Aalto-yliopisto
Informaatio- ja luonnontieteiden tiedekunta
Teknillisen fysiikan ja matematiikan tutkinto-ohjelma

Kimi Ylilammi

Monen kauppamatkustajan ongelman ratkaiseminen kombinatorisena huuto-kauppana

Kandidaatintyö
Espoo, 12. Maaliskuuta 2012

Valvoja: TkT Kimmo Berg
Ohjaaja: Prof. Harri Ehtamo

Aalto-yliopisto
Informaatio- ja luonnontieteiden tiedekunta
Teknillisen fysiikan ja matematiikan tutkinto-ohjelma

KANDIDAATINTYÖN TIIVISTELMÄ

Tekijä:	Kimi Ylilammi
Työn nimi:	Monen kauppamatkustajan ongelman ratkaiseminen kombinatorisena huutokauppana
Päiväys:	12. Maaliskuuta 2012 **Sivumäärä:** 23
Tutkinto-ohjelma:	Teknillinen fysiikka ja matematiikka
Valvoja:	TkT Kimmo Berg
Ohjaaja:	Prof. Harri Ehtamo

Monen kauppamatkustajan ongelma on yleistys tunnetusta yhden kauppamatkustajan ongelmasta, jossa kauppamatkustajan on käytävä kaikissa annetuissa kaupungeissa mahdollisimman nopeasti. Työssä tutkitaan monen kauppamatkustajan ongelman ratkaisemista kombinatorisena huutokauppana, jossa kauppamatkustajat asettavat huutoja järjestelmään. Huonojen reittivaihtoehtojen poistamiseen esitellään kolme menetelmää. Kombinatorisen huutokaupan huono puoli on huutojen viemä tila sekä niiden läpikäyminen parhaan huutokombinaation löytämiseksi. Etäisyyden mittaaminen pienimmästä virittävästä puusta osoittautui parhaimmaksi menetelmäksi, kun kauppamatkustajien ei tarvitse palata alkupisteisiinsä. Kuitenkin jos kauppamatkustajien on palattava alkuun, on parempi mitata etäisyyttä alkupisteestä. Yhden kauppamatkustajan vakio yksikköneliöllä vaikuttaisi olevan sama kuin monen kauppamatkustajan vakio.

Asiasanat:	TSP, mTSP, mTSPNR, kauppamatkustajan vakio
Kieli:	Suomi

Symbolit ja lyhenteet

TSP	Travelling Salesman Problem, kauppamatkustajan ongelma
mTSP	Multiple Travelling Salesman Problem, monen kauppamatkustajan ongelma
mTSPNR	Multiple Travelling Salesman Problem No Return, monen kauppamatkustajan ongelma, jossa kauppamatkustajien ei tarvitse palata alkupisteeseen.
Hamiltonin polku	Polku, joka käy kaikissa kaupungeissa ja palaa alkupisteeseen.
Läheisyysmittari	Mittari, joka kertoo kuinka lähellä kaksi kaupunkia ovat toisiaan.
Alpha-läheisyys	Etäisyys mitattuna pienimmästä virittävästä puusta
EMS	Euklidean Minimun Spanning tree, pienin virittävä puu euklidisessa avaruudessa
NSPH	Nearest Starting Point Heuristic. Heuristiikka, joka mittaa etäisyyttä alkupisteestä.
NNH	Nearest Neighbor Heuristic. Heuristiikka, joka mittaa etäisyyttä edellisestä kaupungista.
Lin - Kernighan -algoritmi	TSP:hen käytetty ratkaisualgoritmi
SEC	Subtour Elimination Constraint, alipolkujen poistamisrajoitus
c_1	Kauppamatkustajan vakio
n	Kaupunkien lukumäärä
m	Kauppamatkustajien lukumäärä
H	Huutojen lukumäärä
L_n	Lyhimmän suljetun polun pituus kaupunkien välillä.
c_{ij}	Kaupunkien i ja j välinen etäisyys.
cm_{ij}	Kaupungin i ja kauppamatkustajan j välinen etäisyys.
α_{ij}	Kaupunkien i ja j välinen alfa-etäisyys.
$MST(i)$	Kaupungin i etäisyys pienimmästä virittävästä puusta.
2-opt ja 3-opt	Lokaaleja optimaalisia reittejä vastaavasi kahden tai kolmen kaupungin välillä.

Sisältö

1 Monen kauppamatkustajan ongelma **5**

2 Menetelmät **6**

2.1 Alustus 8

2.2 Kauppamatkustajien huutojen asetus 8

2.2.1 Yhden kauppamatkustajan ongelma 9

2.2.2 Ympärillä olevat kaupungit 9

2.2.3 Lin - Kernighanin alfa 11

2.2.3.1 Pienin virittävä puu 12

2.3 Puun lehtien luominen 13

2.4 Puurakenteen ratkaiseminen 13

2.5 Kauppamatkustajan vakio 14

3 Tulokset **15**

3.1 Kauppamatkustajan vakion mittaus 15

3.2 Heuristiikkojen vertailua 15

3.2.1 Viisi kauppamatkustajaa 16

3.2.2 Kaksi kauppamatkustajaa 18

3.2.3 Kauppamatkustajien lukumäärän vaikutus 19

3.2.4 Heuristiikat perinteisessä ongelmassa 20

4 Yhteenveto **21**

1 Monen kauppamatkustajan ongelma

Monen kauppamatkustajan ongelma on helppo ymmärtää mutta vaikea laskennallisesti ratkaista, sillä ongelma on tyyppiä $P = NP$. Ongelmaa käytetään monesti eri optimointitehtävissä apuna, sillä moni ongelma suppenee mTSP:aan. mTSP voidaan myös supistaa isoksi TSP:ksi [9], mutta tämän yhdiste-TSP:n ratkaiseminen on useammin kuitenkin vaikeampaa kuin alkuperäisen mTSP:n. Tämä onnistuu lisäämällä $m - 1$ uutta aloituspistettä ja asettamalla äärettömän matkan näiden keskinäisten pisteiden välille. Tämän TSP:n ratkaisusta tulee sama kuin alkuperäisestä mTSP:sta.

Käsittelen ongelmaa, jossa kaupungit ja kauppamatkustajat jaetaan arpomalla satunnaisesti yksikköneliön sisälle. Alueella ei ole seiniä tai muita esteitä vaan kaupunkien välisenä etäisyytenä toimii euklidinen etäisyys $D = \sqrt{x^2 + y^2}$. Etäisyydet talletetaan kaupungeista kaupunkiin matriisiin c_{ij} ja vastaavasti kauppamatkustajan aloituspisteestä kaupunkiin matriisiin cm_{ij}. Kuljettava matka ei riipu kuljetusta suunnasta ja ongelma on tällöin symmetrinen $c_{ij} = c_{ji}$. Ongelma on myös metrinen, eli kolmiosymmetrinen: $c_{ij} + c_{jk} \geq c_{ik}$. Matriisit ovat myös aina positiivisia, koska kyseessä on euklidinen etäisyys.

Euklidiset TSP:t on todettu vaikeammaksi kuin ei-euklidiset TSP:t [10]. Sovelluksissa etäisyyttä voitaisiin myös mitata toisin eli esim. ohitettavien esteiden kuluttamalla ajalla. Kaupunkien keskinäiset etäisyydet voidaan laskea kolmiomatriisiin, mikäli kaupunkeja on vähän. Etäisyyksiä voidaan myös laskea silloin, kun niitä tarvitaan ohjelman suorittamisen aikana.

Tarkoituksena on muodostaa mahdollisimman lyhyt reitti, millä kauppamatkustajat käyvät yhdessä kaikissa kaupungeissa. Tämä on myös tarkoitus pystyä laskemaan mahdollisimman nopeasti mahdollisimman hyvällä tarkkuudella. Tarkastelen muunnosta traditionaalisesta mTSP:sta; kauppamatkustajien ei tarvitse palata alkukaupunkiinsa (mTSPNR, Multiple Salesman Problem No Return).

Tämä voidaan asettaa lineaariseksi optimointitehtäväksi:

$$\min \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{1.1}$$

$$\text{s. e.} \sum_{i=1}^{n} x_{ij} = 1,\ j = 2,...n, \tag{1.2}$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 2,...,n, \tag{1.3}$$

$$\sum_{j=2}^{n} x_{a_j j} = m, \tag{1.4}$$

$$+ \text{ Osapolkujen poistamisrajoitukset}, \tag{1.5}$$

$$x_{ij} \in 0, 1, \forall (i,j) \in A, \tag{1.6}$$

Jossa n on kaupunkien lukumäärä ja m kauppamatkustajien lukumäärä. Muuttuja x_{ij} saa arvon 1 mikäli reitti i:stä j:n on olemassa ja 0 mikäli ei ole. Lausekkeet (1.2), (1.3) ja (1.6) ovat sijoitusehtoja. Lauseke (1.4) takaa, että kauppamatkustajia lähtee liikenteeseen maksimissaan m kappaletta. Muuttuja a_j kuvastaa mistä kaupungista kukin kauppamatkustaja lähtee. Lausekkeessa (1.5) on osapolkujen poistamisrajoitukset (Subtour Elimination Constraint, SEC), joita on mahdollista kirjoittaa useita erilaisia. Ne pakottavat polut alkamaan kunkin kauppamatkustajan alkukaupungista. A on joukko, jossa on kaikki yksittäiset reittipalat (matkat yhdestä kaupungista toiseen).

2 Menetelmät

Monen kauppamatkustajan ongelmalla on useita ratkaisutapoja kuten, kombinatorinen, geneettinen ja lineaarinen optimointi. Käsittelen ongelmaa vain kombinatorisen huutokaupan näkökulmasta eli konstruktiivisella heuristiikalla. Konstruktiivinen heuristiikka on heuristiikka, joka rakentaa kerralla monta ratkaisua ongelmaan. On myös olemassa iteratiivisia menetelmiä, jotka antavat ensin yhden ratkaisun ja sitten parantavat sitä. Nämä iteratiiviset menetelmät ovat kirjallisuudessa suositumpia, sillä niiden muistinkulutus on huomattavasti pienempää.

Kauppamatkustajat käyttävät kombinatorista huutokauppaa asettamalla huutoja järjestelmään. Kauppamatkustajat asettavat ehdotuksen omalle reitille ja asettavat huudon arvoksi matkan pituuden. Kun kaikki kauppamatkustajat ovat asettaneet huutonsa, järjestelmä pyrkii valitsemaan näistä voittavat huudot niin, että kaikissa kaupungeissa käydään ja kukin kauppamatkustaja voi voittaa enintään yhden huudon. Kauppamatkustaja voi myös hävitä kaikki huutonsa ja tällöin kauppamatkustaja pysyy paikallaan.

Esimerkki 1:

1. Valitaan aloitustilanteeksi kaksi kaupunkia ($n = 2$) ja kaksi kauppamatkustajaa ($m = 2$).
2. Kauppamatkustaja 1 asettaa kaikki mahdolliset reittivalinnat järjestelmään, eli kauppamatkustaja 1 asettaa huudot $\{1\}$, $\{2\}$, $\{1, 2\}$, $\{2, 1\}$.
3. Kauppamatkustaja 2 asettaa vastaavasti samat huudot: $\{1\}$, $\{2\}$, $\{1, 2\}$, $\{2, 1\}$.
4. Kaikissa huudoissa huudon arvona on euklidinen etäisyys reitin pisteiden välillä. Tämä saadaan hintamatriisista c_{ij} kaupunkien välillä ja matriisista cm_{ij} kauppamatkustajasta kaupunkiin.
5. Voittajaksi valitaan kauppamatkustajien huutojen unioni, jonka yhteenlaskettu arvo on pienin ja näin saadaan määritettyä lyhin reitti, jota kauppamatkustajien tulisi kulkea.

Ongelma kombinatorisissa huutokaupoissa on se, että kun kauppamatkustajat asettavat kaikki huudot, huutoja tulee runsaasti. Huutojen lukumääräksi tulee

$$H = \sum_{i=1}^{n} \frac{n}{(n-i)!} m. \tag{2.1}$$

Huutojen määrää on jotenkin vähennettävä, jotta ongelma saataisiin laskettua rajallisessa ajassa.

Käyttämäni ratkaisualgoritmi koostuu muutamasta eri vaiheesta, jotka suoritetaan peräjälkeen. Kunkin vaiheen sisällä ongelma voitaisiin jakaa usealle suorittimelle, tai seuraavan vaiheen laskenta aloittaa, kun osa edellisestä vaiheesta on laskettu. Menetelmää käytettäessä tulee valita, käyttääkö enemmän muistia vai prosessoritehoa ongelman ratkaisuun.

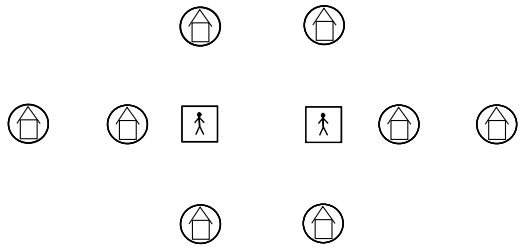
Ongelma ratkaistaan kombinatorisella huutokaupalla seuraavilla vaiheilla:

1. Alustus
2. Kauppamatkustajien huutojen asetus
3. Puun lehtien luominen
4. Puurakenteen ratkaiseminen

2.1 Alustus

Tässä vaiheessa voidaan laskea joitain alkutietoja sijainneista, kuten laskea kaikki mahdolliset etäisyydet matriisiin tai pienin virittävä puu. Kuvassa 2.1 on esimerkkitilanne, jossa on muutama kaupunki sekä kaksi kauppamatkustajaa.

Kuva 2.1: Alkutilanne, kun kaupunkeja on kahdeksan ja kauppamatkustajia on kaksi.



2.2 Kauppamatkustajien huutojen asetus

Ensimmäisessä vaiheessa kauppamatkustajat asettavat huutonsa huutokauppajärjestelmälle. Mitä enemmän huutoja asetetaan, sitä hitaammaksi seuraavat vaiheet tulevat. Tässä vaiheessa pyritään asettamaan huutoja, jotka ovat mahdollisimman hyviä huutoja ja poistamaan huutoja, jotka eivät voi voittaa.

Kauppamatkustajien on turha asettaa samoja kaupunkipermutaatioita huudoista, sillä tiedetään, että vain yksi näistä permutaatioista voi voittaa. Yksittäinen kauppamatkustaja voi selvittää, mikä hänen kaupunkipermutaatioista on paras. Kauppamatkustaja joutuu siis laskemaan TSP:n jokaista kaupunkikombinaatiota kohden. Näin saadaan vähennettyä huudot määrään

$$H = \sum_{i=1}^{n} \frac{n}{(n-i)!i!} m = 2^n m. \tag{2.2}$$

Esimerkissä 1 Kauppamatkustajan 1 ei siis tarvitsisi asettaa molempia huutoja $\{1, 2\}$ ja $\{2, 1\}$, vaan vain katsoa kumpi huudoista on parempi ja valita se. Mikäli toinen huuto on halvempi kuin toinen, ja se sisältää samat kaupungit kuin toinen, sanotaan, että toinen huuto suoraan dominoi edellistä. Dominoituja huutoja ei tarvitse syöttää järjestelmään, sillä määritelmän mukaan on silloin olemassa parempi huuto. Saattaa myös syntyä tilanteita, joissa toinen huuto on halvempi ja sisältää vähintään samat kaupungit kuin toinen, eli dominoivia huutoja, jotka sisältävät samat kohteet tai enemmän kuin toinen, mutta on arvoltaan pienempi. Tälläisiä tilanteita ei kuitenkaan voi syntyä, jos huonot kaupunkikombinaatiot on onnistuneesti poistettu. Edellisestä huolimatta huutojen määrä kasvaa kaupunkimäärän suhteen vieläkin eksponentiaalisesti. Huutojen määrää on vähennettävä, jotta ongelma voidaan ratkaista suurilla kaupunkimäärillä.

2.2.1 Yhden kauppamatkustajan ongelma

Jokaiselle huutokombinaatiolle, jossa on kaupunkeja mukana enemmän kuin yksi on laskettava yhden kauppamatkustajan ongelma. Yhden kauppamatkustajan ongelma voidaan formalisoida lineaarisena optimointiongelmana:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \tag{2.3}$$

$$\text{s. e.} \sum_{i=1}^{n} x_{ij} = 1 \ \forall j, \tag{2.4}$$

$$\sum_{j=1}^{n} \sum_{j=1}^{n} x_{ij} = 1 \ \forall i, \tag{2.5}$$

$$+ \text{ Osapolkujen poistamisrajoitukset}, \tag{2.6}$$

$$x_{ij} \in 0, 1, \forall (i, j) \in A, \tag{2.7}$$

2.2.2 Ympärillä olevat kaupungit

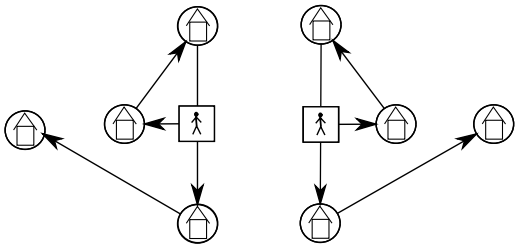
Huutojen määrää voidaan rajoittaa valitsemalla kaupungeista vain osa huutoihin. Valitsemalla kaupunkeja kauppamatkustajan ympäriltä varmistutaan

siitä, että kaupungit ovat lähellä kauppamatkustajaa. Kauppamatkustajan ympärillä olevat kaupungit voidaan valita kahdella tavalla: Valitsemalle ne alkupisteen läheltä tai valitsemalla ne aina edellisen kuljetun kaupungin ympäriltä.

Rajoitetaan huutojen määrää asettamalla kaikki huudot vakiosäteisen ympyrän sisältä. Tällöin jokainen kauppamatkustaja pysyisi pienellä alueella ja valitsisi kohteita, jotka ovat alussa kauppamatkustajan lähellä. Tämä algoritmi tunnetaan nimellä Nearest Starting Point Heuristic (NSPH). Ongelmaksi kuitenkin muodostuu se, että kauppamatkustajat pysyvät vain alkupisteensä läheisyydessä, ja säteen sisällä on kuitenkin iso TSP ratkaistavana. Kuvan 2.1 alkutilanteessa NSPH valitsisi kuvan 2.2 reitin. Joka on kaukana optimaalisesta ratkaisusta, ja joka jopa menee ristiin oman reittinsä kanssa. Kuitenkin mTSP:ssa olisi hyvä, että kauppamatkustajat pysyisivät alkupisteensä lähellä, sillä heidän on palattava myös sinne. Tuloksissa havaitaankin, että ainakin pienillä kaupunkimäärillä tämä algoritmi menestyy hyvin.

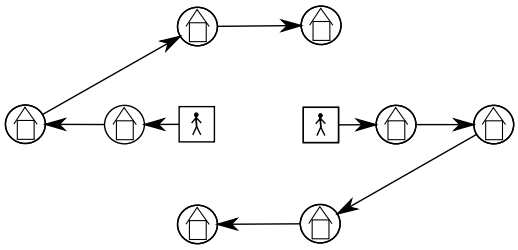
On myös ongelmallista valita ympyrän säde oikein, jos säde kattaa koko yksikköneliön, algoritmi valitsee kaikki mahdolliset huutopermutaatiot. Minimissään säteen on oltava niin pieni, että kaikki kaupungit ovat maksimissaan tämän etäisyyden päästä lähimmästä kauppamatkustajan aloituspisteestä.

Kuva 2.2: NSPH:n valitsema reitti



mTSPNR:ssa on luultavasti parempi, että kauppamatkustaja valitsisi aina lähimmän kaupungin, eikä alkupistettä lähinnä olevan. Näin kauppamatkustaja valitsisi seuraavat kaupungit aina edellisen kaupungin lähipiiristä. Tämä on todennäköisesti järkevä valinta seuraavalle kaupungille. Tällä tavalla huutojen asettamisessa ei tarvitsisi erikseen optimoida kaupunkien läpikäymisjärjestystä, sillä algoritmi valitsee aina lähimpiä kaupunkeja, mikä on luultavasti hyvä reitti. Näin ei myös muodostuisi TSP:ta ratkaistavaksi jokaiselle kauppamatkustajalle. Loppupeleissä ei myöskään valittaisi niin paljon huutoja kuin NSPH:lla. Tämä algoritmi tunnetaan nimellä Nearest Neighbor Heuristic (NNH). Esimerkin alkutilanteessa NNH valitsisi kuvan 2.3 reitin.

Kuva 2.3: NNH:n valitsema reitti alkutilanteeseen



Kun sijoitellaan n kaupunkia satunnaisesti yksikköneliölle ja valitsemalla aina lähin kaupunki, päästään keskimäärin 25 prosenttia pitempään reittiin optimaalisesta [11]. On kuitenkin mahdollista sijoittaa kaupungit niin, että valitsemalla aina lähimmän kaupungin, saadaan huonoin mahdollinen ratkaisu [12].

NNH on ahne algoritmi, eikä se ota huomioon muuta kuin aina seuraavan optimaalisen kaupungin. Algorimi valitsee aluksi lyhyitä reittejä, mutta lopuksi todella huonoja. Tämä johtuu siitä, että algoritmi valitsee ensin lähimmät kaupungit ja sitten kaukaisimmat. Tämä ongelma esiintyy myös NSPH:ssa, kuten nähdään kuvasta 2.2.

NNH:ta käytetään esimerkiksi 2-opt ja 3-opt reittien optimoimiseen, jotka ovat vastaavasti lokaaleja optimaalisia reittejä kahden tai kolmen kaupungin välillä. Niiden laskeminen on optimaalisesti $O(n^k)$, jossa n on kaupunkien lukumäärä ja k lokaalin optimin kaupunkien lukumäärä.

2.2.3 Lin - Kernighanin alfa

Etäisyys ei ole välttämättä paras läheisyysmittari arvioimaan, kuinka hyvä huuto on. Lin - Kernighanin algoritmi [13] on hyvä yhden kauppamatkustajan ongelmaan käytetty heuristiikka. Algoritmissa huudon läheisyytenä käytetään etäisyyden sijasta alfa-etäisyyttä, joka määritellään kaupungista i kaupunkiin j: $\alpha = c_{ij} - MST(j)$ ja kauppamatkustajasta i kaupunkiin j: $\alpha = cm_{ij} - MST(j)$. Funktio $MST()$ on kohteen etäisyys kaupunkien virittävästä pienimmästä puusta ja muuttuja α on huudon etäisyys siitä. Mikäli valittujen huutojen alfa-etäisyyksien summa on nolla, on reitti optimaalisin TSP:ssa, sillä silloin α:n määritelmän mukaan reitti on pienimmällä virittävällä puulla. Alfa-etäisyydellä on siis mahdollista osoittaa, että reitti on paras mahdollinen. On kuitenkin harvinaista saada näiden etäisyyksien summaksi nolla. Kuitenkin mitä pienempi etäisyys on, sitä lupaavampi vaih-

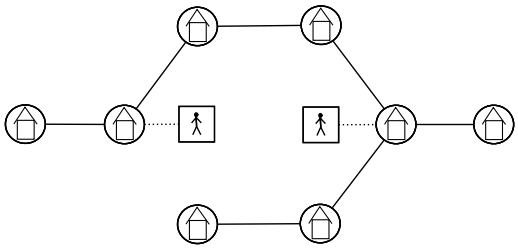
toehto on. mTSP:ssa reitti on myös optimaalinen alfojen summan ollessa nolla.

Kokeellisista tuloksista on havaittu, että $70-80$ prosenttia optimaalisista reiteistä on pienimmällä virittävällä puulla [13], joten α osoittautuu hyväksi läheisyysmittariksi. Etäisyyttä pienimmästä puusta käyttää myöskin hyväkseen Held ja Karp -heuristiikka, joka ei koskaan saavuta huonompaa tulosta kuin $\frac{2}{3}$ optimista euklidisessa ongelmassa [14]. Alfa-etäisyys saattaa osoittautua huonommaksi kuin NNH tilanteissa, joissa MST:ssa esiintyy monta haaraa, sillä silloin ei voida valita reittiä, joka kulkisi pitkään pienimmällä virittävällä puulla.

2.2.3.1 Pienin virittävä puu

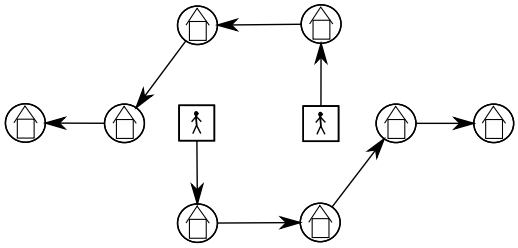
Yhdistämällä kaupungit toisiinsa verkoksi niin, että kustakin kaupungista kulkee yksi yksinkertainen polku toiseen kaupunkiin, saadaan muodostettua kaupunkien virittävä puu. Mikäli puun polkujen yhteenlaskettu summa on pienin mahdollinen, kutsutaan puuta pienimmäksi virittäväksi puuksi. Kuvan 2.1 pienin virittävä puu on kuvassa 2.4. Tarkastaltaessa kuvaa 2.5 huomataan, että tämä on samankaltainen kuin pienimmän virittävän puun reitti.

Kuva 2.4: Pienin virittävä puu kuvan 2.1 aloitustilanteelle



Kauppamatkustajat on liitetty puuhun lehtinä, eikä muut kauppamatkustajat voi kulkea näitä lehtien reittejä pitkin. Puu voidaan laskea Primin algoritmilla, joka aina saavuttaa optimaalisen puun. Puun laskemiseen ei mene kohtuuttomasti aikaa pienillä kaupunkien ja kauppamatkustajien määrillä. Käyttämäni Primin algoritmin implementaation aikakompleksisuus on $O(k^2)$ ja tila $O(k)$.

Kuva 2.5: Alfa-läheisyyttä käyttämällä ratkaistu mTSP tilanteesta 2.1, joka on myös optimaalinen ratkaisu.



2.3 Puun lehtien luominen

Tässä vaiheessa katsotaan yksittäisten kauppamatkustajien asettamia huutoja kokonaisuudessaan. Tiedetään, että vain yksi huuto per kauppamatkustaja on mahdollinen. Voidaan siis asettaa kauppamatkustajien huudot taulukkoon ja aloittaa huutojen läpikäyminen ensimmäisestä kauppamatkustajasta. Valitaan kauppamatkustajan yksi huuto, ja katsotaan seuraavan kauppamatkustajan huudot läpi. Etsitään seuraavan kauppamatkustajan huudoista sellaiset, joissa ei ole samoja kaupunkeja kuin aiemmin valitussa huudossa. Tätä jatketaan niin, että aina seuraavaksi valitussa huudossa ei ole samoja kaupunkeja kuin aikaisemmin valituissa. Näin käymällä läpi kaikki huudot saadaan muodostettua puurakenne, jota voidaan käydä läpi seuraavassa vaiheessa. Tämän algoritmin ongelmaksi muodostuu muistin käyttö. Puuta ei siis kannata tallettaa muistiin, vaan tämä vaihe kannattaa tehdä ratkaisuvaiheessa realiaikaisesti. Kannattaa myös edetä suuntaan, joka vaikuttaa lupaavammalta, esimerkiksi α:n arvion avulla.

Nykyajan ratkaisimet käyttävät useita eri arviointitapoja huudon paremmuuden arvioimiseksi. Erilaisilla arviointitavoilla saadaan muodostettua rajat, joiden sisällä ratkaisujen on pysyttävä. Rajojen laskeminen huudoille on syvyyssuuntaisessa haussa ja branch-and-cut -algoritmissa tärkeässä asemassa.

2.4 Puurakenteen ratkaiseminen

Puurakenteesta optimaalisen reitin voi valita monella eri algoritmilla. Voidaan käyttää esimerkiksi algoritmeja A* tai leveyssuuntaista hakua. Algoritmien yksi ongelma on reitin optimaalisuuden todistaminen tai riittävän

tarkkuuden toteaminen. Voidaan esimerkiksi käydä tietty määrä huutoja läpi ja sopia, että on saavutettu riittävän hyvä tarkkuus. Tällä tavalla voi joskus käydä niin, että etsii parempaa ratkaisua, vaikka paras on jo löydetty tai etsitään liian lyhyen aikaa. Voidaan myös yrittää määrittää joku raja, jonka jälkeen ratkaisu on tarpeeksi hyvä. Joissain tapauksissa voidaan myös todeta, että huuto-kombinaatio on optimaalinen (kuten esimerkiksi α:n tapauksessa) ja voidaan lopettaa ratkaiseminen suoraan. Usein optimaalisuuden todistamiseen käytetään algoritmeissa huomattaviakin ponnistuksia.

2.5 Kauppamatkustajan vakio

On osoitettu [15], että on olemassa kauppamatkustajan vakio $c_1 > 0$, jolle pätee

$$c_1 = \lim_{n\to\infty} \frac{L_n}{\sqrt{n}}, \tag{2.8}$$

jossa L_n on lyhin suljettu polku eli Hamiltonin polku TSP:ssa yksikköneliössä. Vakio c_1 riippuu lähinnä vain alueen muodosta. Kun kaupungit arvotaan satunnaisesti tasaisella jakaumalla yksikköneliöön, voidaan arvioida L_n:n pituutta Beardwood-Halton-Hammersley teoreemalla [15],

$$L_n^* \approx c_1\sqrt{n}. \tag{2.9}$$

Vakiolle c_1 on estimioitu erilaisia arvoja väliltä $0,7124-0,765$ [15-17]. Luvulle saadaan myös laskettua tarkka teoreettinen arvo [18]

$$c_1 = \frac{4(1+2\sqrt{2})\sqrt{51}}{153} = 0,714782... \tag{2.10}$$

Tätä arvoa voidaan käyttää etäisyyksien arviointiin tilastollisesti. Monen kauppamatkustajan ongelmaan ei löydy kirjallisuusviitteitä c_1:n käyttäytymisestä kun kauppamatkustajien määrää lisätään. Vakiota ei ole myöskään ratkaistu tai todettu olevan mTSPNR:ssa.

3 Tulokset

Laskennassa on käytetty toista prosessoriydintä kaksiytimisestä prosessorista Intel(R) Core(TM) 2 Quad CPY Q6600 @ 2,40GHZ ja muistilla Dual DDR2 356,1MHz. Ratkaisin on ohjelmoitu C++ -ohjelmointikielellä, ja kääntäjänä toimii MinGW32. Kaikki ratkaisimet ratkaisevat aina 1000 samaa ongelmaa eri alkuarvoilla. Taulukoissa ja kuvissa näkyvät arvot ovat näiden tuhannen ajon keskiarvoja. Jokaisessa tapauksessa kaikki algoritmit ovat aina saavuttaneet jonkun ratkaisun ongelmaan, eikä ratkaisemattomia ongelmia syntynyt testitapauksissa. Ajan mittauksessa on käytetty WinApin GetTickCount() funktiota ja sen virhe on maksimissaan 16 millisekuntia [19], joten näillä aikamäärillä virhe on olemattoman pieni. Missään testitapauksessa mikään algoritmi ei saavuttanut tietokoneen muistirajoitteita.

3.1 Kauppamatkustajan vakion mittaus

Kuvassa 3.1 on esitetty $L_n/\sqrt{n}$ kauppamatkustajan funktiona. Tästä saamme määritettyä vakion c_1:n arvon yhdeksällä kaupungilla. Pienillä kaupunkimäärillä vakion c_1 arvoksi saadaan $1,035$, joka poikkeaa Johnson, McGeoch ja Rothbergin arvosta $0,7124$ [11] huomattavasti, sillä kaupunkeja on mittauksessa vain yhdeksän. Tässä testissä siis algoritmi laskee kierroksen, eikä TSPNR:aa. Vakio c_1 näyttäisi lähestyvän samaa lukuarvoa eri kauppamatkustajien määrillä. Tämä vaikuttaa loogiselta, sillä kun kaupunkien määrä lähestyy ääretöntä, vakiona pidettävällä kauppamatkustajien lukumäärällä ei ole enää suurta vaikutusta.

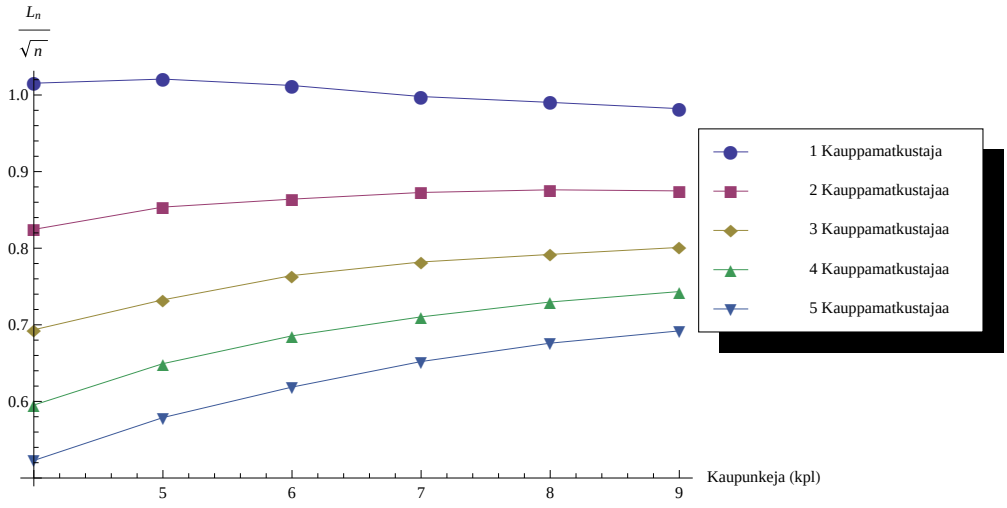
3.2 Heuristiikkojen vertailua

Tarkastellaan kolmea eri huutojen poistotapaa ja verrataan niitä brute-force -ratkaisimeen mTSPNR optimointiongelmassa.

1. Alpha: Sama kuin NNH, mutta käyttää läheisyys mittarina Lin- Kernighanin alfaa.

2. NSPH: Valitsee aloituspistettä lähimmät kaupungit.

Kuva 3.1: Vakio c_1 Näyttäisi lähestyvän samaa vakiota eri kauppamatkustajamäärillä.



3. NNH: Valitsee aina edellisestä lähimmän kaupungin. Algoritmi valitsee aina kolme lähintä kaupunkia jatkaakseen.

4. Brute: Brute-force -algoritmi, joka valitsee kaikki huudot ja ratkaisee ongelman saavuttaen aina optimaalisen ratkaisun.

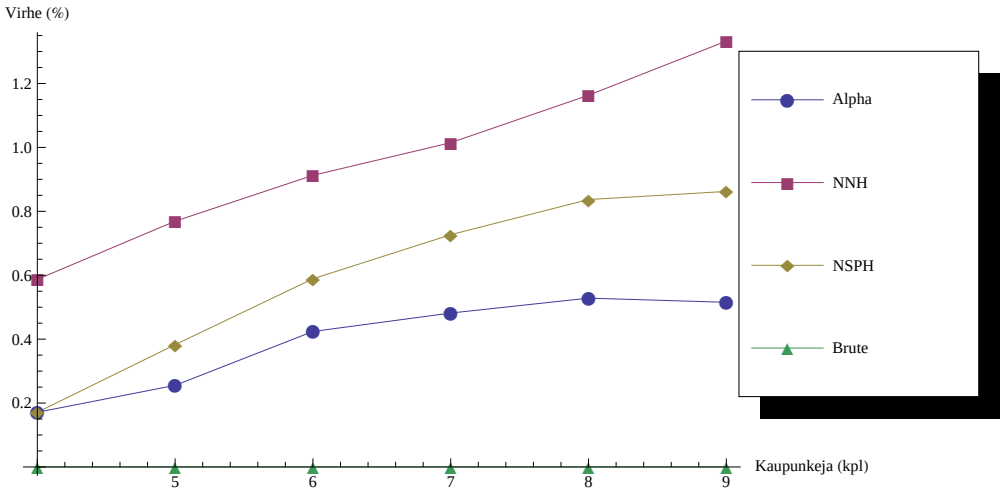
3.2.1 Viisi kauppamatkustajaa

Tarkastellaan ongelmaa viidellä kauppamatkustajalla. Kuvaajassa 3.2 on saatujen tulosten suhteellinen virhe optimista. Alpha menestyy kaikissa tilanteissa paremmin kuin muut algoritmit. Kun kaupunkeja on neljä, Alphalla ja NNH:lla on yhtä pieni virhe. Tapaukset, joissa toiset algoritmit ovat parempia kuin Alpha, ovat harvinaisia (Tuhannen testitapauksen joukossa oli kaksi tapausta, joissa NNH sai paremman tuloksen kuin Alpha). NSPH menestyy muita algoritmeja kehnommin, ja kun kaupunkien lukumäärä kasvaa, sen virhe kasvaa nopeammin kuin muiden.

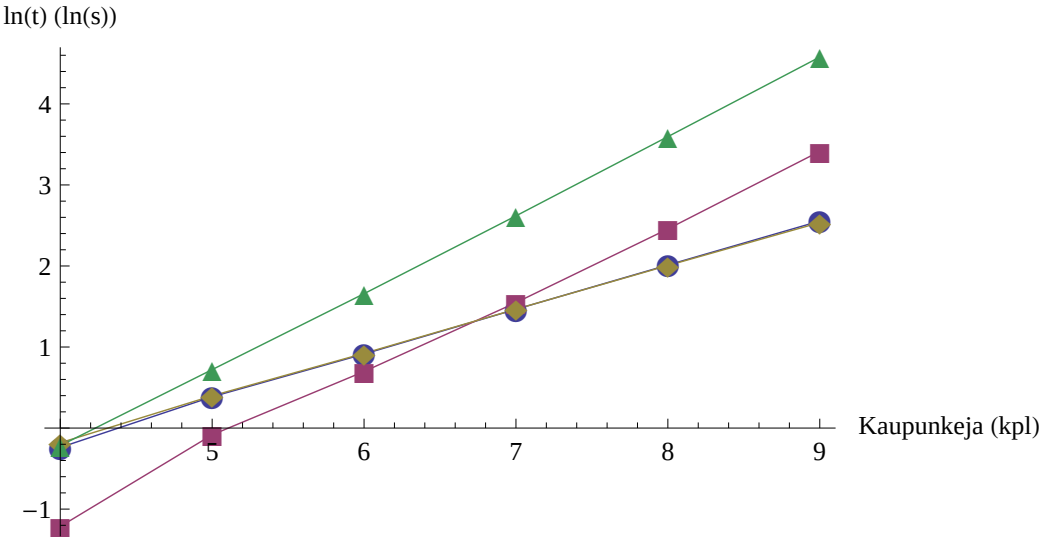
Kuvaajassa 3.3 kuvataan samoja algoritmeja samoilla symboleilla kuin kuvaajassa 3.2 ja seuraavissa kuvaajissa käytetään myös näitä symboleita.

Kuvaajasta 3.3 nähdään algoritmien kuluttama aika logaritmisella asteikoilla. Algoritmit Alpha ja NNH käyttävät suunnilleen yhtä paljon aikaa. Tämä

Kuva 3.2: Suhteellinen virhe viidellä kauppamatkustajalla.



Kuva 3.3: Ajan käyttö viidellä kauppamatkustajalla

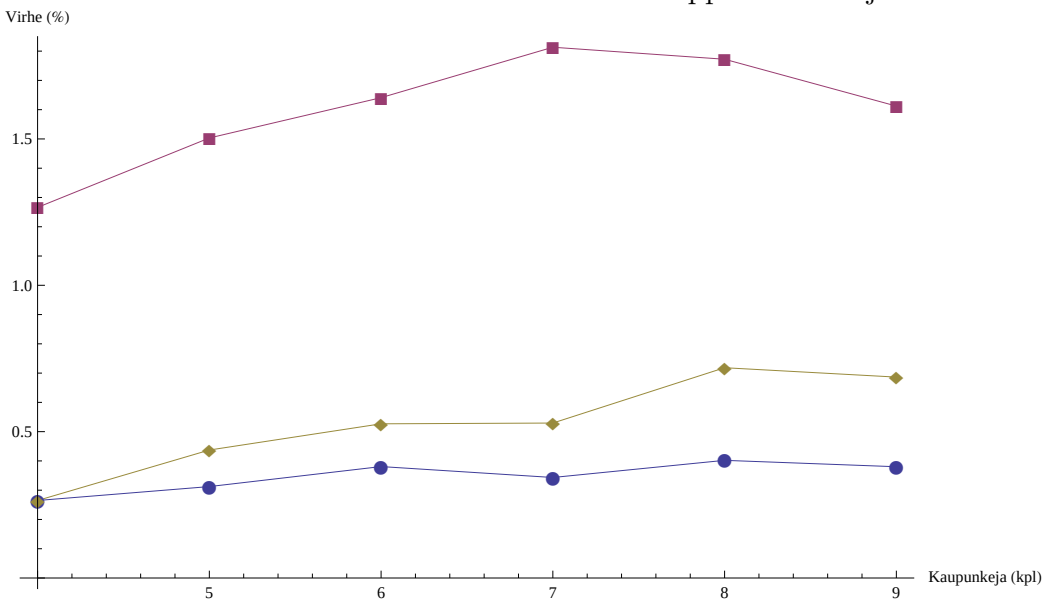


johtuu siitä, että algoritmit eroavat vain hieman toisistaan. Alussa Alpha-algoritmissa lasketaan pienin virittävä puu ja α matriisi, mutta tämän jälkeen algoritmit toimivat samalla tavalla. Alun laskemiseen käytetty aika on siis hyvin pieni, joka ei näy selvästi kuvaajissa. NSPH käyttää alle seitsemällä kaupungilla vähemmän aikaa kuin muut heuristiikat. Koska muut heuris-

tiikat valitsevat aina kolme kaupunkia jatkaakseen, ne päätyvät kuitenkin valitsemaan yhtä monta kaupunkia. On nopeampi siis vain asettaa huudot lähimmistä kaupungeista (kuitenkin NSPH:n virhe on suurempi kuin kahden muun). Kun kaupunkien lukumäärää kasvatetaan, NSPH käyttää enemmän aikaa kuin kaksi muuta algoritmia. Kaikki algoritmit käyttävät eksponentiaalisen ajan ratkaistakseen mTSPNR:n, tämä on suurin ongelma kombinatorisen huutokaupan tavassa ratkaista ongelma. Kuitenkin tekemällä monimutkaisempia algoritmeja tästä ongelmasta päästäisiin eroon.

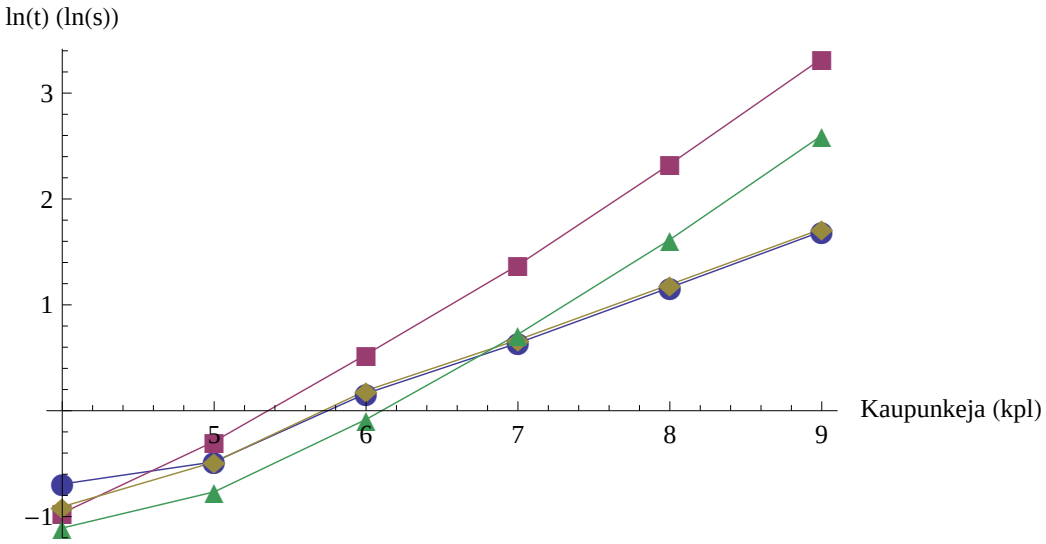
3.2.2 Kaksi kauppamatkustajaa

Kuva 3.4: Suhteellinen virhe kahdella kauppamatkustajalla



Tarkastellaan samaa ongelmaa, mutta vain kahdella kauppamatkustajalla. Kuvassa 3.4 on näkyvissä suhteelliset virheet. Algoritmit menestyvät kaikki yhtä hyvin kuin viidellä kauppamatkustajalla: Virhe-erot ovat kuitenkin suurempia, eikä kuvaajista näy niin suuria virhe-eroja. Viidellä kauppamatkustajalla ja alle kymmenellä kaupungilla kaikille ei aina riitä kaupunkeja vierailtavaksi, jolloin kauppamatkustajat käyvät usein vain yhdessä kaupungissa. Kahdella kauppamatkustajalla yksittäiset kauppamatkustajat joutuvat vierailemaan useammassa kaupungissa. Tämän vuoksi NSPH menestyy huomattavasti huonommin; se joutuu asettamaan reittejä eikä vain sijoittelemaan yksittäisiä kaupunkeja kauppamatkustajille.

Kuva 3.5: Ajan käyttö kahdella kauppamatkustajalla



Ajankäyttökuvaaja 3.5 on huomattavasti erilaisempi kahdella kauppamatkustajalla kuin viidellä. Alpha-algoritmi kuluttaa alussa enemmän aikaa kuin muut algoritmit, sillä se joutuu muodostamaan pienimmän virittävän puun. Tästä johtuen neljällä kaupungilla se kuluttaa enemmän aikaa kuin muut kolme algoritmia. Kun kaupunkeja lisätään, algoritmi kuluttaa suunnilleen yhtä paljon aikaa kuin NNH.

Kun kaupunkeja on alle seitsemän, brute-force -algoritmi kuluttaa vähemmän aikaa kuin kaikki muut algoritmit. Tämä johtuu siitä, että brute-force -algoritmi ei joudu vertailemaan erilaisia läheisyysmittareita toisiinsa vaan luettelee kaikki mahdolliset vaihtoehdot läpi. Tähän kuitenkin kuluu sitä enemmän aikaa, mitä enemmän on kaupunkeja. Kun kaupunkien määrä kasvaa brute-force -algoritmi huononee.

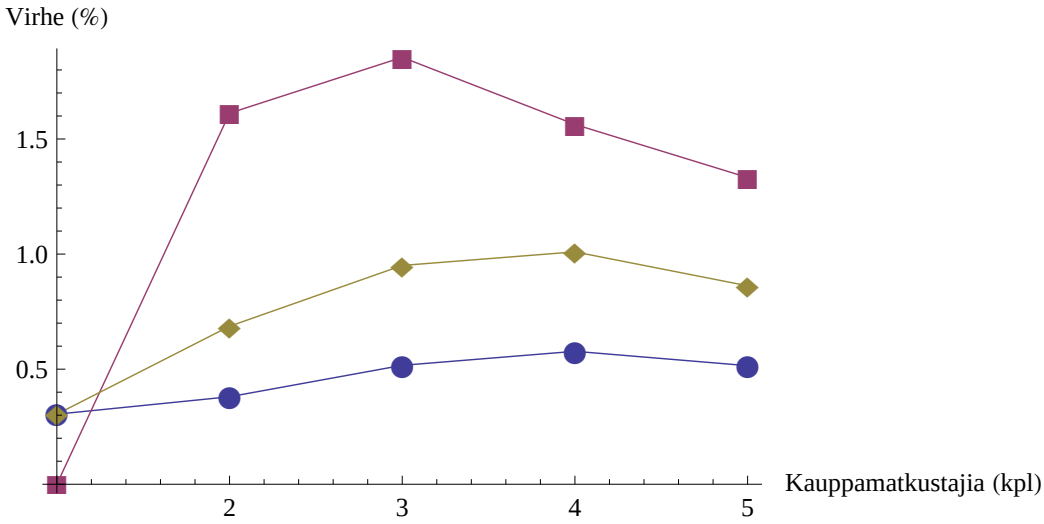
NNH menestyy brute-force -algoritmia huonommin kaikissa tapauksissa. Tämä johtuu siitä, että NNH jakaa koko ongelman kahteen TSPNR:ään ja sen jälkeen yhdistää ne. On nopeampi ratkaista ongelma mTSPNR:na alusta loppuun kuten brute-force -algoritmi tekee.

3.2.3 Kauppamatkustajien lukumäärän vaikutus

Tarkastellaan tilannetta yhdeksällä kaupungilla ja muuttamalla kauppamatkustajien lukumäärää (kuva 3.6). Kahden kauppamatkustajan kohdalla ta-

pahtuu NSPH -algoritmissa suuri hyppäys, sillä algoritmi valitsee sopivan toimintasäteen niin, että kaikki kaupungit tulevat jonkun kauppamatkustajan piiriin. Kahdella kauppamatkustajalla mTSPNR jakautuu siis kahdeksi TSPNR:ksi. Yhdellä kauppamatkustajalla algoritmi toimii kuten brute-force -algoritmi, sillä toimintasäteen on yllettävä kaikkiin kaupunkeihin. Huonoin tilanne NSPH:lla on kolmella kauppamatkustajalla, kun kaupunkeja on yhdeksän. Tilanne on sama myös kuudella kaupungilla. Kaikki algoritmit kuitenkin hyötyvät kauppamatkustajien määrän lisäämisestä, sillä silloin kauppamatkustajalle asetetaan vain muutama kaupunki.

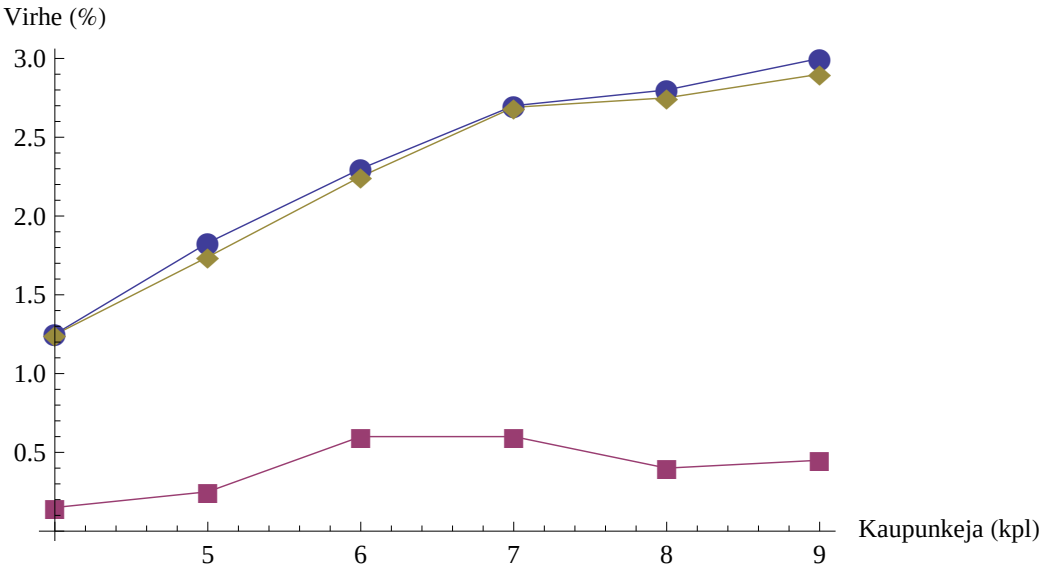
Kuva 3.6: Virhe yhdeksällä kaupungilla



3.2.4 Heuristiikat perinteisessä ongelmassa

Perinteisessä TSP:ssa algoritmit suoriutuvat täysin päinvastaisesti. Kuvasta 3.7 havaitaan, kuinka algoritmien suoriutuminen on täysin toisenlaista kuin mTSPNR:ssa. NSPH menestyy parhaiten, mutta se myös käyttää enemmän aikaa kuin muut algoritmit. Muuttamalla parametreja saadaan NSPH käyttämään vähiten aikaa ja menestymään parhaiten tässä testissä. Tulokset on tehty vain pienillä kaupunkimäärillä, ja tulokset isommilla määrillä voisivat olla erilaiset.

Kuva 3.7: Suhteellisten virheiden vertailua normaalissa mTSP:ssa



4 Yhteenveto

Yhden kauppamatkustajan vakio yksikköneliöllä vaikuttaisi olevan sama kuin monen kauppamatkustajan vakio. Tätä kuitenkin tutkittiin vain muutamalla kaupungilla ja asiaa pitäisi tutkia enemmän useammalla kaupungilla. Pienillä kaupunkimäärillä Lin-Kernighanin alfaan perustuva (Alpha -algoritmi) etäisyyden mittaus mTSPNR:ssa osoittautui parhaimmaksi huutojen poistamismenetelmäksi. Kuitenkin mTSP:ssa etäisyyden mittaus alkupisteeseen (NSPH -algoritmi) menestyi parhaiten. Tutkitut algoritmit vähensivät huutojen määrää, mutta huutojen määrä oli kuitenkin vielä eksponentiaalinen kaupunkien määrän suhteen. Huutojen eksponentiaalisen luonteen poistaminen kaipaisi jatkotutkimusta.

Kirjallisuutta

[1] Fischer R& Richter K (1982). Solving Multi-Objective Traveling Salesman Problem by Dynamic Programming, Mathematicsche operations forschung und Statistic. Series Optimization, 13(2): 247-252.

[2] Sandholm T (2002). Algorithm for optimal winner determination in combinatorial auctions. Artificial Intelligence, 135(1-2): 1-54

[3] IBM. IBM ILOG CPLEX CPOptimizer Specifications (2010). Saatavissa:`http://public.dhe.ibm.com/common/ssi/ecm/en/wsd14044usen/WSD14044USEN.PDF` (viitattu 14. Elokuuta 2011)

[4] Sandholm T, Suri S, Gilpin A & Levine (2001). D.CABOB: A fast optimal algorithm for combinatorial auctions. Proceedings of IJCAI, 1102-1108.

[5] Yan Z, Zhang L, Kang L & Lin G (2003). A new MOEA for multiobjective TSP and its convergence property analysis. Proceedings of Evolutionary Multi-Criterion Optimization, 342-354.

[6] Alsheddy A & Tsangy E (2010). Guided Pareto Local Search and its Application to the 0/1 Multi-objective Knapsack Problems. Proceedings of the eighth metaheuristic international conference, 1--8. Saatavissa:`http://www.essex.ac.uk/csee/research/publications/technicalreports/2010/ces-505.pdf` (viitattu 14. Elokuuta 2011)

[7] Paquete L & Stützle T (2003). A Two Phase Local Search for biobjective Traveling Salesman Problem. Proceedings of the 2nd international conference on Evolutionary multi-criterion optimization, 479-493. Saatavissa:`http://www.essex.ac.uk/csee/research/publications/technicalreports/2010/ces-505.pdf` (viitattu 14. Elokuuta 2011)

[8] Arora S (1998). Polynomial Time Approximation Schemes for Euclidean Traveling Salesman and other Geometric Problems. Journal of the Association for Computing Machinery, 45(5).

[9] Laporte G & Nobert Y (1980). A cutting planes algorithm for the m-salesmen problem. Journal of the Operational Research Society, (31): 10171023.

[10] Bektas T (2006). The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega, 34(209): 209219.

[11] Johnson & McGeoch. (1997). The traveling salesman problem: A case study in local optimization. Local Search in Combinatorial Optimization, 215-310.

[12] Gutin G, Yeo A & Zverovich A(2002). Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. Discrete Applied Mathematics, (117): 81-86.

[13] Helsgaun K (1998). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. European Journal of Operational Research, 126(1): 106130

[14] Held M & Karp M (1970). The Traveling Salesman Problem and Minimum Spanning Trees. Operations Research, 18(6): 1138-1162.

[15] Beardwood J. Halton H & Hammersley M (1959). The Shortest Path Through Many Points. Proceedings of the Cambridge Philosophical Society, 55(4): 299.

[16] Johnson S, McGeoch A, & Rothberg E (1996). Asymptotic Experimental Analysis for the Held-Karp Traveling Salesman Bound. Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms, 341-350.

[17] Percus G & Martin C (1996). Finite Size and Dimensional Dependence in the Euclidean Traveling Salesman Problem. Physical Review Letters, 76(8): 1188-1191.

[18] Normann G & Moscato P (1995). The Euclidean Traveling Salesman Problem and a Space-Filling Curve. Chaos Solitons Fractals, (6): 389-397.

[19] MSDN Kirjaston dokumentaatio GetTickCount funktiosta (2011). Saatavissa:`http://msdn.microsoft.com/en-us/library/ms724408(v=vs.85).aspx` (viitattu 14. Elokuuta 2011)