalne otsing. Lokaalse otsingu tegemiseks on valitud 2-opt algoritm [12]. 2-opt alustab tööd juhuslikust Hamiltoni tsüklist ja hakkab seda sammhaaval lühemaks muutma. Igal sammul valitakse kaks serva $\{u_1, u_2\}$ ja $\{v_1, v_2\}$ (tipud u_1, u_2, v_1, v_2 on paarikaupa erinevad ja esinevad tsüklis antud järjekorras) ja asendatakse need servadega $\{u_1, v_1\}$ ja $\{u_2, v_2\}$ juhul, kui see muudab tsükli lühemaks. Sisuliselt toimub tsüklis leiduva alamlõigu ümber pööramine. Tsükkel on 2-optimaalne, kui ei leidu kahte serva, mille asendamisel tsükkel lühemaks muutuks. Joonisel 2 on näide 2-opt algoritmi poolt tehtavast ümberjärjestusest, mida nimetame 2-opt vahetuseks.



Joonis 2: 2-opt vahetus

Et lokaalne otsing oleks efektiivne (arvutiressursi kasutamise mõttes), tuleb 2-opt vahetust teha kasutades võimalikult vähe operatsioone. Selle eesmärgi nimel teisendatakse konstrueerimise tulemusel saadud linnade massiv satelliitahela andmestruktuuriks. Satelliitahel [13] on sisuliselt tsükliline topeltahel, kus iga linnaga on seotud 2 "satelliiti", mis viitavad vastavalt eelmisele ja järgmisele linnale tsüklis.

Näiteks olgu meil $n = 5$ linna, mis on nummerdatud $0, \ldots, n-1$ (antud juhul siis

$0, 1, 2, 3, 4$), mis moodustavad näiteks tsükli $0 - 1 - 3 - 2 - 4 - 0$. Tabeli 1 teises reas on antud tsükkel esitatud satelliitahelana. Esimese reas on toodud linnad, iga linna all kaks antud linnaga seotud satelliiti, tabeli kolmandas reas on toodud satelliitahela indeksid. Satelliidile indeksiga i vastab linn $\lfloor i/2 \rfloor$ ja vastava linna teise satelliidi indeksi saab leida valemiga $i + 1 - 2(i \bmod 2)$. Tsükli läbimiseks võib alustada satelliitahela suvalisest elemendist, elemendi väärtus viitab tsüklis järgmise linna satelliidi indeksile. Näiteks tabelis 1 satelliitahela elemendi indeksiga 0 (vastab linnale 0) väärtus on 2, mis viitab satelliidile indeksiga 2 (vastab linnale 1) jne.

Antud esituse korral tuleb tsüklis suvalise alamlõigu ümberpööramiseks muuta ainult nelja viida (satelliidi) väärtust ning tsükli uue pikkuse arvutamiseks pärast 2-opt vahetust tuleb esialgsest pikkusest lahutada eemaldatud kaarte pikkused ja liita lisatud kaarte pikkused. Viimane tähelepanek tsükli uue pikkuse arvutamise kohta ei ole küll satelliitahelaga seotud, kuid seda teadmist saame siiski kasutada, et programm ei peaks palju tühja tööd tegema.

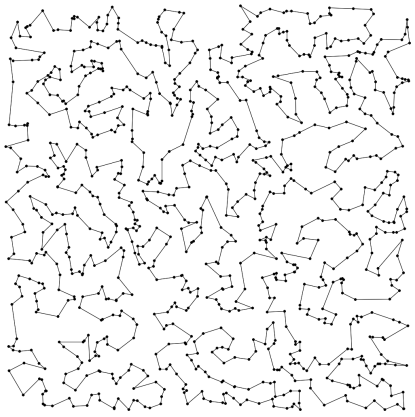
Linn	0		1		2		3		4	
Satelliitahel	2	9	6	1	8	7	4	3	0	5
Satelliitahela indeks	0	1	2	3	4	5	6	7	8	9

Tabel 1: Satelliitahela füüsiline esitus

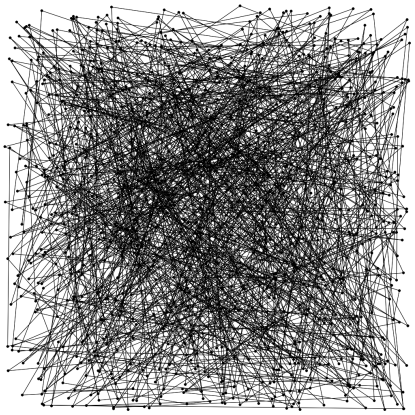
Kui lokaalne otsing jõuab lõpule, siis uuendatakse kasutatud parameetriga r seotud keskmist lahendi väärtust ning võrreldakse saadud tulemust seni parima tulemusega ja asendatakse seni parim tulemus uuega, kui uus lahend on parem.

5.3 Lahenduse tulemused

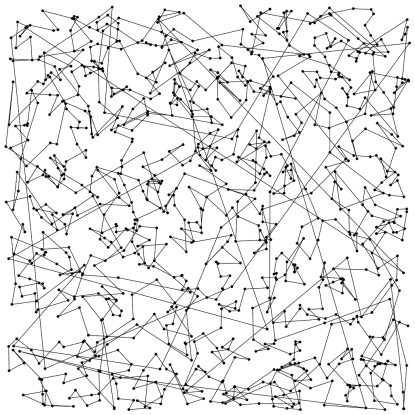
Programm tegi 1000 iteratsiooni, selleks kulus aega 60 h 22 min, ühe iteratsiooni peale kulus keskmiselt 3 min 37 s. Parima leitud tsükli pikkus on 24126, joonisel 3 on näidatud parim lahend ja võrdluseks on kõrval täiesti juhuslik lahend, poolahnelt konstrueeritud ja täielikult ahnelt konstrueeritud lahend (jooniste tegemiseks on kasutatud vahendit SWOG [14]). Tsüklite pikkused on ümardatud ühelisteni.



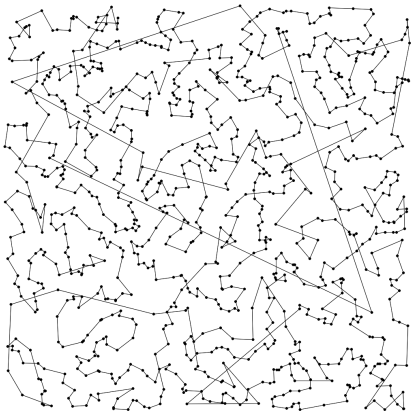
(a) Parim tulemus. Tsükli pikkus 24126



(b) Juhuslik lahend. Tsükli pikkus 523598



(c) Poolahne lahend, $r = 5$. Tsükli pikkus 59922



(d) Ahne lahend, $r = 1$. Tsükli pikkus 29557

Joonis 3: Parim leitud, juhuslik, poolahne ja ahne lahend

Tabelis 2 on toodud esimeses veerus parameetri r võimalikud väärtused, teises veerus viimane vastava väärtuse valimise tõenäosus (esialgu oli see tõenäosus igal väärtusel 1/5), kolmandas veerus on näidatud, mitu korda vastavat väärtust lahendi konstrueerimiseks valiti ja viimases veerus on toodud antud parameetrit kasutades konstrueeritud tsüklite keskmine pikkus.

r	Väärtuse valimise tõenäosus	Valitud kordi	Keskmine tulemus
1	0.2059	197	24579
2	0.1996	208	25359
3	0.1984	179	25515
4	0.1981	216	25556
5	0.1980	200	25549

Tabel 2: Parameetri r valiku mõju lahendile

Saadud tulemuste põhjal jääb mulje, et vähemalt antud juhul ei andnud reageeriv GRASP loodetud kasu ja oleks samahästi oleks võinud parameetri r väärtuse fikseerida. Liiga suurt r väärtust ei ole praktiline valida, sest sel juhul on konstrueeritud lahend liialt juhuslik ja lokaalse otsingu peale kulub palju aega. Jooniselt 3 on näha, et mida juhuslikumalt on konstrueeritud lahend, seda rohkem parandusi on vaja teha, et jõuda lokaalsesse optimumi. Teisalt kui valida r väärtuseks 1 või 2, siis see piirab liialt konstrueeritavate lahendite mitmekesisust. Sobiv r väärtus võiks olla seega vahemikus 3 kuni 5 ja eelnevatel iteratsioonidel saadud info kasutamiseks tuleks kasutada mõnda muud strateegiat. Need järeldused on tehtud ainult vaadeldava ülesande kohta, mitte algoritmi GRASP kasutamise kohta üldiselt.

Kokkuvõte

Käesoleva töö eesmärgiks oli anda ülevaade metaheuristilistest meetoditest. Selle eesmärgi nimel sai kirjeldatud, mida metaheuristilised meetodid endast kujutavad ja miks nad vajalikud on. Sai antud ülevaade erinevatest metaheuristikatest, mille jaotasime kaheks selle alusel, mitu lahendit algoritmil korraga vaatluse all on. Tutvusime trajektoormeetoditega, mis vaatlevad korraga ühte lahendit ja populatsioonimeetoditega, mis vaatlevad korraga rohkem kui ühte lahendit.

Kiire ülevaade sai antud kriitikast, mille osaks on mitmed metaheuristikad saanud, kuna nad ei sisalda uusi ideid, ainult uut terminoloogiat, mis aga muudab metaheuristikate valdkonnas kasuliku ja üleliigse informatsiooni eristamise raskemaks.

Ning lõpuks sai rändkaupmehe ülesande lahendamiseks rakendatud GRASP algoritmi, mille iga iteratsioon koosneb kahest etapist: lahendi poolahne konstrueerimine ja lokaalne otsing. Lokaalse otsingu sooritamiseks sai kasutatud 2-opt algoritmi. Metaheuristikate rakendamisel tuleb tähelepanu pöörata ka sellele, kuidas esitada lahendit. Sobiva andmestruktuuri kasutamisel on võimalik vältida liigsete operatsioonide tegemist arvuti poolt. Antud juhul sai lahendi esitamiseks valitud satelliitahela andmestruktuur.

Overview of metaheuristics and solving the Travelling Salesman problem with GRASP

Bachelor thesis (6 EAP)

Indrek Loolaid

Summary

This bachelor thesis gives an overview of metaheuristics, what they are and what are they used for. Several metaheuristic algorithms are briefly examined which are divided into two groups based on how many solution candidates at a time a given algorithm operates with: trajectory methods, which operate with a single solution candidate at a time and population methods which operate with more than one solution candidates at a time.

A brief criticism towards ”novel” metaheuristics is given which introduce new terminology to describe the algorithm but don’t offer new ideas.

And finally a metaheuristic called GRASP is applied on the Travelling Salesman problem the size of 1000 ”cities”.

Viited

[1] Sean Luke. *Essentials of Metaheuristics*. Lulu, 2009. http://cs.gmu.edu/~sean/book/metaheuristics/.

[2] Kenneth Sörensen. Metaheuristics - the metaphor exposed. *International Transactions in Operational Research*, 2013. http://antor.ua.ac.be/system/files/mme.pdf.

[3] Andrea Roli Christian Blum. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM COMPUTING SURVEYS*, pages 268–308, 2003. http://www-lia.deis.unibo.it/Staff/AndreaRoli/pubs/blum_roli_metaheuristics-preprint.pdf.

[4] Manuel Laguna Fred Glover. Tabu search. http://aiinfinance.com/laguna.pdf.

[5] Thomas Stützle Helena R. Lourenço, Olivier C. Martin. Iterated local search. In *Handbook of Metaheuristics, volume 57 of International Series in Operations Research and Management Science*, pages 321–353. Kluwer Academic Publishers, 2002. http://iridia.ulb.ac.be/~stuetzle/publications/ILS.ps.gz.

[6] Nenad Mladenovic Pierre Hansen. Variable neighborhood search, 1997. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.8989.

[7] Celso C. Ribeiro Mauricio G. C. Resende. Greedy randomized adaptive search procedures, 2002. http://www.research.att.com/~mgcr/doc/sgrasp-hmetah.pdf.

[8] Fabio De Luigi Vittorio Maniezzo, Luca Maria Gambardella. Ant colony optimization. In *Optimization Techniques in Engineering. Springer-Verlag*, pages 101–117. Addison-Wesley, 2004. http://www.idsia.ch/~luca/aco2004.pdf.

[9] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994. http://www.cs.uga.edu/~potter/CompIntell/ga_tutorial.pdf.

[10] Dennis Weyland. A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel"methodology. *Int. J. of Applied Metaheuristic Computing*, 1(2):50–60, 2010. `http://www.idsia.ch/~weyland/harmony_search.pdf`.

[11] `http://python.org/`.

[12] Berthold Vöcking Matthias Englert, Heiko Röglin. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. In *In Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA*, pages 1295–1304, 2007. `http://www.roeglin.org/publications/SODA07.pdf`.

[13] César Rego Colin Osterman. The satellite list and new data structures for symmetric traveling salesman problems, 2004.

[14] `http://biit.cs.ut.ee/swog/`.

Lisa A Linnade koordinaadid

792 600	228 593	728 250	172 500	77 269	189 852	279 956
99 47	327 405	460 739	293 705	54 703	419 265	447 246
841 620	327 908	291 288	230 844	860 992	511 143	291 791
795 79	33 332	533 451	386 33	639 596	605 960	165 968
13 648	495 618	911 884	488 494	566 57	935 466	313 285
8 157	970 676	830 912	610 885	311 113	559 858	984 339
664 336	967 321	209 539	216 529	313 691	287 617	939 439
676 943	210 833	670 322	404 163	315 744	384 15	223 134
992 490	645 398	933 179	615 388	25 312	89 476	351 733
765 129	36 924	240 224	116 483	244 176	532 330	677 930
19 148	988 817	901 181	433 768	921 350	81 248	398 209
80 812	846 62	155 271	868 869	879 125	993 965	437 333
233 24	824 651	869 232	199 624	753 955	619 145	889 772
567 58	340 601	100 577	195 358	404 890	861 629	871 886
439 52	891 5	908 180	825 833	656 421	670 338	941 576
101 47	528 261	414 213	32 541	974 217	563 176	149 757
423 548	667 172	47 704	930 562	398 101	929 916	851 61
274 440	73 288	950 274	480 604	626 456	289 750	50 945
373 810	149 89	498 527	320 470	690 244	602 125	664 164
952 323	405 639	688 37	228 905	397 925	189 556	919 903
892 347	93 581	660 27	844 752	717 162	573 676	26 895
845 22	359 903	201 85	683 388	9 97	810 316	457 791
231 133	937 174	232 761	982 652	960 748	889 951	41 281
110 823	364 835	312 513	211 725	702 901	716 72	892 109
929 137	161 938	981 420	332 454	850 440	237 222	275 911

130 940 452 301 308 791 773 474 642 163 90 397 546 42
121 629 490 28 198 826 423 255 756 536 717 770 415 784
654 991 358 783 449 136 883 507 574 823 381 561 689 536
969 381 93 439 643 926 739 311 836 322 287 245 407 112
527 824 923 333 889 550 753 873 587 941 874 86 750 249
854 797 777 113 919 59 826 824 277 722 431 832 144 156
534 250 801 980 876 61 85 873 136 816 599 891 586 823
706 593 98 102 20 158 872 918 148 594 441 981 274 270
635 496 961 463 629 420 352 729 218 848 643 653 499 744
438 605 145 571 576 625 475 281 952 60 969 822 785 322
872 217 704 114 817 236 476 300 903 926 524 822 985 3
953 745 279 703 30 959 980 915 551 545 220 357 97 12
157 402 316 489 569 417 226 600 162 850 56 667 345 8
486 46 190 334 539 39 770 836 230 288 435 909 833 325
62 922 231 971 604 656 173 49 394 989 324 443 802 637
636 504 599 811 418 506 846 151 832 823 695 986 945 121
424 754 280 969 326 469 906 388 630 795 917 452 211 382
55 563 640 583 90 75 890 695 216 674 848 914 599 812
43 459 261 415 213 682 833 795 245 425 885 124 150 614
96 883 681 775 116 10 144 764 792 308 410 915 314 670
683 106 9 705 209 448 814 564 398 410 992 110 636 465
271 687 830 546 737 424 467 169 818 747 574 515 729 1
504 890 713 125 221 144 244 483 228 526 853 212 170 887
60 369 773 420 526 693 849 494 48 440 778 821 971 581
765 920 742 546 807 522 963 588 404 532 443 584 540 116
354 564 764 957 997 813 715 341 25 702 805 976 144 583
251 954 423 89 706 608 577 838 536 402 693 600 192 168

473 464 623 973 109 730 291 378 412 728 645 624 704 138
876 744 621 690 843 712 128 445 913 108 70 498 126 487
113 834 189 117 549 810 546 559 383 499 235 843 934 536
317 178 799 113 696 914 865 273 869 621 530 809 457 616
498 278 448 48 129 778 890 882 279 585 99 816 19 495
286 892 969 496 765 705 84 77 21 386 305 98 66 865
393 51 862 432 132 297 510 710 635 327 663 535 111 623
497 526 60 324 524 820 646 587 831 126 564 632 551 150
688 982 463 912 179 779 348 821 460 182 939 114 320 388
317 24 981 809 105 827 8 374 868 436 2 514 775 303
477 108 320 460 134 842 266 78 614 787 858 665 166 485
300 799 115 460 165 337 901 422 595 533 241 838 889 732
653 277 282 912 356 420 111 660 867 862 714 678 281 536
845 780 376 958 664 100 849 876 912 23 387 896 560 305
65 363 274 272 672 99 178 971 129 552 765 799 446 163
904 990 325 211 135 334 254 402 208 946 418 778 19 811
151 744 959 439 92 39 852 57 608 880 654 579 949 531
119 938 696 366 483 498 682 632 788 12 38 621 127 657
789 364 485 438 406 851 100 390 613 174 581 999 182 800
226 958 829 324 597 25 127 361 643 419 876 884 919 709
227 167 572 613 946 885 279 28 944 590 640 663 387 619
197 739 231 40 341 98 162 699 416 531 650 151 386 934
65 790 150 389 53 783 784 468 372 901 801 852 546 481
789 215 779 973 253 976 52 603 825 114 645 427 915 811
285 584 560 871 561 500 538 298 430 799 396 504 496 361
665 277 680 762 992 497 725 116 54 948 245 735 380 972
738 879 378 966 352 738 581 13 690 554 816 471 828 496

177 657 617 522 158 29 107 577 342 636 536 155 902 626

881 133 575 478 575 693 780 329 627 16 686 588 706 572

460 123 581 363 935 785 443 470 652 662 286 401 130 598

958 732 64 148 435 332 974 295 981 819 79 602 696 936

754 54 279 327 525 768 959 223 151 435 55 708 317 53

450 15 362 571 845 376 234 422 654 798 147 459 44 37

144 67 654 539 288 601 163 347 906 241 833 289 465 760

263 796 444 166 795 894 506 869 815 918 293 394 61 951

372 138 241 937 447 391 329 385 714 341 526 362 284 80

168 9 220 862 662 824 122 849 523 153 495 791 568 251

609 929 663 9 94 806 969 567 488 646 477 657 93 15

369 580 733 453 582 904 76 242 96 383 523 648 133 168

179 697 555 332 251 181 861 54 638 777 141 763 435 448

1 651 14 920 447 319 960 465 914 791 985 527 471 101

374 450 784 250 274 65 593 802 313 416 805 836 368 237

645 414 670 374 727 686 536 300 903 91 629 485 231 97

170 973 483 451 577 246 709 8 727 47 313 663 934 74

637 69 987 448 964 111 35 304 789 769 357 5 999 907

537 18 531 870 620 646 213 863 508 592 815 589 83 314

439 656 228 680 344 363 82 866 327 831 210 897 478 800

329 568 164 836 996 816 647 242 656 627 16 261 191 280

995 208 535 618 696 4 554 244 886 906 519 625 848 616

883 410 917 436 619 612 401 242 408 428 359 769 228 939

618 562 208 122 278 148 391 490 806 212 520 592 527 898

372 69 928 621 158 378 601 51 205 185 697 604 693 319

906 480 387 899 727 875 460 414 592 299 856 150 654 786

283 233 264 89 965 647 252 162 348 908 416 941 195 171

851 816 672 943 83 115 180 102 961 784 984 3 340 404

255 275 247 369 234 509 239 305 266 235 913 544 200 697

294 556 560 670 92 312 747 980 191 337 737 102 665 69

852 434 926 114 510 604 212 911 507 21 707 986 668 327

938 1 183 342 475 140 171 274 202 11 368 369 417 328

674 826 28 176 536 705 316 225 967 799 595 766 567 130

456 858 352 84 939 874 407 759 705 47 831 314 521 273

61 528 904 728 914 507 781 784 728 168 558 723 791 226

92 987 804 898 726 17 351 558 178 481 454 551 983 44

606 814 564 29 77 387 469 882 988 929 716 115 494 94

100 508 463 731 366 355 907 837 281 525 775 224 489 189

977 479 745 97 631 982 669 459 717 169 880 230 931 76

713 15 955 681 192 890 816 818 948 463 35 145 273 459

442 43 214 578 698 724 15 427 509 725 96 301 418 817

967 439 507 841 268 781 351 519 77 494 994 946 753 374

910 979 129 551 801 994 538 72 183 712 952 556 793 931

98 593 708 86 755 75 144 602 174 537 205 751 694 206

130 231 611 252 802 735 637 907 178 132 430 983 199 642

249 274 627 673 338 913 969 14 349 522 721 412 558 432

615 543 549 416 80 759 525 834 340 541 648 302 425 770

823 829 847 773 800 697 470 640 142 483 387 566 331 360

139 580 240 844 570 268 746 236 481 630 691 334 46 116

917 629 302 445 401 487 170 280 716 359 908 879 655 793

630 444 671 756 636 869 374 322 392 200 655 66 732 710

713 567 306 505 377 539 12 868 195 848 461 725 110 779

269 6 170 967 769 816 275 455 207 738 685 668 735 837

954 35 419 807 869 393 342 770 901 834 316 450 29 588

305 1	921 114	722 351	457 763	452 252	533 203	859 126
264 996	278 941	506 684	896 491	667 33	506 652	298 448
687 934	437 297	503 82	706 809	270 570	446 63	860 225
811 349	728 147	728 259	183 253	580 765	662 216	475 890
213 37	262 692	465 417	269 3	521 366	238 857	382 101
535 902	408 757	411 255	790 873	544 16	612 617	627 877
473 812	476 649	457 291	677 99	354 351	326 23	712 359
234 95	19 863	808 72	381 782	647 200	395 858	239 438
629 584	718 731	535 69	764 901	955 308	862 781	453 463
841 36	384 560	593 850	750 878	262 775	254 895	

Lisa B GRASP programmi kood

```
1  from satelliteList import SatelliteList, getDistance
2  from random import randint, random
3  import itertools
4  import bisect
5  from time import time
6
7  def grasp(vertices, minGreed, stepLimit = 10000, greedEvalInterval
       = 100):
8      sortedNeighbors = getSortedNeighbors(vertices)
9      greedValues = list(range(1, minGreed + 1))
10     greedProbabilities = [1 / minGreed] * minGreed
11     cumdist = list(itertools.accumulate(greedProbabilities))
12     avgResults = {g:(float('inf'), 0) for g in greedValues}
13     best = None
14     bestTourLength = float('inf')
15     steps = 0
16
17     while steps < stepLimit:
18         start = time()
19         if steps % greedEvalInterval == 0 and steps > 0:
20             cumdist = evaluateGreedProbs(greedProbabilities,
                   avgResults, bestTourLength)
21         steps += 1
22         greediness = greedValues[bisect.bisect(cumdist, random() *
               cumdist[-1])]
23         # lahendi konstrueerimine
24         candidate = greedyConstruction(sortedNeighbors, greediness)
25         candidate_sl = SatelliteList(candidate, vertices)
26         # lokaalne otsing
27         candidate_sl.twoOpt()
28         updateAvgResults(greediness, avgResults, candidate_sl.
               tourLength)
29         if candidate_sl.tourLength < bestTourLength:
```

```
30              best = candidate_sl
31              bestTourLength = best.tourLength
32          end = time()
33          print(end - start)
34      print(avgResults)
35      print(greedProbabilities)
36      return best
37
38
39  def getSortedNeighbors(vertices):
40          sortedNeighbors = {}
41          for v in vertices:
42                  dFun = lambda k: getDistance(vertices[v], vertices[k
                        ])
43                  sortedNeighbors[v] = sorted(vertices, key = dFun)[1:]
44          return sortedNeighbors
45
46
47  def greedyConstruction(sortedNeighbors, r):
48      start = randint(0, len(sortedNeighbors) - 1)
49      tour = [start]
50      while len(tour) < len(sortedNeighbors):
51          last = tour[-1]
52          avilableNeighbors = filter(lambda v: v not in tour,
                sortedNeighbors[last])
53          # rcl - Restricted Candidate List
54          rcl = list(islice(avilableNeighbors, r))
55          nextNeighbor = rcl[randint(0, len(rcl) - 1)]
56          tour.append(nextNeighbor)
57      tour.append(start)
58      return tour
59
60  def updateAvgResults(greediness, avgResults, result):
61      avgResult, count = avgResults[greediness]
62      if avgResult == float('inf'):
```

```
63          avgResults[greediness] = (result, 1)
64      else:
65          count += 1
66          newAvg = avgResult + (result - avgResult) / count
67          avgResults[greediness] = (newAvg, count)
68
69  def evaluateGreedProbs(greedProbabilities, avgResults,
        bestTourLength):
70      bestToAvgRatios = [bestTourLength / avgResults[k][0] for k in
            avgResults]
71      ratiosSum = sum(bestToAvgRatios)
72      for i in range(len(greedProbabilities)):
73          greedProbabilities[i] = bestToAvgRatios[i] / ratiosSum
74      return list(itertools.accumulate(greedProbabilities))
```

Programm 1: GRASP algoritm

```
from math import sqrt

class SatelliteList:
    def __init__(self, tour, vertices):
        self.vertices = vertices
        self.tourLength = tourLength(vertices, tour)
        self.tour = [None for _ in range((len(tour) - 1) * 2)]
        for i in range(len(tour)):
            v = tour[i]
            if i != len(tour) - 1:
                self.tour[v*2] = tour[i + 1] * 2
            if i != 0:
                self.tour[v*2 + 1] = tour[i-1] * 2 + 1

    def getTour(self):
        tour = []
        i = 0
        for _ in range(len(self.tour) // 2 + 1):
            tour.append(i // 2)
            i = self.tour[i]
        return tour

    def forward(self, start, steps = 1):
        for _ in range(steps):
            start = self.tour[start]
        return start

    def reverseSubpath(self, a, b, c, d):
        self.tour[a] = c ^ 1
        self.tour[c] = a ^ 1
        self.tour[d ^ 1] = b
        self.tour[b ^ 1] = d
        return (a, c ^ 1, b ^ 1, d)

```

```
35      def twoOpt(self):
36          improved = True
37          steps = 0
38          a = 0
39          while improved:
40              improved = False
41              bestDistance = self.tourLength
42              for i in range(len(self.vertices) - 2):
43                  b = self.forward(a)
44                  c = b
45                  for k in range(len(self.vertices) - i - 2):
46                      steps += 1
47                      c = self.forward(c)
48                      d = self.forward(c)
49                      v_a = a >> 1
50                      v_b = b >> 1
51                      v_c = c >> 1
52                      v_d = d >> 1
53                      dist_ab = getDistance(self.vertices[v_a], self.
                            vertices[v_b])
54                      dist_cd = getDistance(self.vertices[v_c], self.
                            vertices[v_d])
55                      oldDist = dist_ab + dist_cd
56                      dist_ac = getDistance(self.vertices[v_a], self.
                            vertices[v_c])
57                      dist_bd = getDistance(self.vertices[v_b], self.
                            vertices[v_d])
58                      newDist = dist_ac + dist_bd
59                      distChange = oldDist - newDist
60                      if distChange > 0:
61                          self.reverseSubpath(a, b, c, d)
62                          self.tourLength -= distChange
63                          improved = True
64                          break
65                  if improved:
```

```
66                  break
67              a = b
68
69
70  def getDistance(v1, v2):
71      return sqrt(sum(((e1 - e2) ** 2 for e1, e2 in zip(v1, v2))))
72
73  def tourLength(vertices, tour):
74      distance = 0
75      for i in range(len(tour) - 1):
76          v1 = vertices[tour[i]]
77          v2 = vertices[tour[i+1]]
78          distance += getDistance(v1, v2)
79      return distance
```

Programm 2: Satelliitahel

```
from grasp import *
from time import time
from satelliteList import SatelliteList
from TSPUtils import *

vertices = readTSPFromFile('TSP/TSP_1000.txt')
start = time()
best = grasp(vertices, 5)
end = time()
print('time:␣', end - start)
commands = tourToSwogCommands(vertices, best.getTour())
fileName = 'TSP/GRASP/1000_grasp_' + str(best.tourLength) + '.txt'
swogCommandsToFile(commands, fileName)
```

Programm 3: GRASP programmi käivitamine

```
1  from math import sqrt
2
3  def readTSPFromFile(file):
4      nodes = {}
5      key = 0
6      with open(file) as f:
7          for line in f:
8              if line.startswith('#'):
9                  continue
10             coords = tuple(map(int, line.split()))
11             nodes[key] = coords
12             key += 1
13     return nodes
14
15 def tourToSwogCommands(vertices, tour):
16     swogCommands = []
17     swogCommands.append('new␣1100,1100\n')
18     swogCommands.append('coordsys␣west␣south␣right␣up\n')
19     swogCommands.append('origin␣50,50\n')
20     for k in vertices:
21         command = ''.join(('fcircle␣', str(vertices[k][0]), ',' ,
               str(vertices[k][1]), '␣3␣:p', str(k), '\n'))
22         swogCommands.append(command)
23     for i in range(len(tour) - 1):
24         command = ''.join(('line␣(p', str(tour[i]), ')␣(p', str(tour
               [i+1]), ')\n'))
25         swogCommands.append(command)
26     return swogCommands
27
28 def swogCommandsToFile(swogCommands, file):
29     with open(file, 'w') as f:
30         f.writelines(swogCommands)
```

Programm 4: Abifunktsioonid

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina Indrek Loolaid

(sünnikuupäev: 02.04.1988)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
„Ülevaade metaheuristilistest meetoditest ja rändkaupmehe ülesande lahendamine GRASP meetodiga“,

mille juhendaja on 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, 04.06.2013