Kauppamatkustajan ongelman approksimointialgoritmin suunnittelu, toteutus ja kokeellinen tutkimus

Aaro Tuomisto

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi

Pro gradu -tutkielma

Ohjaaja: Martti Juhola

Huhtikuu 2007

Tampereen yliopisto
Tietojenkäsittelytieteiden laitos
Tietojenkäsittelyoppi
Aaro Tuomisto: Kauppamatkustajan ongelman approksimointialgoritmin suunnittelu, toteutus ja kokeellinen tutkimus
Pro gradu -tutkielma, 81 sivua
Huhtikuu 2007

Tässä tutkielmassa kuvataan uusien, tutkielmaa varten kehitettyjen, kauppamatkustajan ongelman approksimointialgoritmien rakennetta, toteutusta ja kokeellista tutkimusta. Lisäksi tutkielma luo katsauksen itse ongelman historiaan ja tutkimukseen. Uusia algoritmeja tutkitaan vertaamalla niitä jo olemassa olevaan konveksia verhoa hyödyntävään lisäysalgoritmiin. Testimateriaalina kokeellisessa tutkimuksessa on käytetty internetistä löytyviä pistesarjoja, jotka ovat yleisesti käytössä kauppamatkustajan ongelman tutkimuksessa.

Kokeellisen tutkimuksen tuloksena saatiin selville, että kehitettyjen algoritmien perusidea on toimiva ja että kehitetyt algoritmit ovat vertailukohtaansa nähden huomattavan nopeita. Lisäksi saatiin selville, että uusien algoritmien antama reitti ei ollut kovin paljon pidempi vertailukohtana käytetyn algoritmin antamaan reittiin verrattuna.

Avainsanat ja -sanonnat: kauppamatkustajan ongelma, approksimointialgoritmi, kokeellinen tutkimus.

Sisällys

1 Johdanto

Tässä tutkielmassa luodaan katsaus kauppamatkustajan ongelman historiaan ja teoriaan ja esitellään sille itse kehitettyjä uusia approksimointialgoritmeja. Kehitetyille algoritmeille suoritetaan kokeellinen tutkimus, jossa niitä verrataan toisiinsa ja jo olemassa olevaan approksimointialgoritmiin.

Kauppamatkustajan ongelmalla voidaan hyvällä syyllä sanoa olevan monta isää, kuten tulemme myöhemmin huomaamaan. Merrill M. Floodin [1956, p. 61] mukaan ongelman esitti, siinä muodossa kuin se nykyisin tunnetaan, Hassler Whitney vuonna 1934. Flood myös toteaa samassa artikkelissa, että ongelmassa on kyse sellaisen järjestyksen $P = (1, i_2, i_3, ..., i_n)$ löytämisestä, joka minimoi summan $a_{1i_2} + a_{i_2 i_3} + a_{i_3 i_4} + ... + a_{i_n 1}$, missä muuttujat $a_{\alpha\beta}$ kuuluvat annettuun ei-negatiivisten reaalilukujen joukkoon. Flood lisää, että nimitys kauppamatkustajan ongelma tulee siitä, että kauppamatkustaja voisi hyödyntää ongelman tehokasta ratkaisua valitessaan reittiä, jota pitkin kotikaupungistaan lähtien kiertäisi tietyt kaupungit.

Ongelman ratkaisemiseen on pyritty monella tavalla. Lineaarisen optimoinnin näkökulmasta ongelmaa lähestyvät sekä Dantzig ja muut [1959] että Miller ja muut [1960], jotka artikkelissaan käsittelevät myös cutting-plane -menetelmää eli leikkaavien tasojen menetelmää hyödyntävää algoritmia. Bellman [1962] pyrkii lähestymään ongelmaa dynaamisen ohjelmoinnin keinoin eli pilkkomalla ratkaistavan tapauksen pienempiin osaongelmiin, joiden ratkaisut sitten yhdistetään. Samaa dynaamisen ohjelmoinnin lähestymistapaa käyttävät myös omalla tahollaan Held ja Karp [1961]. Little ja muut [1963] esittelevät ongelmaan branch-and-bound -menetelmällä toimivan algoritmin. Balas ja Toth [1987] esittävät kattavan kuvauksen erilaisista branch-and-bound -menetelmistä ja huomauttavat, että myös tässä menetelmässä pyritään ongelman ratkaisua lähestymään helpompien osaongelmien kautta. Bellmore ja Nemhauser [1968, pp. 546–548] mainitsevat ratkaisumenetelmän, jossa jollakin tavalla valittua reittiä parannetaan muuttamalla sen kulkua, ja lisäysalgoritmina tunnetun menetelmän, jossa valittua osareittiä täydennetään uusilla pisteillä. Reittiä vaihdoksilla parantavat algoritmit tunnetaan nykyisin paremmin Linin ja Kernighanin [1973] esittämästä Lin-Kernighan algoritmista. Norback ja Love [1977] puolestaan lähestyvät ongelmaa geometrisesta näkökulmasta ja tutkivat tilannetta, jossa konveksia verhoa käytetään lisäysalgoritmin aloitusreittinä. Sama idea esiintyy myös Goldenin ja muiden [1980] esittämässä konveksia verhoa hyödyntävässä algoritmissa. Viimeisimpinä uutuuksina

erilaisten ratkaisuvaihtoehtojen joukossa ovat neuroverkkoja hyödyntävät algoritmit, joista esimerkkinä voidaan mainita Somhoman ja muiden [1997] julkaisema algoritmi.

Samanaikaisesti ongelmaa varten kehitettyjen laskennallisten menetelmien kehittyessä ovat myös ongelman käsittelyyn käytettävät tietokoneet edistyneet huimasti. Kuvaavaa kehitykselle on se, että siinä missä vuonna 1954 saatiin ratkaistua 49 kaupungin ongelma, kuten Dantzig ja muut [1954] osoittavat, niin vuonna 2004 pystyttiin Applegaten ja muiden mukaan [2007, p. 52] ratkaisemaan jo 24978 Ruotsin kaupunkia ja kylää käsittävä ongelma.

Laskennallisten menetelmien ja laskentaan käytettävien laitteiden ohessa on myös kehitetty menetelmiä optimaalisen ratkaisun arviointiin. Yhtenä esimerkkinä voidaan mainita Nicos Christofidesin [1972] esittelemä ongelman osareittejä hyödyntävä menetelmä optimaalisen reitin vähimmäispituuden laskemiseksi.

Oman tärkeän osansa kauppamatkustajan ongelma tutkimuksessa ja käytännön sovelluksissa muodostavat myös helposti ratkaistavissa olevat erikoistapaukset. Gilmore ja muut [1987, p. 88] mainitsevat triviaalin omaisena esimerkkinä tilanteen, jossa kaikki reitit ovat yhtä pitkiä.

Tämän tutkimuksen kannalta keskeisin algoritmi on konveksia verhoa hyödyntävä lisäysalgoritmi. Konveksin verhon valitsemisen aloitusreitiksi tekee houkuttelevaksi Reineltin [1994, p. 88] huomautus, että kauppamatkustajan ongelman optimiratkaisussa konveksin verhon sisältämien pisteiden on pakko sisältyä ratkaisuun konveksin verhon määräämässä järjestyksessä.

Tutkimuksen kunnianhimoisena tavoitteena oli luoda konveksia verhoa hyödyntävän lisäysalgoritmin pohjalta kokonaan uusi kauppamatkustajan ongelman approksimointialgoritmi ja oikeuttaa tämän uuden algoritmin olemassaolo kokeellisen tutkimuksen avulla. Tutkimuksessa käytetyksi aineistoksi valittiin internetistä löytyvä tiedostojen joukko, jota yleisesti käytetään ongelmaa varten luotujen algoritmien testauksessa.

Luvussa 2 esitetään joitakin tutkimuksen kannalta oleellisia termejä ja käsitteitä.

Luvussa 3 käsitellään kauppamatkustajan ongelman historiaa ja teoriaa ja esitellään sen käytännön sovelluksia. Luvussa perehdytään myös kauppamatkustajan ongelman tutkimukseen.

Luvussa 4 esitellään tutkittavat algoritmit ja sellaiset algoritmit ja menetelmät, joita tarvitaan tutkittavien algoritmien toteutuksissa. Luku painottuu omien algoritmien toiminnan ja rakenteen esittelyyn. Luvussa pyritään kuvaamaan omat algoritmit ja

niiden vertailukohtana toimiva konveksia verhoa hyödyntävä lisäysalgoritmi siten, että algoritmien toteutus olisi annettujen tietojen pohjalta mahdollista.

Luvussa 5 kuvataan kokeellisen tutkimuksen tavoitteet ja käytännön toteutus. Ensin luvussa käsitellään kokeellisen tutkimuksen tavoitteet ja sitten testimateriaalin rakenne, tutkimuksessa mitattavat suureet, tulosten kirjaamismenetelmä ja tuloksille tehtävät tarkistustoimenpiteet.

Luvussa 6 esitetään kokeellisen tutkimuksen tulokset ja suoritetaan algoritmien välinen vertailu. Vertailu tehdään konveksia verhoa hyödyntävästä lisäysalgoritmista tehtyä toteutusta vasten. Luvussa keskitytään algoritmien nopeuden ja tarkkuuden tarkasteluun.

Luvussa 7 tehdään johtopäätöksiä kokeellisen tutkimuksen avulla saaduista tuloksista ja esitetään näiden johtopäätöksien perusteella arvio omien algoritmien kantavana voimana toimineen idean hyödyllisyydestä ja käyttökelpoisuudesta.

Luvussa 8 tehdään yhteenveto tutkimuksesta ja esitetään oma arvio sen onnistumisesta.

2 Termejä ja käsitteitä

Tässä luvussa käydään läpi joitakin oleellisia termejä ja käsitteitä.

Työ

Termillä työ viitataan jatkossa tähän pro gradu -tutkielmaan.

Kustannusfunktio

Kustannusfunktiolla tarkoitetaan tämän työn yhteydessä yleisesti menetelmää, jolla arvioidaan johonkin reittiin tehtävän muutoksen hyödyllisyyttä. Tämän työn yhteydessä toteutetuissa algoritmeissa kustannusfunktio laskee, kuinka paljon kahden pisteen välinen matka pitenee, jos kuljetaan jonkin kolmannen pisteen kautta.

Algoritmi

Algoritmille ei ole yleisesti hyväksyttyä määritelmää. Normaalisti algoritmin katsotaan tarkoittavan joukkoa ristiriidattomia käskyjä, joiden avulla ratkaistaan jokin ongelma. Tämän työn yhteydessä algoritmilla voidaan viitata myös yleisempään ohjeeseen, jossa jokin kohta on jätetty määrittelemättä tarkemmin. Määritellyssä algoritmissa saatetaan esimerkiksi kehottaa käyttämään jotakin itse valittua tapaa seuraavan pisteen valinnassa, mutta ei tarkemmin määritellä tuota tapaa.

Heuristiikka

Heuristiikka on algoritmi, joka luopuu täsmälleen oikean ratkaisun tavoittelusta, jotta saavutetaan nopeampi suoritusaika. Sanaa heuristiikka ei käytetä tässä työssä, vaan se on korvattu sanalla algoritmi silloinkin, kun se esiintyy lähdemateriaalissa.

Approksimointialgoritmi

Approksimointialgoritmi antaa jollekin tietylle ongelmalle jonkin ratkaisun. Ratkaisu ei yleensä ole paras mahdollinen.

Deterministinen algoritmi

Deterministisellä algoritmilla tarkoitetaan algoritmia, joka toimii aina samalla syötteellä samalla tavalla.

Epädeterministinen algoritmi

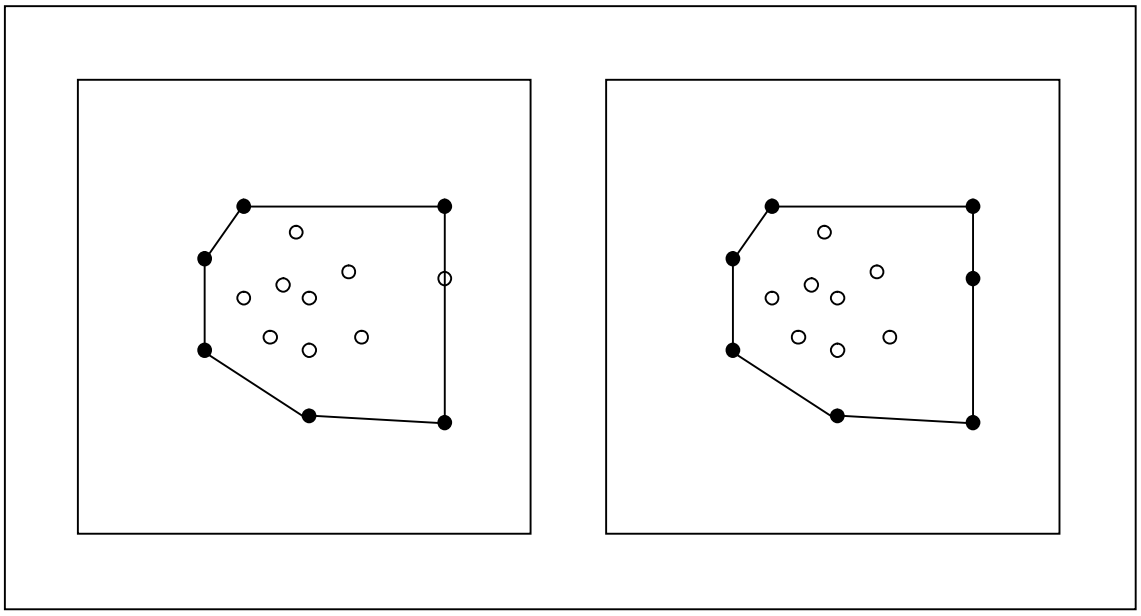
Epädeterministisellä algoritmilla tarkoitetaan algoritmia, joka voi toimia samalla syötteellä eri tavalla eri ajokertoina.

Referenssipiste

Referenssipisteellä tarkoitetaan tämän työn yhteydessä pistettä, jonka suhteen jokin operaatio tehdään.

Konveksi verho

Orponen ja Ernvall [2004, s. 85] esittävät konveksille verholle seuraavan määritelmän. Pistejoukon $Q = \{p_1, ..., p_n\}$ konveksi verho on suppein konveksi monikulmio, joka sisältää kaikki *Q*:n pisteet joko sisällään tai reunallaan. Tämän työn yhteydessä konveksin verhon käsitettä laajennetaan siten, että konveksiin verhoon määritellään kuuluvaksi kaikki kyseiselle monikulmiolle osuvat pisteet, eivätkä vain sen kärkipisteet. Kuva 1 havainnollistaa konveksin verhon käsitettä ja sen laajentamista.

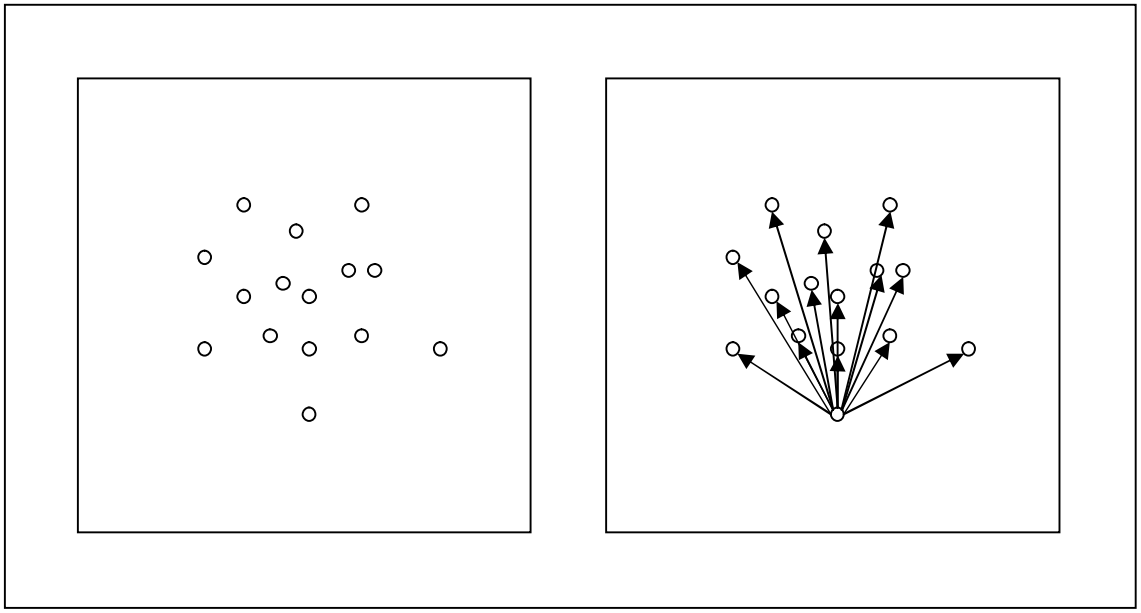


Kuva 1. Oikean puoleisessa esimerkissä konveksia verhoa on laajennettu mainitulla tavalla.

Pistejoukon kulmajärjestys

Pistejoukon kulmajärjestyksellä tarkoitetaan pisteiden kulmajärjestystä jonkin referenssipisteen suhteen. Kuva 2 esittää kulmajärjestystä, joka on tehty käyttäen referenssipisteenä pienimmän y-koordinaatin arvon omaavaa pistettä. Kuvan

kulmajärjestys tulkitaan siten, että oikeapuoleisin nuoli osoittaa pienimmän kulma-arvon omaavaan pisteeseen ja vasemmanpuoleisin suurimman kulma-arvon omaavaan pisteeseen.



Kuva 2. Pistejoukolle on tehty kulmajärjestys pienimmän y-koordinaatin arvon omaavan pisteen suhteen.

Algoritmin läpimenoaika

Algoritmin läpimenoajalla tarkoitetaan aikaa, joka algoritmilta kuluu ongelman alkutilanteesta vastauksen muodostamiseen.

Suure

Termillä suure tarkoitetaan jotakin mitattua algoritmin ominaisuutta, kuten esimerkiksi läpimenoaikaa.

Johdettu suure

Termillä johdettu suure tarkoitetaan jotakin sellaista arvoa, joka saadaan mitatuista suureista laskemalla.

Esiintymä

Esiintymällä tarkoitetaan algoritmille annettavaa syötettä, joka sisältää jonkin tietyn ongelman ilmentymän.

TSP

Kauppamatkustajan ongelmasta käytetään usein lyhennettä TSP, joka tulee englannin kielen sanoista Traveling Salesman Problem.

Karakteristinen funktio

Karakteristisella funktiolla tarkoitetaan funktiota, joka antaa vastaukseksi joko arvon 1 tai 0. Tietojenkäsittelyssä nämä arvot usein tulkitaan totuusarvoiksi kyllä tai ei.

3 Kauppamatkustajan ongelma

Tämän luvussa selvitetään kauppamatkustajan ongelman historiaa, annetaan sen tarkka verkkoteoreettinen määritelmä ja valaistaan siihen kehitettyjä ratkaisumenetelmiä. Lisäksi luvussa luodaan yleissilmäys kauppamatkustajan ongelman käytännön sovelluksiin. Tavoitteena on tarjota, yhdessä edellisen luvun määritelmien kanssa, lukijalle tarvittava pohjatieto myöhempien lukujen ymmärtämistä varten.

3.1 Historia

Seuraavassa tutustutaan kauppamatkustajan ongelman historiaan.

3.1.1 Termin syntyminen

Kauppamatkustajan ongelman perinteisessä muodossa annettu kysymyksenasettelu on Hoffmanin ja Wolfin [1987, p. 1] mukaan seuraava. Myyntimiehen, joka lähtee kotikaupungistaan, täytyy käydä tietyssä joukossa annettuja kaupunkeja ja palata kotiinsa. Hänelle on edullista valita reitti niin, että matka, jonka hän kulkee käydessään eri kaupungeissa, on mahdollisimman lyhyt. Oletuksena on, että hänellä on tiedossa eri kaupunkien väliset etäisyydet. Millä keinolla hän voi päätellä lyhyimmän mahdollisen reitin?

Hoffman ja Wolf [p. 5] jatkavat, että ensimmäisen kerran termiä kauppamatkustajan ongelma käytettiin matemaattisissa piireissä luultavasti vuosien 1931 ja 1932 välisenä aikana. He kuitenkin samalla lisäävät, että jo vuonna 1832 ilmestyi saksankielinen kirja, jossa käsiteltiin myös tapaa, jolla kauppiaan kannattaisi valita kaupunkien välillä kulkemansa reitti. Kyseinen kirja antoi suosituksen kulkea mahdollisimman monen kaupungin kautta vierailematta kahta kertaa samassa kaupungissa.

Tarkkaa tietoa siitä, kuka toi termin matemaattiseen kieleen, ei Hoffmanin ja Wolfin [p. 5] mukaan ole, mutta he samalla korostavat Merrill Floodin merkitystä termin julkaisemisessa matemaattisessa kirjallisuudessa. Flood [1956, p. 61] itse mainitsee, että ongelman esitti, siinä muodossa kuin se nykyisin tunnetaan, Hassler Whitney vuonna 1934 Princetonin yliopistossa. Hoffman ja Wolf [p. 5] kertovat, että Flood olisi kertonut kuulleensa ongelmasta A. W. Tuckerilta, joka puolestaan oli Floodin muistin mukaan ilmoittanut termin isäksi Whitneyn.

3.1.2 Tutkimus

Erityinen merkitys kauppamatkustajan ongelman tutkimukselle on Hoffmanin ja Wolfin mukaan [p. 6] ollut Dantzigin, Fulkersonin ja Johnsonin artikkelilla ”Solution to Large-Scale Traveling-Salesman Problem”, jonka julkaisemista he pitävät käänteentekevänä hetkenä koko optimointiteorian historiassa. Artikkelissa Dantzig ja muut [1954, p. 406] esittävät ratkaisun 49 kaupungin kauppamatkustajan ongelmaan. Hoffmanin ja Wolfin mukaan artikkelin merkitys ei koostunut pelkästään siinä esitetyistä tuloksista, vaan myös kaikesta siitä tutkimuksesta, jonka sen julkaiseminen inspiroi. Vaikka artikkelissa ei esitettykään suoraan algoritmeja, niin monet myöhemmin kehitetyt ratkaisut ovat saaneet alkusysäyksensä siitä. Hoffman ja Wolf [p. 10] mainitsevat, että branch-and-bound -menetelmän käyttö kauppamatkustajan ongelman ratkaisussa, jonka Little ja muut [1963] esittivät, oli saanut innoituksensa Dantzigin, Fulkersonin ja Johnsonin artikkelista.

Toinen merkittävä edistysaskel tapahtui 1960-luvun lopun ja 1970-luvun alun aikana, jolloin Hoffman ja Wolf [p. 11] kertovat eriytymisen niin sanottuihin koviin ja pehmeisiin ongelmiin tapahtuneen. Tällä tarkoitetaan sitä, että pystyttiin tekemään ero helposti ratkaistavissa olevien erityistapausten ja vaikeiden tapausten välillä. Lisäksi havaittiin, että vaikeat eli *NP*-täydelliset ongelmat pystyttiin muuttamaan toisikseen siten, että ratkaisu yhteen niistä toisi ratkaisun kaikkiin muihinkin. *NP*-täydellisten ongelmien olemassaolon tiedostaminen johti uusien approksimointiin perustuvien ratkaisumenetelmien kehittämiseen. Tärkeimpiä approksimointialgoritmeja, jotka esiteltiin tänä aikakautena, olivat Hoffmanin ja Wolfin [p. 13] mukaan Linin ja Kernighanin esittämät sivujen vaihtoon perustuvat algoritmit. Erityisen merkittäväksi on muodostunut Linin ja Kernighanin [1973] esittämä approksimointialgoritmi, joka nykyisin tunnetaan Lin-Kernighan algoritmina.

Hoffman ja Wolf [p. 13] jatkavat, että approksimointialgoritmien tutkimus johti myös niiden analysointimenetelmien kehittymiseen ja suurimman virheen analysointimenetelmän käyttöönottoon. Suurimman virheen analysoinnilla tarkoitetaan sitä, että selvitetään kuinka suuren virheen algoritmi pahimmillaan tekee. Hoffman ja Wolf [p. 13] mainitsevat vielä erikseen Christofidesin vuonna 1976 esittämän algoritmin, jonka voi todistaa tekevän korkeintaan 50 % virheen. Algoritmi tunnetaan nykyisin Christofidesin algoritmina ja sillä on vieläkin pienin todistettavissa oleva suurimman virheen arvo.

Kolmas merkittävä edistysaskel oli Hoffmanin ja Wolfin [p. 14] mukaan 318 kaupungin kauppamatkustajan ongelman ratkaisu, jonka Crowder ja Padberg esittelivät

vuonna 1980. Crowder ja Padberg [1980] hyödynsivät ratkaisun etsinnässään sekä branch-and-bound -menetelmää että leikkaavien tasojen menetelmää. Tämän ratkaisu valoi uskoa siihen, että isojakin ongelmatapauksia pystyttäisiin ratkaisemaan ja vielä niin, että ratkaisun optimaalisuus voitiin todistaa. Myöhemmin ratkaistujen ongelmien koossa on siirrytty tuhansien kaupunkien määriin, kuten Reinelt [1994, p. 1] kertoo. Hän myös lisää, ettei ratkaistujen ongelmien koon kasvu ole johtunut ainoastaan laskentaan käytettyjen koneiden tehojen kasvusta, vaan myös entistä kehittyneemmistä matemaattisista menetelmistä. Viimeisimpiä merkkipaaluja suurten ongelmien ratkaisuissa on Applegaten ja muiden mukaan [2007, pp. 52–56] vuonna 2004 löydetty 24798 Ruotsin kylän ja kaupungin läpi kulkeva optimireitti ja vuonna 2006 löydetty 85900 pisteen ongelman ratkaisu. Pelkästään ratkaistun ongelman koko ei kuitenkaan määritä tapauksen vaikeutta, vaan myös pisteiden jakaumalla on suuri merkitys. Yleisesti maantieteelliset ongelmat tarjoavat hyviä testitapauksia, sillä niissä pisteiden jakauma muodostuu realistiseksi ja satunnaiseksi.

3.2 Kauppamatkustajan ongelman määritelmä

Kohdassa käydään läpi kauppamatkustajan ongelman tarkka verkkoteoreettinen määritelmä. Tässä yhteydessä rajoitutaan verkkoihin, jotka ovat tasossa.

3.2.1 Verkkoteoreettiset käsitteet

Esitellään tarpeelliset verkkoteoreettiset käsitteet.

Etäisyys

Etäisyys eli etäisyysfunktio kertoo joukon pisteiden välisen etäisyyden.

Euklidinen metriikka

Euklidisella metriikalla tarkoitetaan sitä, että pisteiden $p_1 = (x_1, y_1)$ ja $p_2 = (x_2, y_2)$ välinen etäisyys lasketaan kaavalla $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.

Ei-järjestetty tulo

Joukon S ei-järjestetty tulo itsensä kanssa, merkitään $S \,\&\, S$, on ei-järjestettyjen parien $(s \,\&\, t)$ joukko:

$$S \,\&\, S = \{(s \,\&\, t), s \in S, t \in S, (s \,\&\, t) = (t \,\&\, s) \forall s, t \in S\}.$$ [Savolainen, 2001, s. 7]

Abstrakti verkko

Abstrakti verkko eli graafi muodostuu ei-tyhjästä pisteiden eli solmujen joukosta $V = \{v_i\}$, välien joukosta $E = \{e_j\}$ ja verkkoon liittyvästä kuvauksesta $\Phi : E \to V \,\&\, V$. [Savolainen, 2001, s. 7]

Solmujen vierekkäisyys

Solmut v ja w ovat vierekkäisiä solmuja, jos ne ovat saman välin $e \sim (v \,\&\, w)$ päätesolmut. [Savolainen, 2001, s. 11]

Täydellinen verkko

Täydellisessä verkossa kaikki solmut ovat kaikille muille solmuille vierekkäisiä, toisin sanoen jokaisesta solmusta on väli jokaiseen muuhun solmuun. [Savolainen, 2001, s. 20]

Etäisyys verkossa

Luku c_{ij} voidaan joissakin tapauksissa yhdistää verkon solmujen *i* ja *j* väliin (*i*, *j*) ja tästä luvusta voidaan käyttää nimitystä paino, etäisyys tai kustannus [Christofides, 1975, p. 5]. Tämän työn yhteydessä merkinnästä c_{ij} käytetään nimitystä etäisyys.

Etäisyysmatriisi

Verkkoon, jossa on n solmua, liittyvät etäisyydet voidaan ilmoittaa $(n \times n)$ etäisyysmatriisina C, jossa c_{ij} on solmujen i ja j välinen etäisyys.

Kolmioepäyhtälön voimassa olo täydellisessä verkossa

Olkoot *u*, *v* ja *w* täydellisen verkon solmuja. Seuraava ehto määrää kolmioepäyhtälön voimassaolon täydellisessä verkossa: $c_{uv} \le c_{uw} + c_{wv}$.

Välien jono

Verkon välien äärellinen sekvenssi $e_1, e_2, ..., e_n$ muodostaa välien jonon, jos on olemassa sellainen solmujen sekvenssi eli järjestys $v_0, v_1, ..., v_n$, että $e_i \sim (v_{i-1} \,\&\, v_i)$, $i = 1,2,...,n$. [Savolainen, 2001, s. 12]

Suljettu välien jono

Välien jono on suljettu, jos $v_0 = v_n$. [Savolainen, 2001, s. 13]

Ketjujono

Välien sekvenssi, joka muodostaa välien jonon, on ketjujono, mikäli siinä ei esiinny samaa väliä useammin kuin yhden kerran. [Savolainen, 2001, s. 13]

Suljettu ketjujono

Ketjujono on suljettu, jos sen muodostava välien jono on suljettu. [Savolainen, 2001, s. 13]

Silmukka

Silmukka on suljettu ketjujono. [Savolainen, 2001, s. 13]

Yksinkertainen ketju

Jos ketjun kaikki solmut ovat eri solmuja, ketju on yksinkertainen ketju. [Savolainen, 2001, s. 13]

Yksinkertainen silmukka

Jos ketjun $v_0 = v_n$, mutta solmut ovat muuten eri solmuja, niin välien järjestämätöntä joukkoa kutsutaan yksinkertaiseksi silmukaksi. [Savolainen, 2001, s. 13]

Hamiltonin silmukka

Hamiltonin silmukka on sellainen yksinkertainen silmukka, joka kulkee verkon kaikkien solmujen kautta. [Savolainen, 2001, s. 75]

3.2.2 Kauppamatkustajan ongelman tarkka määritelmä

Verkkoteoreettisesti kauppamatkustajan ongelma määritellään seuraavasti.

Kauppamatkustajan ongelma (TSP)

Etsi täydellisen verkon minimipainoinen Hamiltonin silmukka.

Symmetrinen kauppamatkustajan ongelma

Johnson ja Papadimitriou [1987a, p. 61] määrittelevät symmetrisen kauppamatkustajan ongelman seuraavalla tavalla: Merkitään solmujen *i* ja *j* välistä etäisyyttä merkinnällä c_{ij}. Kauppamatkustajan ongelmaa sanotaan symmetriseksi, mikäli $c_{ij} = c_{ji}$.

Kauppamatkustajan ongelma, jossa on voimassa kolmioepäyhtälö

Johnson ja Papadimitriou [1987a, p. 61] määrittelevät kauppamatkustajan ongelman, jossa on voimassa kolmioepäyhtälö, lyhyesti toteamalla, että siinä kauppamatkustajan ongelma rajoitetaan verkkoihin, joissa on voimassa kolmioepäyhtälö.

Euklidinen kauppamatkustajan ongelma

Johnson ja Papadimitriou [1987a, p. 61] määrittelevät euklidiseksi kauppamatkustajan ongelmaksi sellaisen kauppamatkustajan ongelman, jossa on voimassa symmetrisyyden ja kolmioepäyhtälön lisäksi euklidinen metriikka.

3.3 Käytännön sovellukset ja muunnokset

Tutustutaan kauppamatkustajan ongelman käytännön sovelluksiin ja muunnoksiin.

3.3.1 Käytännön sovellukset

Seuraavassa esitetään joitakin kauppamatkustajan ongelman käytännön sovelluksia. Ongelmat ovat Reineltin [1994, pp. 38–39] kuvaamia.

Ajoneuvojen reititys

Ajoneuvojen reititysongelmalla tarkoitetaan esimerkiksi seuraavaa tehtävää. Kaupungissa on n kappaletta postilaatikkoja, jotka täytyy tyhjentää joka päivä tietyn ajanjakson aikana. Ongelmana on löytää pienin määrä ajoneuvoja, joilla työ onnistuu, ja lyhyin aika, jossa tehtävä voidaan kyseisellä määrällä ajoneuvoja suorittaa.

Töiden jaksotus

Töiden jaksotusongelma voidaan kuvata seuraavasti. Tietyt n kappaletta tehtäviä täytyy suorittaa tietyllä koneella. Tehtävän j suorittamiseen kuluva aika on t_{ij}, missä i on se tehtävä, joka suoritetaan koneella juuri ennen tehtävän j suorittamista. Ongelmana on löytää tehtäville sellainen suoritusjärjestys, että tehtävien suorittamiseen kuluva kokonaisaika on mahdollisimman pieni.

Robottien ohjaus

Robottien ohjausongelmassa on kyse seuraavasta. Työvaiheen toteuttamiseksi teollisuusrobotin täytyy tehdä tietty joukko operaatioita. Ongelmana on järjestää työvaiheen vaatimat operaatiot sellaiseen järjestykseen, että robotin työvaiheeseen käyttämä kokonaisaika on mahdollisimman pieni.

3.3.2 Muunnokset

Tutustutaan kahteen kauppamatkustajan ongelman muunnokseen. Tarkoituksena on näyttää, miten kyseiset muunnokset saadaan palautettua takaisin tavalliseen kauppamatkustajan ongelmaan.

Toistuvat kaupungit

Garfinkel [1987, p. 23] toteaa, että toistuvien kaupunkien kauppamatkustajan ongelmassa on kyse normaalista kauppamatkustajan ongelmasta sillä erotuksella, että kussakin kaupungissa tulee vierailla ainakin kerran sen sijaan, että niissä vierailtaisiin täsmälleen kerran. Hän antaa tämän ongelman muuntamiseen normaaliksi kauppamatkustajan ongelmaksi seuraavan ohjeen.

Muodostetaan kaupunkien $(n \times n)$ etäisyysmatriisista C, jossa c_{ij} on kaupunkien i ja j välinen etäisyys, uusi $(n \times n)$ etäisyysmatriisi C', jossa c'_{ij} on kaupunkien i ja j välisen lyhyimmän polun pituus. Garfinkelin mukaan matriisi C' voidaan laske ajassa $O(n^3)$ ja tämän matriisin avulla toistuvien kaupunkien kauppamatkustajan ongelma palautuu kauppamatkustajan ongelman normaaliin tapaukseen.

Monta kauppamatkustajaa

Garfinkelin [1987, pp. 23–24] mukaan monen kauppamatkustajan ongelma on yksinkertaistettu versio aiemmin käsitellystä ajoneuvojen reititysongelmasta. Hän edelleen jatkaa, että kyseissä ongelmassa on n kaupunkia ja m kauppamatkustajaa, joiden avulla tulee jokaisessa kaupungissa vierailla siten, että jokainen kauppamatkustaja lähtee samasta kaupungista ja jokainen työskentelevä kauppamatkustaja j aiheuttaa kiinteän kustannuksen d_j. Tehtävänä ongelmassa on minimoida kiinteiden kustannusten suuruutta ja yhteensä kuljettavan reitin pituutta. Garfinkel antaa seuraavan ohjeen ongelman muuntamiseksi tavalliseksi kauppamatkustajan ongelmaksi.

Indeksoidaan m kauppamatkustajaa arvoille $0-(m-1)$ siten, että $d_0 \le d_1 \le \ldots \le d_{m-1}$. Lisätään $m-1$ uutta kaupunkia, jotka numeroidaan $-1,\ldots,-(m-1)$. Alkuperäiset kaupungit on numeroitu $0,\ldots,n-1$. Otetaan kotikaupungiksi kaupunki, jonka numero on 0 ja lisätään seuraavat välit alkuperäisten välien joukkoon A: väli $(-i, j)$ kaikilla i jos $(0, j) \in A$, väli $(j,-i)$ kaikilla i jos $(j,0) \in A$ ja väli $(-i,-(i-1))$

kaikilla $i = 1,...,m-1$. Välien kustannukset saadaan Garfinkelin mukaan seuraavalla tavalla:

$$c'_{ij} = c_{ij} \qquad i, j = 1,...,n$$

$$c'_{-i,j} = c_{0j} + \tfrac{1}{2} \cdot d_i \qquad i = 1,...,m-1,\ j = 1,...,n$$

$$c'_{j,-i} = c_{j0} + \tfrac{1}{2} \cdot d_i \qquad i = 1,...,m-1,\ j = 1,...,n$$

$$c'_{-i,-(i-1)} = \tfrac{1}{2} \cdot d_{i-1} - \tfrac{1}{2} \cdot d_i \qquad i = 1,...,m-1.$$

Garfinkel [1987, p. 25], huomauttaa että tilanteessa, jossa halutaan ratkaista symmetrinen kauppamatkustajan ongelma, voidaan esitetty muunnos muuntaa symmetriseen muotoon.

3.4 Laskennallinen vaativuus

Kohdassa käsitellään laskennallisuuden vaativuuden käsitteitä.

3.4.1 Analysointimenetelmiä

Algoritmien vaativuuden teoreettisessa arvioinnissa käytetään kahta päämenetelmää. Nämä menetelmät ovat pahimman tapauksen vaativuus ja keskimääräinen vaativuus. [Johnson and Papadimitriou, 1987a, p. 42]

Pahimman tapauksen vaativuus

Algoritmin arvioitua vaativuutta merkitään $O(f(n))$ notaatiolla, jolla tarkoitetaan sitä, että on olemassa sellainen vakio c siten, että kaikilla kokoa n olevilla syötteillä algoritmin läpimenoaika on pienempi tai yhtä suuri kuin $c \cdot f(n)$. Tämä voidaan tulkita siten, että $O(f(n))$ kertoo algoritmin kannalta keskeisimmän operaation suorituskertojen määrän kertaluokan. Arvioitua vaativuutta kutsutaan pahimman tapauksen vaativuudeksi. [Johnson and Papadimitriou, 1987a, p. 42]

Keskimääräisen tapauksen vaativuus

Johnson ja Papadimitriou [pp. 42–43] huomauttavat, että useimmiten algoritmit koostuvat yhdestä tai useammasta osasta, joiden vaativuudet voivat vaihdella tilanteesta riippuen. Tällöin normaalisti arvioidaan sekä pahimman tapauksen vaativuutta että keskimääräisen tapauksen vaativuutta. Hyvänä esimerkkinä tällaisesta tilanteesta on algoritmi, jossa syöteaineisto järjestää jonkin aineiston alkion suhteen. Tällöin algoritmin alkiot järjestävän osuuden vaativuus voi vaihdella merkittävästi sen mukaan, millaisessa järjestyksessä alkiot alun perin on annettu.

3.4.2 Vaativuusluokat *P* ja *NP*

Algoritmien vaativuutta käsittelevä teoria keskittyy niin sanottuihin päätösongelmiin. Tässä luvussa selvitetään yleisellä tasolla, miten kauppamatkustajan ongelman kannalta oleelliset vaativuusluokat *P* ja *NP* määritellään.

Päätösongelmat

Kocayn ja Kreherin [2005, p. 205] mukaan päätösongelmilla tarkoitetaan ongelmia, joihin voidaan vastata kyllä tai ei. Hämäläinen [2003a, s. 64] kuvaa saman hieman toisin sanoin: Joukon S päätösongelmassa tehtävänä on ratkaista, kuuluuko annettu syöte x ennalta määrättyyn joukkoon S eli ongelman vastaus on kyllä tai ei. Hämäläinen jatkaa, että syötteet esitetään yleensä binäärijonoina, jolloin tehtävänä on laskea joukon $S \subseteq \{0,1\}^*$ karakteristinen funktio $X_s : \{0,1\}^* \to \{0,1\}$.

Algoritmien aikavaativuus ja vaativuusluokka *P*

Hämäläinen [2003a, s. 64] määrittelee algoritmin A aikavaativuusfunktion seuraavasti:

$time_A(x) =$ algoritmin A syötteellä x suorittamien laskenta-askelien määrä

$time_A(n) = \max\{time_A(x) : | x | = n\}$.

Vakiintuneen tavan mukaan pidetään kohtuullisina algoritmeja A, joiden aikavaativuutta $time_A(n)$ rajoittaa jokin syötteen pituuden $| x | = n$ polynomi ja kohtuudella ratkeavina ongelmia, joilla on tällainen ratkaisualgoritmi. Syötteen pituuden suhteen polynomisessa ajassa ratkeavien päätösongelmien joukosta käytetään merkintää *P*. [Hämäläinen, 2003a, s. 65]

Vaativuusluokka *NP*

On olemassa suuri joukko ongelmia, joille ei ole löydetty polynomisessa ajassa toimivaa algoritmia. Toisaalta kukaan ei ole myöskään osoittanut, ettei tietyille ongelmille sellaista algoritmia olisi olemassa. Yhteistä näille ongelmille on se, että ratkaisuvaihtoehtojen lukumäärä kasvaa eksponentiaalisesti syötetiedon koon suhteen ja se, että vaikka ongelmien ratkaiseminen onkin vaikeaa, niin annetun ratkaisun tarkastaminen voidaan suorittaa polynomisessa ajassa. [Hämäläinen, 2003a, s. 65]

Epädeterministinen algoritmi on kaksivaiheinen proseduuri, joka saa syötetietonaan päätösongelman esiintymän I ja tekee seuraavaa:

1. Epädeterministinen vaihe: generoidaan mielivaltaisesti valittu merkkijono S, jota pidetään päätösongelman esiintymän I ratkaisuehdotuksena.

2. Deterministinen vaihe: deterministinen algoritmi saa syötteenä sekä esiintymän I että ratkaisuehdotuksen S ja antaa vastauksen kyllä, jos S on päätösongelman esiintymän I ratkaisu.

Epädeterministisen algoritmin sanotaan ratkaisevan päätösongelman, jos ja vain jos ongelman jokaiselle kyllä-esiintymälle se palauttaa vastauksen kyllä jollakin suorituksella. Edelleen ei-deterministinen algoritmi on ei-deterministinen polynominen algoritmi, jos tarkistusvaihe toimii polynomisessa ajassa.

Niiden päätösongelmien joukosta, jotka voidaan ratkaista ei-deterministisellä polynomisessa ajassa toimivalla algoritmilla, käytetään merkintää *NP*. [Hämäläinen, 2003a, s. 65]

3.5 Menetelmät ja approksimointialgoritmit

Tässä kohdassa luodaan katsaus kauppamatkustajan ongelman ratkaisumenetelmiin ja perehdytään tarkemmin sellaisiin approksimointialgoritmeihin, jotka ovat merkittäviä myöhemmin esiteltävien omien algoritmien kannalta.

3.5.1 Menetelmät

Esitetään yleisimmät menetelmät *NP*-täydellisen ongelman ratkaisemiseksi.

Eksponentiaaliset algoritmit

Eksponentiaalisilla algoritmeilla tarkoitetaan algoritmeja, joiden vaativuus kasvaa eksponentiaalisesti käsiteltävän ongelman koon kasvaessa. Hyvänä esimerkkinä voidaan mainita kauppamatkustajan ongelman ratkaiseminen käymällä läpi kaikki mahdolliset reitit ja valitsemalla niistä lyhyin. Ongelmaksi muodostuu se, että jo yhdeksän solmun tapauksessa näitä reittejä on 362880 eli yhdeksän kertoman verran. Käytännössä jotkin teoreettisesti eksponentiaaliset algoritmit toimivat hyvinkin nopeasti. Esimerkki tällaisesta algoritmista on lineaariseen optimointiin perustuva simplex-algoritmi.

Polynomiset erikoistapaukset

Polynomisilla erikoistapauksilla tarkoitetaan niitä jonkin NP-täydellisen ongelman erikoistapauksia, jotka voidaan ratkaista polynomisessa ajassa. Käytännössä siis ongelmasta pyritään tunnistamaan, onko se tällaisen erikoistapauksen ilmentymä, jolloin se voidaan ratkaista nopeasti.

Hyvä esimerkki tällaisesta erikoistapauksesta on niin sanottu 2SAT-ongelma (2-satisfiability). SAT-ongelma tunnetaan paremmin propositiologiikan

toteutuvuusongelmana. 2SAT-ongelma on SAT-ongelma rajoitettuna ja-lausekkeisiin, joissa yhdistetään korkeintaan 2-komponenttisia tai-lausekkeita. Seuraavassa annetaan esimerkki tällaisesta lausekkeesta:

$$(a_{11} \vee a_{12}) \wedge (a_{21} \vee a_{22}) \wedge \ldots \wedge (a_{n1} \vee a_{n2}) .$$

Satunnaisalgoritmit

Hämäläinen [2003b, s. 57] toteaa, että satunnaisalgoritmi on algoritmi, jossa satunnaisuus voi vaikuttaa sen antamaan tulokseen. Hämäläinen lisää, että satunnaisuus voi antaa käyttökelpoisia, yksinkertaisia ja nopeita ratkaisuja ongelmiin, jotka tavallisilla algoritmeilla ovat hankalasti ratkaistavia.

Approksimointialgoritmin hyvyys

NP-täydellisen optimointiongelman tapaukselle määritetään polynomisessa ajassa ratkaisu, joka on enintään k kertaa optimaalista ratkaisua huonompi, jollakin vakiolla k. Joskus voi olla myös $k = k(|x|)$, tapauksen koon suhteen hitaasti kasvava funktio. [Orponen ja Ernvall, 2004, s. 70]

3.5.2 Approksimointialgoritmit

Esitetään tutkimuksen kannalta merkittävät approksimointialgoritmit.

Lähin naapuri -algoritmi

Lähin naapuri -algoritmin perusajatuksena on siirtyä aloituspisteestä lähimpään naapuri pisteeseen, jossa ei ole vielä käyty, ja jatkaa siitä taas lähimpään käymättömään naapuri pisteeseen, kunnes kaikissa pisteissä on käyty. Tämän jälkeen palataan takaisin aloituspisteeseen. Algoritmi esitetään, koska sen voidaan nähdä olevan looginen edeltäjä lisäysalgoritmille. Seuraavassa on Reineltin [1994, pp. 73–74] kuvaus algoritmin toiminnasta:

1. Valitse satunnainen piste *j* ja aseta *l* = *j* ja $T = \{1,2,\ldots,n\} \setminus \{j\}$.
2. Niin kauan, kuin *T*:ssä on pisteitä, toista seuraavia kohtia 3 ja 4.
3. Olkoon *j* sellainen *T*:n piste, että $c_{lj} = \min\{c_{li} \mid i \in T\}$.
4. Yhdistä *l* ja *j* ja aseta $T = T \setminus \{j\}$ ja *l* = *j*.
5. Yhdistä *l* ensimmäisessä askeleessa valittuun aloitussolmuun.

Tämän algoritmin vaativuusluokka on $O(n^2)$. [Reinelt, 1994]

Lisäysalgoritmi

Lisäysalgoritmi on pohjana myöhemmin esiteltävälle konveksia verhoa hyödyntävälle algoritmille ja sitä kautta työtä varten kehitetyille omille algoritmeille. Perusajatuksena lisäysalgoritmissa on valita aloitusreitti jollakin tavalla, mikä triviaalitapauksessa voi tarkoittaa yhden pisteen valintaa, ja täydentää sitten tätä osareittiä lisäämällä siihen jäljellä olevat osareittiin kuulumattomat pisteet. Seuraavassa Reineltin [p. 82] kuvaus algoritmin toiminnasta:

1. Valitse jollakin tavalla reitti k:n pisteen läpi $v_1, v_2, ..., v_k$ $(k \geq 1)$ ja aseta $W = V \setminus \{v_1, v_2, ..., v_k\}$.
2. Niin kauan, kuin *W*:ssä on pisteitä, toista seuraavia kohtia 3 ja 4.
3. Valitse piste $j \in W$ jollakin kriteerillä.
4. Lisää *j* johonkin kohtaa osareittiin ja aseta $W = W \setminus \{j\}$.

Tämän algoritmin vaativuusluokka on $O(n^2)$ tai $O(n^2 \cdot \log(n))$ riippuen lisättävän pisteen valintakriteeristä. [Reinelt, 1994]

Mainittakoon vielä, että Reinelt [p. 88] esittää suosituksen käyttää aloitusreittinä pistejoukon konveksia verhoa. Hän perustelee tätä sillä, että on todistettu pistejoukon konveksin verhon muodostavien pisteiden esiintyvän kyseisessä konveksin verhon antamassa järjestyksessä optimireitissä. Hänen mukaansa tällä valinnalla saadaan myös jonkin verran parempia reittejä. Johnson ja Papadimitriou [1987b, p. 158] mainitsevat, että lisäysalgoritmit ylipäätänsä antavat ratkaisuja, jotka voivat lähestyä kaksinkertaista pituutta optimireittiin verrattuna.

Konveksia verhoa hyödyntävä approksimointialgoritmi

Myöhemmin esiteltäviä omia approksimointialgoritmeja voidaan pitää muunnoksina Stewartin ja Bodinin konveksia verhoa hyödyntävästä approksimointialgoritmista, joka on esitelty Goldenin ja muiden artikkelissa [1980, pp. 698–699]. Tämän algoritmin perustoiminta on seuraava:

1. Muodostetaan pistejoukolle konveksi verho, joka otetaan osareitiksi.
2. Jokaiselle pisteelle *k*, joka ei vielä kuulu osareittiin, etsitään osareitin peräkkäiset pisteet *i* ja *j*, joille $c_{ik} + c_{kj} - c_{ij}$ on pienin.
3. Kaikista reitin muutosvaihtoehdoista *(i, k, j)* etsitään sellainen *(a, b, c)*, jolle $(c_{ab} + c_{bc})/c_{ac}$ on pienin.
4. Lisätään piste *b* osareittiin pisteiden *a* ja *b* väliin.
5. Toistetaan kohdat 2, 3 ja 4, kunnes kaikki pisteet on liitetty reittiin.

Tämän algoritmin vaativuusluokka on $O(n^2 \cdot \log(n))$. [Golden *et al.*, 1980]

Varsinaisessa tutkimuksessa ei tulla suoraan käyttämään tätä algoritmia, vaan siitä tehtyä muunnosta, jossa lisättävän pisteen valinta perustuu suoraan arvoon $c_{ik} + c_{kj} - c_{ij}$. Reittiin siis lisätään se alkio, jonka aiheuttama kustannus on pienin. Lisäksi lisättävän pisteen valinta suoritetaan niin, että algoritmin vaativuudeksi saadaan $O(n^2)$.

Lin-Kernighan algoritmi

Lin-Kernighan algoritmia ei käytetä tämän työn yhteydessä tehdyssä kokeellisessa tutkimuksessa. Se esitellään kuitenkin sen takia, että kyseessä on yksi tärkeimmistä algoritmeista kauppamatkustajan ongelman tutkimuksen kannalta. Oleellista tämän työn kannalta on myös se, että käytetylle testimateriaalille ilmoitetut parhaat löydetyt reittien pituudet on laskettu usein Lin-Kernighan algoritmilla tai sen variaatioilla.

Lin-Kernighan 2-vaihtoalgoritmin perusideana on ottaa lähtökohdaksi jokin reitti ja sitten parantaa sitä vaihtamalla 2 kaarta reitistä, kunnes parannusta ei vaihtoja suorittamalla enää synny. Tällä tarkoitetaan sitä, että poistetaan ratkaisusta 2 kaarta, jotka eivät ole peräkkäin ja tilalle otetaan 2 uutta kaarta, joilla kaksi osareittiä saadaan yhdistettyä yhdeksi silmukaksi. Usein retin muunnos tehdään välittömästi, kun ensimmäinen parannus saadaan aikaan, mutta on myös mahdollista, että tutkitaan kaikki eri vaihtoehdot ja tehdään se reitin muunnos, joka tarjoaa suurimman parannuksen reitin arvoon. [Hämäläinen, 2003b, s. 54]

Hämäläisen [s. 54] kertoo, että käytännön testeissä Lin-Kernighan 3-vaihto algoritmi on osoittautunut parhaaksi kauppamatkustajan ongelman approksimointialgoritmiksi. Kyseisessä algoritmissa poistetaan reitistä 3 kaarta ja laitetaan tilalle 3 uutta kaarta. Tapauksessa, jossa muunnos lyhentää reittiä, on löydetty parempi ratkaisu.

Hämäläinen [s. 55] jatkaa edelleen, että voidaan määritellä yleisesti Lin-Kernighan *k*-vaihto-algoritmi, jossa poistetaan *k* kappaletta kaaria ja tuodaan *k* uutta kaarta tilalle. Hämäläinen kuitenkin mainitsee, että kun *k*:n arvo kasvaa suuremmaksi kuin 3, niin työmäärä on niin suuri, että algoritmia ei yleensä kannata käyttää.

3.6 Approksimointialgoritmien kokeellinen tutkimus

Kohdassa käydään läpi algoritmien kokeellisessa tutkimuksessa huomioitavia asioita.

3.6.1 Kokeellisen tutkimuksen kriteerit

Golden ja Stewart [1987, p. 208] mainitsevat seuraavat viisi kriteeriä, joiden mukaan algoritmeja tulisi vertailla kokeellisessa tutkimuksessa:

1. ratkaisun tarkkuus
2. läpimenoaika
3. implementoinnin vaikeus
4. joustavuus
5. yksinkertaisuus.

Golden ja Stewart jatkavat, että vaikka algoritmin läpimenoaika on usein kriittinen kriteeri valittaessa sopivaa algoritmia, niin myös implementoinnin vaikeutta tulee harkita perusteellisesti. Mikäli algoritmien läpimenoajoissa ei ole merkittävää eroa, niin on useimmiten järkevää valita helpommin toteutettavissa oleva ratkaisu. Algoritmin joustavuudella tarkoitetaan algoritmin kykyä käsitellä ongelman eri variaatioita. Tämäkin ominaisuus voi olla merkittävä käyttötilanteesta riippuen. Algoritmin yksinkertaisuudella tarkoitetaan itse algoritmin muotoilua, eikä niinkään sen implementointia.

3.6.2 Testimateriaalin valinta

Golden ja Stewart [1987, pp. 208–209] painottavat oikeanlaisen testimateriaalin valitsemisen merkitystä. He kehottavat valitsemaan testimateriaalin niin, että sen optimiratkaisut ovat tunnettuja tai että ainakin optimiratkaisujen alarajat ovat tunnettuja. Lisäksi he painottavat sen merkitystä, että valittu testimateriaali luo kattavan kuvan ongelman eri ilmentymistä.

4 Tutkittavien approksimointialgoritmien esittely

Tässä luvussa valaistaan omien algoritmien kantavan idean taustaa, kuvataan omien algoritmien tarvitsemat tietorakenteet ja esitellään keskeiset omissa algoritmeissa hyödynnettävät algoritmit.

4.1 Taustaa

Algoritmien kehitys lähti ajatuksesta, mikä olisi ihmiselle helppo approksimointialgoritmi kauppamatkustajan ongelman ratkaisuun tilanteessa, jossa käytettävissä ei ole muita apuvälineitä kuin paperi, kynä, kumi ja viivoitin. Perusajatuksena oli rajata käsiteltävää tietoa eli pistejoukon käsiteltäviä tapauksia siten, että sen visuaalinen hahmottaminen helpottuisi. Tätä ajatusta johtolankana käyttäen kehitetty algoritmi muistutti niin paljon konveksia verhoa hyödyntävää algoritmia, että heräsi ajatus sen mahdollisesta laskennallisesta tehokkuudesta. Tästä seurasi päätös algoritmin toteutuksen saattamisesta loppuun ja tämän työn tekemisestä kyseisestä algoritmista. Kehitystyön aikana algoritmista syntyi useita versioita, joissa kaikissa perusidea on sama, mutta tekniset toteutusratkaisut vaihtelevat.

Eräs innoittava tekijä algoritmien toimintaan suunnitellessa, oli ystäväni luona näkemäni suunnistuskartat, joiden maaston korkeuseroja kuvaavat käyrät toivat mieleeni pistejoukolle piirtyvät sisäkkäiset konveksit verhot. Tämän ajatuksen johdosta olen itse kutsunut algoritmin perusversiota Suunnistajan algoritmiksi.

4.2 Omissa algoritmeissa tarvittavat tietorakenteet

Algoritmeissa käytetään seuraavia tietorakenteita.

Alkio

Alkio sisältää pisteen koordinaattien lisäksi muita algoritmien tarvitsemia tietoja. Jatkossa sanaa alkio käytetään sanan piste sijaan silloin, kun se on käsiteltävän asian kannalta mielekästä. Alkion tiedot ovat seuraavat:

- tunnistenumero
- x-koordinaatti
- y-koordinaatti
- seuraava
- edellinen
- liitoksessa edeltävä
- liitoksessa seuraava

- poista
- alkuperä.

Näistä tiedoista seuraava ja edellinen ovat tarpeen, kun alkioita käsitellään listoina. Liitoksessa edeltävä ja seuraava kertovat algoritmia suorittaessa, mihin kohtaan olemassa olevaa osareittiä algoritmin kannattaa sijoittaa alkio.

Koska algoritmeissa pisteitä käsitellään suurina joukkoina taulukoita hyödyntäen toteutetuissa tietorakenteissa, niin mahdollisia poistoja ei kannata tehdä muulloin, kun se on ehdottaman välttämätöntä. Alkioita käsiteltäessä ne tarvittaessa merkitään poistettaviksi ja varsinainen poisto toteutetaan silloin, kun se on välttämätöntä.

Samaa pistettä joudutaan joissakin tilanteissa käsittelemään yhtä aikaa eri tietorakenteissa. Tämän takia tarvitaan linkkiä kyseisten tietojen välille. Tämä linkki toteutetaan tiedolla alkuperästä.

Taulukko

Taulukolla tarkoitetaan näiden algoritmien yhteydessä dynaamista tietorakennetta, johon pisteet tallennetaan algoritmin alussa alkioina. Algoritmien edetessä pisteitä siirretään taulukosta reittiin. Reittiin liittämisen yhteydessä alkio tulee poistaa taulukosta. Tämän takia taulukon lukeminen on mahdollistettu listana, jolloin poistettavia alkioita ei tarvitse konkreettisesti poistaa taulukosta, vaan ne ainoastaan merkitään poistettaviksi ja poistetaan listakäsittelystä.

Lista

Algoritmeissa luotava reitti ja konveksi verho on tallennettu listoihin. Tähän ratkaisuun on päädytty siksi, että tehtävät alkioiden lisäykset saadaan nopeiksi. Listarakenteeseen on myös lisätty pino-toiminnallisuudet, jotta sitä voidaan käyttää konveksin verhon muodostamisessa Grahamin pyyhkäisymenetelmällä.

4.3 Omissa algoritmeissa hyödynnettävät algoritmit

Ennen omien algoritmien kuvaamista, käydään yleisellä tasolla läpi ne olemassa olevat algoritmit ja menetelmät, joita käytetään omien algoritmien toteutuksissa. Lisäksi kuvataan joitakin näihin tehtyjä muutoksia.

4.3.1 Pikalajittelu

Konveksia verhoa luodessa käsiteltävät pisteet laitetaan kulmajärjestykseen. Järjestäminen tehdään pikalajittelualgoritmilla. Algoritmin julkaisi vuonna 1960 Hoare ja algoritmin vaativuus on keskimäärin $O(n \cdot \log(n))$ ja pahimmassa tapauksessa $O(n^2)$ [Aho et al., 1983]. Algoritmia ei kuvata tarkemmin sen yleisen tunnettuuden takia.

4.3.2 Pisteiden välisen kulman tutkiminen

Pisteiden välisen kulman tutkimisella tarkoitetaan tämän työn yhteydessä sitä, että onko pisteiden $p_0 = (x_0, y_0)$ ja $p_2 = (x_2, y_2)$ välinen jana vasta- vai myötäpäivään pisteiden p_0 ja $p_1 = (x_1, y_1)$ välisestä janasta. Tämä tutkitaan seuraavalla menetelmällä:

Lasketaan ristitulo $(p_1 - p_0) \times (p_2 - p_0)$ seuraavalla kaavalla:

$$(p_1 - p_0) \times (p_2 - p_0) = (x_1 - x_0) \cdot (y_2 - y_0) - (x_2 - x_0) \cdot (y_1 - y_0).$$

Mikäli ristitulo on suurempi kuin 0, niin pisteiden p_0 ja p_2 välinen jana on vastapäivään pisteiden p_0 ja p_1 välisestä janasta. [Orponen ja Ernvall, 2004]

4.3.3 Käännös vasemmalle vai oikealle

Tämän työn yhteydessä sillä, onko käännös vasemmalle vai oikealle, tarkoitetaan sitä, että onko käännös pisteiden $p_1 = (x_1, y_1)$ ja $p_2 = (x_2, y_2)$ väliseltä janalta pisteiden p_2 ja $p_3 = (x_3, y_3)$ väliselle janalle oikealle vai vasemmalle. Tämä tutkitaan seuraavalla menetelmällä:

Lasketaan ristitulo $(p_2 - p_1) \times (p_3 - p_2)$ seuraavalla kaavalla:

$$(p_2 - p_1) \times (p_3 - p_2) = (x_2 - x_1) \cdot (y_3 - y_2) - (x_3 - x_2) \cdot (y_2 - y_1).$$

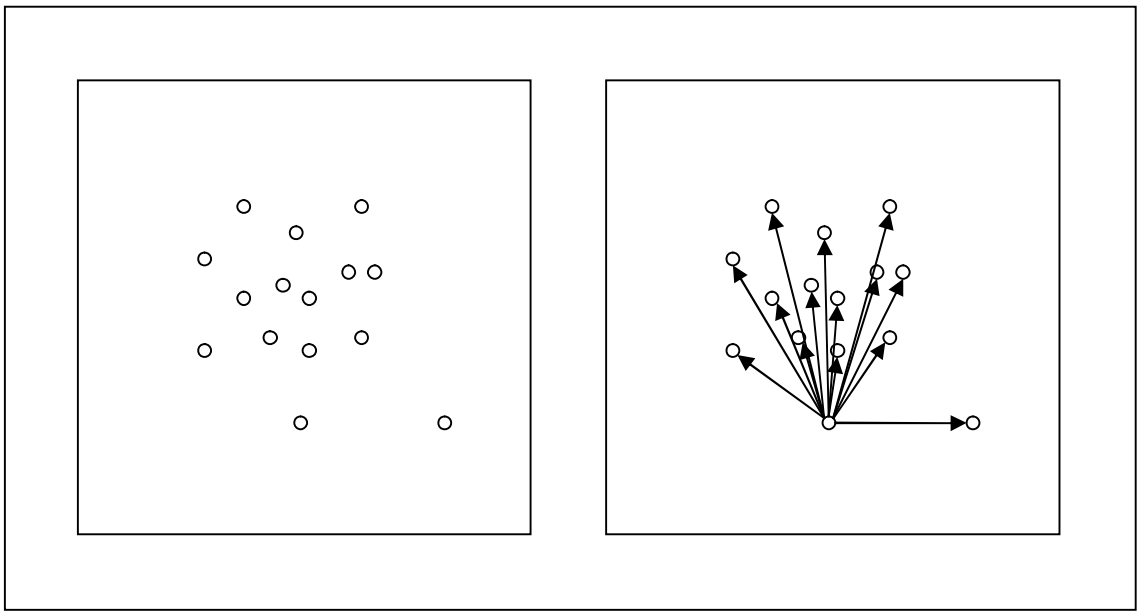
Mikäli ristitulo on suurempi kuin 0, niin käännös pisteiden p_1 ja p_2 väliseltä janalta pisteiden p_2 ja p_3 väliselle janalle on vasempaan. [Orponen ja Ernvall, 2004]

4.3.4 Grahamin pyyhkäisymenetelmä konveksille verholle

Oman algoritmin toteutus sisältää konveksin verhon muodostamisen. Tämä tehdään Grahamin pyyhkäisymenetelmällä, jonka perusrakenteen Orponen ja Ernvall [2004, ss. 85–86] esittävät toimivan seuraavalla tavalla:

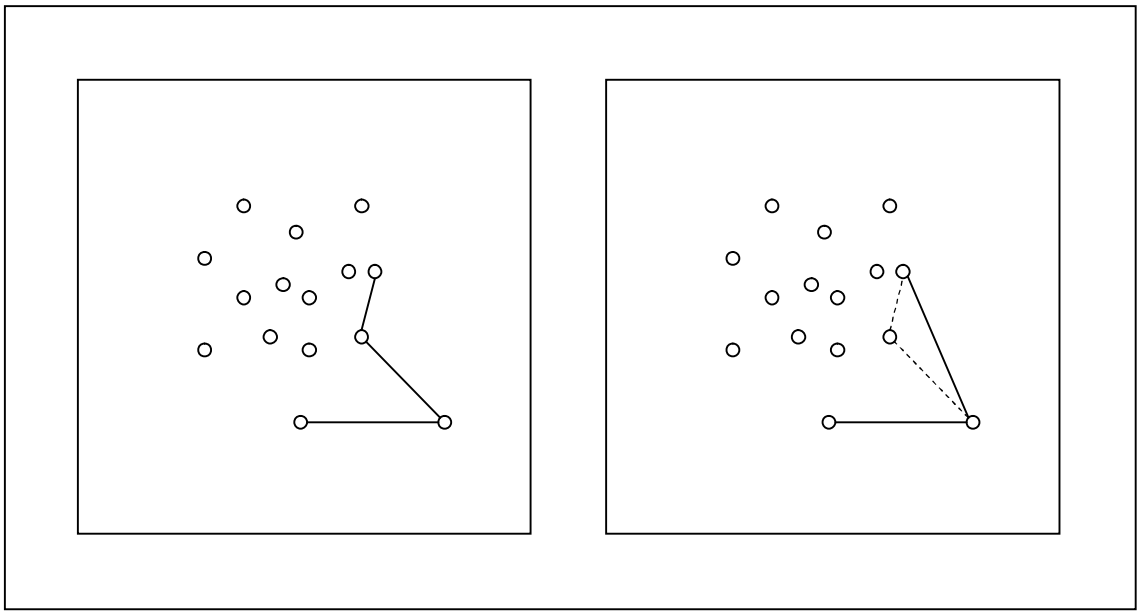
1. Järjestetään pisteet nousevaan kulmajärjestykseen jostain referenssipisteestä lukien. Referenssipisteeksi valitaan piste, jolla on pienin y-koordinaatin arvo. Jos samalla y-koordinaatin arvolla olevia pisteitä on useita, niin

valitaan se, jolla on pienin x-koordinaatin arvo. Kuva 3 esittää kyseisellä tavalla tehtyä kulmajärjestystä.



Kuva 3. Pisteet järjestetään referenssipisteen mukaiseen kulmajärjestykseen.

2. Käydään pisteitä läpi järjestyksessä ja kerätään ehdokaspisteitä pinoon *S*.
3. Kun ehdokasverho tekee ei-konveksin käännöksen (käännös oikealle), poistetaan pinosta *S* pisteitä, kunnes mutka oikenee. Kuva 4 havainnollistaa oikaisun toimintaa.



Kuva 4. Ei-konveksi käännös oikaistaan.

4. Kun kaikki pisteet on käsitelty, on konveksin verhon pisteet pinossa *S* oikeassa järjestyksessä.

Normaalisti konveksi verho toteutetaan siten, että kun pisteet järjestetään referenssipisteen mukaiseen kulmajärjestykseen, niin samalla kulmalla olevista pisteistä säilytetään vain etäisin. Koska omissa algoritmeissa kaikki pisteet tulee kuitenkin käsitellä, niin näin ei ole toimittu, vaan kaikki pisteet otetaan mukaan. Tästä seuraa ongelmia, sillä jos heti referenssipisteen jälkeen pienimmällä kulmalla olevat pisteet ovat samalla selaussuoralla, niin ne tulisi käsitellä siten, että lähin laitetaan pinoon ensin. Myöhemmin tulisi taas toimia niin, että kaukaisin laitetaan pinoon ensin.

Omissa algoritmeissa toimitaan siten, että kun kulmajärjestys tehdään, niin samalla kulmalla olevat pisteet järjestetään niin, että lähemmän kulma-arvo katsotaan suuremmaksi. Pikalajittelun suorittamisen jälkeen tutkitaan, onko välittömästi ensimmäisen pisteen perässä samalla kulma-arvolla olevia alkioita, ja jos on, niin ne käännetään siten, että lähempänä oleva on kauempana olevaa edellä.

4.3.5 Menetelmä konveksin verhon korjaamiseksi

Algoritmien 2, 3 ja 4 toiminnassa oleellisena osana on konveksin verhon korjaaminen. Koska konveksi verho on muodostettu Grahamin pyyhkäisymenetelmällä, jakaantuu tämä korjaaminen kahteen tapaukseen, jotka käsitellään erikseen. Seuraavassa kuvaillaan työn tekijän itse kehittämä menetelmä konveksin verhon korjaamiseksi.

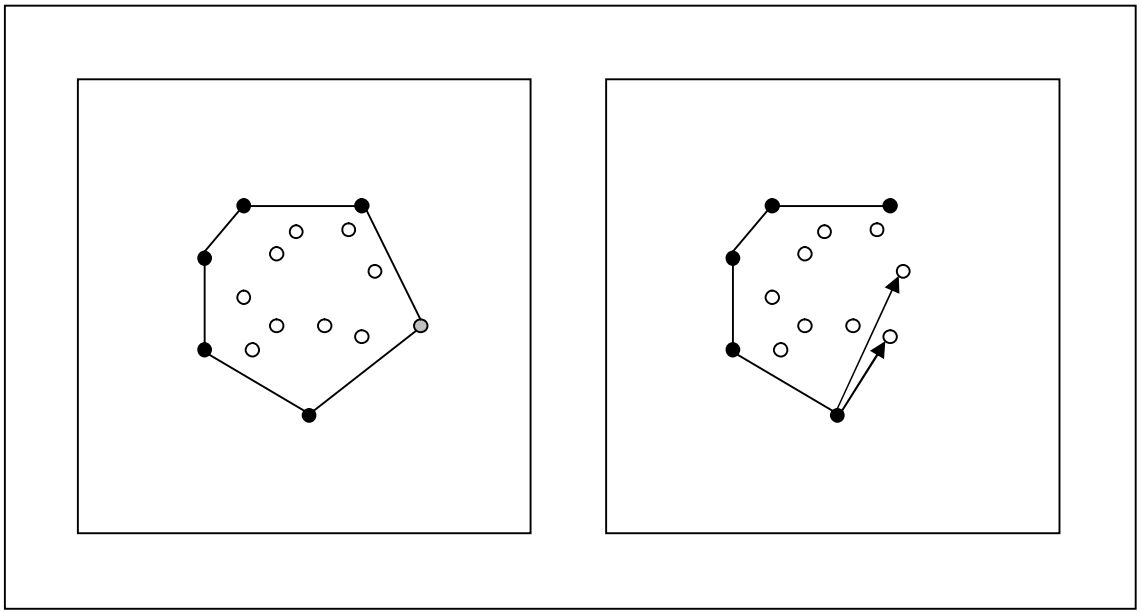
Korjauksen mahdollistavat alkuvalmistelut

Pisteet tallennetaan taulukkoon viitteinä alkioihin ja järjestetään Grahamin pyyhkäisymenetelmässä kuvatulla tavalla. Tämän jälkeen alkiot linkitetään toisiinsa niin, että ne muodostavat kumpaankin suuntaan linkitetyn listan. Konveksi verho muodostetaan siten, että konveksiin verhoon tulevista alkioista otetaan kopiot, joihin tallennetaan linkki alkuperäiseen alkioon, ja nämä kopiot tallennetaan kumpaankin suuntaan linkitettyyn listaan.

Korjaus kun poistettava piste ei ole kulmapiste

Tapauksessa, jossa poistettava piste ei ole kulmapiste, otetaan ylös sitä seuraava ja edeltävä alkio konveksissa verhossa. Poistettavan alkion linkin avulla alkuperäinen alkio merkitään poistettavaksi taulukkoon tallennetusta listasta. Tällöin alkio säilyy taulukossa, mutta poistuu listakäsittelystä.

Talteen otettujen, konveksissa verhossa poistettavaa alkiota edeltävän ja seuraavan alkion sisältämien alkuperäisiin alkioihin osoittavien linkkien avulla voidaan lukea alkiot sisältävästä taulukosta listakäsittelyn avulla ne alkiot, jotka ovat kulma-arvoiltaan edeltävän ja seuraavan alkion välissä. Näille alkioille noudatetaan Grahamin pyyhkäisymenetelmässä kuvattua käsittelyä ja lisätään konveksiin verhoon sinne kuuluvat alkiot. Kuva 5 näyttää, miten kulmajärjestys auttaa valitsemaan käsiteltävät alkiot.



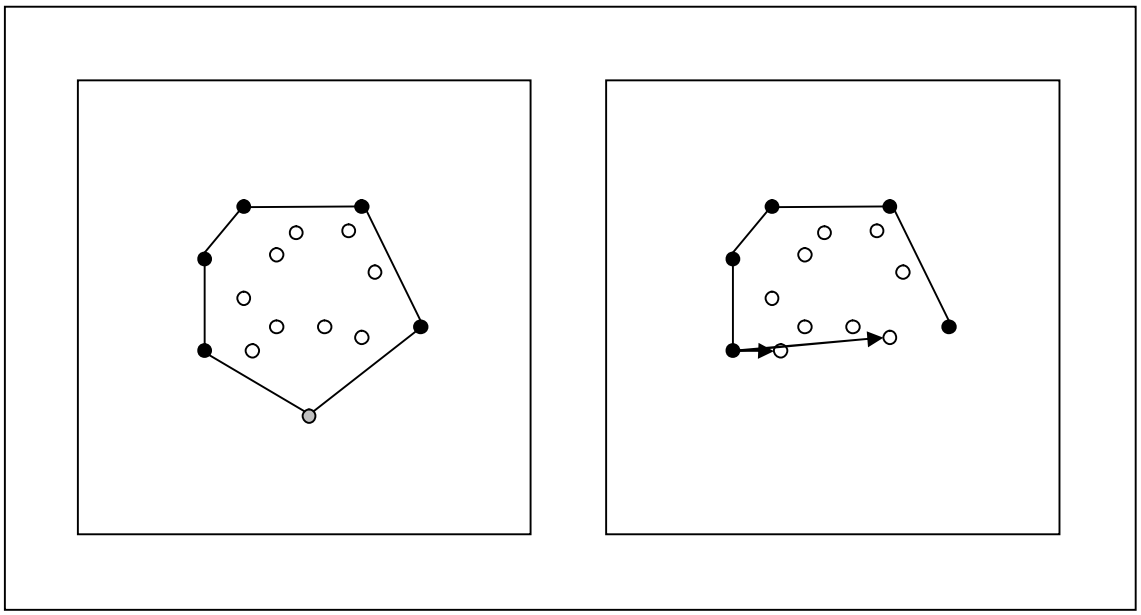
Kuva 5. Kulmajärjestyksen avulla löydetään harmaata eli poistettavaa alkiota edeltävän ja seuraavan alkion välissä olevat käsiteltävät alkiot.

Korjaus kun poistettava piste on kulmapiste

Tapauksessa, jossa poistettava alkio on kulmapiste, merkitään poistettavassa alkiossa olevan linkin avulla alkuperäinen alkio poistettavaksi. Otetaan talteen konveksissa verhossa edeltävänä ja seuraavana oleva alkio ja poistetaan poistettava alkio konveksista verhosta. Tyhjennetään alkioiden viitteiden taulukko niistä viitteistä, jotka osoittavat poistettaviksi merkittyihin alkioihin. Grahamin pyyhkäisymenetelmässä kuvatulla tavalla valitaan uusi kulmapiste ja suoritetaan sen mukainen järjestäminen. Tämän jälkeen alkiot, joihin taulukosta viitataan, linkitetään kumpaankin suuntaan linkitetyksi listaksi.

Huomion arvoista on, että nyt voidaan taas toimia korjauksen suhteen samoin kuin silloin, kun poistettava alkio ei ollut kulmapiste. Tämä johtuu siitä, että poistettavaa alkiota edeltävä ja seuraava alkio konveksista verhosta ovat tallessa ja

niistä löytyy viitteet alkuperäisiin alkioihin. Kuvasta 6 nähdään, miten uuden referenssipisteen mukainen kulmajärjestys auttaa valitsemaan käsiteltävät alkiot.



Kuva 6. Uuden referenssipisteen mukaisen kulmajärjestyksen avulla löydetään harmaata eli poistettavaa alkiota edeltävän ja seuraavan alkion välissä olevat käsiteltävät alkiot.

Hyödyllinen rajoite

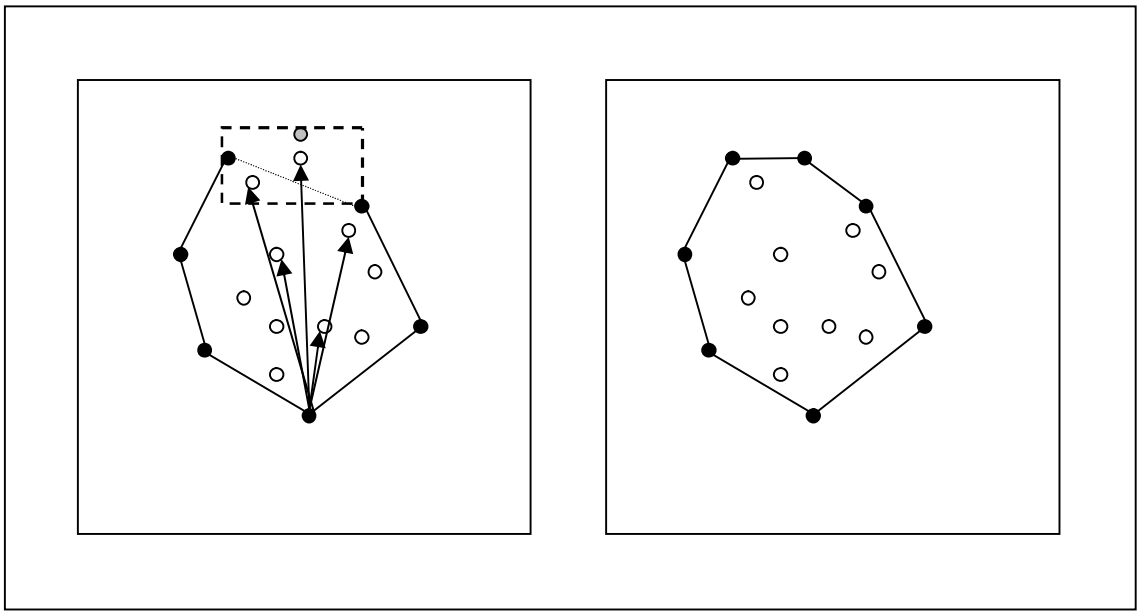
Käsiteltävän pistejoukon ollessa suuri konveksin verhon korjaamista hidastaa se, että edellisen ja seuraavan alkion välissä voi olla paljon alkioita, joiden kulma-arvo sijoittuu kyseiselle välille. Suurin osa näistä alkioista on sellaisia, että ne eivät voi päätyä konveksiin verhoon. Verhon korjaaminen nopeutuu huomattavasti, kun käytetään seuraavassa kuvattavaa rajoitetta, joka estää sen, ettei täysin turhille pisteille tehdä kalliita vasemmalle ja oikealle kääntymisen tarkasteluja.

Otetaan talteen konveksista verhosta poistettavan, sitä edeltävän ja seuraavan alkion koordinaateista suurin ja pienin x-koordinaatti x_s ja x_p sekä suurin ja pienin y-koordinaatti y_s ja y_p. Mahdollisia lisäysalkioita tarkasteltaessa ohitetaan kaikki sellaiset alkiot, joiden x-koordinaatti on pienempi kuin x_p tai suurempi kuin x_s. Samoin ohitetaan kaikki ne alkiot, joiden y-koordinaatti on pienempi kuin y_p tai suurempi kuin y_s.

Niille alkioille, jotka täyttävät x- ja y-koordinaatteihin liittyvät ehdot, suoritetaan aikaisemmin kuvatun kaltainen käännökseen vasemmalle liittyvä tarkastelu

poistettavaa alkiota edeltävän ja seuraavan alkion suhteen. Tarkastelu ikään kuin luo janan edellisen ja seuraavan alkion väliin ja varmistaa, että perusteellisemmin käsiteltävät pisteet ovat janan oikealla puolella. Tämä takaa sen, että Grahamin pyyhkäisymenetelmän mukaiset vaativammat tarkastelut tehdään ainoastaan mahdollisimman todennäköisesti konveksiin verhoon lisättäville alkioille.

Kuva 7 pyrkii selventämään tarkastelujen toimintaa. Kuvassa harmaan eli poistettavan alkion ja sitä konveksissa verhossa edeltävän ja seuraavan alkion avulla luotu x- ja y-koordinaattien tarkastelu esitetään katkoviivalla merkityllä suorakulmiolla. Edeltävän ja seuraavan alkion välillä näkyvä pisteviiva kuvaa toisena rajoitteena käytettävän käännöstarkastelun vaikutusta. Rajoitteiden ansiosta kuvan tilanteessa vaativammat tarkastelut tehdään vain yhdelle alkiolle viidestä kulmajärjestyksen mukaan tutkittavaksi valittavasta alkiosta. Tämän rajoiteyhdistelmän käyttö nopeuttaa konveksin verhon korjausta, sillä suurilla pistejoukolla ja kulmapisteen ollessa kaukana voi potentiaalisia käsiteltäväksi tulevia pisteitä olla erittäin paljon.



Kuva 7. Rajoitteiden toiminta tilanteessa, jossa konveksi verho korjataan.

4.4 Algoritmien perustoiminnot

Seuraavassa kuvataan algoritmien perustoiminnot.

4.4.1 Algoritmi 1

Algoritmi 1 on Stewartin ja Bodinin konveksia verhoa hyödyntävä approksimointialgoritmi, johon on tehty pieniä muutoksia. Sitä voidaan myös pitää sovelluksena Reineltin [1994, pp. 85–86] esittämästä kandidaattipisteiden käytöstä

lisäysalgoritmissa. Algoritmia 1 käytetään muiden algoritmien vertailukohtana ja sen perustoiminta on seuraavanlainen:

1. Muodostetaan pistejoukolle konveksi verho ja otetaan se osareitiksi.
2. Jokaiselle pisteelle *k*, joka ei vielä kuulu osareittiin, etsitään osareitin peräkkäiset pisteet *i* ja *j*, joille $c_{ik} + c_{kj} - c_{ij}$ on pienin. Tämä lisäyskohta tallennetaan pisteen tietoihin. Pidetään yllä tietoa siitä, millä pisteellä kyseinen arvo on pienin ja kutsutaan tätä pistettä *b*:ksi ja sen lisäysvälin pisteitä *a*:ksi ja *c*:ksi.
3. Lisätään piste *b* osareittiin pisteiden *a* ja *c* väliin.
4. Tutkitaan jokainen osareittiin kuulumaton piste *k* osareitin välien (*a*, *b*) ja (*b*, *c*) suhteen. Mikäli lisäyksen (*a*, *k*, *b*) aiheuttama kustannus $c_{ak} + c_{kb} - c_{ab}$ tai lisäyksen (*b*, *k*, *c*) aiheuttama kustannus $c_{bk} + c_{kc} - c_{bc}$ on pienempi kuin pisteelle *k* tallennetun lisäyskohdan aiheuttama kustannus, niin pisteen lisäyskohdaksi tallennetaan pienemmän kustannuksen aiheuttanut. Pisteet, joilla vanhana lisäyskohtana on ollut (*a*, *c*), tutkitaan myös välien (*a*, *b*) ja (*b*, *c*) suhteen ja lisäyskohdaksi otetaan näistä kahdesta se, joka aiheuttaa pienemmän kustannuksen. Pidetään yllä tietoa siitä, millä pisteellä lisäys on edullisin. Kutsutaan tätä pistettä b_u:ksi ja sen lisäysvälin pisteitä a_u:ksi ja c_u:ksi.
5. Lisätään piste b_u osareittiin pisteiden a_u ja c_u väliin.
6. Kutsutaan a_u:ta a:ksi, b_u:ta b:ksi ja c_u:ta c:ksi.
7. Toistetaan kohtia 4, 5 ja 6, kunnes pisteitä ei ole enää jäljellä.

Algoritmin vaativuus koostuu konveksin verhon muodostamisen vaativuudesta ja algoritmin loppuosan vaativuudesta. Konveksin verhon muodostaminen on kuvattu aikaisemmin tässä tutkielmassa. Sen aikavaativuus on keskimäärin luokkaa $O(n \cdot \log(n))$ ja pahimmillaan luokkaa $O(n^2)$. Algoritmin loppuosan vaativuus saadaan seuraavaksi esitettävällä kaavalla. Merkinnöissä *x* tarkoittaa pisteiden määrää konveksissa verhossa ja *n* tarkoittaa pisteiden lukumäärää. Kaavassa käytettävä kerroin 3 tulee siitä, että ensimmäisen pisteen valinnan jälkeisillä kierroksilla kustannusfunktio suoritetaan kolme kertaa kunkin tutkittavan pisteen kohdalla.

$$(a-x)\cdot x + (n-x-1)\cdot 3 + (n-x-2)\cdot 3 + \ldots + [n-(n-2)]\cdot 3 + [n-(n-1)]\cdot 3$$

Tämä saadaan sievennettyä seuraavaan muotoon.

$$(n-x)\cdot x + 3\cdot[(n-x-1) + (n-x-2) + \ldots + 2 + 1]$$

Sen jälkeen hyödynnetään seuraavaa kokonaislukusarjan summakaavaa

$$1+2+3+...+n=\frac{n\cdot(n+1)}{2}.$$

Tämän kaavan avulla saadaan seuraava sievennys

$$(n-x)\cdot x+3\cdot\{\tfrac{1}{2}\cdot[(n-x-1)\cdot(n-x-1+1)]\},$$

joka sievenee edelleen helpommin luettavaan muotoon

$$\tfrac{3}{2}\cdot n^2-2\cdot n\cdot x-\tfrac{3}{2}\cdot n+\tfrac{1}{2}\cdot x^2+\tfrac{3}{2}\cdot x.$$

Sievennettyä kaavaa voidaan arvioida seuraavalla tavalla

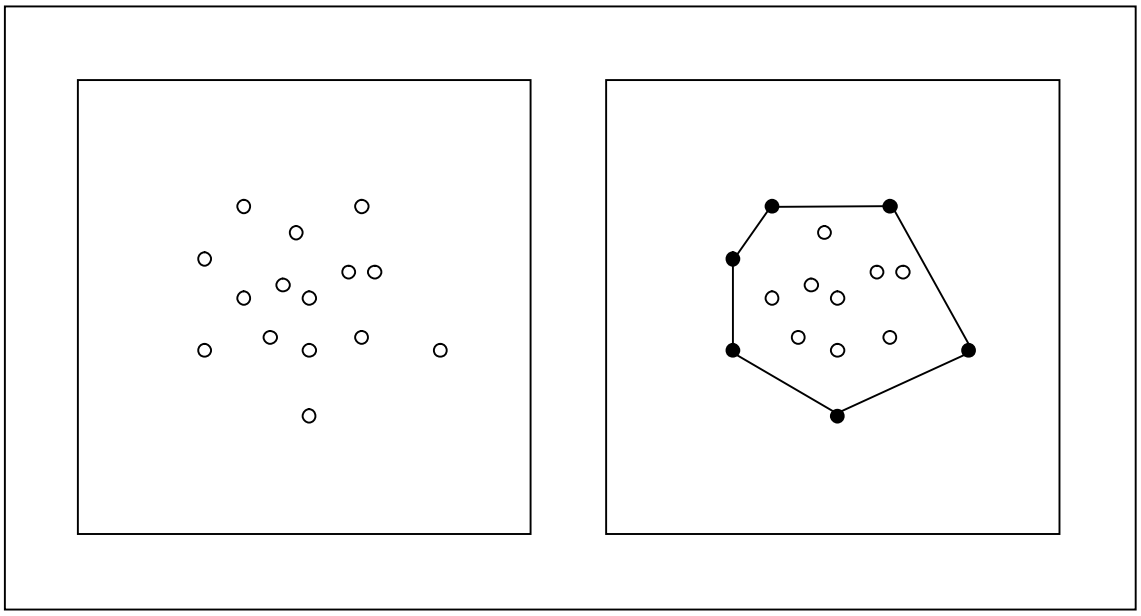
$$\tfrac{3}{2}\cdot n^2-2\cdot n\cdot x-\tfrac{3}{2}\cdot n+\tfrac{1}{2}\cdot x^2+\tfrac{3}{2}x\le\tfrac{3}{2}\cdot n^2+\tfrac{1}{2}\cdot x^2+\tfrac{3}{2}\cdot x$$

$$\le 2\cdot n^2+\tfrac{1}{2}\cdot n^2+\tfrac{3}{2}\cdot n^2=4\cdot n^2.$$

Algoritmin vaativuus on siis luokkaa $O(n^2)$. Seuraavalla kuvasarjalla selvennetään algoritmin toimintaa. Kuvasarjan ei kuvaa oikeaa tilannetta, joten sen esittämät lisäyskohdat eivät välttämättä ole parhaita mahdollisia.

Lähtötilanne ja konveksin verhon luonti

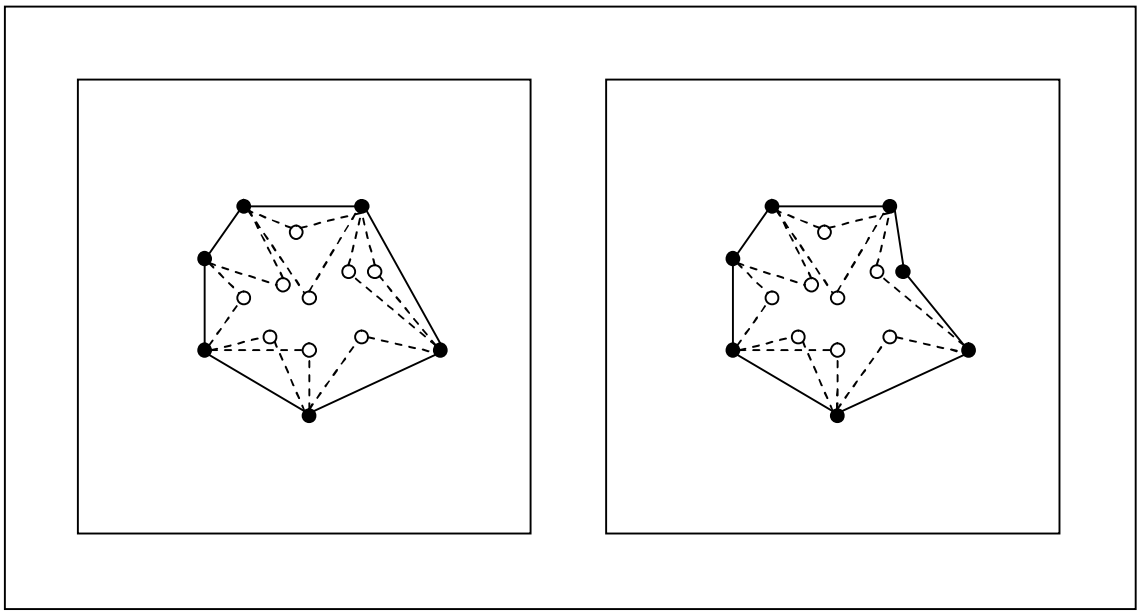
Algoritmi valitsee pistejoukon konveksin verhon osareitiksi (kuva 8).



Kuva 8. Alkuperäiselle pistesarjalle muodostetaan konveksi verho.

Lisäyskohtien etsintä

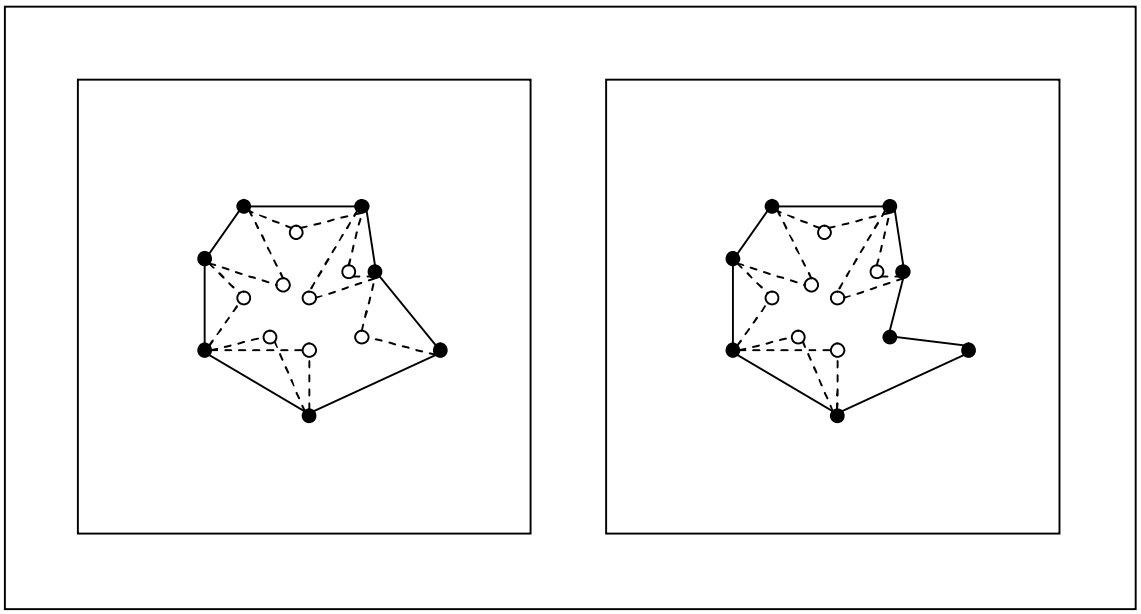
Algoritmi etsii osareitin sisälle jääville pisteille pienimmän kustannuksen aiheuttavan lisäyskohdan ja toteuttaa kaikista pienimmän kustannuksen aiheuttavan lisäyksen (kuva 9).



Kuva 9. Etsitään lisäyskohdat ja toteutetaan pienimmän kustannuksen aiheuttava lisäys.

Tutkinta uusia osareitin osia vasten ja seuraava lisäys

Algoritmi tutkii osareitin sisälle jääneet pisteet kahta uutta osareitin väliä vasten ja toteuttaa pienimmän kustannuksen aiheuttavan lisäyksen (kuva 10).



Kuva 10. Tutkitaan uusia osareitin osia vasten ja tehdään seuraava lisäys.

Algoritmi jatkaa toimintaansa samalla tavalla, kunnes jäljellä ei enää ole osareittiin kuulumattomia pisteitä.

4.4.2 Algoritmi 2 (Suunnistajan algoritmi)

Algoritmi 2 on omien algoritmien perusversio. Sen voidaan nähdä olevan läheistä sukua Algoritmille 1 ja sitä kautta Stewartin ja Bodinin algoritmille. Perusajatuksena Algoritmissa 2 on se, että aloitusreitiksi valitaan pistejoukolle muodostettu konveksi verho. Tämän jälkeen jäljelle jääville pisteille muodostetaan konveksi verho ja tämän konveksin verhon sisältämien pisteiden joukosta liitetään reittiin se piste, jonka synnyttämä kustannus on pienin. Pisteen reittiin liittämisen jälkeen konveksi verho korjataan ja seuraava liitettävä piste valitaan jälleen sen sisältämien pisteiden joukosta. Sama toistuu, kunnes kaikki pisteet on liitetty reittiin.

Seuraavassa kuvataan algoritmin vaiheet yleisellä tasolla:

1. Muodostetaan pistejoukolle konveksi verho ja otetaan se osareitiksi. Kutsutaan tätä reittiä *R*:ksi.
2. Muodostetaan konveksin verhon sisälle jääville pisteille konveksi verho. Kutsutaan tätä verhoa *S*:ksi.
3. Jokaiselle *S*:n pisteelle *k* etsitään *R*:n peräkkäiset pisteet *i* ja *j*, joille $c_{ik} + c_{kj} - c_{ij}$ on pienin. Nämä lisäyskohdat tallennetaan pisteen tietoihin. Lisäksi otetaan talteen tietoa siitä, millä pisteellä kyseinen arvo on pienin ja kutsutaan tätä pistettä *b*:ksi ja sen lisäysvälin pisteitä *a*:ksi ja *c*:ksi.
4. Lisätään piste *b* *R*:ään pisteiden *a* ja *c* väliin.
5. Poistetaan piste *b* *S*:stä ja korjataan *S* niin, että se on jälleen osareittiin kuulumattomien pisteiden konveksi verho.
6. Tutkitaan sellaiset *S*:n pisteet *k*, joilla löytyy jokin lisäyskohta, osareitin lisäyksessä syntyneiden välien (*a*, *b*) ja (*b*, *c*) suhteen. Mikäli lisäyksen (*a*, *k*, *b*) aiheuttama kustannus $c_{ak} + c_{kb} - c_{ab}$ tai lisäyksen (*b*, *k*, *c*) aiheuttama kustannus $c_{bk} + c_{kc} - c_{bc}$ on pienempi kuin pisteelle *k* tallennetun lisäyskohdan aiheuttama kustannus, niin pisteen lisäyskohdaksi tallennetaan pienemmän kustannuksen aiheuttanut. Pisteet, joilla vanhana lisäyskohtana on ollut (*a*, *c*), tutkitaan myös vain välien (*a*, *b*) ja (*b*, *c*) suhteen ja lisäyskohdaksi otetaan näistä kahdesta se, joka aiheuttaa pienemmän kustannuksen. Pidetään yllä tietoa siitä, millä pisteellä lisäys on edullisin. Kutsutaan tätä pistettä b_u:ksi ja sen lisäysväli pisteitä a_u:ksi ja c_u:ksi.
7. Tutkitaan sellaiset *S*:n pisteet *k*, joilla ei löydy lisäyskohtaa seuraavalla tavalla muodostettavan rajoitteen määräämien osareitin välien suhteen. Valitaan sellaiset *S*:n pisteet *m* ja *n*, joille on tallennettu lisäyskohta ja joista

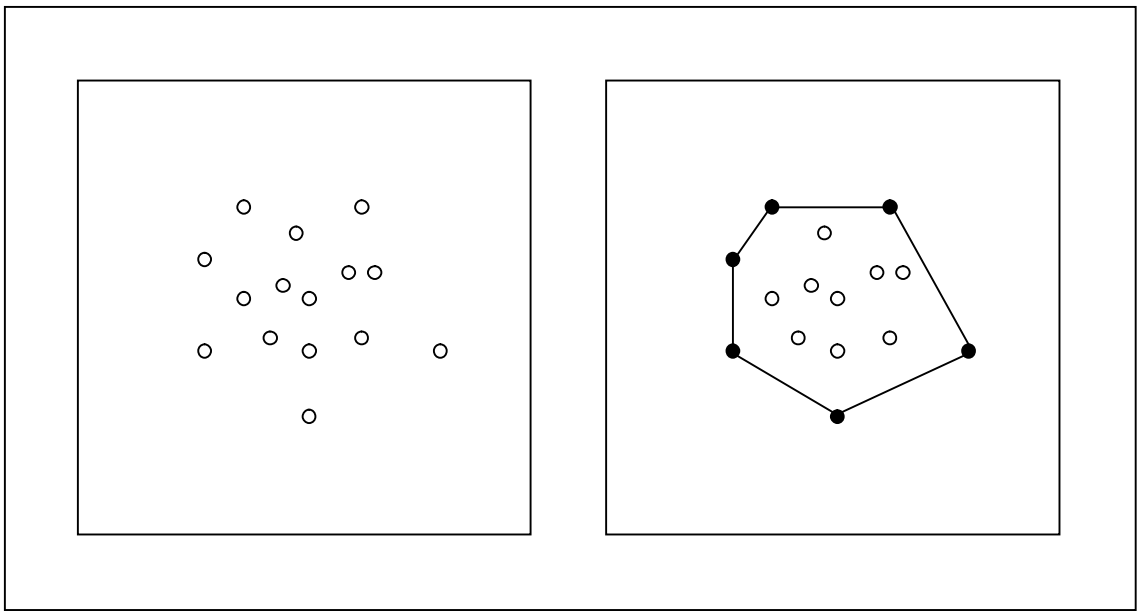
m edeltää *k*:ta *S*:ssä ja *n* seuraa *k*:ta *S*:ssä. Valitaan näiden pisteiden *m* ja *n* lisäyskohtien toisiinsa nähden kauimmaiset *R*:n pisteet ja suoritetaan tutkinta näiden pisteiden rajoittamia välejä vasten. Näiden *R*:n välien joukosta valitaan kullekin *k*:lle pienimmän kustannuksen aiheuttava lisäyskohta. Mikäli jollekin *k*:lle löytyy lisäyskohta, jonka aiheuttama kustannus on pienempi kuin tallessa olevan lisäyksen (a_u, b_u, c_u) aiheuttama kustannus, niin asetetaan kyseinen k b_u:ksi ja kyseinen lisäyskohta (a_u, b_u, c_u):ksi.

8. Lisätään piste b_u osareittiin pisteiden a_u ja c_u väliin.
9. Kutsutaan pistettä a_u a:ksi, pistettä b_u b:ksi ja pistettä c_u c:ksi.
10. Toistetaan kohtia 5–9, kunnes pisteitä ei ole enää jäljellä.

Tämän algoritmin vaativuus voidaan parhaiten selvittää kokeellisesti. Seuraavalla kuvasarjalla selvennetään algoritmin toimintaa. Kuvasarjan ei kuvaa oikeaa tilannetta, joten sen esittämät lisäyskohdat eivät ole välttämättä parhaita mahdollisia.

Lähtötilanne ja konveksin verhon luonti

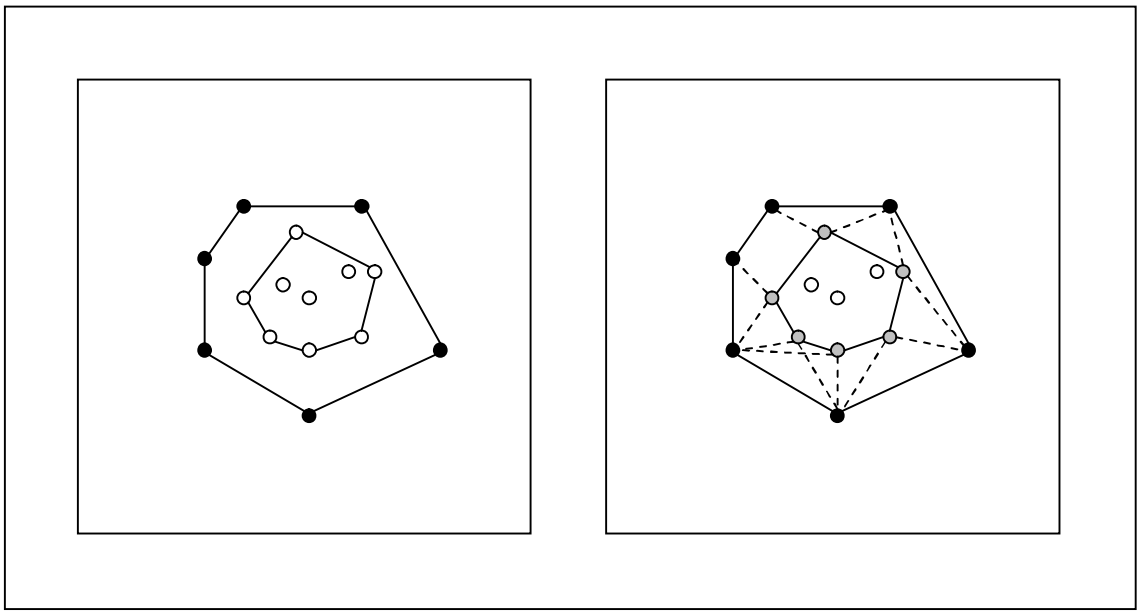
Algoritmi valitsee konveksin verhon osareitiksi (kuva 11).



Kuva 11. Alkuperäiselle pistesarjalle muodostetaan konveksi verho.

Sisemmän konveksin verhon luonti ja lisäyskohtien etsintä

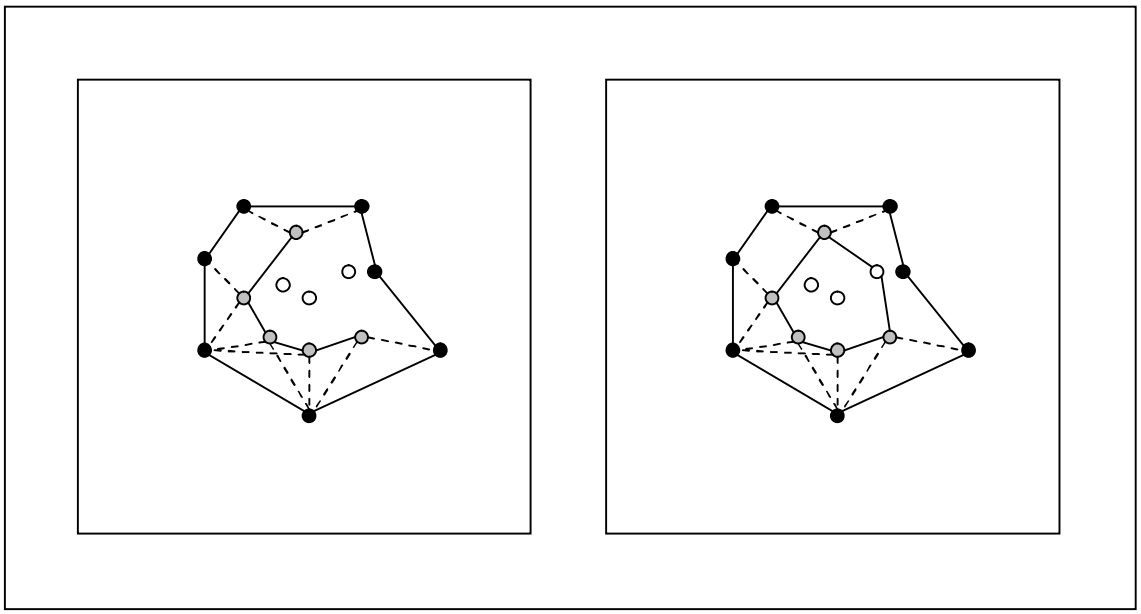
Algoritmi muodostaa osareitin sisälle jääneille pisteille konveksin verhon ja etsii tämän verhon pisteille parhaat mahdolliset lisäyspaikat osareitissä (kuva 12).



Kuva 12. Luodaan sisempi konveksin verho ja etsitään lisäyskohdat sen pisteille.

Pisteen lisäys osareittiin ja sisemmän konveksin verhon korjaus

Sen jälkeen, kun kaikille osareitin sisälle jäävän konveksin verhon pisteille on laskettu paras mahdollinen lisäyskohta, liitetään pisteistä pienimmän kustannuksen aiheuttava osareittiin ja korjataan osareitin sisälle jäävien pisteiden konveksi verho (kuva 13).

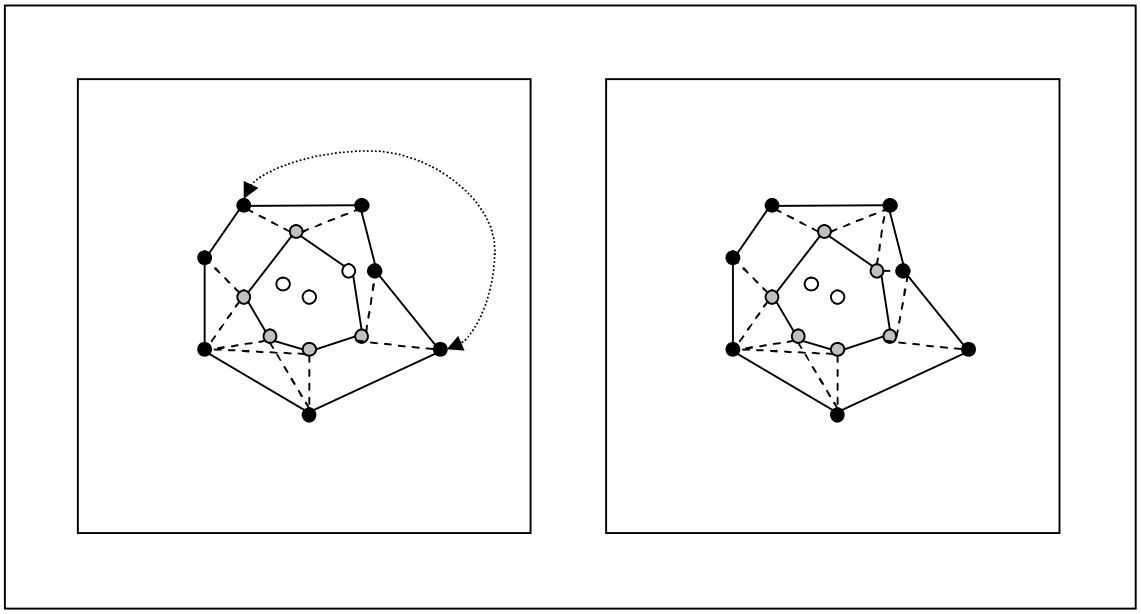


Kuva 13. Liitetään piste osareittiin ja korjataan konveksin verho.

Uusien lisäyskohtien valinta ja rajoitteen toiminta

Etsittäessä seuraavaa lisättävää pistettä tutkitaan ensin ne pisteet, joilla löytyy jokin lisäyskohta, osareitin kahta uutta sivua vasten ja tehdään tarvittavat muutokset. Kuvassa

14 tämä näkyy siten, että konveksin verhon uuden valkoisella merkityn pisteen alapuolella olevan pisteen lisäyskohta on muuttunut, kun sitä verrataan kuvan 13 esittämään tilanteeseen. Tämän jälkeen uudelle pisteelle luodaan tutkimusrajoite ja se tutkitaan rajoitteen sisältämiä osareitin välejä vasten. Tutkimusrajoitetta kuvassa ilmentää pisteviiva, jonka päissä olevat nuolet merkitsevät tutkittavan reitin alueen alkua ja loppua. Kuvan tilanne tulee tulkita siten, että uusi piste tutkitaan osareitin välejä vasten aloittaen alempana olevan nuolen merkitsemästä lisäyskohdasta ja lopettaen ylemmän nuolen merkitsemään lisäyskohtaan. Pisteelle otetaan ylös pienimmän kustannuksen aiheuttava lisäyskohta tutkimusrajoitteen antamalta osareitin alueelta.



Kuva 14. Etsitään uudet lisäyskohdat ja käytetään hakurajoitetta.

Rajoitteen hyödyllisyys tulee ilmi paremmin suuria pistejoukkoja käsiteltäessä. Toisaalta silloin on riskinä se, että osareitti tekee rajoitteen osoittamalla välillä jonkinlaisen mutkan, joka voi sisältää lisäyspaikan etsinnän kannalta täysin turhia pisteitä.

4.4.3 Algoritmi 3

Algoritmin 2 vaiheessa 7 tutkitaan konveksin verhon uudet pisteet. Tämä tehtiin luomalla uuden pisteen eri puolilta löytyvien jo tutkittujen pisteiden lisäyskohtien avulla rajoite osareittiin ja tutkimalla piste tuon rajoitteen sisältämiä mahdollisia lisäyskohtia vasten. Tätä vaihetta pyritään Algoritmissa 3 tehostamaan luomalla seuraavanlainen lisärajoite.

Lisärajoitteen toiminta

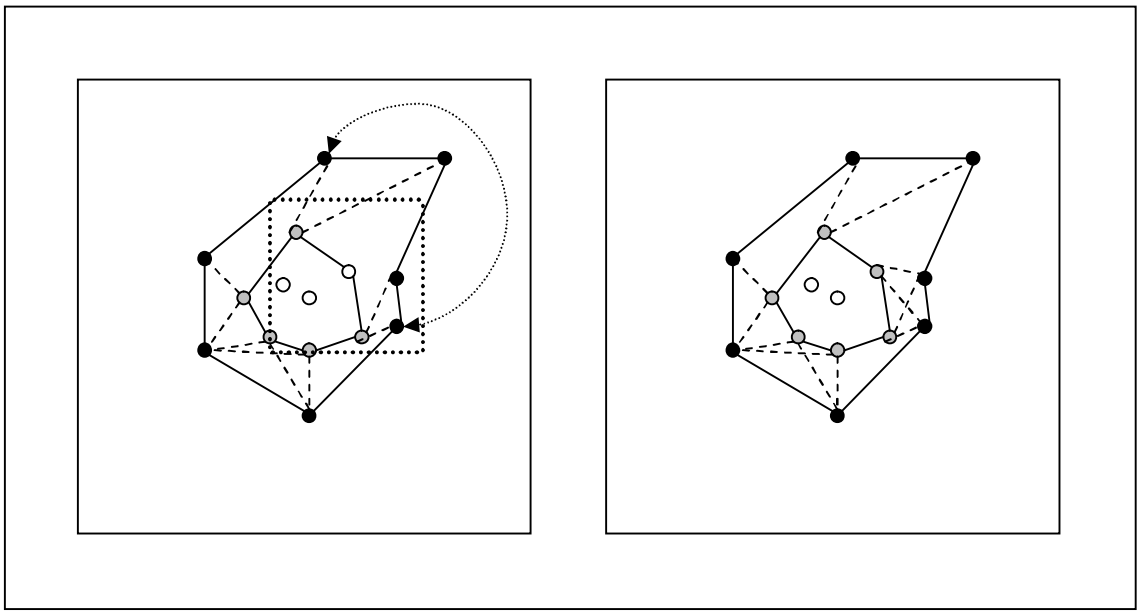
Aina kun uudelle pisteelle löydetään lisäyskohta, jonka aiheuttama kustannus on pienempi kuin edellisen valitun lisäyskohdan, luodaan lisäyskohdan pidemmän sivun avulla neliörajoite ja jatkossa sellaiset osareitin lisäyskohdat, joissa sekä alku- että loppupiste eivät ole tuon neliörajoitteen sisällä, ohitetaan. Tutkittava piste on neliörajoitteen keskellä ja neliön sivun pituus lasketaan seuraavalla tavalla.

Tutkitaan, mikä on suurin ero tutkittavan pisteen ja lisäyskohdan osareittiin kuuluvien pisteiden x- ja y-koordinaattien välillä. Esimerkiksi tilanteessa, jossa tutkittavan pisteen koordinaatit ovat (1, 2) ja lisäyskohdan pisteiden koordinaatit ovat (3, 5) ja (4, 7), niin suurin x-koordinaattien välinen erotus on 3 ja suurin y-koordinaattien välinen erotus on 6, joten suurimmaksi eroksi siis saadaan 6. Kutsutaan jatkossa tätä suurinta erotusarvoa *a*:ksi. Seuraavaksi lasketaan kyseisen arvon avulla neliörajoitteen puolikkaan sivun pituus. Tämä tehdään kaavalla $(3/2) \cdot a + 1$.

Ideana arviossa on se, että halutaan laskea arvo, joka olisi riittävän pieni, mutta kuitenkin niin suuri, että paremmat lisäyskohdat tulevat sen sisälle. Arvio on saatu siten, että approksimoidaan sellaisen tasasivuisen kolmion hypotenuusan pituutta, jonka kummatkin kateetit ovat *a*:n pituisia. Käytännössä siis arvioidaan arvoa, jonka kaava $\sqrt{2 \cdot a^2}$ palauttaisi.

Ensimmäisen lisäyskohdan valinnan jälkeen neliörajoite lasketaan siten, että laskennassa huomioidaan ensimmäinen ja viimeinen mahdollinen lisäyskohta ja edellisen osareittiin tehdyn lisäyksen synnyttämät uudet lisäyskohdat. Kunkin kohdan perusteella lasketuista neliön sivun puolikkaasta pituudesta valitaan se, joka on pienin. Tämä tehdään siksi, että kyseiset lisäyskohdat tiedetään sijaitseviksi tutkittavalla osareitin alueella. Tällöin on mahdollista, että saadaan heti alussa laskettua rajoite mahdollisimman tarkaksi.

Arvion toteutuksen hyvänä puolena on erityisesti se, että joka kerta, kun pisteelle löytyy parempi lisäyskohta, niin neliörajoite tarkentuu automaattisesti. Neliörajoite toimii siis eräänlaisena jatkuvasti tarkentuvana tähtäimenä. Kuva 15 selventää neliörajoitteen toimintaa.



Kuva 15. Neliörajoite sulkee sisällensä järkevät lisäyskohdat uudelle pisteelle.

4.4.4 Algoritmi 4

Algoritmissa 4 on tehostettu Algoritmin 3 toimintaa seuraavalla tavalla. Konveksin verhon muodostus on toteutettu siten, että Grahamin pyyhkäisymenetelmän tarvitsema kulmapiste valitaan mahdollisimman keskeltä pistejoukkoa. Ajatuksena tässä on se, että kun kulmapiste valitaan keskeltä, niin kestää kauemmin ennen kuin se valitaan osareittiin lisättäväksi pisteeksi, ja kulmapistejärjestys täytyy tehdä uudelleen.

Algoritmiin 4 sisältyy myöhemmin tarkemmin kuvattavia ongelmia, joiden vuoksi se ei ole täysin vakaa. Tällä tarkoitetaan sitä, että algoritmin läpimenoa ei kaikilla syötteillä voida taata. Nämä ongelmat lisääntyvät tilanteissa, joissa käsiteltäviä pisteitä on vähän. Tämän vuoksi tehdyssä toteutuksessa on päädytty ratkaisuun, jossa ainakin ensimmäinen kulmajärjestys tehdään menetelmällä, jossa kulmapiste valitaan mahdollisimman keskeltä, mutta myöhemmässä vaiheessa siirrytään käyttämään menetelmää, jossa kulmapiste valitaan kuten Algoritmeissa 2 ja 3. Tällä tavoin toimittaessa kaikki testimateriaalin pistesarjat saatiin käsitellyiksi, mutta takuuta siitä, että ongelmia ei tulisi jollain toisella sarjalla, ei voida antaa.

Syy siihen, miksi Algoritmi 4 on otettu mukaan tähän tutkimukseen, on se, että sen avulla voidaan paremmin arvioida Algoritmien 2, 3 ja 4 perustana olevan idean hyvyyttä. Algoritmin 4 kehitystyötä ei siis ole viety täysin loppuun.

4.4.5 Algoritmi 5

Algoritmi 5 on Algoritmista 4 tehty huippunopea muunnos. Sen tarkoituksena on luoda algoritmien vertailulle alaraja, siinä missä Algoritmi 1 toimii ylärajana. Algoritmissa 5 ei pyritä säilyttämään reitin tarkkuutta, vaan siinä optimoidaan nopeutta.

Perusideana on, että toiminta on muuten kuten Algoritmissa 4, mutta osareitin sisällä olevan konveksin verhon uusien pisteiden lisäyskohta etsitään sen kummaltakin puolelta löytyvien lisäyskohdan omaavien pisteiden lisäyskohtien ja osareittiin edellisen lisäyksen johdosta syntyneiden kahden lisäyskohdan joukosta. Tämä tarkoittaa sitä, että jokainen uusi piste tutkitaan neljää lisäyskohtaa vasten. Tällöin pisteiden lisäyskohtien etsinnästä tulee lineaarista ja algoritmin vaativimmaksi osuudeksi jää konveksin verhon muodostus ja korjaus.

4.5 Omien algoritmien tarkempi kuvaus ja niiden ongelmat

Seuraavassa kuvataan tarkemmin omien algoritmien toimintaa. Käytännössä tarkempi kuvaus keskittyy Algoritmeihin 1 ja 2.

4.5.1 Algoritmi 1

Vaihe 1: aloitus

Ensimmäisessä vaiheessa tehdään alkuvalmistelut varsinaisen algoritmin pääsilmukan toimintaa varten. Vaiheessa tehdään seuraavat toimenpiteet:

1. Luetaan tiedot pisteistä tiedostosta ja tallennetaan ne alkioina taulukkoon *Pisteet*.
2. Haetaan taulukon alkioista se, jolla on pienin y-koordinaatin arvo. Mikäli useammalla kuin yhdellä alkiolla on sama pienin y-koordinaatin arvo, niin valitaan alkioista se, jolla on pienin x-koordinaatin arvo. Kutsutaan alkiota kulmapisteeksi.
3. Lajitellaan taulukon *Pisteet* alkiot haetun kulmapisteen suhteen nousevaan kulmajärjestykseen käyttäen pikalajittelua.
4. Luodaan alkioille konveksi verho Grahamin pyyhkäisymenetelmällä siten, että konveksiin verhoon tulevat alkiot tallennetaan listaan *Reitti* ja merkitään poistettaviksi taulukossa *Pisteet*.

Taulukkoa *Pisteet* voidaan lukea listana siten, että poistettavaksi merkityt alkiot ohitetaan, joten reittiin lisättyjä pisteitä ei tarvitse poistaa siitä. Tämän ominaisuuden hyödyllisyys tulee esille jatkossa, sillä aina kun taulukosta lisätään alkio *Reitti*-listaan, niin poisto taulukosta tapahtuu yhdellä operaatiolla ja seuraavan kierroksen *Pisteet*-

taulukon lukuoperaatiot pienenevät automaattisesti yhdellä. Algoritmissa 2 alkioiden konkreettinen poistaminen on välttämätöntä, sillä siinä taulukkoa joudutaan järjestämään uudelleen.

Vaihe 2: tutkinta suoritetaan ensimmäisen kerran

Vaiheessa 2 etsitään kullekin *Reitti*-listaan tallennetun osareitin sisäpuolelle jääneelle pisteelle pienimmän kustannuksen aiheuttava lisäyskohta osareitissä. Se alkio, jolle löytynyt lisäyskohta aiheuttaa kaikista pienimmän kustannuksen, valitaan lisättäväksi alkioksi.

1. Luetaan *Pisteet*-taulukkoa listana alkaen ensimmäisestä alkiosta, jota ei ole merkitty poistettavaksi. Kyseinen alkio on tallennettu *Pisteet*-tietorakenteeseen. (Käytetään tästä alkiosta jatkossa nimeä tutkittava alkio.)
2. Käydään läpi *Reitti*-listan alkiot ja tallennetaan tutkittavalle alkiolle edeltävä ja seuraava alkio pienimmän kustannuksen aiheuttavassa liitoksessa.
3. Otetaan tutkittava alkio talteen lisättävänä alkiona.
4. Valitaan uudeksi tutkittavaksi alkioksi tutkittavaa alkiota seuraava alkio.
5. Käydään läpi *Reitti*-listan alkiot ja tallennetaan tutkittavalle alkiolle edeltävä ja seuraava alkio pienimmän kustannuksen aiheuttavassa liitoksessa.
6. Mikäli tutkittavan alkion liitoksen kustannus on pienempi kuin lisättävän alkion, tulee siitä uusi lisättävä alkio.
7. Toistetaan kohdat 4, 5 ja 6, kunnes kaikki taulukon *Pisteet* sellaiset alkiot, joita ei ole merkitty poistettavaksi, on käyty läpi.

Vaihe 3: valittu alkio poistetaan käsiteltävien joukosta ja lisätään reittiin

Vaiheessa 3 lisättävä alkio merkitään poistettavaksi, jolloin se poistuu *Pisteet*-taulukon listakäsittelystä. Tämän jälkeen alkio lisätään osareittiin.

1. Merkitään valittu lisättävä alkio poistettavaksi *Pisteet*-taulukkoon.
2. Tutkitaan, mikä on lisättävälle alkiolle talletettu liitoksessa edeltävä alkio.
3. Lisätään lisättävä alkio listaan *Reitti* liitoksessa edeltävän alkion eteen.

Vaihe 4: tutkinta suoritetaan uudelleen

Vaiheessa 4 etsitään seuraava lisättävä alkio. Mikäli *Pisteet*-taulukossa ei enää ole sellaisia alkioita, joita ei ole merkitty poistettaviksi, niin lopetetaan käsittely.

1. Luetaan *Pisteet*-taulukkoa listana alkaen ensimmäisestä alkiosta, jota ei ole merkitty poistettavaksi. Kyseinen alkio on tallennettu *Pisteet*-tietorakenteeseen. (Käytetään tästä alkiosta jatkossa nimeä tutkittava alkio.)
2. Mikäli tutkittavalla alkiolla on jo lisäyskohta, joka on säilynyt edellisen *Reitti*-listaan tehdyn lisäyksen jälkeen, niin verrataan lisäyskohtaa edellisen *Reitti*-listaan tehdyn lisäyksen synnyttämiin kahteen uuteen lisäyskohtaan. Tutkittavalle alkiolle tallennetaan lisäyskohdaksi pienimmän kustannuksen aiheuttava lisäyskohta, mikäli sellainen löytyy uusien lisäyskohtien avulla.
3. Mikäli tutkittavan alkion vanha lisäyskohta tuhoutui *Reitti*-listaan tehdyssä lisäyksessä, niin valitaan alkiolle lisäyskohdaksi edellä mainituista kahdesta uudesta lisäyskohdasta pienemmän kustannuksen aiheuttava lisäyskohta.
4. Valitaan tutkittava alkio lisättäväksi, mikäli sille määrätyn lisäyskohdan aiheuttama kustannus on pienin vaiheessa neljä käsiteltyjen alkioiden lisäyskohtien aiheuttamista kustannuksista.
5. Valitaan uudeksi tutkittavaksi alkioksi tutkittavaa alkiota seuraava alkio *Pisteet*-taulukon listakäsittelystä.
6. Toistetaan kohtia 2, 3, 4 ja 5, kunnes kaikki taulukon *Pisteet* sellaiset alkiot, joita ei ole merkitty poistettavaksi, on käsitelty.

Vaihe 5: tutkinta viedään loppuun

Vaiheessa 5 toistetaan vaiheita 3 ja 4 niin kauan, kuin *Pisteet*-taulukossa on alkioita, joita ei ole merkitty poistettaviksi. Tämän jälkeen reitti on valmis.

4.5.2 Algoritmi 2

Vaihe 1: aloitus

Ensimmäisen vaiheen tarkoituksena on tehdä alkuvalmistelut varsinaisen algoritmin pääsilmukan toimintaa varten. Vaiheessa tehdään seuraavat toimenpiteet:

1. Luetaan tiedot pisteistä tiedostosta ja tallennetaan ne alkioina taulukkoon *Pisteet*.
2. Haetaan taulukon alkioista se, jolla on pienin y-koordinaatin arvo. Mikäli useammalla kuin yhdellä alkiolla on sama pienin y-koordinaatin arvo, niin valitaan niistä se, jolla on pienin x-koordinaatin arvo. Kutsutaan alkiota kulmapisteeksi.
3. Lajitellaan alkiot haetun kulmapisteen suhteen nousevaan kulmajärjestykseen käyttäen pikalajittelua.

4. Luodaan pisteille konveksi verho Grahamin pyyhkäisymenetelmällä siten, että konveksiin verhoon tulevat pisteet tallennetaan listaan *Reitti* ja merkitään poistettaviksi taulukossa *Pisteet*.
5. Poistetaan taulukosta *Pisteet* ne alkiot, jotka on merkitty poistettaviksi.
6. Toistetaan kohdat 2 ja 3.
7. Toistetaan kohta 4 siten, että konveksi verho tallennetaan listaan *Sisä* ja alkioita ei merkitä poistettaviksi.

Vaihe 2: tutkinta suoritetaan ensimmäisen kerran

Vaiheen 2 tarkoitus on tutkia ensimmäisen kerran, mikä *Sisä*-listaan tallennetuista alkioista lisätään *Reitti*-listaan ja mihin kohtaan lisäys tehdään:

1. Luetaan *Sisä*-listan ensimmäinen alkio. (Käytetään tästä alkiosta jatkossa nimeä tutkittava alkio.)
2. Käydään läpi *Reitti*-listan alkiot ja tallennetaan tutkittavalle alkiolle edeltävä ja seuraava alkio pienimmän kustannuksen aiheuttavassa liitoksessa.
3. Otetaan tutkittava alkio talteen lisättävänä alkiona.
4. Valitaan uudeksi tutkittavaksi alkioksi listasta *Sisä* tutkittavaa alkiota seuraava alkio ja toistetaan sille kohta 2.
5. Mikäli tutkittavan alkion liitoksen aiheuttama kustannus on pienempi kuin lisättävän alkion liitoksen aiheuttama kustannus, tulee tutkittavasta alkiosta uusi lisättävä alkio.
6. Toistetaan kohtia 4 ja 5, kunnes kaikki listan *Sisä* alkiot on käyty läpi.

Vaihe 3: valittu alkio poistetaan käsiteltävien joukosta ja konveksi verho korjataan

Vaiheessa 3 lisättävä alkio poistetaan *Sisä*-listasta ja lista korjataan niin, että se on taas osareittiin kuulumattomien alkioiden konveksi verho.

1. Otetaan talteen lisättävää alkiota edeltävä ja seuraava alkio *Sisä*-listassa. Mikäli lisättävä alkio on listan viimeinen alkio, niin seuraava alkio on listan ensimmäinen alkio ja vastaavasti, jos lisättävä alkio on listan ensimmäinen alkio, niin sitä edeltävä alkio on listan viimeinen alkio.
2. Merkitään valittu alkio poistettavaksi *Pisteet*-taulukkoon.
3. Mikäli valittu alkio on toiminut kulmapisteenä konveksia verhoa luodessa, poistetaan taulukosta *Pisteet* poistettaviksi merkityt alkiot, valitaan uusi kulmapiste samalla tavalla kuin valittiin vaiheessa yksi ja järjestetään alkiot tuon kulmapisteen mukaiseen kulmajärjestykseen.

4. Konveksiin verhoon mahdollisesti tulevat uudet alkiot sijoittuvat siitä poistettua alkiota edeltävän ja seuraavan alkion väliin. Itse korjaus voidaan tehdä Grahamin pyyhkäisymenetelmää soveltamalla. Korjauksen suorittaminen on kuvattu tarkemmin jo aiemmin. Ennen korjausta lisättävä alkio poistetaan *Sisä*-listasta.

Vaihe 4: lisättävä alkio lisätään reittiin

Vaiheessa 4 lisättävä alkio lisätään listaan *Reitti*.

1. Lisättävän alkion tiedoista tarkistetaan, mikä on lisättävää alkiota liitoksessa edeltävä alkio.
2. Lisätään lisättävä alkio listaan *Reitti* edeltävän alkion eteen.

Vaihe 5: tutkinta suoritetaan uudelleen

Vaiheessa 5 valitaan seuraava lisättävä alkio, mikäli alkioita on vielä jäljellä.

1. Luetaan listaa *Sisä* siten, että aloitetaan viimeksi poistettua alkiota välittömästi seuranneesta alkiosta. Käytetään tästä alkiosta jatkossa nimeä alkualkio ja tutkittavasta alkiosta nimeä tutkittava alkio.
2. Mikäli tutkittavalla alkiolla on jo lisäyskohta, joka on säilynyt edellisen *Reitti*-listaan tehdyn lisäyksen jälkeen, niin verrataan lisäyskohtaa edellisen *Reitti*-listaan tehdyn lisäyksen synnyttämiin kahteen uuteen lisäyskohtaan. Tutkittavalle alkiolle tallennetaan lisäyskohdaksi pienimmän kustannuksen aiheuttava lisäyskohta, mikäli sellainen löytyy uusien lisäyskohtien avulla.
3. Mikäli tutkittavan alkion vanha lisäyskohta tuhoutui *Reitti*-listaan tehdyssä lisäyksessä, niin valitaan alkiolle lisäyskohdaksi edellä mainituista kahdesta uudesta lisäyskohdasta pienemmän kustannuksen aiheuttava.
4. Mikäli alkiota ei ole aikaisemmin tutkittu, niin sitä tutkitaan listan *Reitti* alkiota vasten siten, että *Reitti*-listan lukeminen aloitetaan alkiosta, joka on tutkittavaa alkion *Sisä*-listassa edeltävän alkion liitoksessa edeltävä *Reitti*-listan alkio ja lopetetaan kohdassa 1 mainitun alkualkion liitoksessa seuraavaan *Reitti*-listan alkioon. Tutkittavalle alkiolle valitaan tältä väliltä se lisäyskohta, joka aiheuttaa pienimmän kustannuksen.
5. Mikäli tutkittavan alkion liitoksen kustannus on pienempi kuin lisättävän alkion, tulee siitä uusi lisättävä alkio.
6. Valitaan uudeksi tutkittavaksi alkioksi tutkittavaa alkiota seuraava alkio.

7. Toistetaan vaiheet 2, 3, 4, 5 ja 6 niin kauan, että kaikki *Sisä*-listan alkiot on tutkittu.

Vaihe 6: tutkinta viedään loppuun.

Vaiheessa 6 toistetaan vaiheita 3, 4 ja 5 niin kauan kuin *Pisteet*-taulukossa on alkioita, joita ei ole merkitty poistettaviksi.

Algoritmin 2 heikkoudet

Algoritmi 2 sisältää kaksi merkittävää rakenteellista heikkoutta.

Osareitin kaareutuminen

Osareitinreitin sisällä olevan konveksin verhon kokonaan uusille pisteille etsitään mahdollinen lisäyskohta käyttämällä rajoitteena pisteen kummaltakin puolelta löytyvien jo tutkittujen pisteiden lisäyskohtia. Tämä ratkaisu on useimmiten hyvin toimiva, mutta kun reittiin lisättyjen pisteiden määrä kasvaa, tapahtuu reitin kaareutumista. Käytännössä tämä tarkoittaa sitä, että reittiin voi syntyä hyvin syviä ja kapeita mutkia. Osareitin mutkaisuudesta seuraa, että käytettävästä rajoitteesta huolimatta tutkitaan monia turhia lisäyskohtia.

Konveksin verhon muodostus

Algoritmin kantavana ideana on hyödyntää osareitin sisäpuolella olevaa konveksia verhoa eräänlaisena tutkimuksen rajoitteena. Tähän sisältyy myös algoritmin selvin heikkous.

Tarvittava konveksi verho muodostetaan Grahamin pyyhkäisymenetelmällä, jossa oleellista on valita pistejoukolle kulmapiste ja lajitella tämän kulmapisteen suhteen pisteet nousevaan kulmajärjestykseen. Algoritmin 2 tarkemman kuvauksessa vaiheen 3 kohdassa 3 mainitaan, että jos osareittiin lisättävä piste on konveksin verhon kulmapiste, niin kulmapiste tulee valita uudestaan ja pisteille on tehtävä uusi kulmajärjestys. Kulmajärjestyksen muodostaminen tehdään pikalajittelulla, joten se on keskimäärin $O(n \cdot \log(n))$ vaativuuden vaativa operaatio.

On vaikea ennustaa, kuinka usein uusi kulmajärjestys täytyy tehdä kunkin pistesarjan kohdalla. On kuitenkin selvää, että tämä algoritmin ominaisuus heikentää sen suorituskykyä merkittävästi.

4.5.3 Algoritmi 3

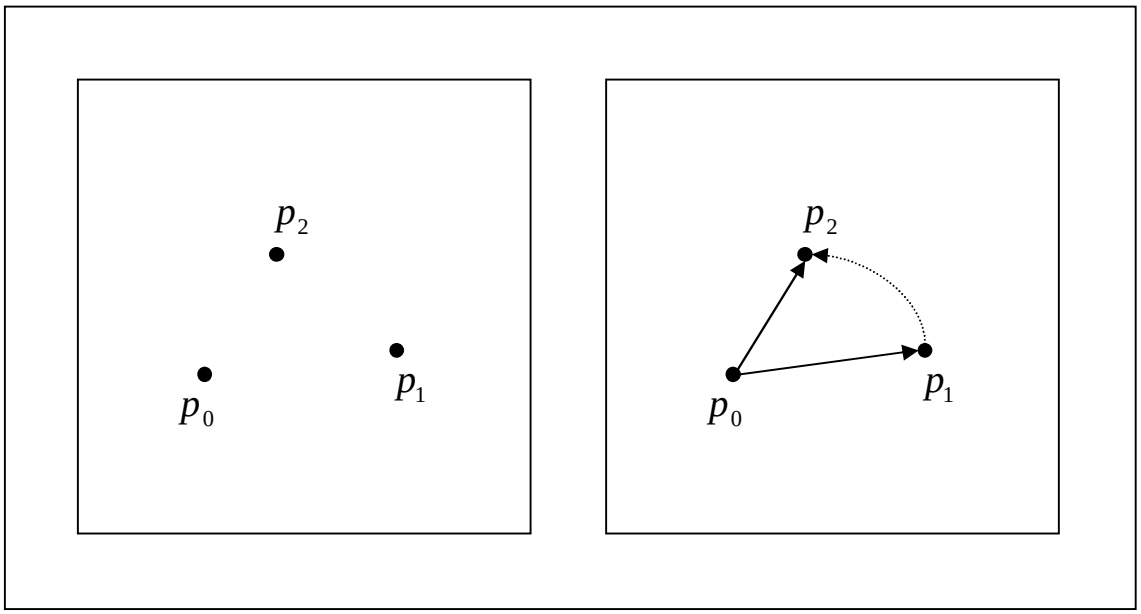
Algoritmi 3 on muuten samanlainen kuin Algoritmi 2, mutta siinä käytetään aiemmin kuvattua neliön mallista lisärajoitetta. Lisärajoitteen avulla pystytään osittain välttämään edellä mainitun osareitin kaareutumisen aiheuttama ongelma. Käytännössä alkeisoperaatioiden määrässä ei tapahdu vähenemistä Algoritmiin 2 verrattuna, mutta alkeisoperaation laatu muuttuu työläästä helpommaksi kokonaislukujen vertailuoperaatioksi.

4.5.4 Algoritmi 4

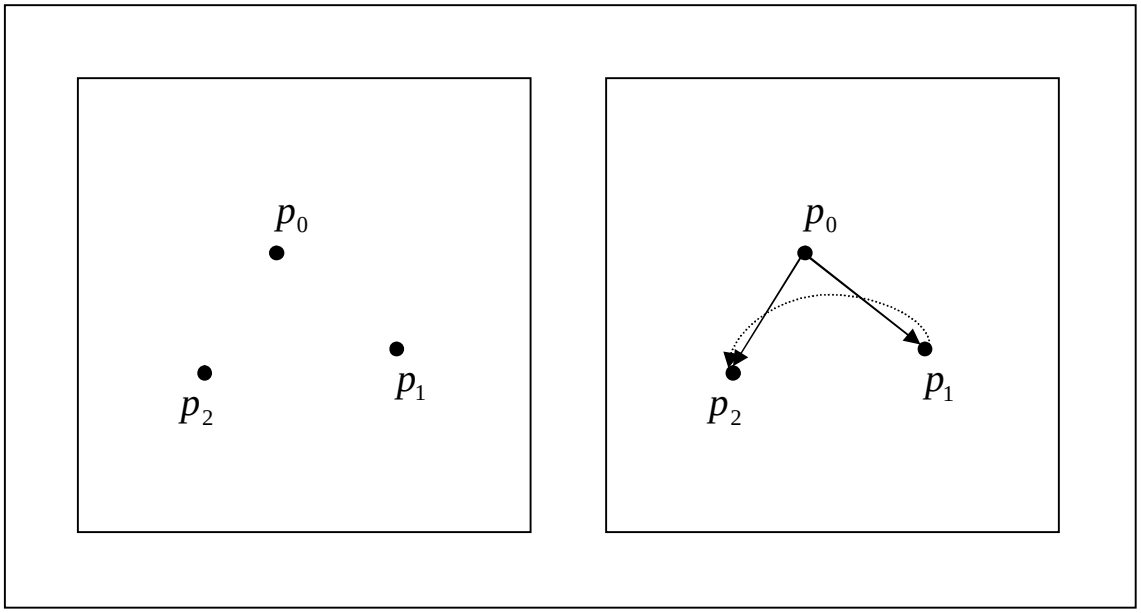
Algoritmi 4 toimii muuten kuten Algoritmi 3, mutta siinä pisteiden kulmajärjestys tehdään mahdollisimman keskeltä löytyvän pisteen mukaan. Tästä seuraa seuraavia ongelmia.

Pisteiden välisen kulman tutkiminen

Konveksin verhon muodostusta varten pisteet laitetaan kulmajärjestykseen tiettyä pistettä referenssipisteenä käyttäen. Kulmajärjestyksen tutkimisessa hyödynnetään ristituloa, jonka avulla voidaan laskea, onko pisteiden $p_0 = (x_0, y_0)$ ja $p_2 = (x_2, y_2)$ välinen jana vasta- vai myötäpäivään pisteiden p_0 ja $p_1 = (x_1, y_1)$ välisestä janasta. Silloin kun kulmajärjestys tehtiin pienimmän y-koordinaatti arvon omaavan pisteen suhteen, menetelmä toimi moitteettomasti, mutta mahdollisimman keskeltä valitun referenssipisteen mukaan laskettaessa se ei enää toimikaan oikein. Tilannetta voidaan hahmottaa kuvien 16 ja 17 avulla.



Kuva 16. Referenssipisteenä toimii pienimmän y-koordinaatin arvon omaava piste.



Kuva 17. Referenssipisteenä toimii mahdollisimman keskeltä löytyvä piste.

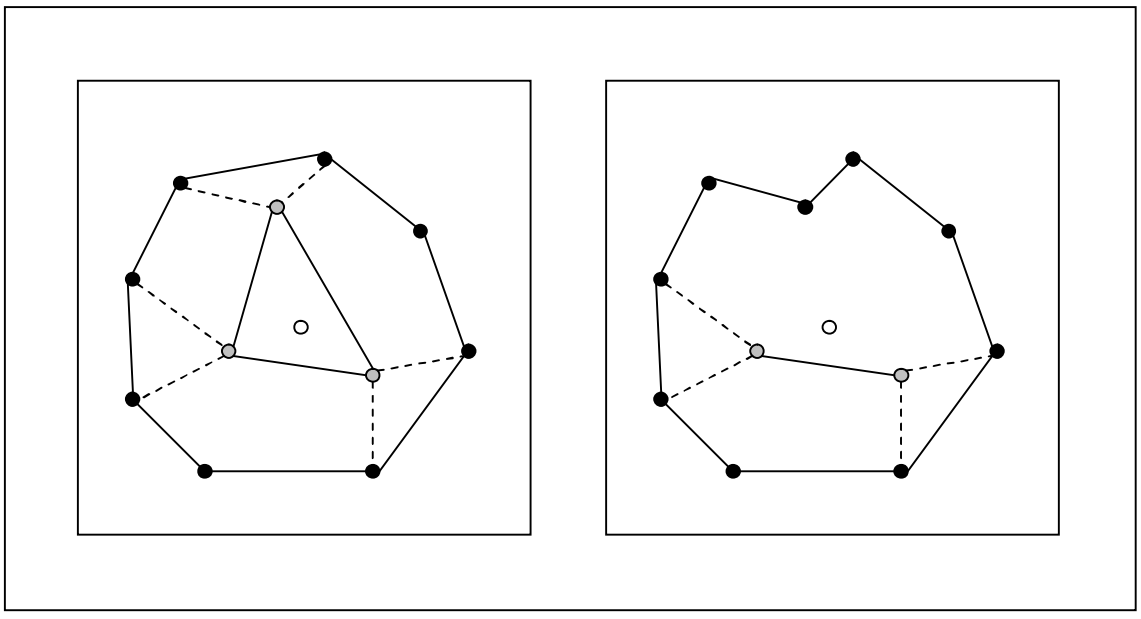
Silloin kun referenssipisteenä on pienimmän y-koordinaatin omaava piste tai jokin muu samalla periaatteella valittu piste, ovat kaikki pisteet referenssipisteen suhteen toisiinsa nähden yksiselitteisesti joko oikealla tai vasemmalla puolella tai samalla selaussuoralla. Tällöin ristitulolla tehty tarkastelu antaa oikean tuloksen, kuten kuvasta 16 voidaan päätellä. Silloin kun referenssipiste valitaan mahdollisimman keskeltä, seuraa välittömästi seuraava ongelma.

Tarkasteltaessa sitä, onko jana toisen janan suhteen oikealla vai vasemmalla, täytyy huomioida se, että kulmajärjestys kulkee ympyrän muotoisesti vastapäivään referenssipisteen suhteen. Tällöin ristitulo tarkastelu ei enää annakaan oikeaa tulosta kaikissa tapauksissa.

Tämä ongelma voidaan kiertää siten, että kun kulmajärjestys tehdään mahdollisimman keskeltä valitun referenssipisteen suhteen, niin ennen ristitulon käyttöä hoidetaan yksinkertaisilla x- ja y-koordinaatti tarkasteluilla hankalat tapaukset niin, että ristituloa käytetään vain pisteisiin, jotka sijaitsevat samassa neljänneksessä referenssipisteen suhteen. Tällä tarkoitetaan sitä, että esimerkiksi referenssipisteen vasemmalla puolella oleva piste saa automaattisesti suuremman kulma-arvon kuin referenssipisteen oikealla puolella oleva piste.

Referenssipisteen ajautuminen reunalle

Toinen merkittävä ongelma on referenssipisteen ajautuminen reunalle eli tilanne, jossa se tultaisiin liittämään osareitin sisäpuolella olevaan konveksiin verhoon. Kuva 18 esittää tätä tilannetta.



Kuva 18. Referenssipiste ajautuu reunalle.

Kuvan 18 tilanteessa osareitin sisäpuolella olevaa konveksia verhoa korjattaessa käydään läpi ne pisteet, joiden kulma-arvo on verhosta poistettua pistettä vastapäivään liikkuen edeltäneen ja seuranneen pisteen välissä. Koska referenssipisteellä on pienin kulma-arvo, se jää käsittelemättä.

Algoritmissa 4 tilanne hoidetaan siten, että tehdään uusi kulmajärjestys, mikäli osareittiin liitettävää pistettä edeltävän ja seuraavan pisteen välinen jana on vasemmalle edeltävän ja referenssipisteen väliseen janaan verrattuna. Lisäksi sen jälkeen, kun pisteitä on jäljellä vähemmän kuin 10000, siirrytään käyttämään kulmajärjestyksessä pienimmän y-koordinaatin arvon omaavan referenssipisteen mukaista menetelmää.

Edellä kerrotun perusteella on selvää, ettei Algoritmi 4 ole täysin vakaa ja viimeistelty. Sen avulla pystytään kuitenkin käsittelemään tutkimuksessa käytetty aineisto. Algoritmin 4 avulla saadaan suuntaa-antava vastaus sille, voidaanko Algoritmin 2 ongelmaksi mainittu konveksin verhon muodostusta tehostaa.

4.5.5 Algoritmi 5

Algoritmi 5 toimii muuten kuten Algoritmi 4, mutta sen osareitin sisäpuolella olevan konveksin verhon uusien pisteiden lisäyskohdan etsintä on muutettu. Algoritmissa 5 näille pisteille etsitään lisäyskohta pisteen kummaltakin puolelta löytyvän lisäyskohdan omaavan pisteen lisäyskohtien ja osareittiin edellisen lisäyksen johdosta syntyneiden kahden lisäyskohdan joukosta. Aiemmin mainitun mukaisesti Algoritmin 5 tarkoituksena on antaa alaraja Algoritmeissa 2, 3 ja 4 käytetyn idean suorituskyvylle.

5 Kokeellisen tutkimuksen tavoitteet ja suorittaminen

Luvussa tarkastellaan kokeellisen tutkimuksen tavoitteita ja käytännön suoritusta.

5.1 Kokeellisen tutkimuksen tavoitteet

Kokeellisella tutkimuksella on kolme päätavoitetta. Halutaan selvittää sekä algoritmien nopeus että tarkkuus. Tämän lisäksi halutaan testitulosten perusteella osoittaa Algoritmille 1 tehdyn teoreettisen vaativuustarkastelun tulos oikeaksi.

Kokeellinen vaativuustarkastelu tehdään, jotta voidaan osoittaa Algoritmin 1 ohjelmallisen toteutuksen olevan määritellyn vaativuusluokan mukainen. Tällöin tiedetään, että muiden algoritmien toteutuksien vertailu tapahtuu sellaiseen Algoritmin 1 toteutukseen, jonka vaativuus todella on $O(n^2)$.

Algoritmin 1 nopeuden kokeellisella tutkimuksella halutaan ensisijaisesti luoda vertailukohta muiden algoritmien analysoinnille, sillä Algoritmien 2–5 teoreettinen vaativuustarkastelu on vaikeaa. Tämän vuoksi niiden vaativuusanalyysi tehdään vertaamalla algoritmien läpimenoaikoja Algoritmin 1 läpimenoaikaan. Tavoitteena on osoittaa, että Algoritmeissa 2–4 käytettyjen ideoiden avulla on mahdollista saavuttaa merkittävästi parempi läpimenoaika, kuin mihin Algoritmi 1 kykenee.

Algoritmien 1–5 tarkkuuden tarkastelu tehdään vertaamalla saatuja tuloksia testimateriaalille valmiiksi laskettuihin arvoihin. Kyseiset arvot ovat joissain tapauksissa optimeja, mutta useimmiten ainoastaan parhaita löydettyjä. Tavoitteena on osoittaa, ettei Algoritmien 2–4 antama reitti ole merkittävästi huonompi verrattuna Algoritmin 1 antamaan reittiin.

5.2 Kokeellisen tutkimuksen suorittaminen

Seuraavassa kuvataan miten kokeellinen tutkimus suoritettiin. Oleellisimpina seikkoina tutkimuksen suorittamisessa kuvataan valittu testimateriaali, mitattavat suureet, testitulosten kirjaaminen ja mitatuille arvoille tehtävät tarkistukset.

5.2.1 Testimateriaali

Kokeellisessa tutkimuksessa käytettiin testimateriaalina internetistä löytyviä pistesarjoja, joista osalle on tutkittu paras mahdollinen reitti ja osalle paras tähän asti löydetty reitti. Pistesarjat löytyvät sivustolta http://www.tsp.gatech.edu/vlsi/index.html. Pistesarjoista pienin käsittää 131 pistettä ja suurin 744710 pistettä. Valtaosa pistesarjoista sijoittuu kooltaan välille 131–59296 pistettä. Tällä välillä on 98 pistesarjaa 102 joukosta. Mainittakoon vielä, että pistesarja sra104815.tsp on virheellisesti nimetty,

sillä kyseinen pistesarja käsittää 104814 pistettä. Pistesarjat on sivustolle toimittanut Andre Rohe ja sivuston mukaan ne perustuvat Bonnin yliopiston diskreetin matematiikan tutkimusinstituutin käyttämiin pistesarjoihin.

Sivustolta löytyvistä pistesarjoista käytettiin Algoritmin 1 tutkimuksessa 100 ensimmäistä. Suurimmat pistesarjat jätettiin Algoritmin 1 kohdalla tutkimatta, sillä algoritmin läpimenoaika kasvoi liian suureksi. Algoritmeille 2–5 tutkimus suoritettiin kaikilla pistesarjoilla.

Huonona puolena valitussa testimateriaalissa on se, että on vaikea arvioida, kuinka täydellisen kuvan erilaisista pisteiden jakaumista testimateriaali antaa. On myös vaikea arvioida, onko pistesarjojen muodostamisessa jokin sellainen ominaisuus, joka vaikuttaa niiden rakenteeseen testituloksia vääristävästi. Lisäksi pistesarjojen alkuperän ja totuudenmukaisuuden todentaminen on hankalaa. Välttämättä siis ei ole takeita siitä, että pistesarjat ovat lähtöisin sieltä mistä väitetään tai että niille ilmoitetut parhaat löydetyt reitit ovat oikeasti olemassa.

Puoltavana tekijänä pistesarjojen käytössä testimateriaalina on se, että ainakin sarjat tarjoavan sivuston mukaan ne ovat lähtöisin Bonnin yliopistosta. Sivuston yleisen tunnettuuden vuoksi on todennäköistä, että Bonnin yliopisto olisi reagoinut, jos asia ei olisi näin. Puoltavaksi tekijäksi voidaan lukea myös se, että pistesarjoille valmiiksi lasketut tulokset tarjoavat todella hyvän vertailukohdan algoritmien tarkkuuden tutkimukselle. Tämä pätee erityisesti suurille pistesarjoille, joille tarkkojen tulosten laskeminen on hyvin työlästä ja aikaa vievää.

5.2.2 Testimateriaalin rakenne

Testimateriaali koostuu tekstitiedostoihin tallennetuista sarjoista pisteitä, joista ilmoitetaan pisteen yksilöivä numero, x-koordinaatti ja y-koordinaatti. Sekä x- että y-koordinaatti ovat kokonaislukuarvoisia. Pistetiedostot ovat niin sanotusta TSPLIB-formaatissa, jossa tiedoston kommenttiosassa kerrotaan tiedoston nimi, tyyppi, koko ja käytettävä mitta. Tyyppi on näiden tiedostojen yhteydessä TSP, koko ilmoittaa pisteiden lukumäärän tiedostossa ja käytettävä mitta on euklidinen.

Edellä mainitulla internet-sivustolla on mahdollista nähdä kunkin pistetiedoston pisteiden jakauma. Tästä voi olla hyötyä, mikäli jokin algoritmien läpimenoajoissa tapahtuva heilahdus antaa aihetta lähempään tarkasteluun.

5.2.3 Mitattavat suureet

Tutkimuksessa mitataan kahta suuretta eli läpimenoaikaa ja reitin tarkkuutta.

Läpimenoaika

Läpimenoajalla tarkoitetaan aikaa, joka kuluu ajon aloittamisesta siihen, että reitti on valmis. Läpimenoaikaan ei lasketa mukaan aikaa, joka kuluu pisteiden lukemiseen tiedostosta, eikä aikaa, joka kuluu tarkistettaessa, että algoritmin antaman reitti käsittää kaikki tiedoston sisältämät pisteet. Myöskään algoritmin yhteydessä mitattujen tunnuslukujen kirjoittaminen lokiin ei näy läpimenoajassa.

Algoritmien 2–4 kohdalla otetaan ylös myös, kuinka paljon aikaa kuluu lisättävää pistettä etsiessä niiden pisteiden tutkimiseen, joilla on jo jokin lisäyskohta, ja niiden pisteiden tutkimiseen, joilla ei ole ennestään lisäyskohtaa. Lisäksi otetaan ylös, kuinka paljon menee aikaa niihin osareitin sisällä olevan konveksin verhon korjauksiin, joiden yhteydessä kulmajärjestystä täytyy muuttaa ja niihin korjauksiin, joissa kulmajärjestystä ei tarvitse muuttaa. Vertailun helpottamiseksi lasketaan myös se, kuinka monta kertaa kumpaakin eri korjaustyyppiä suoritetaan.

Reitin tarkkuus

Kunkin algoritmin antaman reitin pituus lasketaan käyttäen sekä euklidista etäisyyttä että euklidisesta etäisyydestä johdettu etäisyyttä, jota tässä yhteydessä kutsutaan TSPLIB-etäisyydeksi. Kahden pisteen välinen TSPLIB-etäisyys saadaan pyöristämällä niiden euklidinen etäisyys lähimpään kokonaislukuun. Testimateriaalin valmiiksi lasketuissa arvoissa on reitin pituus laskettu käyttäen TSPLIB-etäisyyttä. Tällaisen ratkaisun käyttöä voi perustella Johnsonin ja Papadimitrioun [1987a, pp. 60–61] huomautuksella siitä, että euklidisen metriikan käyttö luo ongelmia pyöristysten kanssa. He jatkavat määrittelemällä vastaavan kaltaisesti uuden metriikan, jossa euklidisella metriikalla saatu arvo pyöristetään kattofunktiolla seuraavaan kokonaislukuun, joka on suurempi tai yhtä suuri kuin alkuperäinen arvo.

Tämän työn yhteydessä toimitaan siten, että kun saatuja tuloksia vertaillaan testimateriaalille valmiiksi laskettuihin arvoihin, niin käytetään TSPLIB-etäisyyden mukaista tulosta. Silloin kun vertaillaan itse toteutettuja algoritmeja keskenään, niin käytetään euklidisen etäisyyden antamaa tulosta.

Testitulosten kirjaaminen

Testitulokset kirjataan ylös kolmen desimaalin tarkkuudella. Koska kyseessä ovat approksimointialgoritmit, niin pyöristysvirheiden käsittely ei ole oleellista. Tämä siksi, että algoritmien perusajatuksena on approksimoida, eikä pienten virheiden merkitys ole tehtävien tarkastelujen kannalta oleellinen. Testitulosten pohjalta luodaan Excel-

tiedosto, jonka avulla mitattavista suureista voidaan luoda tarkasteluja helpottavia kuvia.

5.2.4 Tarkistukset

Algoritmien testaukseen kuuluu oleellisena osana niiden antamien tuloksien oikeellisuuden tutkiminen. Tämän työn yhteydessä tämä tarkoittaa, että algoritmin antama reitti on todella olemassa oleva. Tarkistetaan siis, että jokainen pistesarjan piste on mukana reitissä ja ettei mikään pistesarjan piste ole reitissä useammin kuin yhden kerran.

Ensimmäinen versio tarkistuksesta toteutettiin niin, että käytiin pistesarjan sisältävää tiedostoa läpi ja tarkistettiin, että jokainen piste löytyy reitistä. Lisäksi tarkistettiin, että reitissä oli yhtä monta pistettä kuin pistesarjan sisältävässä tiedostossa ilmoitettiin. Suuremmilla pistesarjoilla tapa osoittautui kuitenkin liian hitaaksi, joten kehitettiin tarkistustapa, joka toimii seuraavasti:

1. Ilmoitetaan, kuinka monta pistettä reittiin kuuluu. Tämän luvun tulee täsmätä pistesarjan ilmoitettuun kokoon. Oikeellisuus voidaan tarkistaa ajon kirjoittamista tiedoista.
2. Luodaan kokonaislukutaulukko, jossa on yhtä monta alkiota kuin reitissä on pisteitä.
3. Käydään reitti läpi ja kasvatetaan taulukossa pisteen tunnistenumeron osoittamaa indeksiä yhdellä.
4. Käydään luotu taulukko läpi ja tarkistetaan, että sen jokaisessa indeksissä on arvo yksi.
5. Mikäli löytyy indeksi, jolla arvo eroaa ykkösestä, ilmoitetaan virheestä.

6 Tulokset

Tässä luvussa käsitellään kokeellisella tutkimuksella saatuja tuloksia. Tarkastelut tehdään niin, että aivan aluksi suoritetaan Algoritmin 1 vaativuustarkastelu. Tämän jälkeen suoritetaan Algoritmien 1 ja 2 välinen keskinäinen vertailu ja Algoritmin 2 tarkempi analyysi. Tämän jälkeen verrataan Algoritmeja 2 ja 3 ja sitten Algoritmeja 3 ja 4 keskenään. Lopuksi verrataan Algoritmia 4 Algoritmiin 5. Tähän järjestelyyn on päädytty siksi, että esitettävät kuvat pysyisivät mahdollisimman selkeinä. Tarkastelu Algoritmien 1 ja 2 välillä tehdään pistesarjoilla, joiden koot ovat väliltä 131–232025, koska suuremmilla pistesarjoilla Algoritmin 1 läpimenoaika kasvaa liian suureksi. Muita algoritmeja verrattaessa tarkastelut tehdään koko testimateriaalilla eli pistesarjoilla joiden koot ovat välillä 131–744710.

6.1 Algoritmin 1 vaativuustarkastelu

Lukumäärän kasvulla tarkoitetaan sitä, kuinka monikertainen kukin tiedosto on verrattuna edelliseen sitä pienempään tiedostoon. Ensimmäiselle tiedostolle tätä arvoa ei lasketa. Esimerkiksi tiedosto xqg237.tsp on 1,81-kertainen verrattuna tiedostoon xqf131.tsp. Kullekin tiedostolle, jolle on laskettu *lukumäärän kasvu*, lasketaan myös kuinka suuri tuo suure on, kun se korotetaan toiseen potenssiin.

Vastaavasti *ajan kasvulla* tarkoitetaan sitä, kuinka moninkertainen kunkin tiedoston läpimenoaika on verrattuna edellisen tiedoston läpimenoaikaan. Ensimmäiselle tiedostolle tätä arvoa ei lasketa. Esimerkiksi tiedoston pbk411.tsp läpimenoaika on 1,03-kertainen verrattuna tiedoston pbl395.tsp läpimenoaikaan.

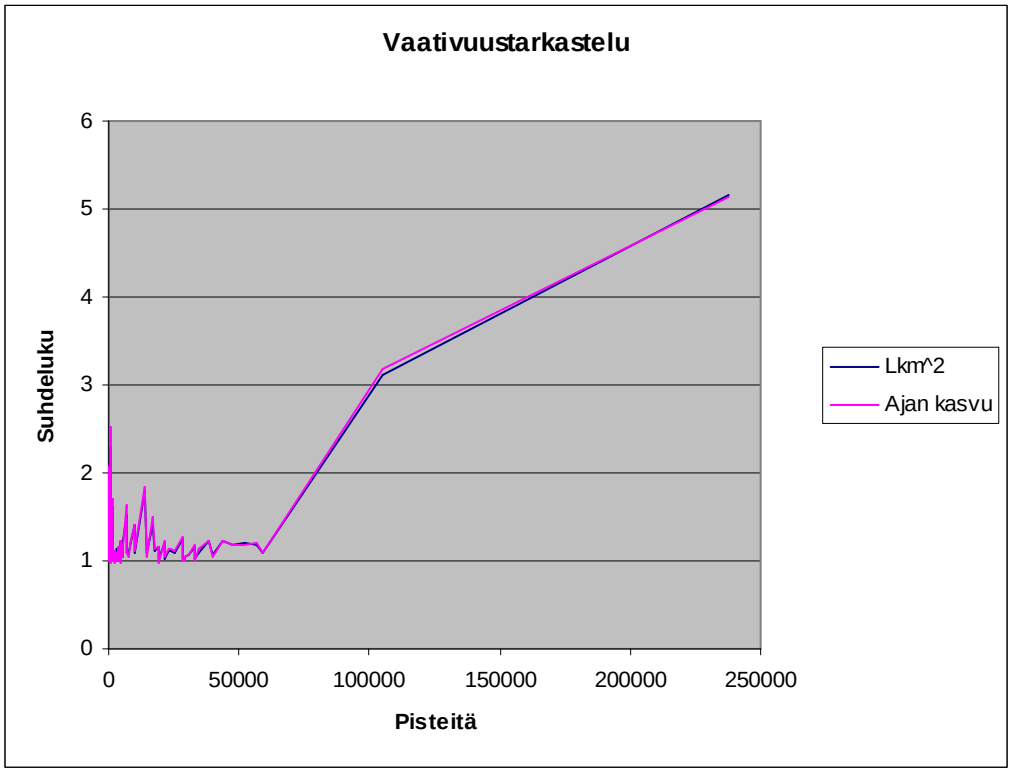
Itseisarvo erotuksesta *lukumäärän kasvu potenssiin kaksi - ajan kasvu* kertoo, kuinka suuri kyseisten suureiden erotuksen itseisarvo on.

Tehtävä tarkastelu perustuu seuraavaan huomioon. Käsiteltävien tapauksien kasvaessa kertoimella *x*, noudattaa alkeisoperaatioiden määrän kasvu $O(n^2)$ vaativuuden omaavassa algoritmissa seuraavalla kaavalla saatua tulosta:

$$(x \cdot n)^2 \Big/ n^2 = x^2 \cdot n^2 \Big/ n^2 = x^2 .$$

Laskemalla *lukumäärän kasvu potenssiin kaksi*, saadaan alkeisoperaatioiden määrän kasvu, mikäli kyseessä on $O(n^2)$ algoritmi. Tätä lukua vertailtaessa suureeseen *ajan kasvu*, saadaan vastaus kysymykseen, onko kyseessä kokeellisen tutkimuksen perusteella $O(n^2)$ vaativuuden omaava algoritmi. Selvästikin suureen *ajan kasvu* täytyisi tällöin suuruudeltaan vastata suuretta *lukumäärän kasvu potenssiin kaksi*.

Laskettaessa keskiarvo itseisarvoille, jotka saadaan suureiden *lukumäärän kasvu potenssiin kaksi* ja *ajan kasvu* erotuksille, saadaan tulokseksi 0,04. Tämä tulos antaa vahvan viitteen siitä, että Algoritmi 1 on vaativuudeltaan $O(n^2)$ luokkaan kuuluva. Kuvassa 19 esitetään sekä suureen *ajan kasvu* että suureen *lukumäärän kasvu potenssiin kaksi* kehitys. Se antaa selkeän kuvan siitä, kuinka tarkasti arvot mukailevat toisiaan.



Kuva 19. Algoritmin 1 vaativuustarkastelu.

Saatujen tuloksien perusteella voidaan todeta havaituksi se, että Algoritmi 1 on vaativuudeltaan $O(n^2)$ luokkaan kuuluva.

6.2 Algoritmien 1 ja 2 vertailu

Algoritmien 1 ja 2 vertailulla pyritään luomaan pohja Algoritmeissa 2, 3 ja 4 käytetyn perusidean arvioinnille.

6.2.1 Tarkkuus

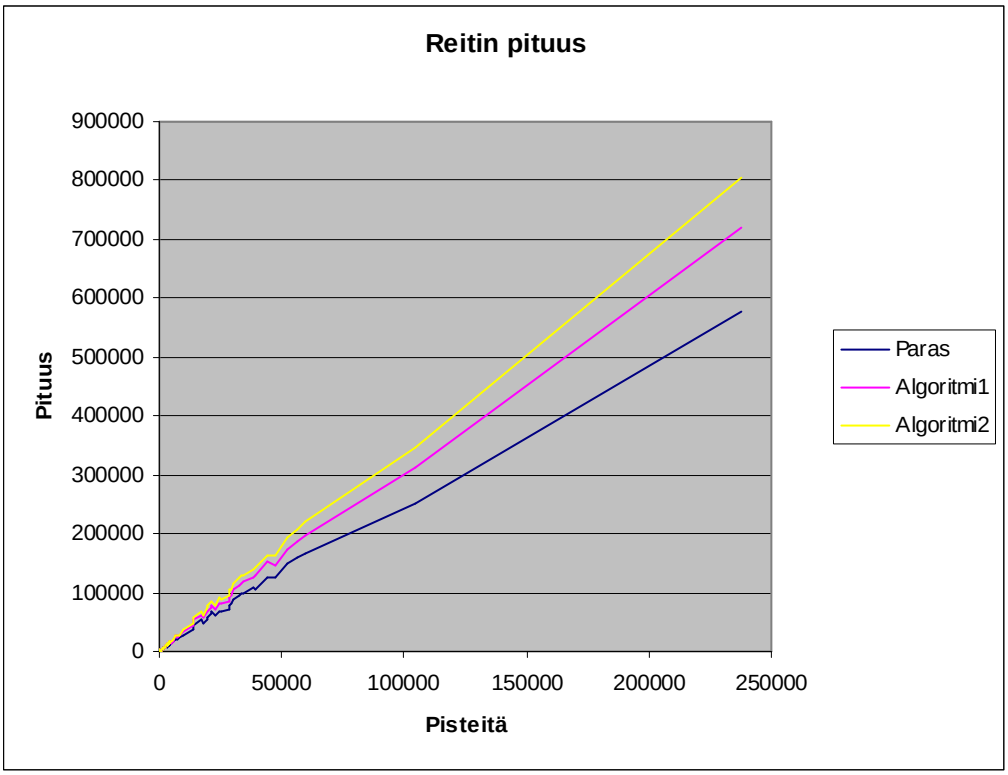
Kokeellisen tutkimuksen perusteella Algoritmin 2 tarkkuus ei poikennut käytetyillä pistesarjoilla merkittävästi Algoritmin 1 tarkkuudesta. Algoritmi 1 antoi keskimäärin 1,17-kertaisen tuloksen verrattuna pistesarjoille ilmoitettuihin parhaisiin löydettyihin reitteihin, kun taas Algoritmi 2 antoi keskimäärin 1,26-kertaisen tuloksen. Suurimmillaan Algoritmin 1 antama reitti oli 1,25-kertainen parhaaseen löydettyyn

verrattuna ja pienimmillään 1,06-kertainen. Algoritmilla 2 vastaava suurin arvo oli 1,39-kertainen ja pienin 1,06-kertainen.

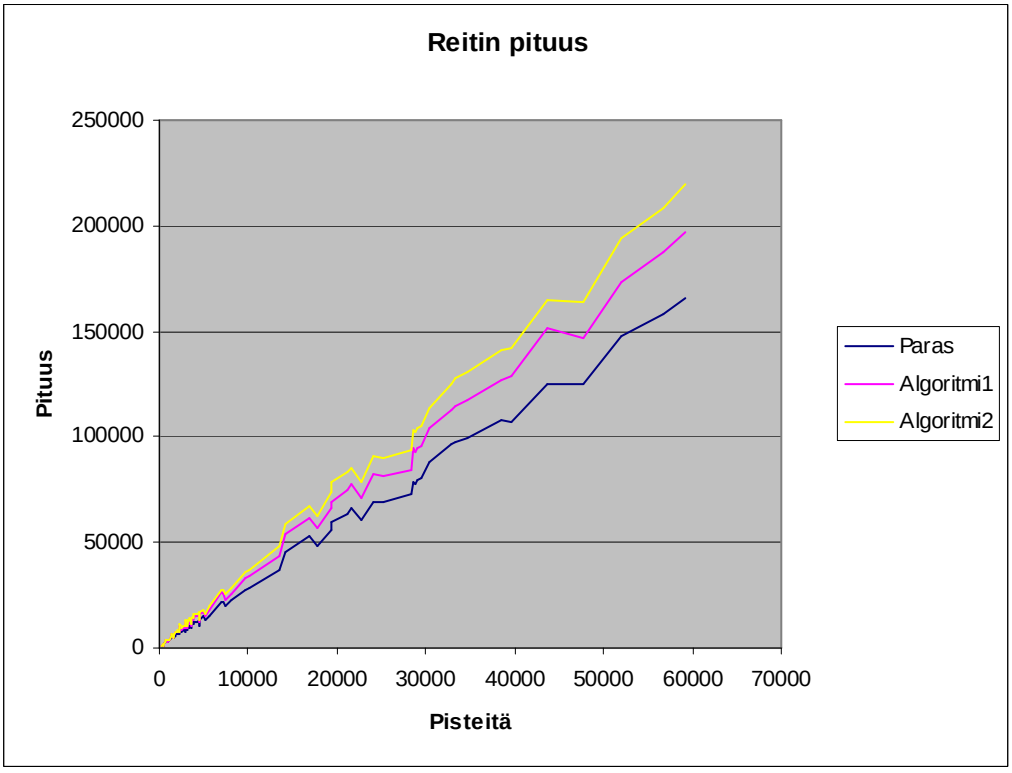
Algoritmin 2 antama reitti TSPLIB-etäisyydellä laskettuna oli keskimäärin 1,08-kertainen verrattuna Algoritmin 1 antamaan reittiin. Suurimmillaan Algoritmin 2 antama reitti oli 1,14-kertainen ja pienimmillään 1,00-kertainen verrattuna Algoritmin 1 antamaan approksimaatioon. Näissä arvoissa ei tapahtunut merkittävää muutosta, kun siirryttiin euklidiseen etäisyyteen. Tällöin Algoritmi 2 laskema reitti oli pituudeltaan keskimäärin 1,07-kertaisen verrattuna Algoritmin 1 antamaan reittiin. Suurimmillaan Algoritmin 2 laskema reitti oli 1,13-kertainen ja pienimmillään 1,00-kertainen vertailukohtaan nähden.

Algoritmilla 1 saatiin sen antaman reitin pituuden ja parhaan löydetyn reitin pituuden suhteen keskihajonnaksi 0,03 ja Algoritmilla 2 vastaava luku oli 0,06. Tästä voitaisiin päätellä, että pisteiden lukumäärä ei merkittävästi vaikuta algoritmien antamien reittien tarkkuuteen. Tämä ei kuitenkaan välttämättä pidä paikkaansa, sillä pistejoukon jakauma koon suhteen ei ole tasainen, vaan valtaosa pistesarjoista sijoittuu kooltaan välille 131–59296 pistettä. Keskihajonta antaa kuitenkin viitettä siitä, että mahdollinen reitin huononeminen pistesarjojen koon kasvaessa tapahtuu maltillisesti.

Algoritmien 1 ja 2 käyttäytymisestä saa hyvän käsityksen seuraavien kuvien avulla. Kuvassa 20 esitetään algoritmien antamien reittien pituuksien kehitys pistesarjoilla, joiden koot ovat väliltä 131–232025, ja kuvassa 21 pistesarjoilla, joiden koot ovat väliltä 131–59296. Kuvista näkee selvästi, kuinka Algoritmien 1 ja 2 antamat reitit mukailevat toisiaan pituuden suhteen. Kuvista on myös hyvin havaittavissa, kuinka ero parhaan löydetyn reitin pituuden ja Algoritmien 1 ja 2 antaman approksimaation välillä hitaasti suurenee pistesarjojen koon kasvaessa.



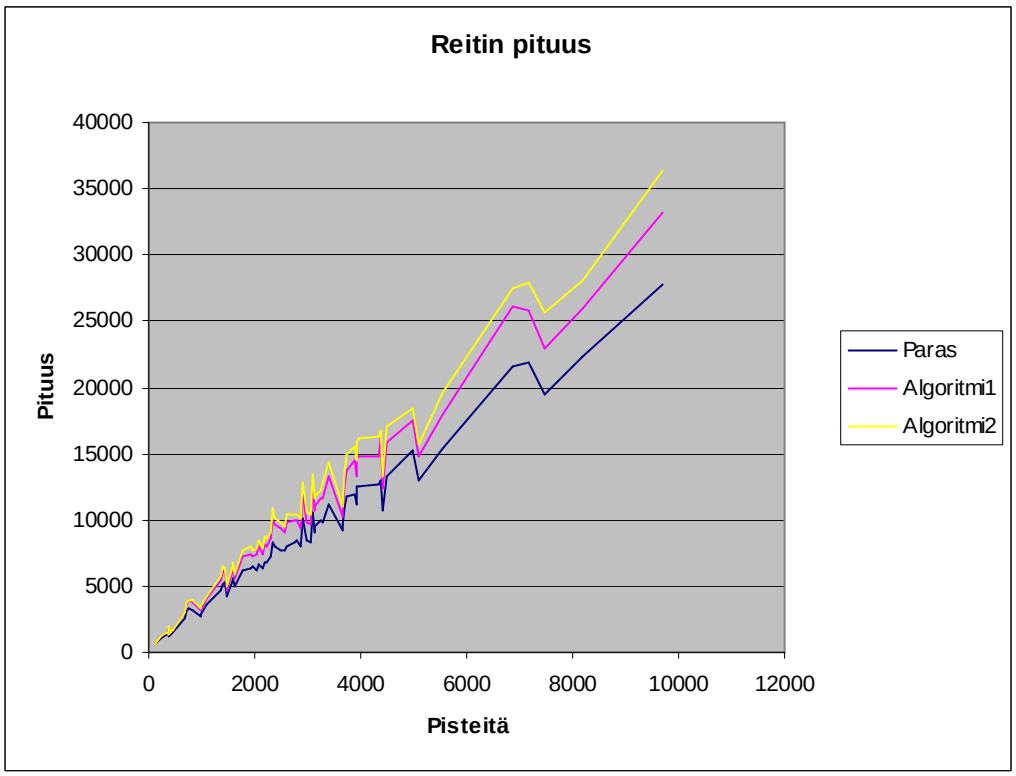
Kuva 20. Reittien pituudet, kun pistesarjojen koot ovat välillä 131–232025.



Kuva 21. Reittien pituudet, kun pistesarjojen koot ovat välillä 131–59296.

Kuva 22 antaa tilanteesta vielä tarkemman käsityksen, sillä kuvassa käsiteltävät pistesarjat, joiden koot ovat välillä 131–9698, käsittävät 71 kappaletta testimateriaalina

toimivista 102 tiedostosta. Kuvan perusteella voidaan silmämääräisesti havaita, kuinka suhde algoritmien antaman approksimaation ja parhaan löydetyn ratkaisun välillä pysyy samankaltaisena aiempiin kuviin verrattuna. Kuten keskihajonta antoi ymmärtää, ero suurenee approksimaatioreittien ja vertailureitin suhteen hyvin maltillisesti.



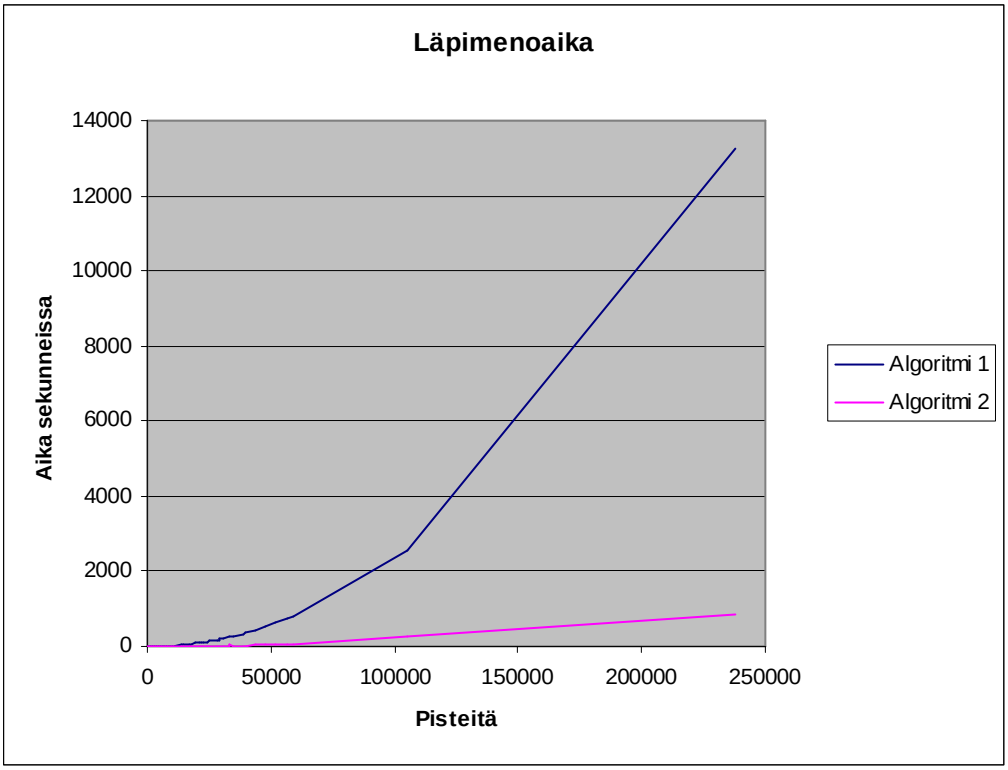
Kuva 22. Reittien pituudet, kun pistesarjojen koot ovat välillä 131–9698.

Tutkimuksen perusteella voidaan sanoa, että Algoritmi 1 ja Algoritmi 2 näyttäisivät olevan ainakin 1-approksimointialgoritmeja eli ne tekevät enintään 100 prosentin virheen. Voidaan myös melko varmasti sanoa niiden olevan tai ainakin näyttävän olevan $\frac{1}{2}$-approksimointialgoritmeja eli, että ne tekevät enintään 50 prosentin virheen. Tulosten perusteella voidaan myös sanoa, ettei Algoritmin 2 tarkkuus poikkea merkittävästi Algoritmin 1 tarkkuudesta. Voidaan kuitenkin selvästi todeta Algoritmin 2 olevan tarkkuuden osalta alisteinen Algoritmille 1.

6.2.2 Nopeus

Kokeellisessa tutkimuksessa selvisi, että Algoritmin 2 läpimenoaika poikkeaa merkittävästi Algoritmin 1 läpimenoajasta. Algoritmin 1 läpimenoaika oli keskimäärin 9,02-kertainen verrattuna Algoritmin 2 läpimenoaikaa. Tässä tarkastelussa jätettiin huomioimatta ne pistesarjat, joissa algoritmin läpimenoaika oli niin pieni, että mittaustulokseksi tuli 0 sekuntia. Suurimmillaan Algoritmin 1 läpimenoaika oli 22,5-

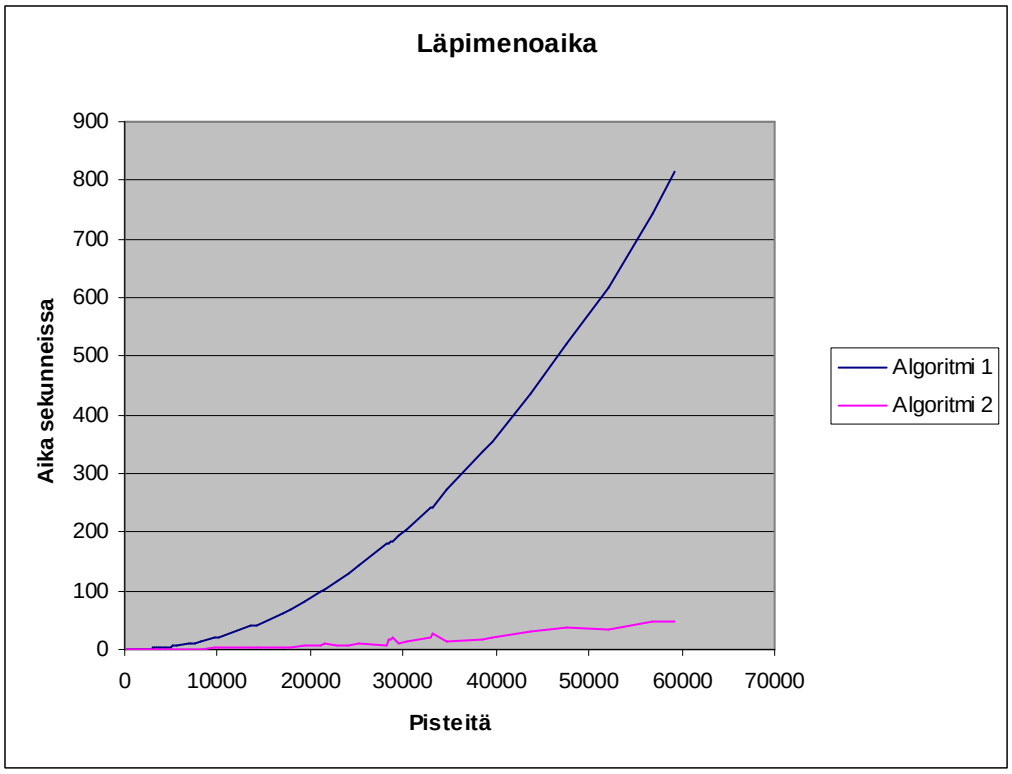
kertainen Algoritmiin 2 verrattuna. Tiedostolla xqg238.tsp Algoritmi 1 oli nopeampi kuin Algoritmi 2. Kyseisen tiedoston koko on kuitenkin niin pieni, että on vaikea sanoa, onko kyseessä jokin Algoritmi 2 heikkouden ilmentymä vai testausympäristön epävakauden tulos. Kuvassa 23 esitetään Algoritmien 1 ja 2 läpimenoaikojen kehitys.



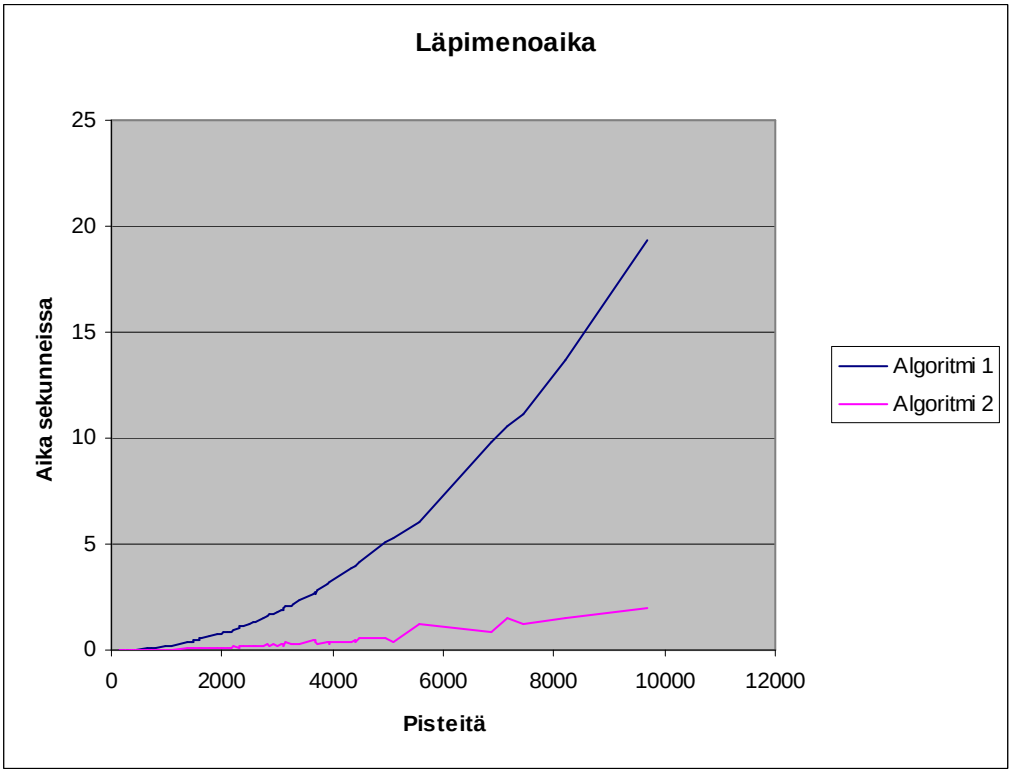
Kuva 23. Läpimenoajan kehitys pistesarjoilla, joiden koot ovat välillä 131–232025.

Kuvista 24 ja 25 nähdään, kuinka ilmeisesti Algoritmin 1 läpimenoajan kehitys mukailee n^2-käyrän muotoa ja miten Algoritmin 2 läpimenoaika poikkeaa siitä. Tämä ominaisuus tulee paremmin ilmi tarkasteltaessa tiedostoja, joissa on vähemmän pisteitä, sillä suuret yksittäiset pistesarjat vääristävät kuvaa.

Kuvat 24 ja 25 voivat antaa harhaanjohtavan mielikuvan Algoritmin 2 läpimenoajan lineaarisuudesta. Tästä ei kuitenkaan ole suoranaisesti ole kyse. Algoritmissa 2 $O(n^2)$-tyyppinen käytös enemminkin ilmenee loivempana kuin Algoritmissa 1. Tähän pureudutaan paremmin seuraavassa luvussa, jossa analysoidaan Algoritmia 2.



Kuva 24. Läpimenoajan kehitys pistesarjoilla, joiden koot ovat välillä 131–59296.



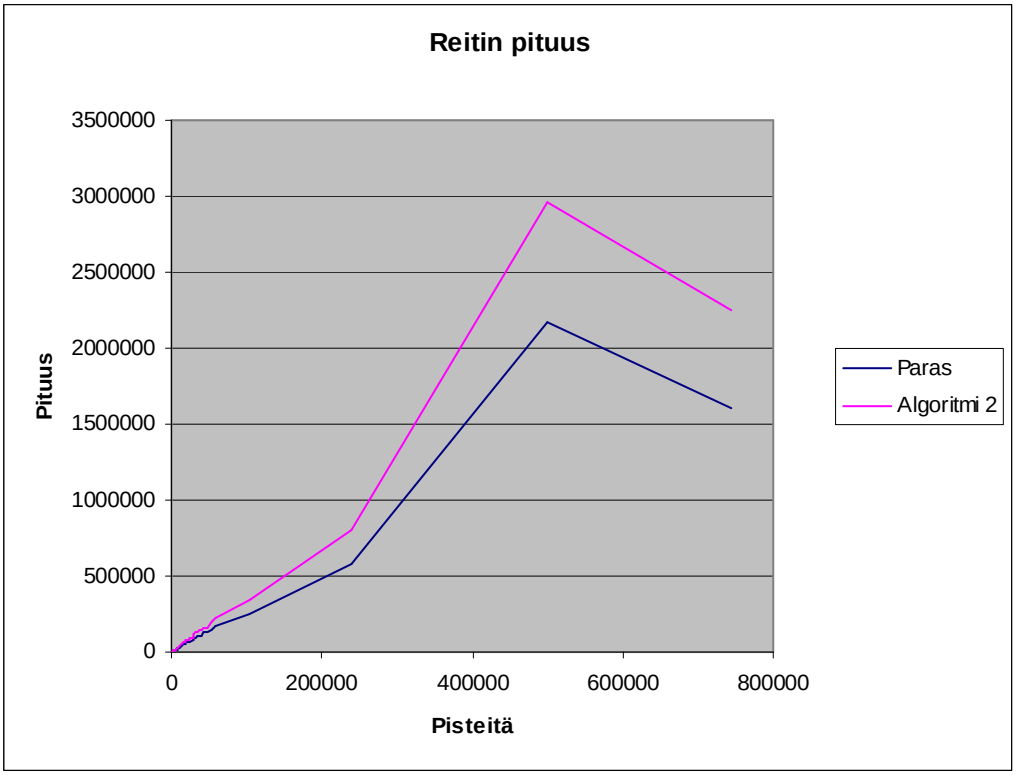
Kuva 25. Läpimenoajan kehitys pistesarjoilla, joiden koot ovat välillä 131–9698.

6.3 Algoritmin 2 tarkempi analyysi

Algoritmin 2 tarkempi analyysi suoritetaan siksi, että kyseessä on oman algoritmi-idean perusversio, jonka toiminnallisuutta Algoritmit 3 ja 4 mukailevat. Algoritmin 2 tarkemmassa analyysissa keskitytään pääasiassa sen läpimenoajan tutkimiseen. Koska Algoritmilla 2 pystyttiin käsittelemään laajempi testimateriaali kuin Algoritmilla 1, niin aivan aluksi tarkastellaan Algoritmin 2 tarkkuutta koko testimateriaalilla.

6.3.1 Tarkkuus

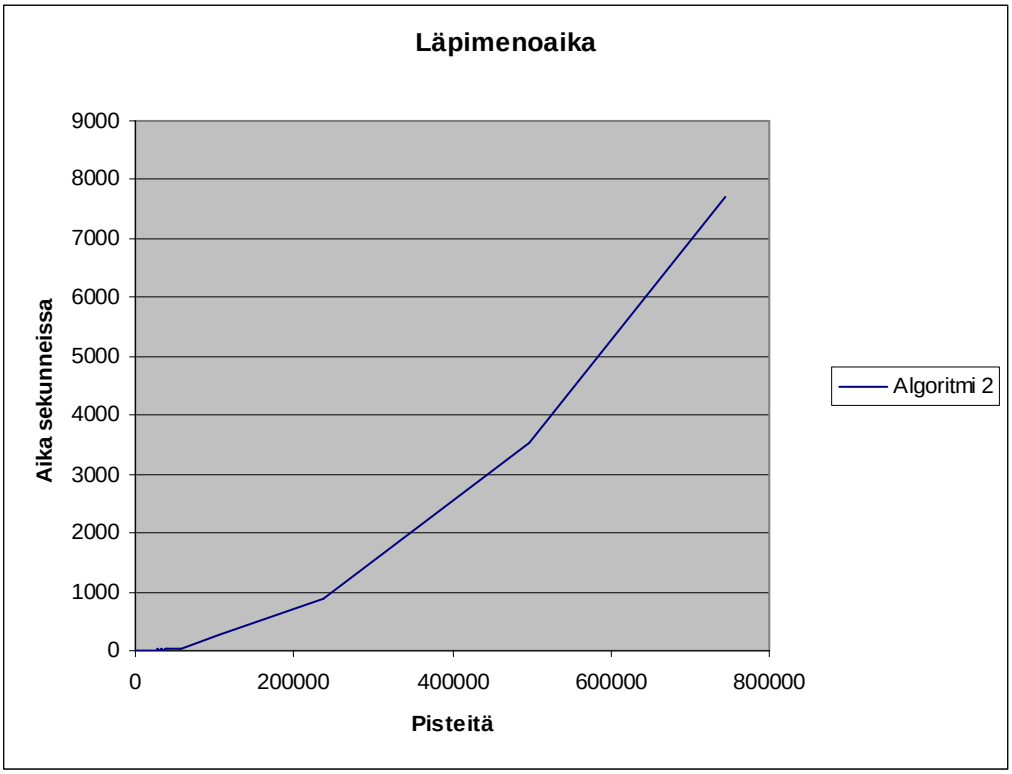
Suurempien testiaineistojen mukana olo ei merkittävästi vaikuttanut Algoritmin 2 tuloksiin tarkkuuden suhteen. Koko testimateriaalilla Algoritmi 2 antoi keskimäärin 1,26-kertaisen tuloksen verrattuna pistesarjoille ilmoitettuihin parhaisiin löydettyihin reitteihin. Suurimmillaan Algoritmi 2 antoi 1,40-kertaisen reitin parhaaseen löydettyyn verrattuna. Vastaavat luvut olivat 1,26 ja 1,39, kun käytössä ei ollut koko testimateriaali. Pienin löydetty suhde ei muuttunut, vaan se oli edelleen 1,06-kertainen. Suurin arvo saatiin testimateriaalin viimeisellä pistesarjalla, johon kuuluu 744710 pistettä. Kuva 26 esittää Algoritmin 2 tarkkuutta, kun aineistona on koko testimateriaali.



Kuva 26. Reitin pituudet kehitys koko testimateriaalilla.

6.3.2 Nopeus

Algoritmin 2 nopeutta tutkiessa on tavoitteena selvittää, miten algoritmin eri osat vaikuttavat läpimenoajan muodostumiseen. Aluksi kuitenkin perehdytään hieman tarkemmin Algoritmin 2 nopeuteen koko testimateriaalin osalta. Kuvassa 27 esitetään Algoritmin 2 läpimenoajan kehitys koko testimateriaalilla.



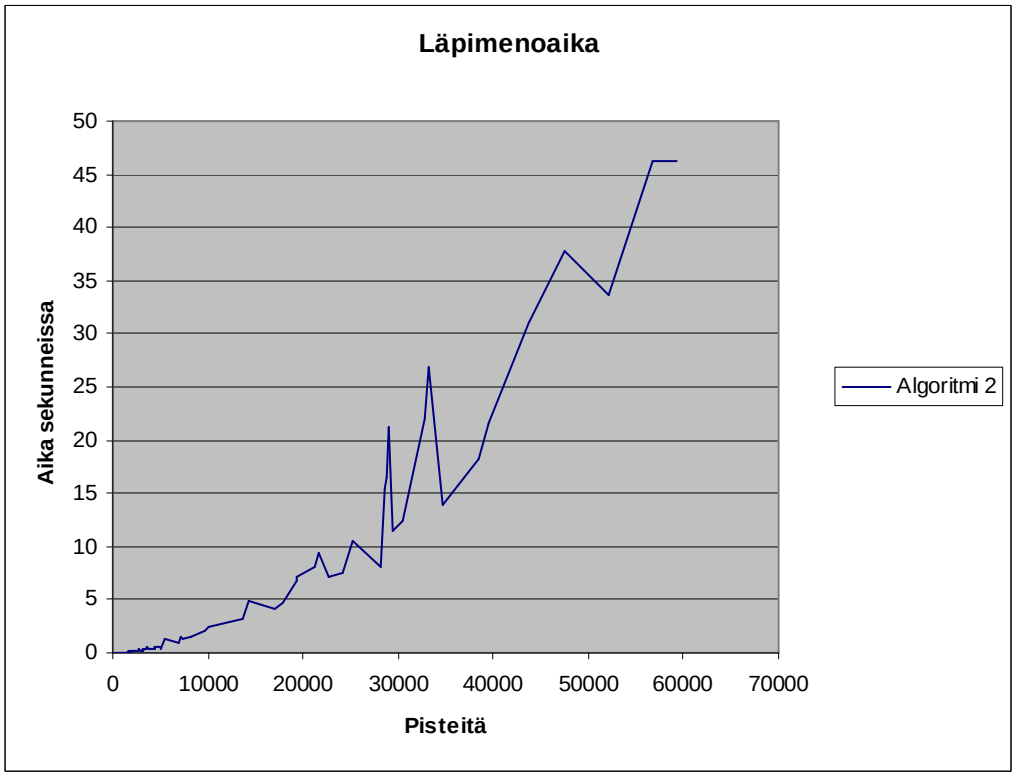
Kuva 27. Algoritmin 2 läpimenoajan kehitys koko testimateriaalilla.

Kuvasta 27 nähdään, miten Algoritmin 2 läpimenoaika kasvaa jyrkemmin pistesarjojen koon kasvaessa. Kuvassa on silmämääräisesti nähtävissä $O(n^2)$-tyyppistä läpimenoajan kasvua, mikä antaisi viitettä siitä, että Algoritmissa 2 ei onnistuta vähentämään alkeisoperaatioiden määrää Algoritmiin 1 verrattuna, vaan muuttamaan niiden laatua. Verrattuna Algoritmiin 1 käsitys Algoritmin 2 nopeudesta vahvistuu, kun huomataan se, että Algoritmi 2 laski 744710 pisteen sarjalle approksimaation 2,14 tunnissa, siinä missä Algoritmi 1 käytti 238025 pisteen sarjaan 3,68 tuntia. Algoritmi 2 pystyi siis selvittämään yli kolme kertaa suuremmalle tiedostolle reitin 1,54 tuntia nopeammin.

Kuten aiemmin on sekä teoreettisesti että kokeellisesti näytetty, kuuluu Algoritmi 1 vaativuusluokkaan $O(n^2)$. Tämä tarkoittaa sitä, että pisteiden määrän kaksinkertaistuessa sen läpimenoajan tulisi nelinkertaistua. Tämä tulos saatiin kaavasta

$(2 \cdot n)^2 / n^2 = 4 \cdot n^2 / n^2 = 4$. Samalla periaatteella voidaan laskea, että tilanteessa, jossa pisteiden määrä kolminkertaistuu, tulisi Algoritmin 1 läpimenoajan 9-kertaistua. Arvion mukaan Algoritmi 1 tulisi siis käyttämään noin 9 kertaa enemmän aikaa pistesarjalle, jossa on 744710 pistettä, verrattuna pistesarjaan, jossa on 238025 pistettä. Voidaan siis arvioida, että suurimpaan pistesarjaan, johon Algoritmi 2 käytti 2,14 tuntia, kuluisi Algoritmilta 1 noin 33,12 tuntia.

Seuraavaksi perehdytään tarkemmin Algoritmin 2 läpimenoajan muodostukseen. Tätä varten esitetään kuva 28, jossa on Algoritmin 2 läpimenoajat tiedostoilla, joiden koot ovat väliltä 131–59296 pistettä.

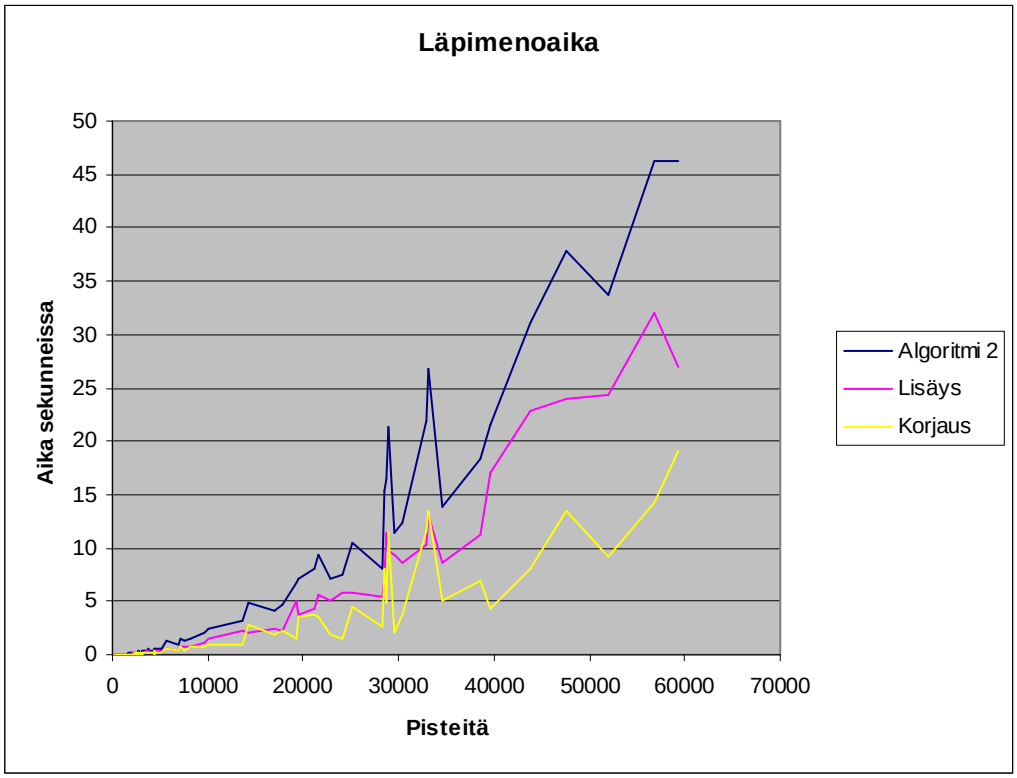


Kuva 28. Algoritmin 2 läpimenoajan kehitys pistesarjoilla 131–59296.

Algoritmin 2 läpimenoajan kehityksessä näkyy selkeitä piikkejä. Seuraavaksi yritetään selvittää, mistä nämä piikit johtuvat.

Algoritmi 2 koostuu pääpiirteissään kahdesta pääosasta, jotka ovat seuraavan reittiin lisättävän pisteen etsintä ja reitin sisäpuolisen konveksin verhon korjaus. Kuten aikaisemmin on mainittu, niin näihin kumpaankin pääosaan sisältyy omat ongelmansa. Reittiin lisättävän pisteen etsinnässä kyseinen ongelma oli approksimaatioreitin kaareutumisen vaikutus ja konveksin verhon korjauksessa tilanne, jossa kulmapiste

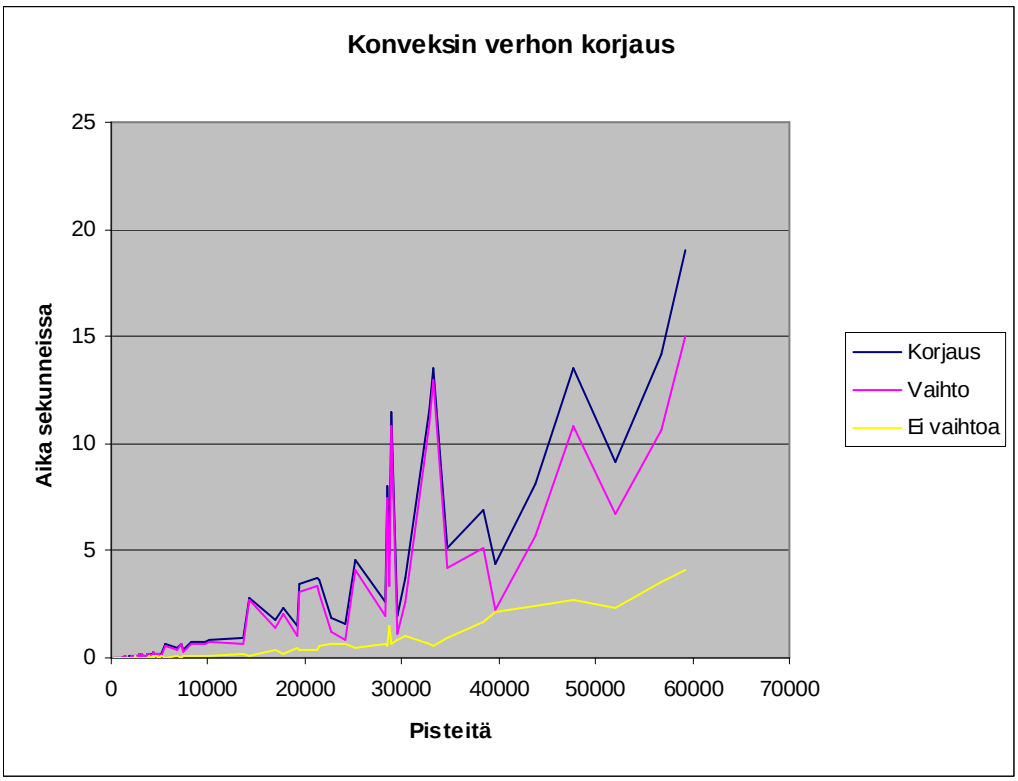
joudutaan vaihtamaan. Kuva 29 antaa yleiskäsityksen siitä, millaisen osuuden nämä kaksi eri pääosaa vievät kokonaisläpimenoajasta.



Kuva 29. Algoritmin 2 pääosien vaikutus läpimenoajan kasvuun.

Kuten kuvasta 29 nähdään, ovat molemmat pääosat lähes yhtä merkittävässä asemassa kokonaisläpimenoajan muodostumisessa. Kuvasta huomataan myös, miten korjauksien suorittaminen lisää Algoritmin 2 läpimenoajassa tapahtuvia vaihteluja. Oleellista on tutkia, selittyvätkö näkyvissä olevat piikit niillä ongelmilla, joiden olemassaolo jo tiedetään. Aloitetaan tämä tarkastelu tutkimalla tarkemmin algoritmin toisen pääosan eli approksimaatioreitin sisällä olevan konveksin verhon korjauksen aikavaativuutta.

Kuvassa 30 esitetään, millaiset osuudet konveksin verhon korjauksen käyttämän ajan kehityksestä vievät tilanteet, joissa kulmapistettä täytyy vaihtaa ja tilanteet, joissa riittää pelkkä korjaus.

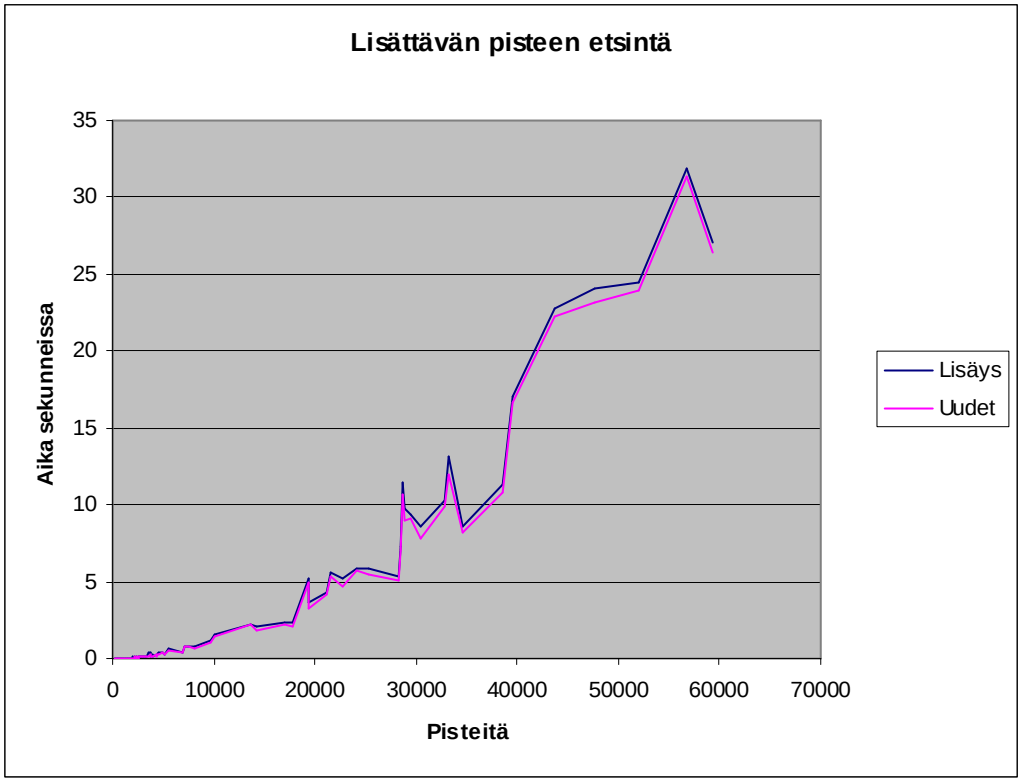


Kuva 30. Kulmajärjestyksen vaihtamisen vaikutus korjausten vaatimaan aikaan.

Kuvan 30 perusteella on selvää, että tapaukset, joissa kulmajärjestystä joudutaan muuttamaan, vievät merkittävän osan korjauksien vaatimasta kokonaisajasta. Ilmiön merkittävyys on sitäkin suurempi huomioitaessa, että tilanteita, joissa tehdään pelkkä korjaus, on pistesarjoilla paljon enemmän kuin tilanteita, joissa tehdään myös uusi kulmajärjestys. Esimerkiksi pistesarjalla boa28924.tsp kulmajärjestyksen uusiminen ja korjaus tapahtui 1094 kertaa ja se vei aikaa 10,9 sekuntia, kun taas pelkkä korjaus tapahtui 27502 kertaa ja se vei aikaa 0,61 sekuntia. Pistesarjalla fry33203.tsp puolestaan kulmajärjestyksen uusiminen ja korjaus tapahtui 972 kertaa kuluttaen aikaa 13,0 sekuntia, kun taas pelkkä korjaus tapahtui 31448 kertaa ja siihen kului aikaa 0,55 sekuntia. Testitulosten perusteella on siis ilmeistä, että aiemmin mainittu konveksin verhon korjaukseen liittyvä ongelma vaikuttaa merkittävästi Algoritmin 2 läpimenoaikaan.

Algoritmin toinen pääosa eli lisättävän pisteen etsintä muodostuu kahdesta osasta. Toisessa tutkitaan pisteitä, joille on jo löydetty jokin lisäämiskohta, reittiin syntyneitä uusia lisäyskohtia vasten ja toisessa tutkitaan konveksiin verhoon tuotuja uusia pisteitä, osareittiin pisteiden sivuilta löytyvien jo tutkittujen pisteiden luomaan rajoittumaa vasten. Käytännön testeissä havaitaan, että kokonaan uusien pisteiden

tutkinta kulutti lähes kaiken sen ajan, joka kului lisättävän pisteen etsintään. Kuva 31 selventää tilannetta.



Kuva 31. Kokonaan uusien pisteiden osuus lisättävän pisteen etsinnästä.

Konveksiin verhoon tuotujen uusien pisteiden tutkiminen vie siis käytännössä kaiken algoritmin ensimmäisen vaiheen kuluttaman ajan. Testitulosten perusteella ei suoraan voi päätellä, että tämä aiheutuisi luvussa aiemmin kuvatusta approksimaatioreitin kaareutumisesta. Voidaan kuitenkin arvioida, että uusien pisteiden tutkimista tehostamalla, voitaisiin saada merkittävää säästöä Algoritmin 2 läpimenoajassa.

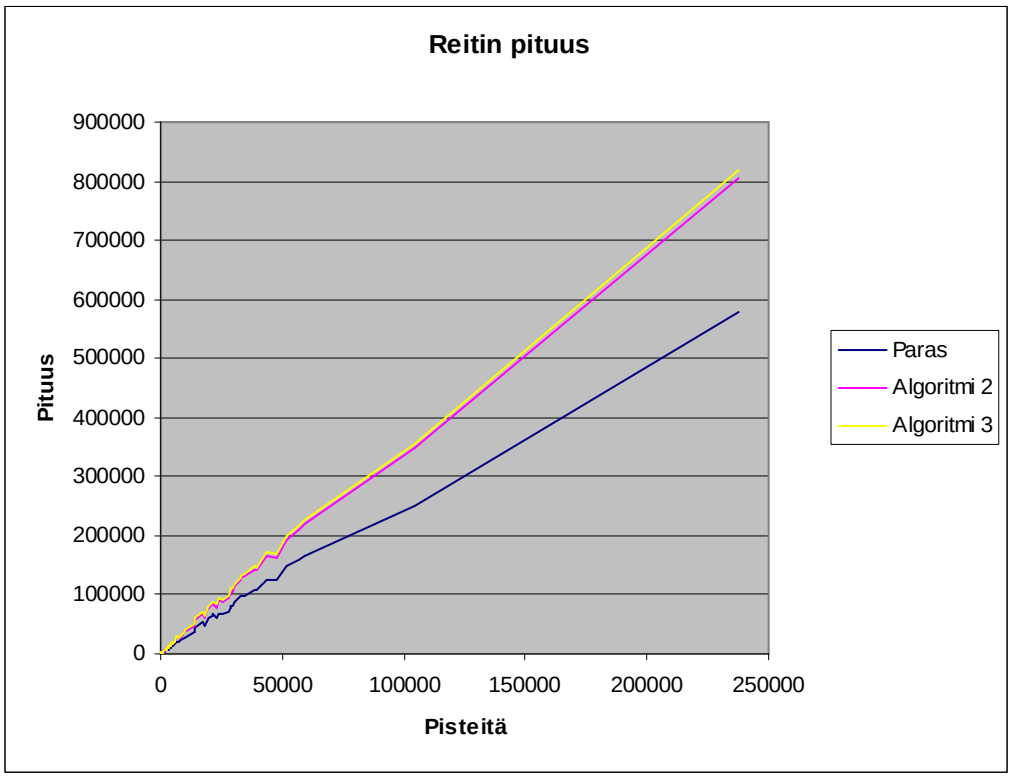
6.4 Algoritmi 3

Algoritmi 3 on saatu Algoritmista 2 sillä muutoksella, että aiemmin kuvatun neliörajoitteen avulla pyritään vähentämään mahdollisen osareitin kaareutumisen aiheuttamaa turhien lisäyskohtien tutkimista.

6.4.1 Tarkkuus

Algoritmin 3 käyttämä neliörajoite ei aiheuta merkittävää muutosta reitin pituuteen, kun sitä verrataan Algoritmin 2 antamaan tulokseen. Algoritmin 3 antama reitti oli TSPLIB-etäisyyttä käytettäessä keskimäärin 1,03-kertainen verrattuna Algoritmin 2 antamaan

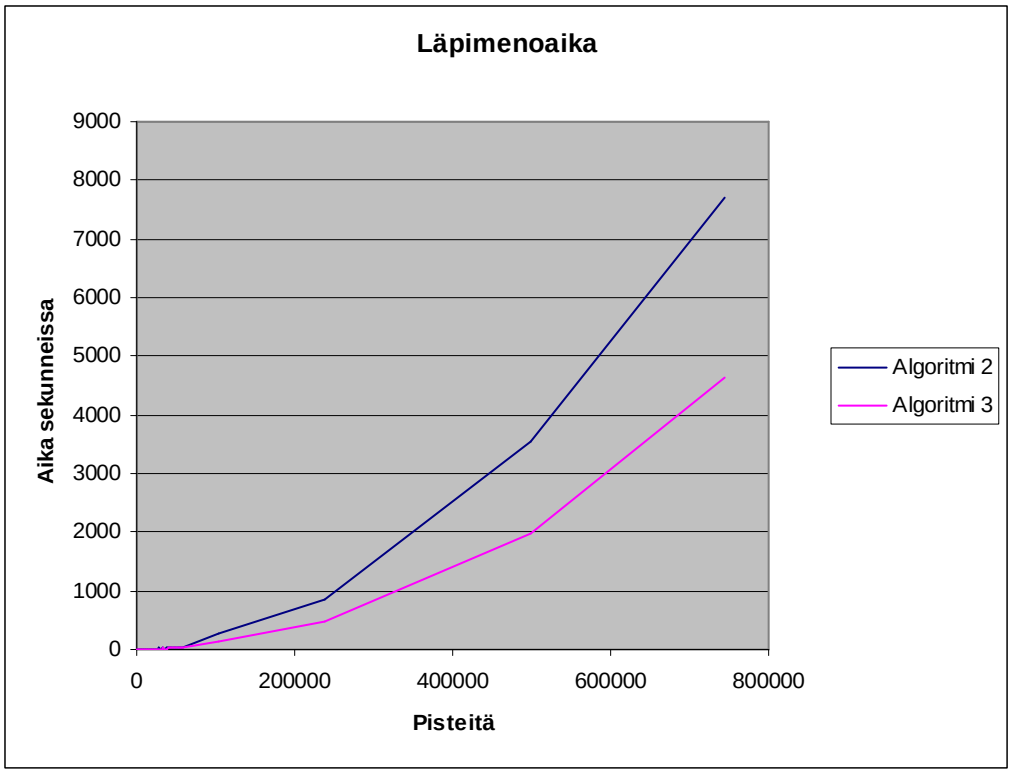
reittiin ja keskimäärin 1,30-kertainen, kun sitä verrataan parhaaseen löydettyyn reittiin. Euklidisella etäisyydellä laskettuna suhteessa ei tapahtunut muutosta. Kuva 32 havainnollistaa, kuinka vähän arvot poikkeavat toisistaan.



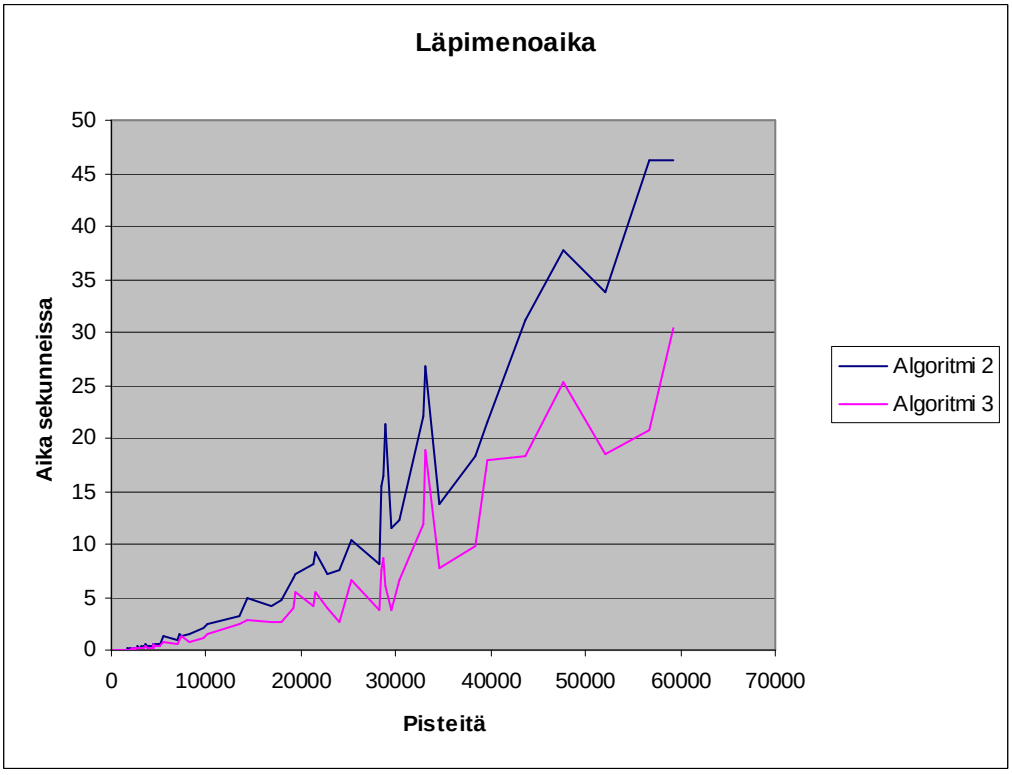
Kuva 32. Reitin pituuden kehitys Algoritmilla 3.

6.4.2 Nopeus

Testeissä Algoritmi 3 paljastui merkittävästi nopeammaksi kuin Algoritmi 2. Algoritmin 2 läpimenoaika on keskimäärin 1,57-kertainen verrattuna Algoritmin 3 läpimenoaikaan. Verrattuna Algoritmiin 3 Algoritmin 1 läpimenoaika oli keskimäärin 15,1 kertaa suurempi, kun taas Algoritmiin 2 verrattuna Algoritmin 1 läpimenoaika oli keskimäärin vain 9,0 kertaa suurempi. Suurimmillaan Algoritmin 1 läpimenoaika oli jopa 50,4-kertainen verrattuna Algoritmin 3 läpimenoaikaan. Kuvat 33 ja 34 näyttävät Algoritmien 2 ja 3 välisen tilanteen kehityksen ensin koko testimateriaalilla ja sitten pistesarjoilla, joissa on 131–59296 pistettä.



Kuva 33. Algoritmien 2 ja 3 läpimenoajan kehitys pistesarjoilla 131–744710.



Kuva 34. Algoritmien 2 ja 3 läpimenoajan kehitys pistesarjoilla 131–59296.

Kuvasta 34 nähdään, että vaikka Algoritmin 3 läpimenoaika onkin laskenut tasaisesti koko aineistolla, niin aiemmin havaitut piikit löytyvät edelleen tuloksista.

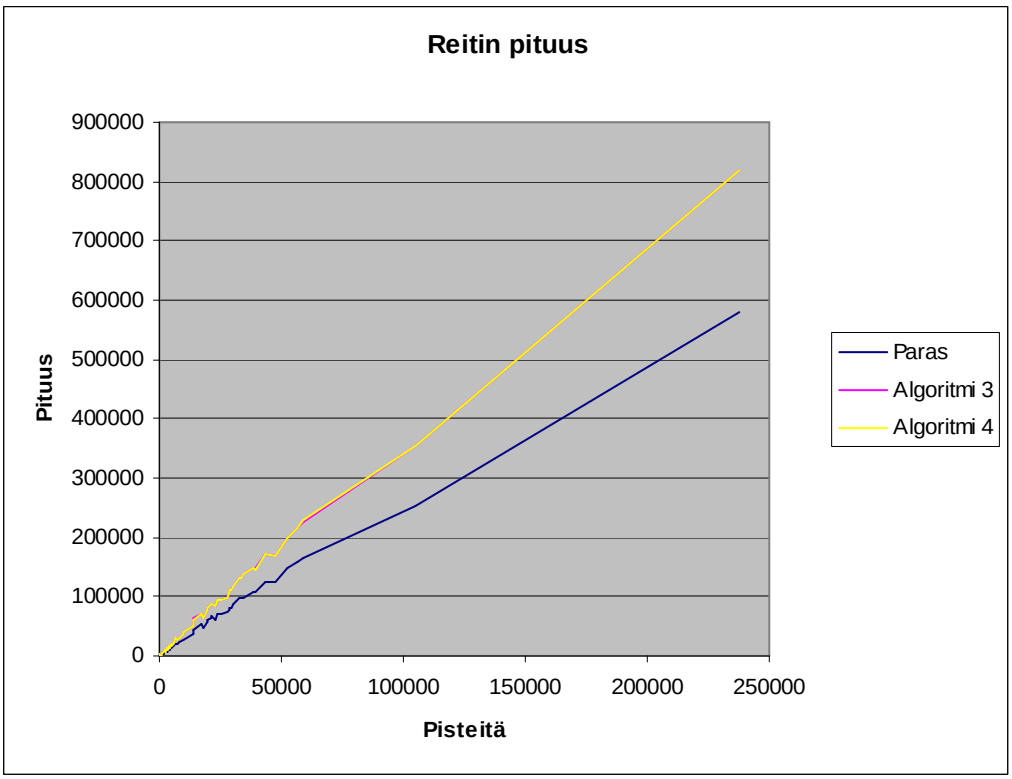
Tämä on loogista, sillä Algoritmin 3 sisältämä muutos vaikutti ainoastaan Algoritmin 2 pääosan eli uuden lisäyspisteen etsinnän toimintaan.

6.5 Algoritmi 4

Algoritmi 4 saatiin Algoritmista 3 sillä muutoksella, että konveksien verhojen muodostuksessa käytetään menetelmää, jossa kulmapiste valitaan mahdollisimman keskeltä aineistoa.

6.5.1 Tarkkuus

Algoritmin 4 sisältämä muutos on luonteeltaan sellainen, että sen ei tulisi aiheuttaa tarkkuuden heikkenemistä tai paranemista suhteessa Algoritmiin 3. Yleisesti ottaen näin onkin, mutta muutamia poikkeuksia mahtuu aineistoon. Tämä johtuu siitä, että kun osareitin sisäpuolella oleva konveksi verho muodostetaan eri tavalla, niin pisteet tulevat siihen eri järjestyksessä. Tällä tarkoitetaan sitä, että lista joka sisältää konveksin verhon saattaa alkaa eri alkiosta. Tällöin alkiot käsitellään eri järjestyksessä ja joissain tapauksissa voi käydä niin, että yhtä suuren kustannuksen aiheuttavista lisäyksistä tulee eri lisäys tehdyksi. Tarkkuudeltaan Algoritmit 3 ja 4 ovat niin lähellä toisiaan, että tämän tarkempi tarkastelu ei ole mielekästä, sillä esimerkiksi kuvasta 35 näiden kahden eri algoritmin antaman matkan kehitystä ei voi erottaa toisistaan.

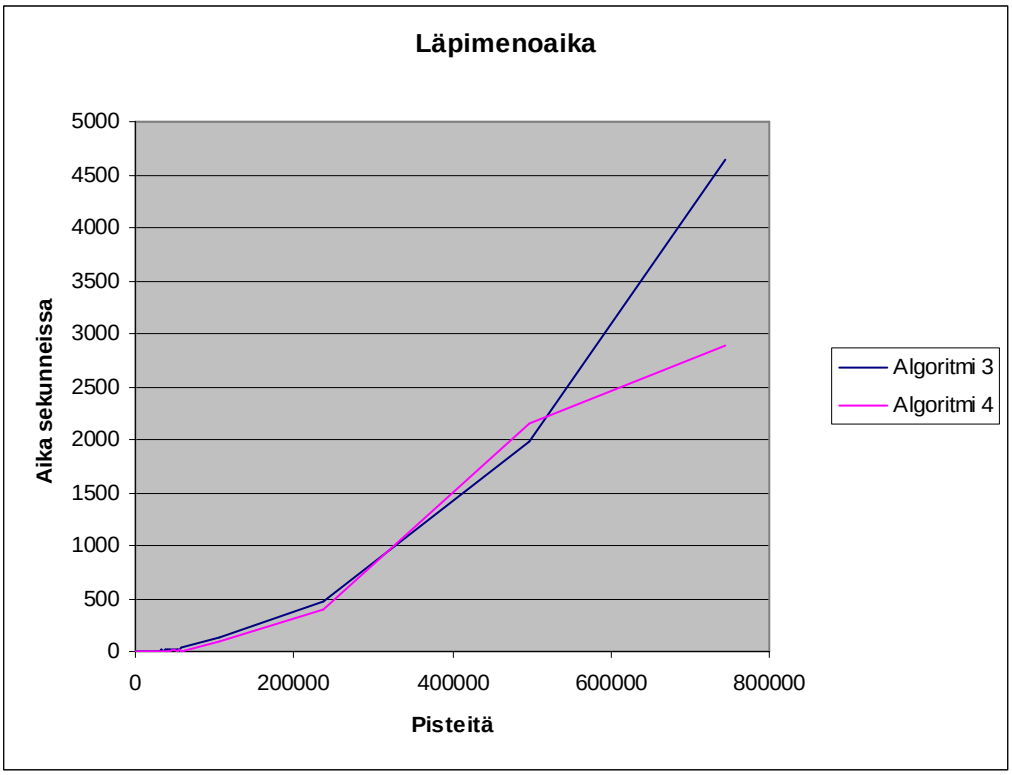


Kuva 35. Reitin pituuden kehitys Algoritmilla 4.

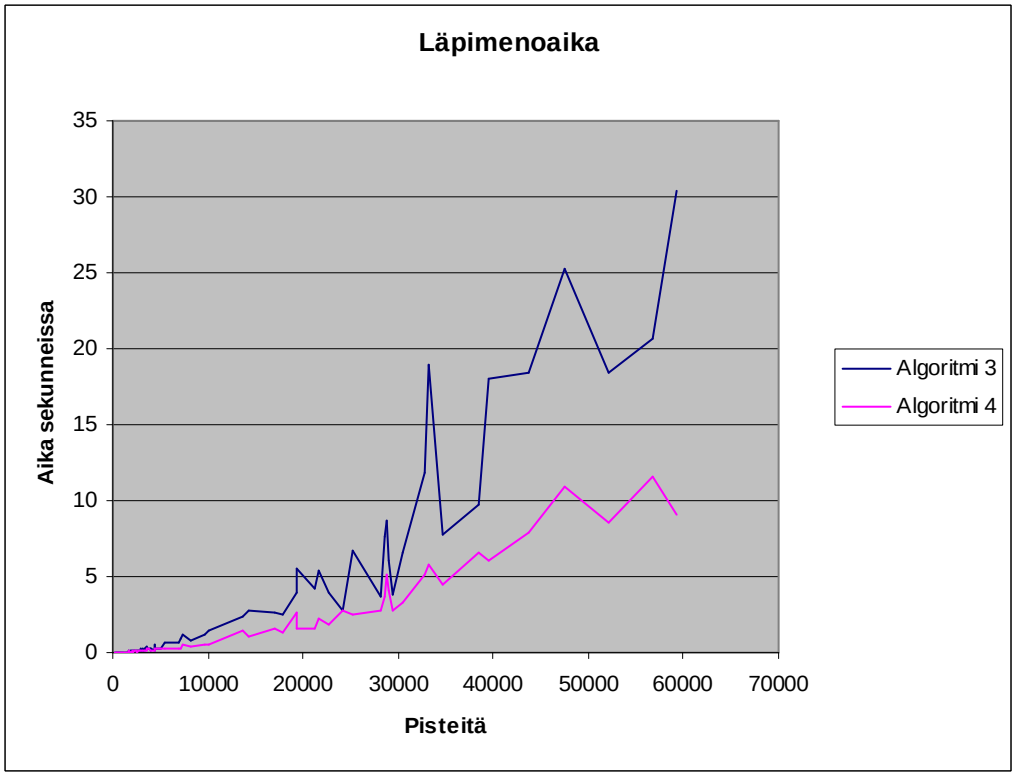
6.5.2 Nopeus

Testitulokset osoittavat Algoritmin 4 useimmissa tapauksissa nopeammaksi verrattaessa Algoritmiin 3. Algoritmin 3 läpimenoaika oli keskimäärin 1,79-kertainen verrattuna Algoritmin 4 läpimenoaikaan. Algoritmin 2 läpimenoaika oli keskimäärin 2,75-kertainen ja Algoritmilla 1 27,8-kertainen verrattuna Algoritmin 4 läpimenoaikaan. Suurimmillaan Algoritmin 1 läpimenoaika oli 89,6-kertainen verrattuna Algoritmiin 4 ja pienimmillään 4,88-kertainen.

Algoritmi 3 oli yksittäisissä tapauksissa nopeampi kuin Algoritmi 4, mutta tämä on selitettävissä sillä, että kyseisillä tiedostoilla Algoritmi 3 ei joutunut tekemään monia kulmajärjestyksiä. Kuvissa 36 ja 37 vertaillaan Algoritmeja 3 ja 4. Koko testimateriaalin ollessa kyseessä yksittäinen suurella tiedostolla tapahtuva tilanne, jossa Algoritmi 3 on nopeampi kuin Algoritmi 4, hieman vääristää kuvaa 36.



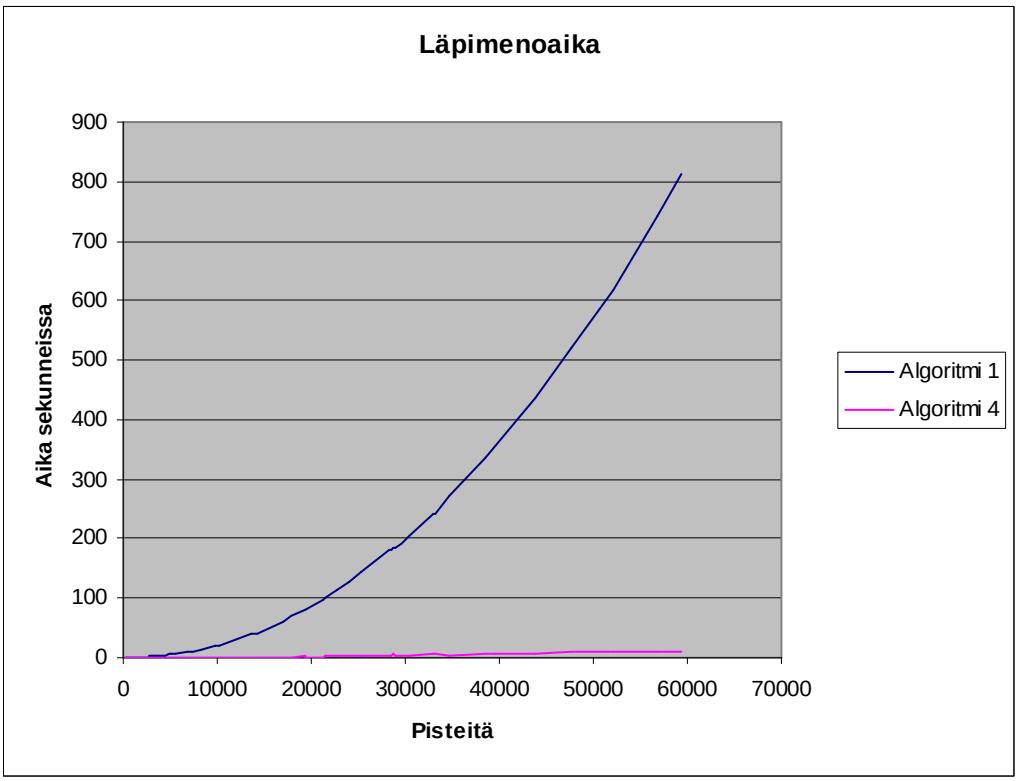
Kuva 36. Algoritmin 4 läpimenoajan kehitys pistesarjoilla 131–744710.



Kuva 37. Algoritmin 4 läpimenoajan kehitys pistesarjoilla 131–59296.

Erityisesti kuvasta 37 nähdään, kuinka menetelmä, jossa kulmajärjestys tehdään mahdollisimman keskeltä löytyvän pisteen mukaan, tasoittaa läpimenoajan kehitystä.

Algoritmin 4 nopeudesta pistesarjoilla, joiden koot ovat väliltä 131–59296, saa parhaiten mielikuvan, kun sitä verrataan Algoritmin 1 nopeuteen samoilla sarjoilla. Kuva 38 esittää tätä tilannetta.



Kuva 38. Algoritmien 1 ja 4 vertailu pistesarjoilla, joiden koot ovat 131–59296.

Kuten kuvasta 38 voidaan nähdä, on Algoritmi 4 huomattavasti nopeampi kuin Algoritmi 1. Lisäksi tulee muistaa, että Algoritmin 4 antama reitti ei ole merkittävästi huonompi verrattuna Algoritmin 1 antamaan reittiin.

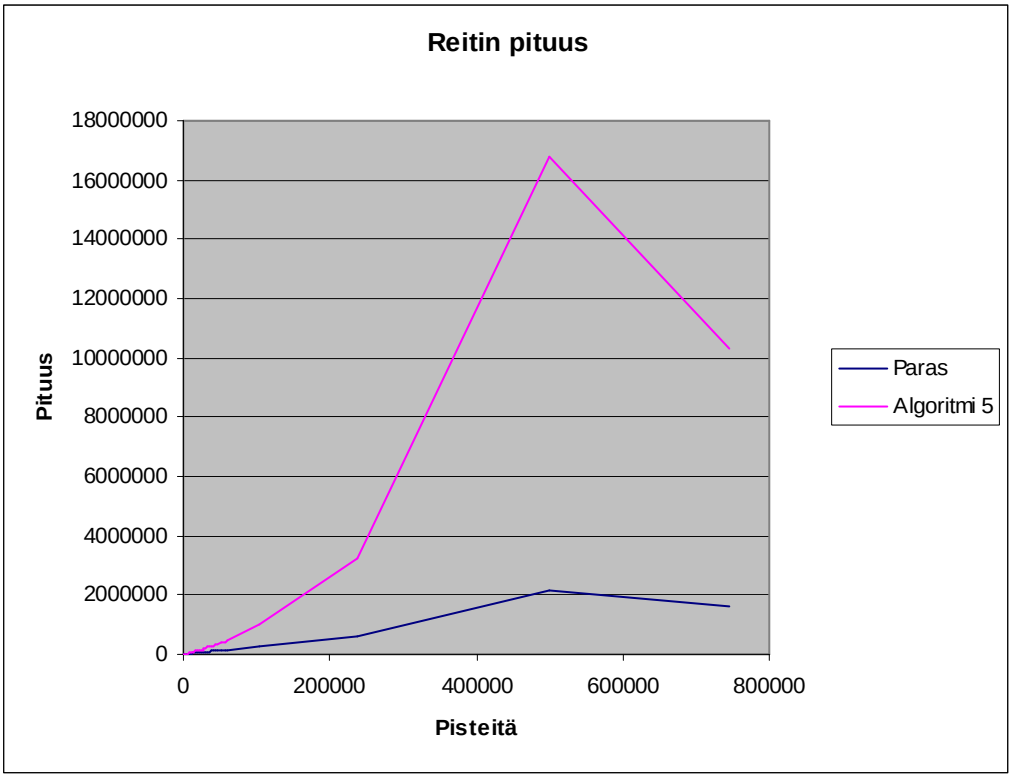
6.6 Algoritmi 5

Algoritmi 5 toimii muuten samoin kuin Algoritmi 4, mutta sen osareitin sisäpuolella olevan konveksin verhon uusien pisteiden lisäyskohdan etsintä on muutettu. Algoritmissa 5 näille pisteille etsitään lisäyskohta valitsemalla lisäyskohta pisteen kummaltakin puolelta löytyvän lisäyskohdan omaavan pisteen lisäyskohtien ja osareittiin edellisen lisäyksen johdosta syntyneiden kahden lisäyskohdan joukosta.

Algoritmin 5 tarkoituksena on antaa alaraja Algoritmeissa 2, 3 ja 4 käytetyn idean suorituskyvylle. Tällöin voidaan paremmin arvioida erityisesti Algoritmin 4 suorituskyvyn tasoa.

6.6.1 Tarkkuus

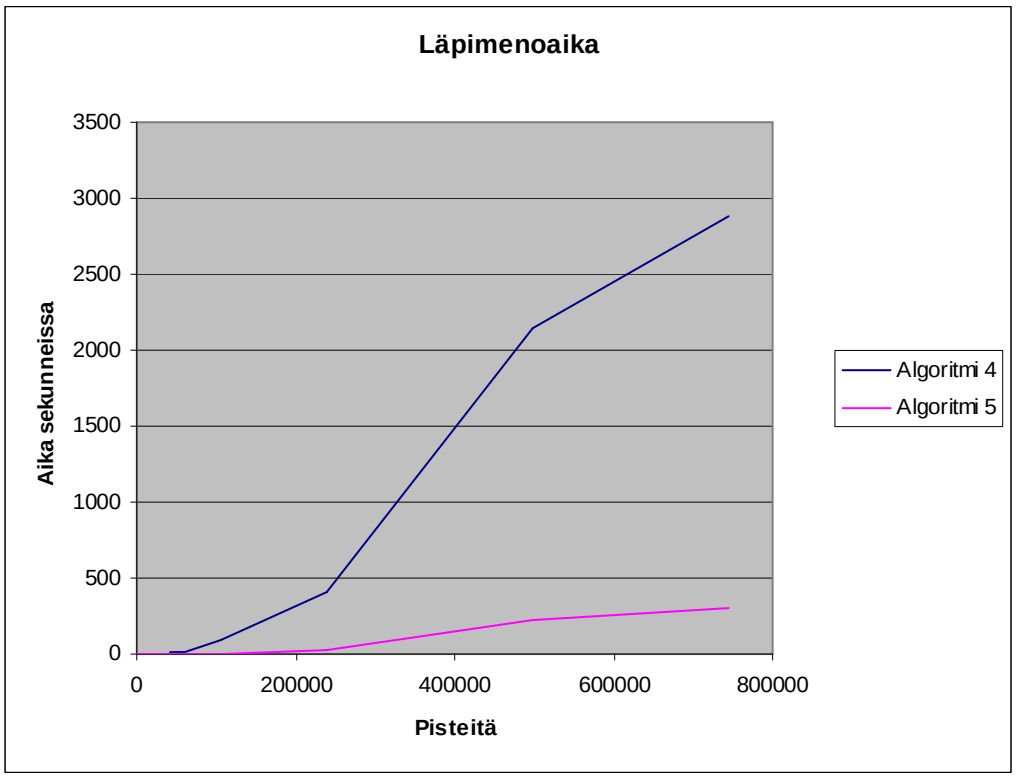
Algoritmin 5 kohdalla reitin tarkkuuden taso romahtaa nopeasti pistesarjojen koon kasvaessa, mutta pienemmille pistesarjoille se antaa melko tarkan arvion. Keskimäärin Algoritmin 5 antama reitti oli 2,01-kertainen parhaaseen löydettyyn reittiin verrattuna. Suurimmillaan reitti oli 7,75-kertainen ja pienimmillään 1,20-kertainen. Sinänsä yllättävää on se, että Algoritmi 5 antaa pienemmillä pistesarjoilla niinkin tarkan arvion kuin se nyt antaa. Kuten kuva 39 osoittaa, huononee algoritmin antama reitti tasaisesti pistesarjojen koon kasvaessa.



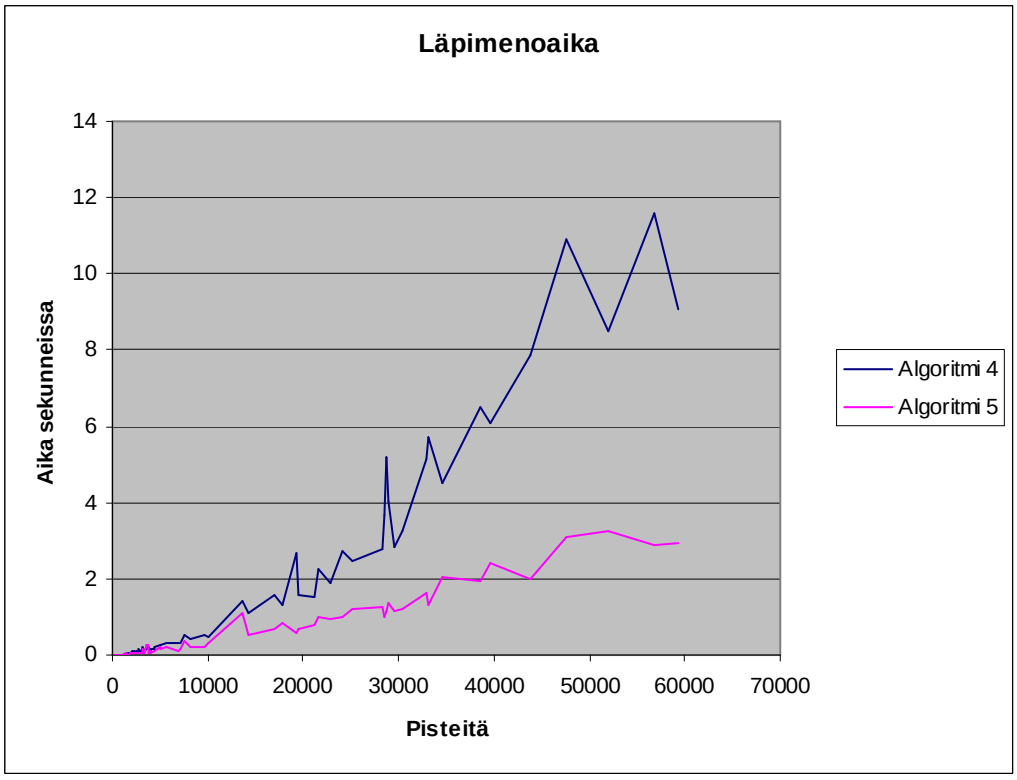
Kuva 39. Algoritmin 5 reitin pituuden kehitys pistesarjoilla 131–744710.

6.6.2 Nopeus

Algoritmin 5 nopeus on niin suuri, että sen vertailu muihin kuin Algoritmin 4 nopeuteen ei ole mielekästä. Yllättävää kuitenkin on se, että Algoritmin 4 läpimenoaika on keskimäärin vain 2,55-kertainen Algoritmin 5 läpimenoaikaan verrattuna. Suurimmillaan tämä arvo on 15,4-kertainen, mutta tiedostolla fjs3649.tsp Algoritmi 4 oli jopa nopeampi kuin Algoritmi 5. Kuvat 40 ja 41 selventävät tilannetta.

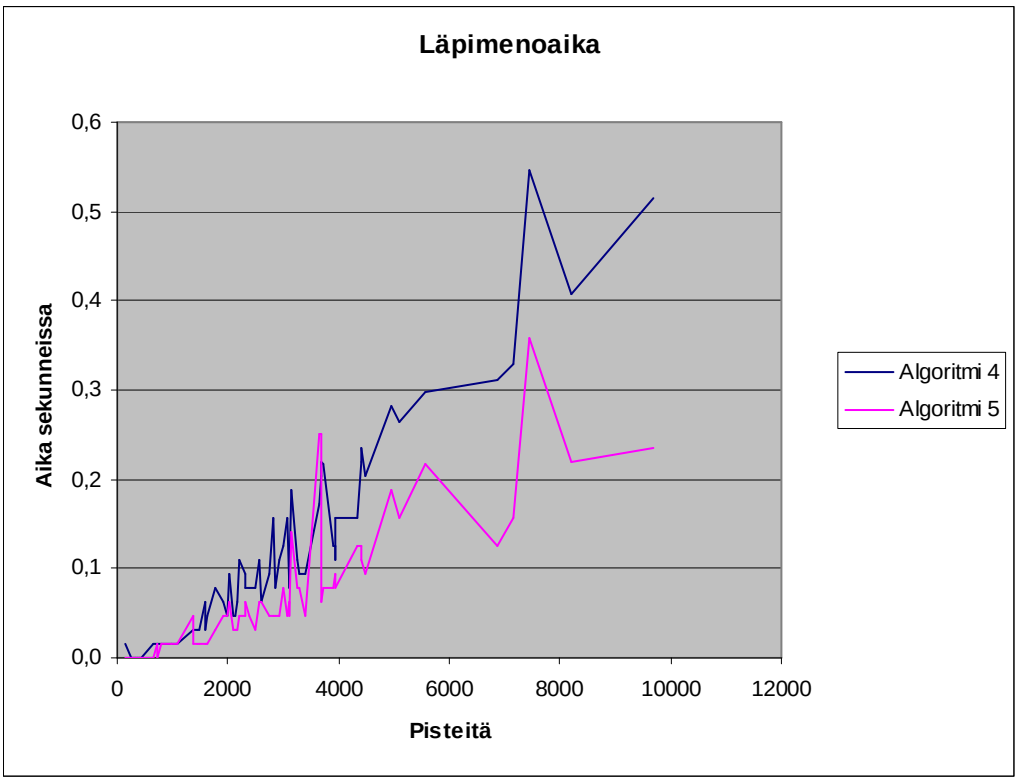


Kuva 40. Algoritmien 4 ja 5 läpimenoajat pistesarjoilla 131–744710.



Kuva 41. Algoritmien 4 ja 5 läpimenoajat pistesarjoilla 131–59296.

Erityisesti kuvasta 41 nähdään, että vaikka Algoritmi 4 onkin hitaampi kuin Algoritmi 5, niin sen läpimenoaika pysyy kuitenkin melko lähellä Algoritmin 5 läpimenoaikaa. Vielä pienemmillä pistesarjoilla ero käy niin pieneksi, että Algoritmin 5 käyttäminen edes pelkästään nopeuden takia ei ole kovin mielekästä. Kuva 42 havainnollistaa tilannetta.



Kuva 42. Algoritmien 4 ja 5 läpimenoajat pistesarjoilla 131–9698.

7 Johtopäätöksiä ja arvioita algoritmien hyödyllisyydestä

Esitetään johtopäätöksiä ja arvioidaan algoritmien hyödyllisyyttä.

7.1 Johtopäätöksiä

Tässä kohdassa tehdään yhteenveto luvun 6 johtopäätöksistä.

7.1.1 Tarkkuus

Kokeellisen tutkimuksen perusteella voidaan sanoa, että Algoritmit 1, 2, 3 ja 4 näyttävät olevan ainakin 1-approksimointialgoritmeja eli ne tekevät enintään 100 prosentin virheen. Voidaan myös melko varmasti sanoa niiden olevan tai ainakin näyttävän olevan $\frac{1}{2}$ - approksimointialgoritmeja eli ne tekevät enintään 50 prosentin virheen.

Tulosten perusteella voidaan myös sanoa, ettei Algoritmien 2, 3 ja 4 tarkkuus poikkea merkittävästi Algoritmin 1 tarkkuudesta. Voidaan kuitenkin sanoa Algoritmien 2, 3 ja 4 olevan tarkkuuden osalta alisteisia Algoritmille 1.

7.1.2 Nopeus

Tutkimustulosten perusteella voidaan sanoa Algoritmien 2, 3 ja 4 olevan selvästi nopeampia kuin Algoritmi 1. Läpimenoaikojen ero on erityisesti suuremmilla pistesarjoilla niin merkittävä, että tehtäessä nopeaa approksimointia suurille pistesarjoille ei Algoritmin 1 käyttämisille ole perusteita. Voidaan siis sanoa Algoritmin 1 olevan nopeuden suhteen alisteinen Algoritmeille 2, 3 ja 4.

Erittäin mielenkiintoista tutkimustuloksissa on se, että Algoritmin 4 läpimenoajat olivat melko lähellä Algoritmin 5 läpimenoaikoja. Tämän tuloksen voidaan katsoa osoittavan, että Algoritmin 4 sisältämät tutkimusta nopeuttavat rajoitteet ja konveksin verhon korjausmenetelmä toimivat erittäin hyvin.

7.2 Arvioita algoritmien hyödyllisyydestä

Lopuksi arvioidaan vielä Algoritmien 2, 3 ja 4 hyödyllisyyttä.

7.2.1 Algoritmi 2

Algoritmin 2 peruslogiikka voidaan todeta toimivaksi. Myös sen voidaan sanoa olevan hyvä korkeintaan vaativuusluokan $O(n^2)$ approksimointialgoritmi. Algoritmin 2 toiminta on selkeästi ennustettavaa reitin muodostumisen suhteen, mutta ei niin selkeästi ennustettavaa siihen kuluvan ajan suhteen. Huomionarvoista on kuitenkin, että

algoritmi pysyy nopeuden suhteen kilpailukykyisenä huolimatta sen läpimenoajassa ilmenevästä voimakkaasta vaihtelusta.

7.2.2 Algoritmi 3

Koska Algoritmi 3 on selkeästi nopeampi kuin Algoritmi 2, ja Algoritmin 3 voidaan katsoa olevan matkan tarkkuuden osalta hyvin lähellä Algoritmia 2, niin ei voida nähdä perusteita Algoritmin 2 käyttämiselle Algoritmin 3 sijaan. Algoritmin 3 toiminta on Algoritmin 2 tavoin selkeästi ennustettavaa reitin muodostumisen suhteen, mutta ei niin selkeästi ennustettavaa siihen kuluvan ajan suhteen.

7.2.3 Algoritmi 4

Algoritmin 4 voidaan sanoa olevan erittäin nopea. Vertailussa Algoritmiin 5 nähtiin, että Algoritmin 4 uuden lisäyksen tutkiva osuus toimi nopeudella, joka joissain tapauksissa lähestyi Algoritmin 5 kyseisen osan lineaarista nopeutta. Tulosten perusteella Algoritmi 4 näyttäisi olevan tarkkuudeltaan hyvin lähellä $\frac{1}{2}$-approksimointialgoritmia. Se on siis sekä erittäin nopea että vertailukohtiin nähden tarkka. Algoritmin 4 perustoiminta on siis selkeästi ennustettavaa sekä reitin muodostumisen että siihen kuluvan ajan suhteen.

Algoritmin 4 puutteeksi voidaan nähdä sen kehitystyön osittainen keskeneräisyys, mutta voidaan kuitenkin perustellusti sanoa, että Algoritmin 4 toteutuksen merkittävimmät ongelmat on saatu ratkaistua. Voidaan myös todeta, että siitä on mahdollista kehittää täysin vakaa versio.

8 Yhteenveto

Tutkielmassa tarkasteltiin uusien, tutkielmaa varten kehitettyjen, kauppamatkustajan ongelman approksimointialgoritmien toimintaa suorittamalla niille kokeellinen tutkimus, jossa niitä verrattiin jo olemassa olevaan konveksia verhoa hyödyntävän algoritmin toteutukseen. Tutkimuksessa käytettiin testimateriaalina 102 eri aineistoa, jotka löytyvät internetistä sivustolta http://www.tsp.gatech.edu/vlsi/index.html. Sivuston mukaan aineisto perustuu Bonnin yliopiston diskreetin matematiikan tutkimusinstituutin käyttämiin pistesarjoihin ja aineiston on toimittanut sinne Andre Rohe. Aineiston pienin pistesarja sisältää 131 pistettä ja suurin 744710 pistettä. Osalle pistesarjoista on laskettu paras mahdollinen optimireitti ja osalle paras tähän asti löydetty reitti.

Tutkimus keskittyi algoritmien nopeuteen ja niiden muodostaman reitin tarkkuuteen. Lisäksi tutkimuksessa pyrittiin kokeellisesti todentamaan omien algoritmien sisältämiä ongelmia ja osoittamaan niihin tehdyt korjaukset toimiviksi. Tutkimuksen yhteydessä suoritettiin myös vertailukohtana olleen algoritmin teoreettisen vaativuustarkastelun todentaminen kokeellisella tutkimuksella saatujen tulosten perusteella. Tutkimuksen selkeänä tuloksena voidaan sanoa, että uusien algoritmien käyttämiselle, vertailukohtana olleen algoritmin sijaan, on olemassa vahvat perusteet tilanteessa, jossa algoritmin läpimenoaika on tärkeässä asemassa. Kaikki idean pohjalta toteutetut uudet algoritmit olivat selvästi nopeampia kuin vertailukohtana toimiva algoritmi, eikä niiden antamien reittien pituudessa tapahtunut dramaattista huononemista vertailukohdan antamaan vastaukseen nähden. Uusien algoritmien tarjoama nopeuden kasvu on niin suurta, että edes algoritmien rakenteen monimutkaisuutta, ja sitä kautta algoritmien vaikeampaa implementointia, ei voida pitää esteenä niiden käyttämiselle.

Tutkittava aihe on ollut mielenkiintoinen, mutta erittäin haastava. Aiheen haastavuus on syntynyt tietosisällön vaikeudesta ja tutkimuksen vaatimasta kehitys- ja suunnittelutyöstä. Kritiikkinä tätä työtä kohtaan voitaisiin sanoa, että Algoritmin 4 kehitystyö olisi ollut hyvä saattaa täysin loppuun. Esteenä kehitystyön täydelliselle loppuun saattamiselle oli kuitenkin omien aika-arvioiden ylittäminen. Kehitystyö alkoi vuoden 2006 heinäkuussa ja vuoden 2007 tammikuussa se oli pakko lopettaa ja siirtyä kokeellisen tutkimuksen pariin. Tutkimuksen toisena heikkoutena voidaan pitää teoriaosuuden lyhyyttä ja monessa suhteessa pintaraapaisun omaista suppeutta. Käsiteltävä aihealue on kuitenkin laajimpia koko optimoinnin teoriassa ja tarkempi yksityiskohtiin paneutuminen olisi nopeasti paisuttanut kirjallisen osuuden liian pitkäksi. Oli siis tehtävä rajuja rajauksia sen suhteen, mitä asioita esitellään

syvällisemmin ja mitä vain pintapuolisesti. Tämän takia kirjallisuuskatsaus ja kauppamatkustajan ongelman teoreettinen tarkastelu pyrittiin tekemään sellaisiksi, että ne palvelisivat työtä kokonaisuutena, eivätkä vain sen osina.

Henkilökohtaisesti pidän tutkimuksen tavoitetta saavutettuna. Kehitettyä algoritmi-ideaa voidaan pitää niin tehokkaana, että sen olemassa ololle on oikeutus.

Viitteet

[Aho *et al.*, 1983] Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman, *Data Structures and Algorithms*. Addison-Wesley, 1983.

[Applegate *et al.*, 2007] David l. Applegate, Robert E. Bixby, Vasek Chavátal and William J. Cook, *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[Balas and Toth, 1987] E. Balas, P. Toth, Branch and bound methods. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Bellman, 1962] Richard Bellman, Dynamic Programming Treatment of the Travelling Salesman Problem, *Journal of the Association for Computing Machinery*, **9**, (1962), 61–63.

[Bellmore and Nemhauser, 1968] M. Bellmore, G. L. Nemhauser, The Traveling Salesman Problem: A Survey, *Operations Research*, **16**, 3, (1968) , 538–558.

[Christofides, 1972] Nicos Christofides, Bounds for the Traveling-Salesman Problem, *Operations Research*, **20**, 5, (1972), 1044–1056.

[Christofides, 1975] Nicos Christofides, *Graph Theory An Algorithmic Approach*. Academic Press Inc., 1975.

[Crowder and Padger, 1980] Harlan Crowder and Manfred W. Padger, Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality, *Management Science*, **26**, 5, (1980), 495–509.

[Dantzig *et al.*, 1954] G. Dantzig, R. Fulkerson and S. Johnson, Solution of a Large-Scale Traveling-Salesman Problem, *Journal of the Operations Research Society of America*, **2**, 4, (1954), 393–410.

[Dantzig *et al.*, 1959] G. Dantzig, R. Fulkerson and S. Johnson, On a Linear-Programming, Combinatorial Approach to the Traveling-Salesman Problem, *Operations Research*, **7**, 1, (1959), 58–66.

[Flood, 1956] Merrill M. Flood, The Traveling Salesman Problem, *Operations Research*, **4**, 1, (1956), 61–75.

[Garfinkel, 1987] R. S. Garfinkel, Motivation and modeling. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Gilmore *et al.*, 1987] P. C. Gilmore, E. L. Lawler and D. B. Shmoys, Well solved special cases. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Golden and Stewart, 1987] B. L. Golden, W. R. Stewart, Empirical analysis of heuristics. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Golden *et al.*, 1980] B. Golden, L. Bodin, T. Doyle, and W. Stewart, Jr., Approximate Traveling Salesman Algorithms, *Operations Research*, **28**, 3, Part II, (1980), 694–711.

[Held and Karp, 1961] Michael Held and Richard M. Karp, A dynamic programming approach to sequencing problems, *Proceedings of the 16th Association for Computing Machinery Meeting* , (1961), 71.201–71.204.

[Hoffman and Wolfe, 1987] A. J. Hoffman, P. Wolfe, History. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Hämäläinen, 2003a] Pentti Hämäläinen, *Algoritmit 1*. Jyväskylän yliopisto, 2003.

[Hämäläinen, 2003b] Pentti Hämäläinen, *Algoritmit 2*. Jyväskylän yliopisto, 2003.

[Johnson and Papadimitriou, 1987a] D. S. Johnson, C. H. Papadimitriou, Computational complexcity. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Johnson and Papadimitriou, 1987b] D. S. Johnson, C. H. Papadimitriou, Performance guarantees for heuristics. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Karp and Steele, 1987] R. M. Karp, J. M. Steele, Probabilistic analysis of heuristics. In: E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D.B Shmoys (eds.), *The Traveling Salesman Problem*. Wiley-Interscience, 1987.

[Kocay and Kreher, 2005] William Kocay, Donald L. Kreher, *Graphs, Algorithms and Optimization*. Chapman & Hall/CRC Press, 2005.

[Lin and Kernighan, 1973] S. Lin, B. W. Kernighan, An Effective Heuristic Algorithm for the Traveling-Salesman Problem: A Survey, *Operations Research*, **16**, 3, (1968), 538–558.

[Little *et al.*, 1963] John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel, An Algorithm for the Traveling Salesman Problem, *Operations Research*, **11**, 6, (1963), 972–989.

[Miller *et al.*, 1960] C. E. Miller, A. W. Tucker and R. A. Zemlin, Integer Programming Formulation of Traveling Salesman Problems, *Journal of the Association for Computing Machinery*, **7**, (1960), 326–329.

[Norback and Love, 1977] John P. Norback, Robert F. Love, Geometric Approaches to Solving the Traveling Salesman Problem, *Management Science*, **23**, 11, (1977), 1208–1223.

[Orponen ja Ernvall, 2004] Pekka Orponen ja Jarno Ernvall, *Algoritmitekniikka 2004*. Jyväskylän yliopisto, 2004.

[Reinelt, 1994] Gerhard Reinelt, The Traveling Salesman Computational Solutions for TSP Applications. Springer-Verlag, 1994.

[Savolainen, 2001] Vesa Savolainen, Verkkoteoria. Docendo, 2001.

[Somhom *et al.*, 1997] S. Somhom, A. Modares and T. Enkawa, A Self-Organising Model for the Traveling Salesman Problem, *The Journal of the Operational Research Society*, **48**, 9, (1997), 919–928.