Joni Brigatti

Lyhimmän polun laskennan tehostaminen tieverkossa

Tietotekniikan
(simulointi ja optimointi)
pro gradu -tutkielma
15. kesäkuuta 2009

JYVÄSKYLÄN YLIOPISTO
TIETOTEKNIIKAN LAITOS

Jyväskylä

Tekijä: Joni Brigatti
Yhteystiedot: joni.brigatti@jyu.fi
Työn nimi: Lyhimmän polun laskennan tehostaminen tieverkossa
Title in English: Speeding Up Shortest Path Computation in Road Network
Työ: Tietotekniikan (simulointi ja optimointi) pro gradu -tutkielma
Sivumäärä: 28
Tiivistelmä: Tutkielmassa esitetään kaksi menetelmää, joilla voidaan tehostaa lyhimmän polun laskentaa tieverkossa. Lyhimmän polun laskentaa voidaan tehostaa perinteisillä tai nykyaikaisilla kiihdytystekniikoilla, joista jälkimmäiset vaativat esiprosessointia. Esitetyt menetelmät eivät vaadi esiprosessointia ja niitä käytetään, kun kohdepisteet on annettu.
English abstract: In this thesis, two methods for speeding up the shortest path computation in road network is presented. The efficiency of shortest path computation can be increased with conventional or modern speedup techniques, of which the latter require preprocessing. Presented methods do not require preprosessing and are used after target nodes are known.
Avainsanat: Lyhimmän polun ongelma, tieverkko, kiihdytystekniikka, väliprosessointi
Keywords: Shortest path problem, road network, speed up technique, midprosessing

Sisältö

Kiitokset

Tämä tutkielma on kiitos kaikille niille ihmisille, jotka ovat omalla tavallaan vaikuttaneet tutkielman valmistumiseen.

Haluan kiittää tutkielmani ohjaajia, professori Olli Bräysyä ja tutkija Tuukka Purasta, tuesta prosessin aikana. Erityisesti kiitän heitä arvokkaista keskusteluista, ideoista, palautteen määrästä ja kärsivällisyydestä. Lisäksi haluan kiittää Anna-Maija Rahkosta palautteesta kieliasun suhteen ja rohkaisevista kommenteista. Arvostan myös yliassistentti Timo Männikön, yliassistentti Jani Kurhisen, professori Raino A.E. Mäkisen ja professori Tuomo Rossin viimeisiä kommentteja ja korjausehdotuksia.

Haluan kiittää professori Olli Bräysyä luottamuksesta ja mahdollisuudesta työskennellä hänen ryhmässään Agora Centerissä, Jyväskylän yliopistossa. Haluan kiittää Tekesiä ja asiakasyrityksiä projektien OPT-LOG ja TRANSOPT rahoittamisesta, joiden aikana tutkielma on tehty.

Haluan muistaa myös vanhempiani, Jukkaa ja Arjaa, sekä veljiäni, Kimmoa ja Miikaa, kannustuksesta ja yhteisistä hetkistä.

Sisälletyt artikkelit

PI T. Puranen, J. Brigatti, *Topology-based Optimal Road Network Reduction - a Method for Speeding up Shortest Path Computation*, Evolutionary Methods for Design, Optimization and Control, 2007, sivut 409–415.

PII J. Brigatti, *Kohdepisteiden topologinen koostaminen lyhimmän polun ongelmassa*, Reports of the Department of Mathematical Information Technology, Series B4/2009, Scientific Computing, University of Jyväskylä, ISBN 978-951-39-3598-6.

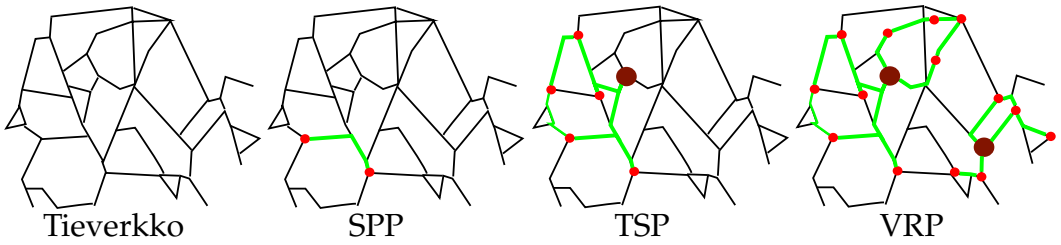
Artikkelissa **PI** tekijöinä olivat tutkielman tekijä ja tutkija Tuukka Puranen. Tutkija Puranen toteutti artikkelissa kuvatun menetelmän. Tutkielman tekijä laati kirjallisuuskartoituksen ja toteutti menetelmän testauksen sekä muut vaaditut algoritmit (Dijkstran algoritmi, A*-algoritmi). Artikkelin kirjoitus ja menetelmän ideointi olivat yhteistyötä.

Raportissa **PII** tekijänä oli tämän tutkielman tekijä. Raportissa käytettiin hyväksi edellistä artikkelia varten luotuja menetelmiä. Uudet luodut menetelmät ovat tutkielman tekijän kehittämiä.

1 Johdanto

Tässä tutkielmassa tutustutaan monesta moneen -lyhimmän polun ongelmaan tieverkossa. Lyhimmän polun ongelma (engl. *Shortest Path Problem*, SPP) [1] on paljon tutkittu ongelma, jossa ratkaistaan lyhin polku tai lyhin etäisyys kahden kohdepisteen välille annetussa verkossa. Monesta moneen -lyhimmän polun ongelmassa (engl. *Many-to-Many Shortest Path Problem*, MTMSSP) ratkaistaan lyhimmät polut tai lyhimmät etäisyydet kaikkien annettujen kohdepisteiden välillä.

Monesta moneen -lyhimmän polun ongelma joudutaan usein ratkaisemaan osana maantieliikenteeseen liittyviä optimointiongelmia. Tällaisia ongelmia ovat esimerkiksi kauppamatkustajan ongelma (engl. *Traveling Salesman Problem*, TSP) [30] ja kaluston reitinoptimointiongelma (engl. *Vehicle Routing Problem*, VRP) [40]. Kauppamatkustajan ongelmassa etsitään optimireitti kohdepisteiden välille siten, että reitti on lyhin mahdollinen ja jokaisessa kohdepisteessä käydään vain kerran. Kaluston reitinoptimointiongelma koostuu yhdestä tai useammasta kauppamatkustajan ongelmasta. Esitellyissä optimointiongelmissa kohdepisteet voivat olla asiakkaita, terminaaleja, varastoja tai muita määriteltyjä pisteitä. Molemmat optimointiongelmat ovat kombinatorisia eivätkä ole ratkaistavissa polynomisessa ajassa NP-täydellisyyden vuoksi tämän hetkisen tietämyksen valossa [23, 37].



Kuva 1.1: Maantieliikenteen optimoinnin vaiheet.

Useimmat TSP- ja VRP-algoritmit vaativat täydellisen etäisyysmatriisin kohdepisteiden välille (ks. esimerkiksi [27, 31]). Etäisyysmatriisi on tietorakenne, johon on tallennettu tarvittavien lyhimpien polkujen etäisyydet. Ongelmien NP-täydellisyyden ja tieverkon koon vuoksi lyhimmän polun laskenta on suoritettava mahdollisimman nopeasti, että koko optimointiprosessi olisi mahdollista suorittaa lyhimmässä mahdollisessa ajassa. Kuvassa 1.1 havainnollistetaan optimointiprosessia, jossa ensin lasketaan lyhimpien polkujen etäisyydet tieverkosta. Lyhimmän po-

lun laskennan avulla muodostetaan etäisyysmatriisi, joka vaaditaan TSP- tai VRP-optimointiongelmien ratkaisuun.

Tieverkot voivat olla dynaamisia tai staattisia. Staattisissa tieverkoissa verkon solmujen väliset etäisyydet eivät muutu. Dynaamisia tieverkkoja käytetään esimerkiksi ruuhka-aikojen huomioimiseen lyhimmän polun laskennassa [7]. Jos solmujen väliset etäisyydet voivat muuttua ajan suhteen, on tämä huomioitava kaluston reitinoptimoinnissa. Edellä kuvattiin optimointiprosessi, jossa etäisyysmatriisi rakennettiin ennen kaluston reitinoptimoinnin ratkaisemista. Jos ongelma on liian dynaaminen eli etäisyysmatriisin jatkuva päivittäminen ei ole mahdollista, voidaan etäisyydet laskea yksitellen ja vain tarvittaessa ilman etäisyysmatriisin rakentamista.

Lyhimmän polun laskennalla ja kaluston reitinoptimoinnilla on useita käytännön sovelluskohteita. Lyhimmän polun laskentaa käytetään esimerkiksi autonavigaattoreissa ja reittihakupalveluissa. Autonavigaattorit etsivät lyhimmän polun sen hetkisestä sijainnista haluttuun kohteeseen. Reittihakupalvelut etsivät lyhimmän polun halutusta sijainnista haluttuun kohteeseen. Erona autonavigaattoreihin reittihakupalvelut kertovat lisäksi, mitä kulkuneuvoja on käytettävä reitin kulkemiseen, jolloin on huomioitava reitin alkamis- tai loppumisaika ja kulkuneuvojen aikataulut. Lyhimmän polun laskenta on myös kriittinen osa kaluston reitinoptimointiohjelmistoja, joissa etäisyysmatriisi on rakennettava varsinaista optimointia varten.

Sovellusalue	Sovelluskohde
Raaka-aineet	Öljyn, kaasun, puun tai vastaavan kuljetus
Kauppa	Tuotteiden toimitus
Julkinen sektori	Jätehuolto ja katujen ylläpito
Teollisuus	Huolto, robotit, materiaalivirrat ja varastointi
Rakennus	Materiaalien, täydennyksien ja työkalujen kuljetus
Media	Sanomalehtien, mainoksien ja postin jakelu
Kuljetus	Lähetys- ja kuljetusyritykset
Maatalous	Tuotteiden keruu, ruoan jakelu
Henkilökuljetus	Joukkoliikenne ja taksipalvelut
Pankki	Pankkiautomaattien tyhjennys ja täydennys
Terveydenhuolto	Kotisairaanhoito ja ruokapalvelu

Taulukko 1.1: Kaluston reitinoptimoinnin sovellusalueita.

Kaluston reitinoptimointia voidaan käyttää esimerkiksi jätehuollon [18] tai kotisairaanhoidon [32] logistiikan optimointiin. Jätehuollon tarkoitus on palvella annetut asiakkaat mahdollisimman tehokkaasti. Asiakkaita ovat esimerkiksi taloyh-

tiöiden, kauppojen tai yrityksien jätteenkeräysastiat. Optimoinnin avulla pyritään etsimään paras ratkaisu kalustolle ja asiakkaiden palvelujärjestykselle. Paras ratkaisu voi olla esimerkiksi kustannuksien, kaluston tai kilometrien minimointi. Kotisairaanhoidon tarkoitus on tarjota sairaanhoitoa potilailla heidän kotonaan ja auttaa arkisissa töissä kuten aamupalan tekemisessä ja pukeutumisessa. Optimoinnin avulla pyritään etsimään paras ratkaisu henkilökunnan allokoinnille ja potilaiden palvelujärjestykselle. Paras ratkaisu voi olla esimerkiksi palveluajan maksimointi, kustannuksien tai kilometrien minimointi. Taulukossa 1.1 luetellaan muita sovellusalueita.

Lyhimmän polun laskenta ja kaluston reitinoptimointi ovat edellisten esimerkkien perusteella käytännönläheisiä tutkimusalueita. Kuilu käytännön ja akateemisen tutkimuksen välillä on kuitenkin suuri. Lisäksi akateemiset yhteisöt keskittyvät erikoistumaan tiettyihin alueisiin eikä käytäntöön soveltuvia lähestymistapoja ole tutkittu merkittävästi. Tosielämän vaatimuksien kasvaessa siirtyminen lähestymistapoihin, jotka soveltuvat myös käytäntöön on välttämätöntä. Merkittäviä vaatimuksia ovat esimerkiksi reaaliaikaisen informaation hallinta ja optimointiongelmien suuremmat suunnittelukokonaisuudet. Tämä tutkielma tutustuu lyhimmän polun laskentaan kaluston reitinoptimoinnin näkökulmasta.

Tutkielmassa luodaan katsaus lyhimmän polun ongelmaan staattisessa tieverkossa ja esitetään kaksi uutta menetelmää, jotka tehostavat lyhimmän polun laskentaa. Luvussa 2 käydään läpi aiheeseen liittyvä kirjallisuus ja tarvittavat esitiedot. Luvussa 3 esitellään tutkielman tieteellinen sisältö, joka muodostuu konferenssiartikkelista ja laitosraportista. Luvussa 4 esitetään jatkotutkimusaiheet artikkelin ja raportin pohjalta. Luvussa 5 muodostetaan yhteenveto.

2 Esitiedot ja kirjallisuus

Tässä luvussa käydään läpi tarvittavat esitiedot ja tieverkkoihin liittyvä lyhimmän polun laskennan kirjallisuus.

2.1 Tieverkko ja lyhimmän polun ongelma

Karttatietokannoissa tieverkko kuvataan tie-elementtivektoreiden joukkona, jossa elementtien päätepisteet ovat verkon solmuja, joita yhdistävät verkon kaaret. Esimerkki tämänkaltaisesta karttatietokannasta on Suomen Tiehallinnon ylläpitämä tie- ja katutietojärjestelmä Digiroad[1]. Tieverkon solmuilla on tieto siihen saapuvista ja lähtevistä kaarista. On olemassa algoritmeja, jotka vaativat lisätietoa esimerkiksi solmujen koordinaateista (ks. luku 2.3.2). Kaarilla on oltava ominaisuutena mitattava pituus, joka on tie-elementin päätepisteiden välinen etäisyys. Tieverkossa kaarien pituus on ei-negatiivinen. Muita yleisiä ominaisuuksia kaarille ovat nopeusrajoitus, sallittu maksimipaino, sallittu maksimikorkeus ja vastaavat tarvittavat ja halutut ominaisuudet. Kadut muodostuvat yleensä useasta tie-elementistä, jolloin tien geometrinen muoto on mallinnettavissa vektoreilla. Tieverkon risteykset ovat aina tie-elementtien päätepisteitä. Akateemisessa tutkimuksessa verkkojen kaarien ominaisuuksina ovat pääasiassa etäisyys ja aika.

Tieverkkoa kuvataan yhtenäisellä graafilla $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, mikä muodostuu solmuista $v \in \mathcal{V}$ ja kaarista $e = (v_i, v_j, l) \in \mathcal{E}$, missä $v_i, v_j \in \mathcal{V}$ ja $l \geq 0$. Lisäksi $|\mathcal{V}| = \hat{v}$ ja $|\mathcal{E}| = \hat{e}$. Yhtenäisessä verkossa $\exists \langle v_i, \ldots, v_j \rangle, \forall v_i, v_j \in \mathcal{V}$ eli jokaisesta solmusta v_i on olemassa polku jokaiseen verkon solmuun v_j. Polulla $\langle v_i, \ldots, v_j \rangle$ on pituus $P\langle v_i, \ldots, v_j \rangle$. Lyhimmän polun ongelmassa etsitään lyhin polku $\langle v_i, \ldots, v_j \rangle^*$ siten, että $P\langle v_i, \ldots, v_j \rangle^* \leq P\langle v_i, \ldots, v_j \rangle$.

2.2 Algoritmien perusteet

Lyhimmän polun algoritmit ovat rekursiivisia ja yleiseltä rakenteeltaan neliosaisia:

- tietorakenteiden alustus,

[1]Tiehallinto, Digiroad - Tietolajien kuvaus, versio 1.2

- tutkittavan solmun valinta,
- solmujen laajennus ja
- lopetusehdon tarkistus.

Tietorakenteet alustetaan kerran lyhimmän polun algoritmin alkaessa. Tutkittava solmu löydetään tietorakenteesta, jonka jälkeen solmuja laajennetaan halutuilla kriteereillä. Laajennetut solmut lisätään tietorakenteeseen, jonka jälkeen ne voidaan tutkia. Algoritmin toiminta päättyy, kun lopetusehto toteutuu. Algoritmien suurimmat erot ovat tietorakenteissa, joilla hallitaan laajennettuja solmuja, ja tavoissa valita tutkittava solmu.

2.3 Perinteiset algoritmit

Algoritmi on yksi yhteen (engl. *one-to-one*) -lyhimmän polun algoritmi, jos se löytää lyhimmän polun tai lyhimmän etäisyyden kohdepisteiden a ja b välille. Algoritmi on yksi moneen (engl. *one-to-many*, *single source*) -lyhimmän polun algoritmi, jos se löytää lyhimmät polut tai lyhimmät etäisyydet kohdepisteestä a kaikkiin muihin annettuihin kohdepisteisiin.

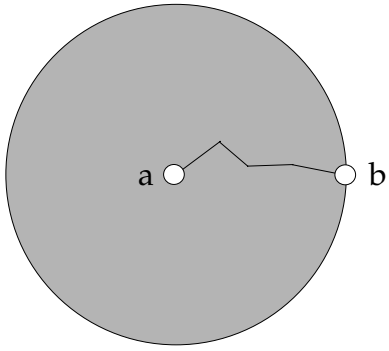
Kaksi yleisesti käytettyä lyhimmän polun algoritmia tieverkossa ovat Dijkstran algoritmi [9] ja A*-algoritmi [19].

2.3.1 Dijkstran algoritmi

Dijkstran algoritmi [9] on nimikkeitä asettava, yksi moneen -lyhimmän polun algoritmi, joka vaatii verkon kaarilta positiiviset pituudet. Nimikkeitä asettaville algoritmeille (engl. *Label-Setting*, LS) [1] on ominaista laajennettujen solmujen järjestäminen tietorakenteeseen siten, että niiden arvona on siihen mennessä kuljettu lyhin matka kyseiseen solmuun. Tutkittavaksi solmuksi valitaan se tietorakenteen solmu, jolla on pienin arvo. Tämän jälkeen tutkittava solmu poistetaan tietorakenteesta. Tutkittavan solmun naapurisolmut laajennetaan ja lisätään tietorakenteeseen. Jos laajennettava solmu on jo tietorakenteessa, solmun arvo päivitetään. Algoritmi päättää toimintansa, kun lopetusehto toteutuu. Lopetusehtona voi olla halutun kohdepisteen löytyminen tai kohdepistejoukon kaikkien solmujen löytyminen.

Tietorakenteeksi voidaan valita järjestetty lista, jossa järjestyksen ylläpito on pahimmassa tapauksessa $O(\hat{v}^2)$ [9], binäärikeko $O(\hat{e} \log \hat{v})$ [42], hienostunut kokonaislukuprioriteettijono $O(\hat{e} + \hat{v} \log \log \hat{v})$ [39] tai fibonaccikeko $O(\hat{v} \log \hat{v} + \hat{e})$ [12].

Verkon kaarien painojen l ollessa $0 \leq l \leq C$ on esitetty säiliötietorakenne (engl. *buckets*) $O(\hat{e} + \hat{v}C)$ [8], jota on parannettu $O(\hat{e} + \hat{v} \log \log C)$ [39] ja $O(\hat{e} \log \log C)$ [11]. Lineaarisessa ajassa toimivia tietorakenteita yksi moneen -ongelmille on esitetty suuntaamattomille verkoille [38].



Kuva 2.1: Dijkstran algoritmin eteneminen.

Kuvassa 2.1 esitetään Dijkstran algoritmin eteneminen. Tummennettu alue kuvaa laajennettuja solmuja.

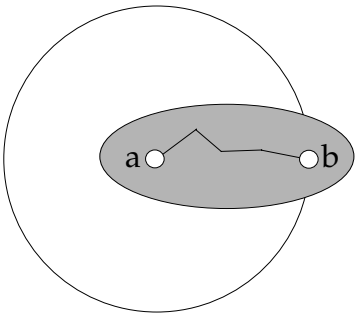
2.3.2 A*-algoritmi

A*-algoritmi [19, 34] on heuristinen yksi yhteen -algoritmi, joka vaatii verkon kaarilta positiivisen pituuden. Algoritmi toimii Dijkstran algoritmin tavoin, mutta tutkittavien solmujen valinta tapahtuu heuristisen funktion avulla. Heuristinen funktio määrää solmulle v arvon

$$f(v) = g(v) + h(v), \tag{2.1}$$

missä $g(v)$ on lyhin kuljettu etäisyys solmuun v ja $h(v)$ arvioitu lyhin etäisyys etsittävään kohdepisteeseen. Algoritmi löytää lyhimmän polun, jos estimoiva funktio $h(v)$ ei yliarvioi loppumatkaa. Estimoiva funktio muodostaa loppumatkasta arvion, joka voi olla esimerkiksi todellinen loppumatka tai euklidinen etäisyys. Todellisen loppumatkan pituutta ei tiedetä ja euklidisen etäisyyden laskeminen vaatii solmujen koordinaattitiedot. Laajennettujen solmujen tietorakenteeksi voidaan valita luvussa 2.3.1 esitettyjä tietorakenteita.

Kuvassa 2.2 havainnollistetaan A*-algoritmin etenemistä ja solmujen laajentamista. Kuvia 2.1 ja 2.2 verrattaessa huomataan merkittävä ero laajennettujen solmujen määrässä Dijkstran algoritmin ja A*-algoritmin välillä. A*-algoritmi on Dijkstran algoritmia nopeampi yksi yhteen -lyhimmän polun laskennassa, mutta ei tue yksi moneen -lyhimmän polun laskentaa kuten Dijkstran algoritmi.



Kuva 2.2: A*-algoritmin eteneminen.

2.4 Perinteiset kiihdytystekniikat

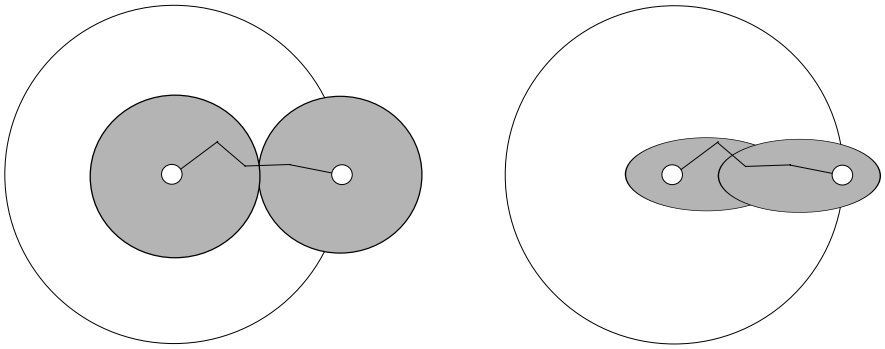
Lyhimmän polun laskentaa voidaan tehostaa perinteisillä kiihdytystekniikoilla. Perinteisiä kiihdytystekniikoita ovat menetelmät, jotka eivät vaadi esiprosessointia. Esiprosessointi on operaatio, joka tehdään ennen kohdepisteiden antamista. Perinteisiä kiihdytystekniikoita ovat kaksisuuntainen haku ja optimaalisten välimaalien käyttö.

2.4.1 Kaksisuuntainen haku

Perinteiset lyhimmän polun algoritmit ovat yksisuuntaisia, eli haku aloitetaan kohdepisteestä a ja lopetetaan kohdepisteeseen b. Kaksisuuntainen haku [33] on tekniikka, jossa etsitään lyhin polku samanaikaisesti kohdepisteestä a kohdepisteeseen b ja kohdepisteestä b kohdepisteeseen a. Edellisiä hakuja kutsutaan eteneviksi tai takeneviksi hakuoperaatioiksi. Dijkstran algoritmin (ks. luku 2.3.1) kaksisuuntaisen haun lopetusehtona on saman solmun tutkiminen molemmissa hakuoperaatioissa. Jos löydetty yhteinen solmu on v, niin etenevä haku on löytänyt lyhimmän polun $\langle a, \ldots, v \rangle^*$ ja takeneva haku lyhimmän polun $\langle b, \ldots, v \rangle^*$. Nyt lyhin polku $\langle a, \ldots, b \rangle^* = \langle a, \ldots, v \rangle^* \cup \langle v, \ldots, b \rangle^*$.

Kaksisuuntaisen A*-algoritmin [22] lopetusehto on monimutkaisempi eikä sitä kuvata tässä tutkielmassa.

Kuvassa 2.3 havainnollistetaan kaksisuuntaisten hakujen eroa solmujen laajentamisen suhteen. Kuvan vasemmalla puolella on Dijkstran kaksisuuntainen haku ja oikealla puolella A*-algoritmin kaksisuuntainen haku.

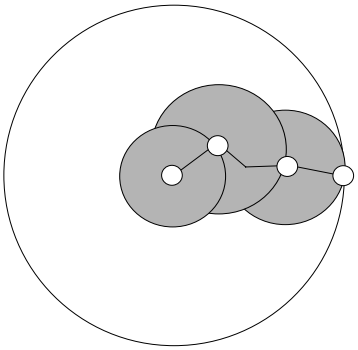


Kuva 2.3: Kaksisuuntaisten hakujen eteneminen.

2.4.2 Optimaaliset välimaalit

Lyhimmän polun laskeminen kohdepisteiden a ja b välille voidaan jakaa useaksi lyhimmän polun ongelmaksi, jos tiedetään, että solmut $v_1, \ldots, v_n$ kuuluvat esitetyssä järjestyksessä lyhimpään polkuun $\langle a, \ldots, b \rangle^*$. Tällöin riittää ratkaista lyhimmät polut solmupareille $(a, v_1), (v_1, v_2), \ldots, (v_{n-1}, v_n), (v_n, b)$, jonka jälkeen haluttu alkuperäinen lyhin polku voidaan kasata yhdistämällä lasketut lyhimmän polut esitetyssä järjestyksessä. Lähestymistapa esitettiin ensin yhdelle välimaalille [5] ja myöhemmin usealle välimaalille [10]. Välimaalien käytön tehokkuus perustuu siihen, että etsittävät reitit ovat lyhyempiä kuin alkuperäinen reitti, jolloin laajennettavien solmujen määrä on pienempi ja haku tehokkaampi.

Menetelmä on vähän käytetty, sillä nopeita menetelmiä välimaalien tunnistamiseksi ei ole kehitetty. Jos välimaaliksi valitaan solmu, joka ei ole lyhimmässä polussa, ei haluttua lyhintä polkua löydetä. Kuvassa 2.4 havainnollistetaan laajennettuja solmuja, kun käytetään optimaalisia välimaaleja.



Kuva 2.4: Optimaalisten välimaalien vaikutus laajennettujen solmujen määrään.

2.5 Esiprosessoitavat kiihdytystekniikat

Jos tutkittavaa verkkoa käsitellään merkittävästi ennen lyhimmän polun laskentaa ja kohdepisteiden määrittämistä, lyhimmän polun algoritmien toimintaa on mahdollista nopeuttaa merkittävästi. Näitä tekniikoita kutsutaan esiprossessoiviksi tai moderneiksi kiihdytystekniikoiksi.

2.5.1 Maamerkit

ALT-algoritmi (*A* search, Landmarks, and the Triangle inequality*) [14] pyrkii parantamaan luvussa 2.3.2 esitetyn A*-algoritmin heuristisen funktion $h(n)$ arviota. Heuristista funktiota parannetaan määrittämällä joukko maamerkkejä $m \in \mathcal{M}$, joita käytetään kolmioepäyhtälön avulla. Maamerkkien käyttö edellyttää solmujen $v \in \mathcal{V}$ etäisyyksien tuntemista kaikkiin maamerkkeihin. Nyt $m_i \in \mathcal{M}, i = 1, \ldots, n$ ja kohdepisteille a ja b on kolmioepäyhtälön mukaan voimassa

$$\begin{aligned} d^1(a,b) &:= d(m_1,b) - d(m_1,a) \leq d(a,b) \\ &\ldots \\ d^n(a,b) &:= d(m_n,b) - d(m_n,a) \leq d(a,b), \end{aligned} \tag{2.2}$$

jolloin estimoiva funktio algoritmille A* on

$$h(n) := max(d^1(a,b), \ldots, d^n(a,b)). \tag{2.3}$$

Menetelmän heikkoutena on tilavaatimus kaikille etäisyyksille maamerkkien ja muiden solmujen välillä.

2.5.2 Geometriset säiliöt

Yleisesti tiedetään, että jokaiselle kaarelle e on olemassa joukko $G(e)$, joka sisältää kaikki ne solmut, jotka voidaan saavuttaa lyhimmällä polulla solmusta v valitsemalla ensimmäiseksi kaareksi e. Joukkoa $G(e)$ kutsutaan kaaren e geometriseksi säiliöksi (engl. *geometric container*) [41]. Nyt on huomioitava, että solmulla v on kaaret $e_1, \ldots, e_n$ ja $G(e_1) \cap \cdots \cap G(e_n) = \emptyset$.

Lyhimmän polun laskennassa geometrisia säiliöitä voidaan käyttää solmujen tehokkaassa laajentamisessa. Tutkittaessa solmu ja sen naapureihin johtavien kaarien geometriset säiliöt voidaan karsia ne kaaret, joiden geometrisissa säiliöissä ei ole kohdepistettä. Geometristen säiliöiden esiprosessointi vaatii kaikkien lyhimpien polkujen laskemisen kaikille solmuille, mikä on laskennallisesti työlästä.

2.5.3 Kaarien merkitseminen

Tieverkon jakaminen alueiksi on intuitiivinen tapa suunnitella reitti paikasta, kuten esimerkiksi kunnasta, toiseen. Kun verkko jaetaan useaan eri alueeseen (engl. *region*), voidaan jokaiselle kaarelle antaa aluekohtainen lippu, jos on olemassa kaaresta alkava lyhin polku, joka kulkee alueen jonkin solmun kautta. Menetelmää kutsutaan kaarien merkitsemiseksi (engl. *Arc Flag*[2], AF) [26, 28, 29].

Nyt verkossa on k aluetta ja jokaiselle kaarelle e ja alueelle r on olemassa lippu, jos kaari e sijaitsee jollain lyhimmällä polulla, joka kulkee alueen r jonkin solmun v kautta.

Olkoon $R_1, \ldots, R_k$ verkon alueita siten, että $v \in \mathcal{V}$ kuuluu vain ja ainoastaan yhteen alueeseen. Nyt jokaiselle kaarelle $e \in \mathcal{E}$ on olemassa vektori f_e siten, että $|f_e| = k$. Nyt $f_{e_i} = 1, i = 1, \ldots, k$, jos on olemassa lyhin polku, joka kulkee kaaren e kautta solmuun $v \in R_i$. Etsittäessä lyhin polku solmuun $b \in R_i$, huomioidaan lyhimmän polun algoritmissa ainoastaan ne kaaret e, joille $f_{e_i} = 1$.

Menetelmän heikkoutena on aikaa vaativa esiprosessointi, jolloin kaaritiedot lasketaan lyhimmän polun menetelmillä. Esiprosessointia on myöhemmin tehostettu niin, että Länsi-Euroopan tieverkko on onnistuttu esiprosessoimaan 17 tunnissa hakuaikojen ollessa tuhansia kertoja lyhyemmät kuin Dijkstran algoritmilla [20].

2.5.4 Ulottuvuusmetriikka

Solmun v ulottuvuutta (engl. *reach*) merkitään $R(v) := max_{s,t \in \mathcal{V}} R_{st}(v)$, missä

$$R_{st}(v) := min(d(s, v), d(v, t)). \tag{2.4}$$

On osoitettu [17], että jos solmun ulottuvuus on tarpeeksi pieni, se voidaan karsia, sillä kyseisen solmun kautta ei voi kulkea haluttu lyhin polku. Menetelmä on myöhemmin yhdistetty luvussa 2.5.1 esitettyyn ALT-algoritmiin [15, 16].

2.5.5 Moottoritiehierarkiat

Moottoritiehierarkiat (engl. *Highway Hierarchies*, HH) [35, 36] ovat lyhimmän polun tarkkuuden säilyttävä kiihdytystekniikka. Menetelmä perustuu lähestymistapaan, jossa kohdepisteiden ympäristöissä, paikallisilla alueilla, huomioidaan kaikki solmut ja ei-paikallisilla alueilla eli moottoritieverkossa ainoastaan ”tärkeät tiet”. Paikallisen alueen solmulle $v \in \mathcal{V}$ määrittelee H lähintä solmua. Tämän jälkeen kaari $e = (v_k, v_l, l) \in \mathcal{E}$ on osa moottoritieverkkoa, jos ei ole olemassa solmuja $v_i, v_j \in \mathcal{V}$

[2]Tunnetaan myös nimellä edge flag.

siten, että kaari e kuuluu lyhimpään polkuun solmusta v_i solmuun v_j, v_l ei ole H:n lähimmän solmun joukossa solmusta v_i eikä v_k ole H:n lähimmän solmun joukossa solmusta v_j. Nyt moottoritieverkosta voidaan karsia eristyneet solmut ja puut, ja korvata polut, jotka sisältävät vain kaksiasteisia solmuja yhdellä kaarella. Esitettyä rakennusprosessia voidaan toistaa iteratiivisesti.

Menetelmää pidetään ensimmäisenä kiihdytystekniikkana, joka kykenee käsittelemään suuria tieverkkoja siten, että lyhimmän polun hakuajat olivat millisekunteja.

Moottoritiehierarkiat ratkaisevat 10 000 $\times$ 10 000 etäisyysmatriisin noin minuutissa [25]. Moottoritiehierarkioiden yhdistämistä ALT-algoritmiin [6] (ks. luku 2.5.1) on myös tutkittu, mutta sitä ei pidetä yhtä houkuttelevana yhdistelmänä kuin esimerkiksi geometrisia säiliöitä (ks. luku 2.5.2) tai siirtosolmujen käyttöä (ks. luku 2.5.6).

2.5.6 Siirtosolmujen käyttö

Siirtosolmujen käyttö (engl. *transit nodes*, TN) [2] perustuu huomioon, että on olemassa pieni joukko siirtosolmuja, jolloin kohdepisteiden a ja b ollessa tarpeeksi kaukana toisistaan, lyhin polku $\langle a, \ldots, b \rangle^*$ kulkee vähintään yhden siirtosolmun kautta. Lisäksi jokaiselle solmulle on olemassa pieni joukko yhteyssolmuja (engl. *access nodes*), jotka kohdataan ennen siirtosolmuja, kun etsitään tarpeeksi pitkää lyhintä polkua. Kun jokaisen solmun etäisyys yhteyssolmuihin ja jokaisen yhteyssolmun etäisyys siirtosolmuihin on laskettu, voidaan tarpeeksi pitkät etäisyydet laskea muutamalla nopealla suorahakuoperaatiolla.

2.5.7 Monitasomenetelmä

Monitasomenetelmässä (engl. *Multi-Level*) [21] verkolle $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on olemassa verkkoja

$$\begin{aligned} \mathcal{G}_1^1 &= (\mathcal{V}_1^1, \mathcal{E}_1^1) \\ &\ldots \\ \mathcal{G}_1^n &= (\mathcal{V}_1^n, \mathcal{E}_1^n), \end{aligned} \tag{2.5}$$

joille $\mathcal{V}_1^1 \subseteq \mathcal{V}, \ldots, \mathcal{V}_1^n \subseteq \mathcal{V}$ siten, että lyhimmät polut verkoissa $\mathcal{G}_1^1, \ldots, \mathcal{G}_1^n$ löytyvät myös verkosta $\mathcal{G}$. Lisäksi $\mathcal{G}_1^1 \cap \cdots \cap \mathcal{G}_1^n = \emptyset$. Nyt verkko $\mathcal{G}$ on taso 0 ja verkot $\mathcal{G}_1^1, \ldots, \mathcal{G}_1^n$ ovat taso 1. Tasoja voidaan rakentaa iteratiivisesti lisää.

Hakualgoritmi on kaksivaiheinen, jossa ensin valitaan operoitava aliverkko ja toiseksi suoritetaan haku valitussa aliverkossa. Tämän jälkeen valitaan jälleen ali-

verkko, jossa operoidaan ja menetelmää toistetaan, kunnes on löydetty haluttu kohdepiste.

2.5.8 Solmujen supistaminen

Solmujen supistaminen (engl. *Contraction Hierarchy*, CH) [13] on menetelmä, jossa solmut järjestetään ”tärkeyden” mukaan, jonka jälkeen hierarkia rakennetaan iteratiivisesti supistamalla pois vähemmän tärkeät solmut. Solmun v supistaminen tarkoittaa sitä, että solmun v kautta kulkevista lyhimmistä poluista tehdään oikopolut.

Hakualgoritmina on kaksisuuntainen haku, jossa etenevä haku tutkii ainoastaan ne kaaret, jotka johtavat ”tärkeämpiin” solmuihin ja takeneva haku tutkii ainoastaan kaaret, jotka tulevat ”tärkeämmistä” solmuista.

Menetelmä kykenee ratkaisemaan 10 000 $\times$ 10 000 etäisyysmatriisin noin 10 sekunnissa ja tilavaatimus verkolle on negatiivinen, eli syötteenä oleva verkko on suurempi kuin esiprosessoinnin jälkeinen verkko.

2.5.9 Yhdistelmät

Edellä esitettyjä menetelmiä on myöhemmin yhdistetty toistensa kanssa. REAL-algoritmissa[3] [14, 15] yhdistetään luvun 2.5.4 ulottuvuusmetriikka ja luvun 2.5.1 ALT-algoritmi. SHARC-algoritmissa[4] [3] yhdistetään moottoritiehierarkiat ja kaarien merkitseminen. Lisäksi on esitetty CALT-algoritmia[5] [4], CHASE-algoritmia[6] [4] ja TNR+AF-algoritmia[7] [4].

Nopeimmat algoritmit [4] tällä hetkellä ovat CH, TNR+AF ja CHASE. TNR+AF-algoritmilla on kaikista nopeimmat hakuoperaatiot, mutta esiprosessointi on työläs. CH-menetelmällä esiprosessointi on nopea ja rakennetun verkon koko on pienempi kuin alkuperäisen verkon, mutta hakuoperaatiot ovat noin 80 kertaa hitaammat kuin TRN+AF-algoritmilla. CHASE-algoritmi on kompromissi näiden kahden väliltä. Esiprosessointi on työläämpi kuin CH-algoritmilla, mutta nopeampi kuin TRN+AF-menetelmällä. CHASE-algoritmin hakuoperaatiot ovat noin 12 kertaa hitaammat kuin TRN+AF-menetelmällä.

[3] REach-based routing with Landmark-based A* search
[4] SHortcuts and ARC-flag
[5] Core-based routing and ALT
[6] Constraction Hierarchy, Arc-flagS and highwaynodE routing
[7] Transit Node Routing and Arc-Flags

2.6 Väliprosessoitavat kiihdytystekniikat

Väliprosessoitavat kiihdytystekniikat ovat menetelmiä, jotka suoritetaan lyhimmän polun laskennan nopeuttamiseksi, kun tutkittavat kohdepisteet tunnetaan. Väliprosessoitavia menetelmiä ei ole juurikaan tutkittu ja ainoa kirjallisuudesta löytyvä menetelmä on verkon karsinta, kun kohdepisteet on annettu.

Verkon karsinta [24] perustuu niiden solmujen ja kaarien karsintaan, joita ei tarvitse tutkia, kun kohdepisteet on annettu. Verkosta etsitään etukäteen 1- ja 2-asteiset solmut. Nyt voidaan karsia kaikki 1-asteiset solmut, jotka eivät ole kohdepisteitä. 2-asteiset solmut poistetaan verkosta siten, että solmun naapurit yhdistetään toisiinsa. Testitapauksessa menetelmällä poistettiin 56% solmuista, solmujen määrän ollessa 3 miljoonaa.

3 Artikkelien esittely

Pro gradu -tutkielman tieteellinen osuus jakaantuu konferenssiartikkeliin ja laitosraporttiin. Konferenssiartikkelissa **[PI]** esitetään menetelmä, jossa tieverkosta karsitaan iteratiivisesti 1- ja 2-asteiset solmut, jolloin laajennettavien solmujen lukumäärä vähenee ja lyhimmän polun laskenta tehostuu. Kuvattu solmujen karsinta ei vaikuta optimaalisen lyhimmän polun löytymiseen. Artikkelin kuva 1 on päivitetty tähän tutkielmaan. Laitosraportissa **[PII]** esitetään menetelmä, jossa lyhimmän polun laskennan nopeutta tehostetaan laskemalla osa lyhimmistä poluista perinteisillä algoritmeilla ja osa suorahakuoperaatioilla. Suorahakuoperaatioissa käytetään älykkäästi tunnettuja etäisyyksiä. Molemmat menetelmät ovat ns. väliprosessointimenetelmiä, jotka suoritetaan kun kohdepistejoukko on annettu.

Artikkelissa **[PI]** esitetty menetelmä on kehitetty täysin erillään verkon karsinnasta [24]. Esitetty menetelmä toimii iteratiivisesti toisin kuin verkon karsinta. Verkon karsinnan ja esitetyn menetelmän vuoksi solmujen aste pienenee, kun solmun naapuri karsitaan. Kun 3-asteiselta solmulta karsitaan yksi naapurisolmu, 4-asteiselta solmulta kaksi naapurisolmua tai 5-asteiselta solmulta 3 naapurisolmua muuttuu tutkittava solmu 2-asteiseksi. Verkon karsinta [24] ei huomioi näitä tilanteita, koska 1- ja 2-asteiset solmut määritetään ennen verkon karsintaa. Iteratiivisessa verkon karsinnassa nämä huomioidaan, mikä mahdollistaa sellaisten solmujen karsimisen, jotka ovat alkuperäisessä verkossa yli 2-asteisia, mutta ovat tutkittavalla hetkellä 1- tai 2-asteisia.

Menetelmän suorituskyvyn mittausta varten rakennetut testitapaukset luotiin aidosta tieverkosta satunnaisella asiakasdatalla. Keski-Suomen tieverkko saatiin Digiroad-kartta-aineistosta. Aineisto sisältää 96 020 solmua ja 102 161 kaarta. Kohdepisteiden lukumäärät olivat 50, 100 ja 200. Testitapauksissa verrattiin karsimattoman ja karsitun verkon vaikutusta lyhimmän polun laskennan tehokkuuteen. Jokaisessa testitapauksessa havaittiin parannus lyhimmän polun laskentaan. Testitapaukset ratkesivat verkon karsinnan jälkeen 8,1–18,4 kertaa nopeammin algoritmista, testitapauksen koosta ja satunnaisotoksesta riippuen. Lisätyön kuluttama aika oli jokaisessa testitapauksessa alle 1 millisekunti, joka on vähän verrattuna modernien kiihdytystekniikoiden vaatimiin esiprosessointiaikoihin.

Raportissa **[PII]** esitetään koostamistekniikka, joka tehostaa lyhimmän polun laskentaa. Menetelmän idea perustuu optimaalisten välimaalien käyttöön lokaalissa

ympäristössä. Optimaalisten välimaalien avulla lyhimmän polun laskenta voidaan tehdä useassa eri osassa ja osa alireiteistä tarvitsee laskea vain kerran, jolloin laskenta tehostuu. Esitetty idea on uusi eikä vastaavaa tutkimusta ole aiemmin tehty.

Menetelmän testaamiseksi luotiin satunnaisia ja tiheitä testitapauksia Keski-Suomen tieverkkoon. Testitapauksia generoitiin 12 ja ne muodostuivat 64, 128, 256 tai 512 kohdepisteestä, jotka olivat 2, 4 tai 8 klusterissa. Dijkstran yksi yhteen -menetelmällä testitapauksien ratkaisemiseen kului aikaa 12–1300 minuuttia, iteratiivisella verkon karsinnalla ja Dijkstran yksi yhteen -menetelmällä 1–150 minuuttia ja koostamistekniikalla 10–540 sekuntia.

Artikkelin ja raportin tuloksista huomataan, että väliprosessointimenetelmät nopeuttavat lyhimmän polun laskentaa lyhyellä prosessointiajalla. Väliprosessointimenetelmien vaatima prosessointi on merkittävästi kevyempi kuin esiprosessointimenetelmien prosessointi. Väliprosessointimenetelmien kevyet prosessoinnit perustuvat annettujen kohdepisteiden topologiseen analysointiin. Tätä ei voida tehdä esiprosessointivaiheessa, koska kohdepisteitä ei tunneta.

4 Jatkotutkimus

Artikkelissa **[PI]** esitettiin tieverkon karsintamenetelmä lyhimmän polun laskennan tehostamiseksi. Menetelmä perustui iteratiiviseen karsintaan 1- ja 2-asteisten solmujen kohdalla. Verkon karsinnan todettiin nopeuttavan lyhimmän polun laskentaa, mutta ei niin merkittävästi kuin esiprosessoivat kiihdytystekniikat.

Tehokkaampia karsimistekniikoita, jotka eivät perustu ainoastaan solmujen asteeseen, on tutkittava. Tehokkaampi topologiatiedon käyttö osana verkon karsintaa voi mahdollistaa suurempien aliverkkojen karsimista, mikä nopeuttaisi lyhimmän polun laskentaa.

Raportissa **[PII]** esitettiin menetelmä lyhimmän polun laskennan tehostamiseksi. Menetelmä perustui koostettujen kohdepisteklustereiden käyttöön. Klusterit on muodostettu lyhimmän polun laskennan tehostamiseksi, mutta niitä voidaan käyttää osana kaluston reitinoptimointia ilman luontikustannuksia.

Nykyiset lyhimmän polun laskennan menetelmät keskittyvät nopeaan etäisyysmatriisin rakentamiseen ja sen hallintaan. Tutkielmassa esitetyt menetelmät eivät kykene yhtä nopeaan etäisyysmatriisin rakentamiseen kuin esiprosessoitavat kiihdytystekniikat. Koostamistekniikka ja esiprosessoitavat kiihdytystekniikat eroavat toisistaan myös käyttötarkoitukseltaan. Koostamistekniikkaa voidaan pitää merkittävänä avauksena uudelle tutkimukselle, joka pyrkii keräämään, tuottamaan ja analysoimaan lisäinformaatiota lyhimmän polun laskennan aikana tieverkossa tapahtuvalle logistiselle optimoinnille. Lisäinformaatio tarkoittaa informaatiota, mikä tulee etäisyysmatriisin lisäksi. Uusi tutkimus pyrkii vastaamaan seuraaviin kysymyksiin:

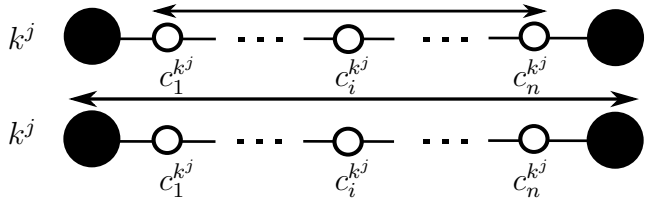
- Millaista lisäinformaatiota on olemassa?
- Voiko lisäinformaation kerätä ja analysoida tehokkaasti?
- Kuinka lisäinformaatiota voidaan käyttää tehokkaasti hyödyksi optimoinnissa?

Koostamistekniikka muodostaa tieverkon logistisille optimointiongelmille topologiatietoon pohjautuvia klustereita. Jos kerätty lisäinformaatio käytetään lyhimmän polun laskennan tehostamiseen, on lisäinformaation kerääminen optimointiongelman näkökulmasta ilmainen operaatio. Jos lisäinformaatiota ei käytetä ly-

himmän polun laskennan tehostamiseen, on lisäinformaation keräämiselle mahdollista laskea kustannus. Koostamistekniikassa käytetään lisäinformaatiota lyhimmän polun laskennan tehostamiseksi, jolloin lisäinformaation kerääminen optimointiongelman näkökulmasta on ilmainen operaatio. Kyseisen lisäinformaation käyttöä osana tieverkon logistista optimointia ei ole tutkittu.

Kaluston reitinoptimoinnin näkökulmasta luotuja klustereita voidaan käyttää alkuratkaisujen luomisessa tai algoritmien siirtäessä suuria kokonaisuuksia esimerkiksi asiakasjoukkoja reitiltä toiselle. Esitetyn klusteroinnin käyttöä rajoitteita sisältävässä kaluston reitinoptimoinnissa olisi tutkittava.

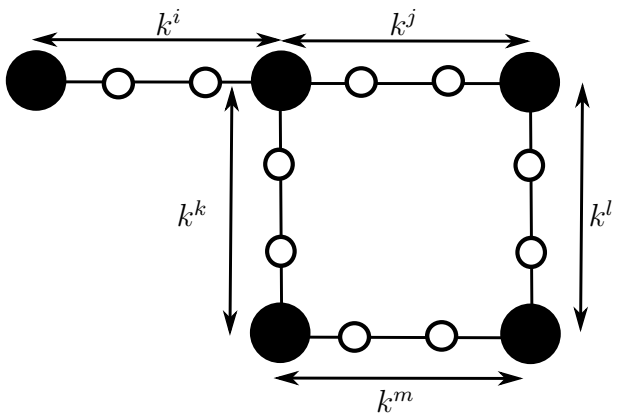
Kun koostaminen tehdään topologian pohjalta, klusterit voivat olla topologisesti yhteydessä toisiinsa. Artikkelissa esitetyssä menetelmässä klusterin muodostavat vain kohdepisteet. Jos tätä lähestymistapaa laajennetaan sisältämään myös lähimmät tieverkkosolmut, voi tulla tilanteita, jolloin kahdella eri klusterilla on sama solmu. Kuvassa 4.1 esitetään koostetun klusterin alkuperäinen ja laajennettu rakenne. Jos kahdella eri klusterilla on sama solmu, voidaan sanoa, että näiden klustereiden välillä on topologinen linkki. Tämän vaikutusta lyhimmän polun laskentaan on tutkittava, sillä tällöin voi olla mahdollista korvata useampia perinteisiä lyhimmän polun operaatioita suorahakuoperaatioilla.



Kuva 4.1: Kohdepisteiden koostaminen.

Topologisilla linkeillä on mahdollista muodostaa suurempia topologisia klustereita. Tästä on hyötyä myös kaluston reitinoptimoinnille. Kuvassa 4.2 on esitetty tilanne, jossa klusterit ovat yhdistetty toisiinsa topologisten linkkien avulla. Klustereita yhdistäessä topologisten linkkien avulla voi tulla vastaan tilanne, jolloin yhdistetyt klusterit ovat niin isoja, että ne ovat käyttökelvottomia. Tämän vuoksi on tutkittava menetelmiä, joilla koostetut klusterit voidaan pilkkoa pienemmiksi klustereiksi mahdollisimman hyvin topologiatietoa hyväksikäyttäen ja kaluston reitinoptimointia ajatellen.

Ehdotettuja menetelmiä lyhimmän polun laskennan tehostamiseksi on verrattava muihin kehitettyihin menetelmiin. Menetelmien vertaamiseen on luotava uusia mittareita, sillä nykyiset mittarit lyhimmän polun laskennassa mittaavat esiprosessoinnin ja hakuoperaatioiden kuluttamaa aikaa. Lisäinformaation arvoa kaluston



Kuva 4.2: Topologisten linkkien avulla muodostettu klusteri.

reitinoptimoinnille eli esimerkiksi klustereiden luomista lyhimmän polun laskennan aikana, ei voida kyseisillä mittareilla mitata.

5 Yhteenveto

Tässä tutkielmassa esitettiin kaksi menetelmää, jotka eivät vaadi esiprosessointia ja tehostavat lyhimmän polun laskentaa. Alan aiemmin tehty kirjallisuus käytiin läpi ja havaittiin, että vastaavaa tutkimusta ei ole juurikaan tehty. Sen sijaan esiprosessointia vaativat lyhimmän polun laskennan tekniikat ovat paljon tutkittuja.

Tutkielman tieteellinen sisältö muodostui konferenssiartikkelista **[PI]** ja laitosraportista **[PII]**. Artikkelissa ja raportissa esitettiin kaksi menetelmää, joiden todettiin tehostavan lyhimmän polun laskentaa.

Tuloksien ja johtopäätöksien avulla huomattiin selkeä tarve lisätutkimukselle. Lisätutkimuksen kohteena ovat esiteltyjen menetelmien jatkokehittäminen ja yhdistäminen kaluston reitinoptimointiin.

6 Lähteet

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993, sivut 93–157.

[2] H. Bast, S. Funke, P. Sanders, D. Schultes, *Fast Routing in Road Networks with Transit Nodes*, Science, vol. 316, 2007, sivu 566.

[3] R. Bauer, D. Delling, *SHARC: Fast and Robust Unidirectional Routing*, In Workshop on Algorithm Engineering and Experiments, 2008, sivut 13–26.

[4] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, D. Wagner, *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm*, Lecture Notes in Computer Science, 2008, sivut 303–318.

[5] P.P. Chakrabarti, S. Ghose, S.C. Desarkar, *Heuristic Search through Islands*, Artificial Intelligence, vol. 29, 1986, sivut 339–348.

[6] D. Delling, P. Sanders, D. Schultes, D. Wagner, *Highway Hierarchies Star*, 9th DIMACS Implementation Challenge, 2006.

[7] C. Demetrescu, G. F. Italiano, *Experimental Analysis of Dynamic All Pairs Shortest Path Algorithms*, ACM Transactions on Algorithms, vol. 2, 2006, sivut 578–601.

[8] R.B. Dial, *Algorithm 360: Shortest-Path Forest with Topological Ordering*, Communications of the ACM, vol. 12, 1969, sivut 632–633.

[9] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, vol. 1, 1959, sivut 269–271.

[10] J.F. Dillenburg, P.C. Nelson, *Improving Search Efficiency using Possible Subgoals*, Mathematical and Computer Modelling, vol. 22, 1995, sivut 397–412.

[11] P. van Emde Boas, R. Kaas, E. Ziljstra, *Design and Implementation of an Efficient Priority Queue*, Theory of Computing Systems, vol. 10, 1976, sivut 99–127.

[12] M.L. Fredman, R.E. Tarjan, *Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms*, Journal of the ACM, vol. 22, 1987, sivut 596–615.

[13] R. Geisberger, P. Sanders, D. Schultes, D. Delling, *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*, Lecture Notes in Computer Science, vol. 5038, 2008, sivut 319–333.

[14] A.V. Goldberg, C. Harrelson, *Computing the Shortest Path: A Search Meets Graph Theory*, 16th annual ACM-SIAM symposium on Discrete Algorithms, 2005, sivut 156–165.

[15] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Reach for A*: Efficient Point-To-Point Shortest Path Algorithms*, Microsoft Technical Report: MSR-TR-2005-132, 2005.

[16] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Better Landmarks within Reach*, In 6th on Experimental Algorithms, vol. 4525, 2007, sivut 38–51.

[17] R. Gutman, *Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks*, 6th Workshop on Algorithm Engineering and Experiments, 2004.

[18] A. Hallamäki, *Jätehuollon optimoinnista*, Pro gradu -tutkielma, Jyväskylän yliopisto, 2008.

[19] P.E. Hart, N.J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transaction of Systems Science and Cybernetics, vol. 2, 1968, sivut 100–107.

[20] M. Hilger, *Accelerating Point-To-Point Shortest Path Computations in Large Scale Networks*, Diploma Thesis, Technische Universität Berlin, 2007.

[21] M. Holzer, F. Schulz, D. Wagner, *Engineering Multi-Level Overlay Graphs for Shortest-Path Queries*, In Workshop on Algorithm Engineering and Experiments, 2006, sivut 156–170.

[22] T. Ikeda, M. Hsu, H. Imai, S. Nishimura, H. Shimoura, T. Hashimoto, K. Tenmoku, K. Mitoh, *A Fast Algorithm for Finding Better Routes by AI Search Techniques*, Vehicle Navigation and Information Systems Conference IEEE, 1994, sivut 291–296.

[23] D.S. Johnson, *Local Optimization and the Traveling Saleman Problem*, Lecture Notes in Computer Science, vol. 443, sivut 446–461.

[24] G.A. Klunder, H.N. Post, *The Shortest Path Problem on Large-Scale Real-Road Networks*, Networks, vol. 48, 2006, sivut 182–194.

[25] S. Knopp, P. Sanders, D. Schultes, F. Schulz, D. Wagner, *Computing Many-To-Many Shortest Paths using Highway Hierarchies*, Workshop on Algorithm Engineering and Experiments, 2007.

[26] E. Köhler, R.H. Möhring, H. Schilling, *Acceleration of Shortest Path and Constrained Shortest Path Computation*, Experimental and Efficient Algorithms, vol. 3503, 2005, sivut 126–139.

[27] G. Laporte, *The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms*, European Journal of Operational Research, vol. 59, 1992, sivut 345–358.

[28] U. Lauther, *Slow Preprocessing of Graphs for Extremely Fast Shortest Path Calculations*, Lecture at the Workshop on Computational Integer Programming at ZIB, 1997.

[29] U. Lauther, *An Experimental Evaluation of Point-To-Point Shortest Path Calculation on Roadnetworks with Precalculated Edge-Flags*, In 9th DIMACS Implementation Challenge, 2006.

[30] F. Lawler, A. Rinnooy-Kan, *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, 1985.

[31] S. Lin, B.W. Kernighan, *An Effectice Heuristics Algorithm for the Traveling-Salesman Problem*, Operations Research, vol. 21, 1973, sivut 498–516.

[32] P. Nakari, O. Bräysy, W. Dullaer, *Communal Transportation: Challenges for Large-Scale Routing Heuristics*, Reports of the Department of Mathematical Information Technology, Series B6/2007, Scientific Computing, University of Jyväskylä, ISBN 978-951-39-2823-0.

[33] T.A.J. Nicholson, *Finding the Shortest Route between Two Points in a Network*, The Computer Journal, vol. 9, 1966, sivut 275–280.

[34] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, 1984.

[35] P. Sanders, D. Schultes, *Highway Hierarchies Hasten Exact Shortest Path Queries*, In 13th European Symposium on Algorithms, vol. 3669, 2005, sivut 568–579.

[36] D. Schultes, *Fast and Exact Shortest Path Queries using Highway Hierarchies*, Master's thesis, Universität Karlsruhe, 2005.

[37] E. Taillard, *Parallel Iterative Search Methods for Vehicle Routing Problems*, Networks, vol. 23, 1993, sivut 661–673.

[38] M. Thorup, *Undirected Single Source Shortest Paths in Linear Time*, Journal of the ACM, vol. 46, 1999, sivut 362–394.

[39] M. Thorup, *Integer Priority Queues with Decrease Key in Constant Time and the Single Source Shortest Paths Problem*, Journal of Computer and System Sciences, vol. 69, 2004, sivut 330–353.

[40] P. Toth, D. Vigo, *The Vehicle Routing Problem*, Siam, 1987.

[41] D. Wagner, T. Willhalm, *Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Grahps*, European Symposium on Algorithms, vol. 2832, 2003, sivut 776–787.

[42] J.W.J. Williams, *Heapsort*, Communications of the ACM, vol. 7, 1964, sivut 347–348.

EVOLUTIONARY METHODS FOR DESIGN, OPTIMIZATION AND CONTROL
P. Neittaanmäki, J. Périaux and T. Tuovinen (Eds.)

TOPOLOGY-BASED OPTIMAL ROAD NETWORK REDUCTION — A METHOD FOR SPEEDING UP SHORTEST PATH COMPUTATION

Tuukka P. Puranen

Faculty of Information Technology
Agora Innoroad Laboratory
University of Jyväskylä
Mattilanniemi 2, 40100 Jyväskylä, Finland
Email: tuukka.puranen@jyu.fi
Web page: http://www.jyu.fi/erillis/agora/en

Joni J. Brigatti

Faculty of Information Technology
Agora Innoroad Laboratory
University of Jyväskylä
Mattilanniemi 2, 40100 Jyväskylä, Finland
Email: joni.brigatti@jyu.fi
web page: http://www.jyu.fi/erillis/agora/en

Abstract. In this paper, we present a new approach for speeding up the shortest path computation in road networks requiring only linear processing time. The main idea is to reduce, once the set of target nodes is known, the number of road edges and nodes that need to be considered. The suggested method conducts an analysis of nodes in the graph based on its topological characteristics. We present computational results in real road network using Dijkstra and A* shortest path algorithms and demonstrate that significant speed up factors are possible. We will argue that combining our approach with other speed-up techniques should be straightforward.

Keywords: Shortest Path, Road Network, Graph Reduction, Midprocessing, Speed-up Technique

1 INTRODUCTION

Shortest Path Problem (SPP) is an integral part of problems in road network applications, such as vehicle routing. In the Vehicle Routing Problem (VRP) a Many-to-Many Shortest Path Problem (MTMSSP) needs to be solved for a defined subset of the network. Solving MTMSPP exactly, i.e., calculating all distances between target nodes, is a laborious task and yields extensive result sets. There exists a set of general algorithms for SPP, but leveraging knowledge of problem type may be used to enhance the execution. Some approaches also take advantage of *preprocessing*. However, processing of the road network usually takes place before the set of target nodes is known. We present an algorithm which reduces the number of nodes that need to be considered in shortest path computation after the set of target nodes is given.

We review the related work in Section 2, present our algorithm in Section 3,

and analyze the results in Section 4. Directions of future research are presented in Section 5, and conclusions are given in Section 6.

2 RELATED WORK

The shortest path computation has been under active study for decades. The basic method for solving shortest paths in graphs is presented by Dijkstra[i]. Goal-directed A*-search is presented by Hart et al.[ii] Conventional speed-up techniques, like bidirectional search (see for example[iii]) and optimal subgoals[iv, v], do not need preprocessing. However, bidirectional search has a low speed-up factor and optimal subgoals are hard to implement in applications.

Modern speed-up techniques have much higher speed-up factors but often require preprocessing. Highway Hierarchies (HH) developed, and later improved, by Sanders and Schultes[vi, vii], offers extremely quick queries with acceptable preprocessing time. Reach-based routing by Gutman[viii], geometric containers by Wagner et al.[ix], ALT[1]-algorithm by Goldberg et al.[x], multi-level methods by Holzer et al.[xi] and Delling et al.[xi], and arc-flag approach by Lauther[xiii], offer useful techniques for speeding up the shortest path computation, but are not able to compete individually with HH. ALT-algorithm and reach-based routing has later been combined by Goldberg et al.[xiv, xv], and HH and ALT-algorithm by Delling et al.[xvi] Many-to-Many computation is presented by Knopp et al.[xvii] and transit node approach by Bast et al.[xviii]

Despite the vast number of existing methods, new sophisticated techniques to solve optimal MTMSSP can be developed. For example, Group-to-Group methods, i.e., computing shortest paths between sets of target nodes instead of individual targets, have not been widely studied. Some modern speed-up techniques require a modified shortest path algorithm. Our method does not require a specific algorithm, which eases the process of combining it with other methods.

3 ROAD NETWORK REDUCTION

As mentioned, preprocessing usually takes place before the set of target nodes is known. Complementary to this approach, we propose a method that reduces the road graph as soon as the target nodes are known. During this *midprocessing* phase, we reduce the number of nodes by starting from nodes of degree 1, and continue by removing nodes iteratively by traversing the graph until encountering a node we need to visit, i.e., a target, or one we cannot, judging by its degree and neighbors, safely remove. In addition, we can safely remove all nodes of degree 2 given that there are no targets in adjoining edges or in the node[2]. The visualization of algorithm is presented in Figure 1 and exact execution of the method in Algorithm 1.

A road graph G is defined as a set of vertices V, i.e., nodes; and a set of edges E, i.e., roads; and is accompanied by a set of targets T, that is, nodes we need to find shortest paths to. C is a set of nodes that have not yet been processed. In line 3, we start from an arbitrary node and process it based on its degree $d_G(v)$ and number of neighbors $|N_G(v)|$. We can *Remove* a node if it is not in the set of nodes we need to visit, and is that of degree one, or degree two with only a single

[1] A search, landmarks, and triangle inequality

[2] The orders can effectively be in the nodes, the edges, or both. In the formalization in this paper, the set of targets is defined as a subset of vertices.

Algorithm 1 *Reduction(G, T)*

Require: $G = (V, E)$
Require: $T \subseteq V$
1: $C \leftarrow V$
2: **while** $C \neq \emptyset$ **do**
3: $C \leftarrow C \setminus \{v\}$, $v \in C$
4: **if** $d_G(v) = 0$ **then**
5: $V \leftarrow V \setminus \{v\}$
6: **continue**
7: **else if** $v \in T$ **then**
8: **continue**
9: **else if** $d_G(v) > 2$ **then**
10: **continue**
11: **end if**
12: **while** $d_G(v) < 3$ **do**
13: **if** $d_G(v) = 1$ **then**
14: $v \leftarrow$ *Remove(G, v)*
15: **if** $v \in T$ **then**
16: **break**
17: **end if**
18: $C \leftarrow C \setminus \{v\}$
19: **else if** $d_G(v) = 2$ **then**
20: **if** $|N_G(v)| = 1$ **then**
21: $v \leftarrow$ *Remove(G, v)*
22: **if** $v \in T$ **then**
23: **break**
24: **end if**
25: $C \leftarrow C \setminus \{v\}$
26: **else**
27: *Merge(G, v)*
28: **break**
29: **end if**
30: **else**
31: **break**
32: **end if**
33: **end while**
34: **end while**

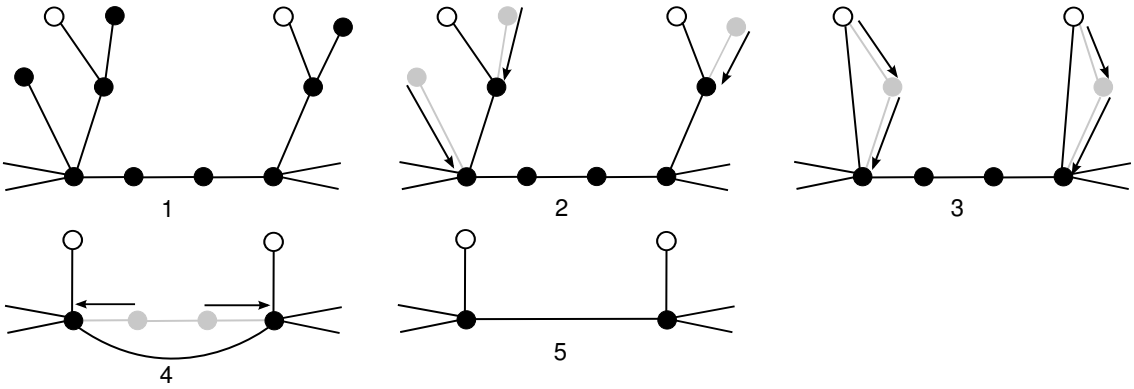


Figure 1: Step-by-step illustration of the execution of the algorithm in a distinct subgraph.

distinct neighbor. *Remove* operation removes the node v from the graph and returns its neighbor. Nodes with degree two and two distinct neighbors can, likewise, be removed using *Merge*. The two neighbors of v are connected with an edge that is, essentially, a sum of the edges of v.

Note that in *Merge* operation, summing the edges does not cause information loss in the constraints[3], due to the fact that there are no targets in the removed node, and thus if the road is to be used, it is traveled from end to end in the optimal route. If the road is not traveled from end to end, the edges can be left untouched where constraints occur. At this stage, however, these constraints are not included in the graph structure. Note, from Section 4, that our test data does include nodes of degree two, which are essentially from the raw data representation of the road network. It is assumed here, for the sake of the computational experiment, that the number of these nodes, which do not alter the topology, is roughly equivalent to the number of constraints in the actual road structure. With our method, the graph can be arbitrarily accurate[4] and still be returned to optimal graph for shortest path computation.

During the graph reduction, each node shall be checked at most $d_G(v) - 1$ times and at least once if $d_G(v) \geq 2$. If $d_G(v) = 1$, the node shall be checked exactly once. Thus, the algorithm has linear time complexity in the number of nodes in the graph.

4 Computational Results

The graph reduction and shortest path algorithms were implemented in C♯. We performed the experiments in Windows XP, Intel Core 2 Duo T7300 processor clocked at 2.00 GHz and with 2,00 GB (777 MHz) of main memory. The commercial data used was a section of Central-Finland, containing 96 020 nodes and 102 161 roads. The data was converted from ESRI shapefiles to data structures, i.e., any preprocessing has not been done. Cases were generated randomly, and it was ensured that the graph representing the road network was connected. A total of 15 test cases, of three different sizes (50, 100, and 200 targets), were solved. The midprocessing phase was run 10 000 times. In the tests, whole distance matrix was

[3]For example speed limits; when the maximum speed of every vehicle is higher than the possible speed limits.

[4]The more accurate the graph is, the more edges are needed to represent a road.

computed with *one-to-one* searches. The results of graph reduction and average speed-up factors are presented in Table 1. Speed-up is defined as a ratio between the CPU time for shortest path computation in original graph, and the CPU time in midprocessed graph plus time required by the processing.

As we can see in results, the graph reduction is extremely fast (1 ms) in current graph, and its execution time does not depend on the set of target nodes. Speed-up factors for A* (11,9-18,4) are in general higher than for Dijkstra's algorithm (8,1-10,6). The reason for speed-up is the huge number of removed nodes. As can be seen in Table 1, over half of the graph comprises of nodes of degree 1 or 2, which are largely removed by the process.

Original data			
	node deg.	# in orig.	%
	1	24605	25,625%
	2	36434	37,944%
	3	33098	34,470%
	4	1860	1,937%
	5	23	0,024%
50 customers			
	node deg.	avg. # in reduced	%
	1	26	0,342%
	2	21	0,270%
	3	7016	92,949%
	4	477	6,320%
	5	9	0,119%
	avg. # scan. in orig.	avg. # scan. in reduced	avg. speed-up factor
A*	34 911 629	3 174 045	18,4
Dijkstra	236 671 796	27 934 280	10,6
Processing time			1 ms
100 customers			
	node deg.	avg. # in reduced	%
	1	50	0,648%
	2	42	0,541%
	3	7044	92,405%
	4	480	6,289%
	5	9	0,118%
	avg. # scan. in orig.	avg. # scan. in reduced	avg. speed-up factor
A*	145 836 285	13 142 869	11,9
Dijkstra	989 888 810	95 779 424	8,1
Processing time			1 ms
200 customers			
	node deg.	avg. # in reduced	%
	1	102	1,310%
	2	81	1,032%
	3	7115	91,391%
	4	478	6,142%
	5	9	0,116%
	avg. # scan. in orig.	avg. # scan. in reduced	avg. speed-up factor
A*	596 989 340	54 598 010	16,8
Dijkstra	4 056 656 143	2 429 590 136	8,9
Processing time			1 ms

Table 1: Results from random generated cases with three different problem sizes.

5 Future Research

The proposed method is under active study, and future research consists of implementing constraints, both in form of road and vehicle. In order to evaluate the applicability of our method in real life cases, it is also necessary to measure its performance in different road networks, and to combine it with other speed-up techniques. The combination of our method, which reduces only the search space, and methods, which do not require a special type of network structure, should be straightforward.

When relaxing the optimality requirement, heuristic versions of the reduction can be developed to further narrow the search space. Another major area of research is the processing of target nodes, i.e., optimal clustering and aggregation. The reduction method can be modified to aggregate nodes instead of reducing them. This can be done without losing the optimality of the shortest path computation.

6 Conclusion

For example in the Vehicle Routing Problem, a Many-to-Many Shortest Path Problem needs to be solved for a subset of the road network. A number of shortest path algorithms have been used to solve the problem, and several techniques have been proposed to speed-up the search, including a number of preprocessing techniques. Complementary to preprocessing, we proposed a midprocessing method that reduces the road network graph as soon as the target nodes are known. The reduction is based on road topology, and heavily leans on the topological characteristics of the network in question. With this method the accuracy of road network does not affect the shortest path computation time. Computational results indicate that, in certain conditions, a major speed-up is possible. In addition, the CPU time required by our method is insignificant compared to the CPU time required to solve the shortest path problem.

7 Acknowledgements

This paper was supported by TEKES (Finnish Funding Agency for Technology and Innovation) via OPT-LOG project. This support is acknowledged with great gratitude. We would like to thank adjunct professor Olli Bräysy (Agora Innoroad Laboratory) for invaluable feedback and comments during the process, and Antti Hallamäki and Pekka Hotokka for ideas and suggestions.

REFERENCES

[1] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, 1959, pages 269-271.

[2] P.E. Hart, N.J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minumun Cost Paths*, Systems Science and Cybernetics, IEEE Transactions on, Vol. 4, 1968, pages 100-107.

[3] T.A.J. Nicholson, *Finding the Shortest Route between Two Points in a Network*, The Computer Journal, Vol. 9, 1966, pages 275-280.

[4] P.P. Chakrabarti, S. Ghose, S.C. Desarkar, *Heuristic Search through Islands*, Artificial Intelligence, Vol. 29, 1986, pages 339-348.

[5] J.F. Dillenburg, P.C. Nelson, *Improving Search Efficiency Using Possible Subgoals*, Mathematical and Computer Modelling, vol. 22, 1995, pages 397-414.

[6] P. Sanders, D. Schultes, *Highway Hierarchies Hasten Exact Shortest Path Queries*, Lecture Notes in Computer Science, Vol. 3669, 2005, pages 568-579.

[7] P. Sanders. D. Schultes, *Engineering Highway Hierarchies*, Lecture Notes in Computer Science, Vol. 4168, 2006, pages 804-816.

[8] R. Gutman, *Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks*, Proceedings in 6th International Workshop on Algorithm Engineering, 2004.

[9] D. Wagner, T. Willhalm, *Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs*, Lecture Notes in Computer Science, Vol. 2832, 2003, pages 776-787.

[10] A.V. Goldberg, C. Harrelson, *Computing the Shortest Path: A Search meets Graph Theory*, Proceedings of the 16th annual ACM-SIAM symposium on Discrete algorithms, 2005, pages 156-165.

[11] M. Holzer, F. Schulz, D. Wagner, *Engineering Multi-Level Overlay Graphs for Shortest-Path Queries*, Workshop on Algorithm Engineering and Experiments, 2006.

[12] D. Delling, M. Holzer, K. Müller, F. Schulz, D. Wagner, *High-Performance Multi-Level Graphs*, 9th DIMACS Challenge on Shortest Paths, 2006.

[13] U. Lauther, *Slow Preprocessing of Graphs for Extremely Fast Shortest Path Calculations*, Lecture at the Workshop on Computational Integer Programming at ZIB, 1997.

[14] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Better Landmarks within Reach*, Lecture Notes in Computer Science, Vol. 4525, 2007, pages 38-51.

[15] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Reach for A*: Efficient Point-to-Point Shortest Path Algorithms*, Workshop on Algorithm Engineering & Experiments, 2006.

[16] D. Delling, P. Sanders, D. Schultes, D. Wagner, *Highway Hierarchies Star*, 9th DIMACS Implementation Challenge, 2006.

[17] S. Knopp, P. Sanders, D. Schultes, F. Schulz, D. Wagner, *Computing Many-to-Many Shortest Paths using Highway Hierarchies*, Workshop on Algorithm Engineering and Experiments, 2007.

[18] H. Bast, S. Funke, P. Sanders, D. Schultes, *Fast Routing in Road Networks with Transit Nodes*, Science, Vol. 316, 2007, page 566.

Reports of the Department of Mathematical Information Technology
Series B. Scientific Computing
No. B. 4/2009

Kohdepisteiden topologinen koostaminen lyhimmän polun ongelmassa

Joni Brigatti

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
FI–40014 University of Jyväskylä
FINLAND
fax +358 14 260 2731
http://www.mit.jyu.fi/

ISBN 978-951-39-3598-6
ISSN 1456-436X

Kohdepisteiden topologinen koostaminen lyhimmän polun ongelmassa

Joni Brigatti
joni.brigatti@jyu.fi

9.6.2009

Tiivistelmä

Lyhimmän polun etsintä on klassinen ongelma tietojenkäsittelytieteissä. Tässä raportissa esitetään menetelmä, jolla voidaan vähentä monesta moneen -lyhimmän polun laskennan operaatioiden määrää koostamalla kohdepisteitä klustereiksi. Operaatiot korvataan suorahakuoperaatioilla menettämättä kuitenkaan lyhimmän polun laskennan tarkkuutta. Menetelmän osoitetaan toimivan hyvin tiheissä testitapauksissa.

1 Johdanto

Tässä raportissa tarkastellaan lyhimmän polun etsimistä tieverkossa. Lyhimmän polun ongelma (engl. *Shortest Path Problem*, SPP) [1] on klassinen ongelma, jossa on tehtävänä selvittää lyhin polku tai etäisyys kohdepisteiden a ja b välillä annetussa verkossa. Monesta moneen -lyhimmän polun ongelmassa (engl. *Many-To-Many Shortest Path Problem*, MTMSPP) ratkaistaan lyhimmät polut kaikkien annettujen kohdepisteiden välillä. Kauppamatkustajan ongelma (engl. *Traveling Salesman Problem*, TSP) [21] ja kaluston reitinoptimointiongelma (engl. *Vehicle Routing Problem*, VRP) [26] ovat keskeisiä logistiikan optimointiongelmia ja niiden ratkaiseminen vaatii tiedot kaikkien pisteiden etäisyyksistä toisiinsa.

Tässä raportissa esitetään koostamistekniikka, jolla voidaan ratkaista monesta moneen -lyhimmän polun ongelma käsittelemällä osaa kohdesolmuista perinteisillä lyhimmän polun laskennan menetelmillä ja osaa

kohdesolmuista suorahakuoperaatioilla. Menetelmä tuo merkittäviä ajallisia säästöjä, kun rakennetaan etäisyysmatriiseja tiheille optimointiongelmille. Tiheitä ja suuria käytännön kuljetustenoptimointiongelmia ovat esimerkiksi jätteidenkeräys [17] ja sanomalehtien jakelu [16].

Raportti on jaettu 7 lukuun. Luvussa 2 käydään läpi aiheeseen liittyvä aiempi kirjallisuus. Luvussa 3 käydään läpi tarvittava termistö. Luvuissa 4 ja 5 esitetään koostamismenetelmä ja analysoidaan laskennalliset tulokset. Luvussa 6 esitetään jatkotutkimusaiheita ja luvussa 7 muodostetaan yhteenveto.

2 Kirjallisuus

Lyhimmän polun laskentaa on tutkittu tieteellisessä kirjallisuudessa vuosikymmeniä. Alan tunnetuin menetelmä, ja yksi käytetyimmistä, on Dijkstran algoritmi [7]. Myöhemmin on kehitetty A*-algoritmi [14], joka Dijkstran algoritmista poiketen hakeutuu haluttua pistettä kohden. A*-algoritmi tutkii verkkoa Dijkstran algoritmia vähemmän, mutta tehokkaammin ja on keskimäärin yksi yhteen -hakuoperaatioissa nopeampi. Kaksisuuntainen haku (katso esimerkiksi [22]) ja optimaaliset välimaalit [4, 8] ovat perinteisiä kiihdytystekniikoita, jotka nopeuttavat laskentaa, mutta joiden merkitys suurien käytännön ongelmien kohdalla on pieni.

Moderneilla kiihdytystekniikoilla voidaan ratkaista suuria käytännön ongelmia, koska menetelmien kiihdytyskertoimet ovat suuria. Kiihdytyskerroin määritellään perinteisen lyhimmän polun laskennan menetelmän käyttämän laskenta-ajan suhteesta kiihdytystekniikan tehostaman menetelmän käyttämään laskenta-aikaan.

Kiihdytystekniikoita ovat Highway Hierarchies [24, 25], Reach-based routing [13], geometriset säiliöt [27], ALT[1]-algoritmi [10], Transit-node routing [2], Multi-level- [5, 15] ja Arc-Flag-lähestymistapa [20].

Edellä esitettyjä menetelmiä on myöhemmin yhdistetty toisiinsa, kuten Reach-based routing ja ALT-algoritmi [11, 12], ja Highway Hierarchies ja ALT-algoritmi [6]. Lisäksi on esitetty CALT [3]-algoritmi, jossa yhdistetään Core-Based routing ja ALT-algoritmi, sekä CHASE [3]-algoritmi, jossa yhdistetään Contraction Hierarchy [9], Arc-flags ja Highway node routing.

Suurien etäisyysmatriisien laskemista on tutkittu [9, 19]. Contraction Hierarchy [9] rakentaa 10 000 $\times$ 10 000 etäisyysmatriisin 10 sekunnissa ja Highway Hierarchies [19] vastaavan ongelman noin minuutissa. Aikoihin ei ole huomioitu menetelmien vaatimaa esiprosessointia.

[1] *A* search, Landmarks, and Triangle inequality

Verkon karsimismenetelmiä ei ole juurikaan tutkittu. Ensimmäisenä verkon karsimista ehdotti Klunder [18], ja edellistä hieman paranneltuna ja erillään keksittynä Puranen [23]. Siinä missä Klunder etukäteen määrittelee 1- ja 2-asteiset solmut karsittavaksi, Puranen tutkii jokaisen solmun ja karsii sen, jos se on tutkittavalla hetkellä 1- tai 2-asteinen. Ero on oleellinen, sillä verkon karsinnan tuloksena solmujen aste saattaa muuttua. Tällöin voidaan karsia solmuja, joiden aste alkuperäisessä verkossa on ollut suurempi kuin 2.

Lyhimmän polun laskentaa ja reitin tarkkuuden säilyttäviä koostamistekniikoita ei ole tutkittu tieteellisessä kirjallisuudessa tapauksissa, joissa tutkittavana on tieverkko. Luvussa 4 esitetään reitin tarkkuuden säilyttävä koostamistekniikka, jolla on mahdollista nopeuttaa lyhimmän polun laskentaa.

3 Termistö

Symboli	Selitys
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Verkko
$\mathcal{V}$	Verkon solmut
$\mathcal{E}$	Verkon kaaret
$v \in \mathcal{V}$	Solmu
$e \in \mathcal{E}$	Kaari
$\mathcal{N}_{\mathcal{G}}(v)$	Solmun v naapureiden joukko
$d_{\mathcal{G}}(v) \in \mathbb{N}$	Solmun v aste
$\langle a, \ldots, b \rangle$	Polku solmusta a solmuun b
$\langle a, \ldots, c, \ldots, b \rangle$	Polku solmusta a solmuun b solmun c kautta
$P\langle a, \ldots, b \rangle$	Polun pituus solmusta a solmuun b
$\langle a, \ldots, b \rangle^*$	Lyhin polku solmusta a solmuun b
$d(a, b) \in \mathbb{R}$	Lyhimmän polun pituus solmusta a solmuun b
$\mathcal{K}$	Klusterit
$\langle c_1^{k^j}, \ldots, c_i^{k^j} \rangle = k^j \in \mathcal{K}$	Klusteri k^j
$\mathcal{C}$	Kohdepisteet
$c \in \mathcal{C}$	Kohdepiste
$\mathcal{M}$	Tunnettujen etäisyyksien joukko

Taulukko 1: Käytetyt symbolit ja niiden selitykset.

Taulukossa 1 esitetään tässä raportissa käytetyt symbolit. Yhtenäinen ja suuntaamaton verkko $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ muodostuu solmuista $v \in \mathcal{V}$ ja kaarista

$e = (v_i, v_j, l) \in \mathcal{E}$, missä $v_i, v_j \in \mathcal{V}$, kaaren pituus $l \in \mathbb{R}, l > 0$. Solmun v naapurijoukko $\mathcal{N}_\mathcal{G}(v)$ sisältää kaikki ne solmut $a_1, \ldots, a_n \in \mathcal{V}$, joilla on kaari $e = (v, a_i, l_i) \in \mathcal{E}, i = 1, \ldots, n$. Solmun v aste $d_\mathcal{G}(v)$ määräytyy kaarien $e = (v, a_i, l_i) \cup (v, v, l_j) \in \mathcal{E}, i = 1, \ldots, n, j = 1, \ldots, m$ lukumäärästä, missä $\{a_1, \ldots, a_n\} = \mathcal{N}_\mathcal{G}(v)$.

Polkua solmusta $a \in \mathcal{V}$ solmuun $b \in \mathcal{V}$ merkitään $\langle a, \ldots, b\rangle$. Lisäksi polkua solmusta $a \in \mathcal{V}$ solmuun $b \in \mathcal{V}$ solmun $c \in \mathcal{V}$ kautta merkitään $\langle a, \ldots, c, \ldots, b\rangle$. Merkitään solmun c liittämistä polkuun $\langle a, \ldots, b\rangle$ seuraavasti: $\langle a, \ldots, b\rangle \cup \{c\} \Rightarrow \langle a, \ldots, b, c\rangle$.

$P\langle a, \ldots, b\rangle$ on polun $\langle a, \ldots, b\rangle$ sisältämien kaarien pituuksien summa. Polku $\langle a, \ldots, b\rangle^*$ on lyhin polku, jos ja vain jos $P\langle a, \ldots, b\rangle^* \leq P\langle a, \ldots, b\rangle$. Etäisyys $d(a, b)$ on lyhimmän polun pituus $P\langle a, \ldots, b\rangle^*, \langle a, \ldots, b\rangle^* \subset \mathcal{V}$. Olkoon lisäksi $\mathcal{M}$ tunnettujen etäisyyksien joukko.

$\langle c_1^{k^j}, \ldots, c_i^{k^j}, \ldots, c_n^{k^j}\rangle$ on klusteri $k^j \in \mathcal{K}$ eli polku kohdepisteitä $c_i^{k^j} \in \mathcal{C}$.

4 Koostamistekniikka

Koostamistekniikkaa voidaan käyttää lyhimmän polun laskennan tehostamiseen. Lähestymistavan tarkoitus on vähentää alireittien $\langle u, \ldots, v\rangle^*$ laskennan lukumäärää tapauksissa, joissa useat eri reitit, kuten esimerkiksi $\langle a, \ldots, u, \ldots, v, \ldots, b\rangle^*$ ja $\langle c, \ldots, u, \ldots, v, \ldots, d\rangle^*$, sisältävät saman alireitin. Nyt esitettyjen kahden eri reitin etsiminen voidaan uudelleenmuotoilla ongelmaksi, jossa lasketaan 5 eri reittiä $\langle a, u\rangle^*$, $\langle v, b\rangle^*$, $\langle c, u\rangle^*$, $\langle v, d\rangle^*$ ja $\langle u, v\rangle^*$. Reitit kootaan tämän jälkeen kahdeksi alkuperäiseksi reitiksi. Nyt alipolku $\langle u, v\rangle^*$ lasketaan vain kerran, vaikka se on osa molempia reittejä.

Lähestymistapa voidaan jakaa kahteen eri vaiheeseen: kohdepisteiden koostamiseen ja etäisyysmatriisin rakentamiseen. Kohdepisteiden koostamisessa muodostetaan kohdepisteklusterit, jotta reitit voidaan jakaa alireiteiksi ja etäisyysmatriisin rakentamisessa lasketaan alireitit, jotka kasataan alkuperäisiksi reiteiksi. Alireitit lasketaan joko suorahakuoperaatioilla tai perinteisillä lyhimmän polun laskennan menetelmillä. Suorahakuoperaatioita ovat tunnettujen etäisyyksien käyttö. Tunnettuja etäisyyksiä ovat jo lasketut etäisyydet ja kahden naapurisolmun välinen etäisyys.

4.1 Kohdepisteiden koostaminen

Koostamistekniikassa keskitytään kohdepisteisiin $c \in \mathcal{C} \subseteq \mathcal{V}$, jotka sijaitsevat verkossa $\mathcal{G}$ toistensa naapureina. Oletetaan, että $d_\mathcal{G}(c) = 2, \forall c \in \mathcal{C}$. Menetelmä luo klusterit $\langle c_1^{k^1}, \ldots, c_n^{k^1}\rangle, \ldots, \langle c_1^{k^m}, \ldots, c_n^{k^m}\rangle = k^1, \ldots, k^m \in \mathcal{K}$ ehdoilla $|\mathcal{N}_\mathcal{G}(c_1^{k^j}) \cap \mathcal{C}| \leq 1, |\mathcal{N}_\mathcal{G}(c_n^{k^j}) \cap \mathcal{C}| \leq 1, j = 1, \ldots, m$. Nyt klusterissa

k^j on vain kohdepisteitä. Ehdot varmistavat, että klusterin ensimmäisen kohdepisteen $c_1^{k^j}$ ja viimeisen kohdepisteen $c_n^{k^j}$ naapurijoukot sisältävät korkeintaan yhden kohdepisteen.

Algoritmi 1 Koostettujen klustereiden rakentaminen

Require: Kohdepistejoukko $\mathcal{C}$, klusterijoukko $\mathcal{K}$

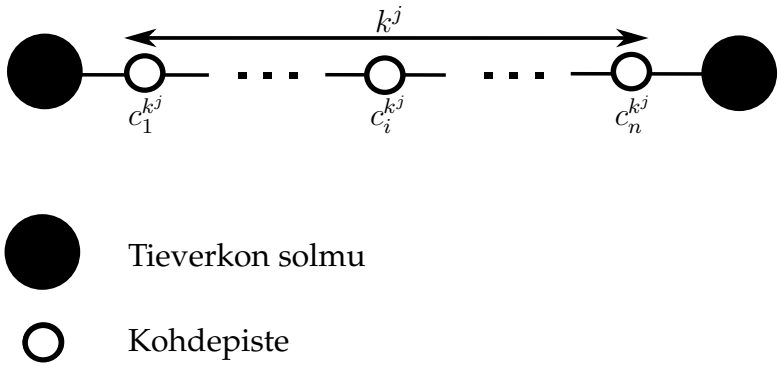
1: **while** $C \neq \emptyset$ **do**
2: $c_x \leftarrow \{c_x \in \mathcal{C} : |\mathcal{N}_\mathcal{G}(c_x) \cap \mathcal{C}| \leq 1\}$
3: $k^j \leftarrow k^j \cup \{c_x\}$
4: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c_x\}$
5: **while** $|\mathcal{N}_\mathcal{G}(c_x) \cap \mathcal{C}| = 1$ **do**
6: $c_x \leftarrow \{c \in \mathcal{N}_\mathcal{G}(c_x) \cap \mathcal{C}\}$
7: $k^j \leftarrow k^j \cup \{c_x\}$
8: $\mathcal{C} \leftarrow \mathcal{C} \setminus \{c_x\}$
9: **end while**
10: $\mathcal{K} \leftarrow \mathcal{K} \cup \{k^j\}$
11: **end while**

Menetelmästä esitetään toteutus algoritmissa 1. Rivillä 1 tarkistetaan kohdepistejoukon koko. Jos $\mathcal{C} = \emptyset$, algoritmi päättää toimintansa. Rivillä 2 valitaan kohdepiste $c_x \in \mathcal{C}$, jolla on naapurina korkeintaan yksi kohdepiste. Rivillä 3 kohdepiste c_x lisätään klusterin k^j ensimmäiseksi kohdepisteeksi. Rivillä 4 kohdepiste c_x poistetaan kohdepisteiden joukosta $\mathcal{C}$ eikä kohdepistettä c_x käsitellä tämän jälkeen koostamisvaiheessa. Jos kohdepisteen c_x naapurina ei ole kohdepistettä, klusteri k^j on valmis ja se lisätään klusterijoukkoon $\mathcal{K}$. Tämän jälkeen siirrytään riville 1. Jos kohdepisteen c_x naapurina on kohdepiste, siirrytään riville 6. Rivillä 6 valitaan uusi kohdepiste c_x. Rivillä 7 kohdepiste c_x lisätään klusteriin k^j ja rivillä 8 kohdepiste c_x poistetaan kohdepisteiden joukosta $\mathcal{C}$ ja kohdepistettä c_x ei enää käsitellä koostamisvaiheessa. Tämän jälkeen toistetaan rivejä 5–8 kunnes kohdepisteellä c_x ei ole naapuria joukossa $\mathcal{C}$. Näin on käyty läpi yhtenäinen polku kohdepisteitä ja ne muodostavat klusterin k^j. Tämän jälkeen siirrytään riville 10 ja lisätään klusteri k^j klustereiden $\mathcal{K}$ joukkoon. Rivejä 1–11 toistetaan kunnes $\mathcal{C} = \emptyset$.

Kuvassa 1 esitetään klusteri k^j, jolle $|k^j| = n$.

4.2 Etäisyysmatriisin rakentaminen

Tässä menetelmässä etäisyysmatriisi rakennetaan koostettujen klustereiden avulla. Etäisyysmatriisi on tietorakenne, joka sisältää kaikkien kohde-



Kuva 1: Koostettu klusteri.

pisteiden etäisyydet kaikkiin kohdepisteisiin. Rakentaminen tehdään neljässä vaiheessa. Ensimmäisessä vaiheessa lasketaan juurisolmujen väliset etäisyydet ja toisessa vaiheessa klusterin k^j juurisolmujen ja sisäsolmujen väliset etäisyydet sekä sisäsolmujen ja juurisolmujen väliset etäisyydet. Kolmannessa vaiheessa lasketaan klusterin k^j sisäsolmujen väliset etäisyydet ja neljännessä vaiheessa jäljelle jäävät etäisyydet eri klustereiden välille, jotka ovat klustereiden k^j ja k^i juurisolmujen ja sisäsolmujen väliset etäisyydet ja klustereiden k^j ja k^i sisäsolmujen väliset etäisyydet. Juurisolmuja ovat kohdepisteet $c_1^{k^j}, c_n^{k^j} \in k^j$, jos $|k^j| \geq 2$, muuten juurisolmu on $c_1^{k^j} \in k^j$. Sisäsolmuja ovat kohdepisteet $c_i^{k^j} \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$, jos $|k^j| \geq 3$.

4.2.1 Juurisolmujen väliset etäisyydet

Ensimmäisessä vaiheessa lasketaan kaikkien juurisolmujen väliset etäisyydet perinteisellä lyhimmän polun algoritmilla kuten Dijkstran algoritmilla. Tarvittavat etäisyydet on laskettu, kun kaikki juurisolmuparit on käyty läpi. Tämän jälkeen lyhimmän polun algoritmia ei käytetä vaan loput etäisyydet lasketaan suorahakuoperaatioilla.

Algoritmissa 2 lasketaan juurisolmujen väliset etäisyydet. Algoritmissa käydään läpi kaikki klusteriparit k^j ja k^i riviltä 1 lähtien. Riveillä 2–4 käsitellään tapaukset $|k^i| \geq 2$, jolloin klusterilla k^i on kaksi juurisolmua. Riveillä 3–4 lasketaan etäisyydet $d(c_1^{k^i}, c_n^{k^j}), d(c_n^{k^i}, c_1^{k^j})$. Jos $k^j = k^i$, niin tutkittavat klusterit ovat samat ja lasketut etäisyydet $d(c_1^{k^j}, c_n^{k^j}), d(c_n^{k^j}, c_1^{k^j})$. Riveillä 6–17 lasketaan lisää etäisyyksiä tilanteessa $k^j \neq k^i$. Riveillä 7–8 lasketaan etäisyydet $d(c_1^{k^j}, c_1^{k^i}), d(c_1^{k^i}, c_1^{k^j})$. Nämä etäisyydet lasketaan kaikille klustereille k^j ja k^i riippumatta niiden koosta. Riveillä 10–11 lasketaan etäisyydet $d(c_n^{k^j}, c_1^{k^i}), d(c_1^{k^i}, c_n^{k^j})$, jos $|k^j| \geq 2$. Riveillä 14–15 lasketaan etäisyydet $d(c_n^{k^j}, c_n^{k^i}), d(c_n^{k^i}, c_n^{k^j})$, jos $|k^j| \geq 2$ ja $|k^i| \geq 2$. Algoritmi päättää toi-

Algoritmi 2 Juurisolmujen etäisyyksien laskeminen

Require: Klusterijoukko $\mathcal{K}$, lyhimmän polun algoritmi A, tunnettujen etäisyyksien joukko $\mathcal{M}$

for all Klusteri $K_i, K_j \in \mathcal{K}$ **do**
  if $|k^i| \geq 2$ **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^j}, c_n^{k^i}, \text{A.RATKAISE}(c_1^{k^j}, c_n^{k^i}))\}$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^i}, c_1^{k^j}, \text{A.RATKAISE}(c_n^{k^i}, c_1^{k^j}))\}$
  end if
  if $i \neq j$ **then**
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^j}, c_1^{k^i}, \text{A.RATKAISE}(c_1^{k^j}, c_1^{k^i}))\}$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^i}, c_1^{k^j}, \text{A.RATKAISE}(c_1^{k^i}, c_1^{k^j}))\}$
    if $|k^j| \geq 2$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^j}, c_1^{k^i}, \text{A.RATKAISE}(c_n^{k^j}, c_1^{k^i}))\}$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^i}, c_n^{k^j}, \text{A.RATKAISE}(c_1^{k^i}, c_n^{k^j}))\}$
    end if
    if $|k^j| \geq 2$ ja $|k^i| \geq 2$ **then**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^j}, c_n^{k^i}, \text{A.RATKAISE}(c_n^{k^j}, c_n^{k^i}))\}$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^i}, c_n^{k^j}, \text{A.RATKAISE}(c_n^{k^i}, c_n^{k^j}))\}$
    end if
  end if
end for

mintansa, kun kaikki klusteriparit on käyty läpi.

4.2.2 Sisäsolmujen ja juurisolmujen väliset etäisyydet klusterin sisällä

Toisessa vaiheessa lasketaan klusterin k^j sisäsolmujen etäisyydet juurisolmuihin ja juurisolmujen etäisyydet sisäsolmuihin, kun $|k^j| > 2$. Määritellään kohdepisteet $a \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}, b \in \{c_1^{k^j}, c_n^{k^j}\}$ ja $c = \{c_1^{k^j}, c_n^{k^j}\} \setminus \{b\}$. Nyt $d(a,b) = min(d', d'')$, missä

$$d' = P\langle a, \ldots, b\rangle^* \tag{1}$$

ja

$$d'' = P\langle a, \ldots, c\rangle^* + P\langle c, \ldots, b\rangle^*. \tag{2}$$

Lisäksi $d(b,a) = min(d', d'')$, missä

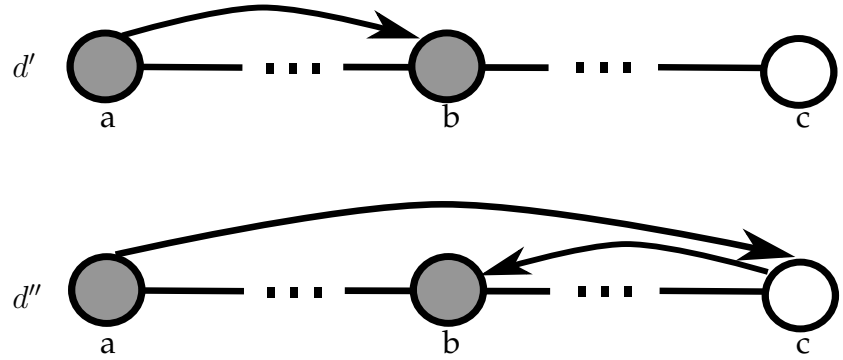
$$d' = P\langle b, \ldots, a\rangle^* \tag{3}$$

ja

$$d'' = P\langle b, \ldots, c\rangle^* + P\langle c, \ldots, a\rangle^*. \tag{4}$$

Nyt $\langle a, \ldots, b\rangle^*, \langle a, \ldots, c\rangle^*, \langle b, \ldots, a\rangle^*, \langle c, \ldots, a\rangle^* \subset k^j$. Lisäksi on voimassa $P\langle c, \ldots, b\rangle^*, P\langle b, \ldots, c\rangle^* \in \mathcal{M}$. Kuvassa 2 on vaihtoehdot potentiaalisille juurisolmun ja sisäsolmun välisille lyhimmille poluille. Sisäsolmujen ja

juurisolmujen väliset potentiaaliset lyhimmät polut muodostetaan vastaavasti.



Kuva 2: Juurisolmun ja sisäsolmun väliset potentiaaliset lyhimmät polut.

Algoritmi 3 Klusterin juurisolmujen ja sisäsolmujen väliset etäisyydet

Require: Klusterijoukko $\mathcal{K}$, tunnettujen etäisyyksien joukko $\mathcal{M}$

1: **for all** $k^j \in \mathcal{K} : |k^j| > 2$ **do**
2: $\quad \mathcal{C}^{k^j} \leftarrow k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$
3: $\quad$ **while** $\mathcal{C}^{k^j} \neq \emptyset$ **do**
4: $\quad\quad c_x \leftarrow c \in \mathcal{C}^{k^j}$
5: $\quad\quad \mathcal{C}^{k^j} \leftarrow \mathcal{C}^{k^j} \setminus \{c_x\}$
6: $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{(c_x, c_1^{k^j}, min(P\langle c_x, \ldots, c_1^{k^j}\rangle^* : \langle c_x, \ldots, c_1^{k^j}\rangle^* \subset k^j,$
$\qquad\qquad P\langle c_x, \ldots, c_n^{k^j}\rangle^* + d(c_n^{k^j}, c_1^{k^j}) : \langle c_x, \ldots, c_n^{k^j}\rangle^* \subset k^j))\}$
7: $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{(c_x, c_n^{k^j}, min(P\langle c_x, \ldots, c_n^{k^j}\rangle^* : \langle c_x, \ldots, c_n^{k^j}\rangle^* \subset k^j,$
$\qquad\qquad P\langle c_x, \ldots, c_1^{k^j}\rangle^* + d(c_1^{k^j}, c_n^{k^j}) : \langle c_x, \ldots, c_1^{k^j}\rangle^* \subset k^j))\}$
8: $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^j}, c_x, min(P\langle c_1^{k^j}, \ldots, c_x\rangle^* : \langle c_1^{k^j}, \ldots, c_x\rangle^* \subset k^j,$
$\qquad\qquad d(c_1^{k^j}, c_n^{k^j}) + P\langle c_n^{k^j}, \ldots, c_x\rangle^* : \langle c_n^{k^j}, \ldots, c_x\rangle^* \subset k^j))\}$
9: $\quad\quad \mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^j}, c_x, min(P\langle c_n^{k^j}, \ldots, c_x\rangle^* : \langle c_n^{k^j}, \ldots, c_x\rangle^* \subset k^j,$
$\qquad\qquad d(c_n^{k^j}, c_1^{k^j}) + P\langle c_1^{k^j}, \ldots, c_x\rangle^* : \langle c_1^{k^j}, \ldots, c_x\rangle^* \subset k^j))\}$
10: $\quad$ **end while**
11: **end for**

Algoritmissa 3 lasketaan klusterin k^j juurisolmujen etäisyydet sisäsolmuihin ja sisäsolmujen etäisyydet juurisolmuihin. Algoritmi käy läpi kaikki klusterit $|k^j| > 2$. Rivillä 2 muodostetaan joukko $\mathcal{C}^{k^j}$, joka sisältää klusterin k^j sisäsolmut. Jos $\mathcal{C}^{k^j} = \emptyset$, palataan riville 1, muuten siirrytään riville 4. Rivillä 4 käsitellään kohdepistettä $c_x \in \mathcal{C}^{k^j}$ ja rivillä 5 poistetaan kohdepiste c_x joukosta $\mathcal{C}^{k^j}$. Rivillä 6 muodostetaan potentiaaliset vaihtoehdot lyhimmälle polulle $\langle c_x, \ldots, c_1^{k^j}\rangle^* \subset \mathcal{V}$, valitaan lyhin etäisyys ja lisätään se tunnettujen etäisyyksien joukkoon $\mathcal{M}$. Rivillä 7 toimitaan kuten rivillä 6,

mutta etsittävän lyhimmän polun pituus on polun $\langle c_x, \ldots, c_n^{k^j} \rangle^* \subset \mathcal{V}$ pituus. Rivillä 8 etsittävä lyhin etäisyys on polun $\langle c_1^{k^j}, \ldots, c_x \rangle^* \subset \mathcal{V}$ pituus ja rivillä 9 polun $\langle c_n^{k^j}, \ldots, c_x \rangle^* \subset \mathcal{V}$ pituus. Tämän jälkeen siirrytään riville 3. Algoritmi lopettaa toimintansa, kun kaikkien klustereiden sisäsolmut on käyty läpi.

4.2.3 Sisäsolmujen väliset etäisyydet klusterin sisällä

Kolmannessa vaiheessa lasketaan klusterin k^j sisäsolmujen väliset etäisyydet, jos $|k^j| \geq 4$, jolloin $\exists\ a, b \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}, a \neq b$. Tällöin tiedetään, että $d(a, b) = min(d', d'', d''')$, missä

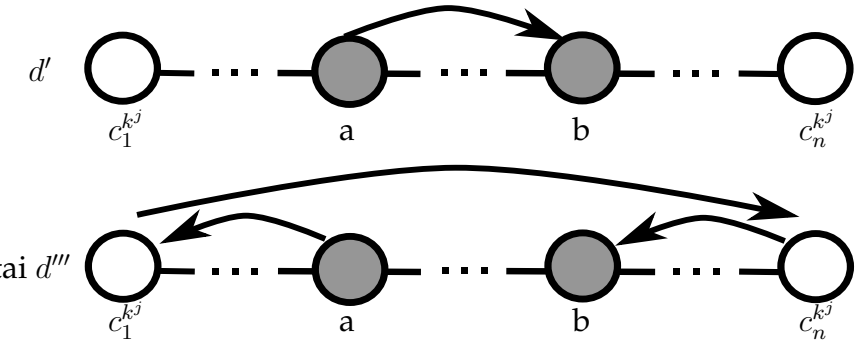
$$d' = P\langle a, \ldots, b \rangle^*, \tag{5}$$

$$d'' = P\langle a, \ldots, c_1^{k^j} \rangle^* + P\langle c_1^{k^j}, \ldots, c_n^{k^j} \rangle^* + P\langle c_n^{k^j}, \ldots, b \rangle^*, \tag{6}$$

ja

$$d''' = P\langle a, \ldots, c_n^{k^j} \rangle^* + P\langle c_n^{k^j}, \ldots, c_1^{k^j} \rangle^* + P\langle c_1^{k^j}, \ldots, b \rangle^*, \tag{7}$$

kun $\langle a, \ldots, b \rangle^* \subset k^j$. $P\langle a, \ldots, c_1^{k^j} \rangle^*, P\langle c_1^{k^j}, \ldots, c_n^{k^j} \rangle^*, P\langle c_n^{k^j}, \ldots, b \rangle^* \in \mathcal{M}$ ja $P\langle a, \ldots, c_n^{k^j} \rangle^*, P\langle c_n^{k^j}, \ldots, c_1^{k^j} \rangle^*, P\langle c_1^{k^j}, \ldots, b \rangle^* \in \mathcal{M}$. Kuvassa 3 on kuvattu vaihtoehdot potentiaalisille lyhimmille poluille kahden sisäsolmun välille. On huomioitava, että potentiaalisista lyhimmistä poluista d'' ja d''' tarvitaan vain se, joka ei sisällä polkua $\langle a, \ldots, b \rangle^* \subset k^j$. Jos tiedetään sisäsolmujen järjestys, valitaan d'', jos a ennen b, muuten d'''.



Kuva 3: Klusterin sisäsolmujen väliset potentiaaliset lyhimmät polut.

Algoritmissa 4 kuvataan klusterin k^j sisäsolmujen välisten etäisyyksien laskeminen. Algoritmi käy läpi kaikki klusterit k^j, joille $|k^j| > 4$. Rivillä 3 käydään läpi kaikki parit $c_j, c_k \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$. Rivillä 4 määritetään potentiaaliset vaihtoehdot lyhimmälle polulle $\langle c_j, \ldots, c_k \rangle^* \in \mathcal{V}$ ja valitaan

Algoritmi 4 Klusterin sisäsolmujen etäisyydet

Require: Klusterijoukko $\mathcal{K}$, tunnettujen etäisyyksien joukko $\mathcal{M}$

for all $k^j \in \mathcal{K}$ **do**
  if $|k^j| \geq 4$ **then**
    for all $c_j, c_k \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$ **do**
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_j, c_k, min(P(\langle c_j, \ldots, c_k \rangle^*) : \langle c_j, \ldots, c_k \rangle^* \in k^j,$
        $d(c_j, c_1^{k^j}) + d(c_1^{k^j}, c_n^{k^j}) + d(c_n^{k^j}, c_k),$
        $d(c_j, c_n^{k^j}) + d(c_n^{k^j}, c_1^{k^j}) + d(c_1^{k^j}, c_k))))\}$
    end for
  end if
end for

lyhin etäisyys kohdepisteiden c_j, c_k välille. Algoritmi lopettaa toimintansa, kun kaikki klusterit on käyty läpi.

4.2.4 Jäljelle jäävät etäisyydet eri klustereiden välillä

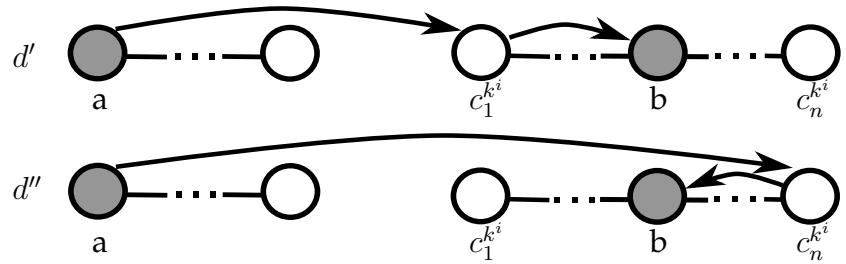
Neljännessä vaiheessa lasketaan juurisolmujen ja sisäsolmujen väliset etäisyydet eri klustereiden välillä sekä sisäsolmujen väliset etäisyydet eri klustereiden välillä. Klusterin k^j juurisolmujen ja klusterin k^i sisäsolmujen väliset etäisyydet lasketaan, jos $|k^i| > 2$. Merkitään juurisolmua $a \in \{c_1^{k^j}, c_n^{k^j}\}$, jos $|k^j| \geq 2$, muuten $a = c_1^{k^j}$. Merkitään solmua $b \in k^i \setminus \{c_1^{k^i}, c_n^{k^i}\}$. Etäisyys $d(a, b) = min(d', d'')$, missä

$$d' = P\langle a, \ldots, c_1^{k^i} \rangle^* + P\langle c_1^{k^i}, \ldots, b \rangle^* \tag{8}$$

ja

$$d'' = P\langle a, \ldots, c_n^{k^i} \rangle^* + P\langle c_n^{k^i}, \ldots, b \rangle^*. \tag{9}$$

Nyt $P\langle a, \ldots, c_1^{k^i} \rangle^*, P\langle c_1^{k^i}, \ldots, b \rangle^*, P\langle a, \ldots, c_n^{k^i} \rangle^*, P\langle c_n^{k^i}, \ldots, b \rangle^* \in \mathcal{M}$. Kuvassa 4 on esitetty klusterin k^j juurisolmun potentiaaliset lyhimmät polut klusterin k^i sisäsolmuun.



Kuva 4: Klusterin k^j juurisolmun ja klusterin k^i sisäsolmun väliset potentiaaliset lyhimmät polut.

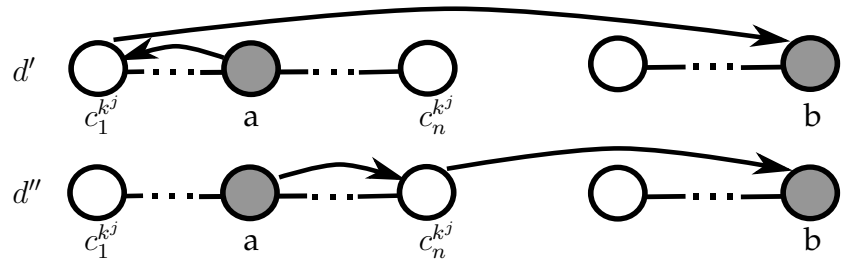
Klusterin k^j sisäsolmujen ja klusterin k^i juurisolmujen väliset etäisyydet lasketaan, jos $|k^j| > 2$. Merkitään sisäsolmua $a \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$ ja merkitään juurisolmua $b \in \{c_1^{k^i}, c_n^{k^i}\}$, jos $k^i \geq 2$, muuten $b = c_1^{k^i}$. Nyt $d(a,b) = min(d', d'')$, missä

$$d' = P\langle a, \ldots, c_1^{k^j}\rangle^* + P\langle c_1^{k^j}, \ldots, b\rangle^* \tag{10}$$

ja

$$d'' = P\langle a, \ldots, c_n^{k^j}\rangle^* + P\langle c_n^{k^j}, \ldots, b\rangle^*. \tag{11}$$

Nyt $P\langle a, \ldots, c_1^{k^j}\rangle^*, P\langle c_1^{k^j}, \ldots, b\rangle^*, P\langle a, \ldots, c_n^{k^j}\rangle^*, P\langle c_n^{k^j}, \ldots, b\rangle^* \in \mathcal{M}$. Kuvassa 5 on esitetty klusterin k^j sisäsolmun potentiaaliset lyhimmät polut klusterin k^i juurisolmuun.



Kuva 5: Klusterin k^j sisäsolmun ja klusterin k^i juurisolmun väliset potentiaaliset lyhimmät polut.

Klusterin k^j ja k^i sisäsolmujen väliset etäisyydet lasketaan, jos $|k^j| \geq 2$ ja $|k^i| \geq 2$. Merkitään $a \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$ ja $b \in k^i \setminus \{c_1^{k^i}, c_n^{k^i}\}$. Nyt määrätään $d(a,b) = min(d', d'', d''', d'''')$, missä

$$d' = P\langle a, \ldots, c_1^{k^j}\rangle^* + P\langle c_1^{k^j}, \ldots, c_1^{k^i}\rangle^* + P\langle c_1^{k^i}, \ldots, b\rangle^*, \tag{12}$$

$$d'' = P\langle a, \ldots, c_1^{k^j}\rangle^* + P\langle c_1^{k^j}, \ldots, c_n^{k^i}\rangle^* + P\langle c_n^{k^i}, \ldots, b\rangle^*, \tag{13}$$

$$d''' = P\langle a, \ldots, c_n^{k^j}\rangle^* + P\langle c_n^{k^j}, \ldots, c_1^{k^i}\rangle^* + P\langle c_1^{k^i}, \ldots, b\rangle^* \tag{14}$$

ja

$$d'''' = P\langle a, \ldots, c_n^{k^j}\rangle^* + P\langle c_n^{k^j}, \ldots, c_n^{k^i}\rangle^* + P\langle c_n^{k^i}, \ldots, b\rangle^*. \tag{15}$$

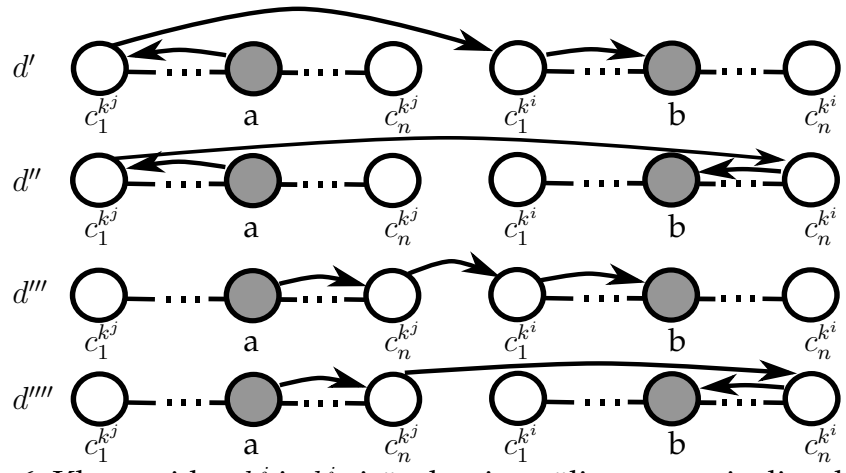
Edellä esitettyjen osareittien pituudet tunnetaan.

Algoritmissa 5 käydään läpi etäisyydet, joissa tutkittavana on kaksi klusteria $k^j \neq k^i$ ja toinen tutkittavista solmuista on sisäsolmu. Rivillä 1 valitaan klusterit $k^j, k^i, k^i \neq k^j$. Riveillä 2–5 tutkitaan tapaukset, joissa klustereille $|k^j| = 1$ ja $|k^i| > 2$. Riveillä 3–5 käydään läpi kaikki solmut

Algoritmi 5 Klustereiden väliset jäljelle jääneet etäisyydet

Require: Klusterijoukko $\mathcal{K}$, tunnettujen etäisyyksien joukko $\mathcal{M}$

1: **for all** $k^j, k^i \in \mathcal{K} : k^j \neq k^i$ **do**
2: **if** $|k^i| > 2$ **then**
3: **for all** $c_x \in k^i \setminus \{c_1^{k^i}, c_n^{k^i}\}$ **do**
4: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_1^{k^j}, c_x, min(d(c_1^{k^j}, c_1^{k^i}) + d(c_1^{k^i}, c_x), d(c_1^{k^j}, c_n^{k^i}) + d(c_n^{k^i}, c_x)))\}$
5: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_x, c_1^{k^j}, min(d(c_x, c_1^{k^i}) + d(c_1^{k^i}, c_1^{k^j}), d(c_x, c_n^{k^i}) + d(c_n^{k^i}, c_1^{k^j})))\}$
6: **if** $|k^j| \geq 2$ **then**
7: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_n^{k^j}, c_x, min(d(c_n^{k^j}, c_1^{k^i}) + d(c_1^{k^i}, c_x), d(c_n^{k^j}, c_n^{k^i}) + d(c_n^{k^i}, c_x)))\}$
8: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_x, c_n^{k^j}, min(d(c_x, c_1^{k^i}) + d(c_1^{k^i}, c_n^{k^j}), d(c_x, c_n^{k^i}) + d(c_n^{k^i}, c_n^{k^j})))\}$
9: **end if**
10: **end for**
11: **end if**
12: **if** $|k^j| > 2$ ja $|k^j| > 2$ **then**
13: **for all** $c_x \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}, c_y \in k^i \setminus \{c_1^{k^i}, c_n^{k^i}\}$ **do**
14: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(c_x, c_y, min(d(c_x, c_1^{k^j}) + d(c_1^{k^j}, c_1^{k^i}) + d(c_1^{k^i}, c_y), d(c_x, c_1^{k^j}) + d(c_1^{k^j}, c_n^{k^i}) + d(c_n^{k^i}, c_y), d(c_x, c_n^{k^j}) + d(c_n^{k^j}, c_1^{k^i}) + d(c_1^{k^i}, c_y), d(c_x, c_n^{k^j}) + d(c_n^{k^j}, c_n^{k^i}) + d(c_n^{k^i}, c_y)))\}$
15: **end for**
16: **end if**
17: **end for**



Kuva 6: Klustereiden k^j ja k^i sisäsolmujen väliset potentiaaliset lyhimmät polut.

$c_x \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}$. Rivillä 4 etsitään potentiaaliset polut lyhimmälle polulle $\langle c_1^{k^j}, \ldots, c_x \rangle^* \in \mathcal{V}$ ja valitaan lyhimmän polun etäisyys etäisyydeksi $d(c_1^{k^j}, c_x)$. Rivillä 5 tehdään samoin etäisyydelle $d(c_x, c_1^{k^j})$. Riveillä 6–9 käydään läpi tapaukset, joissa $|k^j| > 2$. Riveillä 7–8 toiminta on sama kuin riveillä 4–5, mutta tutkittavat etäisyydet ovat $d(c_n^{k^j}, c_x)$ ja $d(c_x, c_n^{k^j})$. Riveillä 12–15 käydään läpi tapaukset, joissa $|k^j| > 2$ ja $|k^i| > 2$. Rivillä 13 käydään läpi solmuparit $c_x \in k^j \setminus \{c_1^{k^j}, c_n^{k^j}\}, c_y \in k^i \setminus \{c_1^{k^i}, c_n^{k^i}\}$. Rivillä 14 valitaan potentiaaliset polut lyhimmälle polulle $\langle c_x, \ldots, c_y \rangle^* \in \mathcal{V}$ ja valitaan lyhin pituus etäisyydeksi $d(c_x, c_y)$. Kuvassa 6 on esitetty klusterin k^j sisäsolmun potentiaaliset lyhimmät polut klusterin k^i sisäsolmuun.

Näiden vaiheiden jälkeen kaikki etäisyydet tunnetaan ja etäisyysmatriisi on rakennettu.

5 Menetelmän testaus ja analysointi

Testitapauksia varten toteutettiin verkon hallinta, geokoodaus, karsintamenetelmä RNR[23], Dijkstran algoritmi, koostamistekniikka ja testidatageneraattori C♯-ohjelmointikielellä. Testit ajettiin tietokoneella, jossa oli Pentium 4 3.00GHz prosessori ja 3GB keskusmuistia. Käyttöjärjestelmänä oli Windows XP SP3.

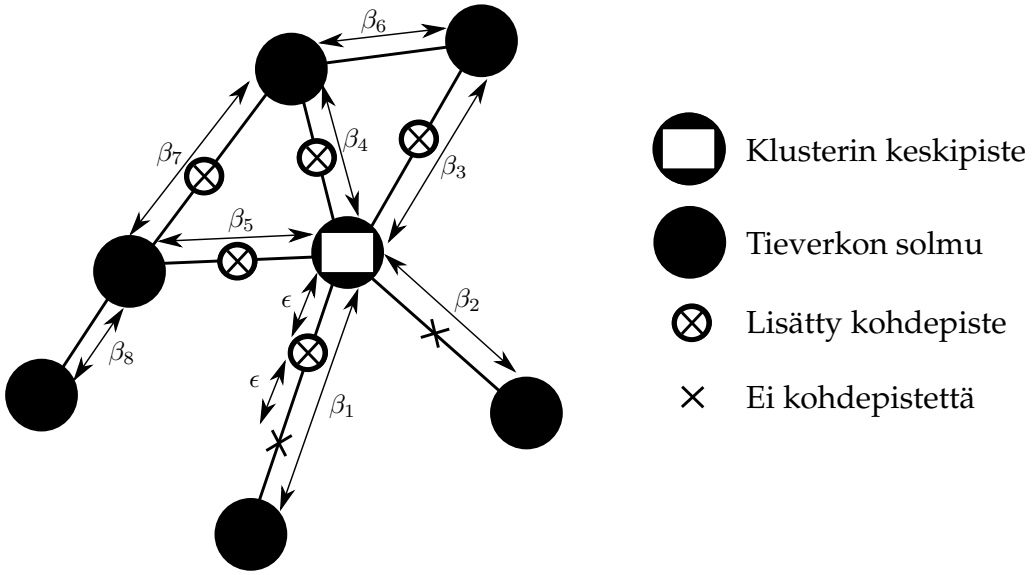
Koostamistekniikka testattiin suuntaamattomassa verkossa $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Verkko muodostettiin Digiroad-kartta-aineistosta Keski-Suomen osalta ja se muodostuu 96 020 solmusta ja 102 161 kaaresta. Solmut esiintyvät ver-

kossa taulukon 2 mukaisesti.

Aste	Määrä	Osuus
1	24605	25,625%
2	36434	37,944%
3	33098	34,470%
4	1860	1,937%
5	23	0,024%

Taulukko 2: Solmujen jakauma verkossa.

Algoritmin tehokkuutta mitattiin joukolla testitapauksia, joissa kohdepisteet sijaitsevat lähellä toisiaan. Tällä pyrittiin simuloimaan jätteidenkeräyksen ja sanomalehtien jakelun optimointiongelmaa, joita varten on rakennettava etäisyysmatriisi. Testitapauksia generoitiin 12. Testidatageneraattorille annettiin parametreina klustereiden määrä α, tiesegmenttien yhteispituus β klusterissa, kohdepisteiden lukumäärä klusteria kohden γ, kohdepisteiden esiintymistodennäköisyys δ ja kohdepisteiden välinen etäisyys ϵ. Jokainen tiesegmentti jaettiin ϵ-välein ja kohdepiste asetettiin kyseiseen väliin todennäköisyydellä δ. Kuvassa 7 on kuvattu testitapaus, jossa $\alpha = 1$, $\gamma = 5$, $\epsilon = 20$ ja $\sum_{i=1}^{8} \beta_i = \beta$. Testitapaukset esitellään taulukossa 3.



Kuva 7: Klusterin muodostus.

#	α	γ	$\alpha \times \gamma$	β	δ	ϵ
1	2	32	64	10 000	0.6	20
2	2	64	128	10 000	0.6	20
3	2	128	256	10 000	0.6	20
4	2	256	512	10 000	0.6	20
5	4	16	64	10 000	0.6	20
6	4	32	128	10 000	0.6	20
7	4	64	256	10 000	0.6	20
8	4	128	512	10 000	0.6	20
9	8	8	64	10 000	0.6	20
10	8	16	128	10 000	0.6	20
11	8	32	256	10 000	0.6	20
12	8	64	512	10 000	0.6	20

Taulukko 3: Testitapaukset ja niiden parametrit.

Testitapaukset ratkaistiin kolmella menetelmällä: Dijkstran yksi yhteen -menetelmällä, verkon karsinnalla ja Dijkstran yksi yhteen -menetelmällä sekä koostamistekniikalla. Dijkstran yksi yhteen -menetelmä ratkaisee yhden kohdepisteparin etäisyyden kerrallaan. Koostamistekniikassa käytetty lyhimmän polun algoritmi oli Dijkstran yksi yhteen -menetelmä. Dijkstran algoritmin tietorakenteena oli prioriteettijono[2]. Koostamistekniikassa verkko karsittiin ennen koostamisen suorittamista.

Ratkaistujen testitapauksien tulokset on esitetty taulukossa 4. Dijkstran yksi yhteen -menetelmä ratkaisi testitapaukset hitaimmin kuluttaen aikaa 12–1300 minuuttia testitapauksesta riippuen. Verkon karsinnan avulla Dijkstran yksi-yhteen menetelmän tehokkuus parani, jolloin testitapaukset ratkesivat 1–150 minuutissa. Koostamistekniikka ratkaisi testitapaukset 10–540 sekunnissa. Suorahakujen kuluttama aika oli 0,05–1,00 sekuntia testitapauksesta riippuen. Lisäksi määriteltiin uusi metriikka, koostamiskerroin, joka kertoo juurisolmujen ja kaikkien kohdepisteiden lukumäärän suhteen. Koostamiskerroin $\zeta, 0 < \zeta \leq 1$, on sitä pienempi, mitä koostetumpi testitapaus on. Koostamiskertoimet olivat 0,020–0,282 testitapauksesta riippuen.

Suurissa testitapauksissa 94% etäisyyksistä laskettiin alle kahdessa sekunnissa. Tämä tarkoittaa vähintään 246 415 lyhintä polkua, testitapauksesta riippuen. Testitapauksissa, joissa kohdepisteiden määrä oli sama, mutta klusterimäärä oli eri, ei havaittu merkittävää eroa suoritusajoissa

[2]Tietorakenteessa toiminnot lisää O(log n), poistaPienin O(log n) ja katsoPienin O(log n).

#	Dijkstra aika Dijk. oper.	RNR + Dijkstra aika Dijk. oper.	Koostamistekniikka aika Dijk. oper. taulukko-operaatiot aika	ζ
1	12.41 min 4096	1.17 min 4096	9,422 s 529 0,047 s	0,129
2	46.22 min 16 384	5.56 min 16 384	2,532 s 324 0,032 s	0,020
3	140.57 min 65 536	16.49 min 65 536	1.15 min 5776 1s	0,088
4	1125.42 min 262 144	147.02 min 262 144	8.44 min 14 400 1 s	0,055
5	12.03 min 4096	1.18 min 4096	8,984 s 576 0,031 s	0,141
6	47.16 min 16 384	5.03 min 16 384	19,1 s 1024 0,1 s	0,063
7	243.13 min 65 536	25.36 min 65 536	1.19 min 4225 1 s	0,064
8	1053.39 min 262 144	78.33 min 262 144	4.49 min 14 162 1 s	0,055
9	15.26 min 4096	2.40 min 4096	24,9 s 1156 0,1 s	0,282
10	65.21 min 16 384	7.01 min 16 384	43,1 s 1936 0,1 s	0,118
11	283.23 min 65 536	34.53 min 65 536	2.60 min 4096 1 s	0,063
12	1300.50 min 262 144	125.13 min 262 144	7.08 min 15 376 1 s	0,059

Taulukko 4: Testien tulokset.

eikä koostamiskertoimissa. Dijkstran yksi yhteen -operaatioiden määrän nousu olisi ollut odotettavaa klustereiden määrän lisääntyessä, mutta tätä ei testitapauksissa havaittu. Eroa suorahakuoperaatioiden kuluttamassa ajassa ei myöskään havaittu.

Koostamistekniikan suoritusajasta valtaosa kuluu perinteisen lyhimmän polun algoritmin laskentaan. Tuloksien perusteella koostamiskerroin ja koostamistekniikan tehokkuus monesta moneen -lyhimmän polun ongelmassa riippuvat toisistaan.

6 Jatkotutkimus

Edellä esitetty koostamistekniikka on tuloksien perusteella kehitysaskel etäisyysmatriisin rakentamiseen tapauksissa, joissa kohdepisteet sijaitsevat lähellä toisiaan.

Jatkotutkimuksissa tullaan pohtimaan koostettujen klustereiden muodostamista siten, että klustereiden juurisolmut ovat tieverkon solmuja. Tämän seurauksena klustereiden juurisolmut voivat olla asteiltaan ≥ 2 ja eri klustereilla voi olla samoja juurisolmuja. Kun kahdella tai useammalla eri klusterilla on sama juurisolmu, voidaan klusterit linkittää toisiinsa ja on mahdollista luoda suurempia klustereita.

Lisäksi aiotaan tutkia etäisyysmatriisin rakentamista ja koostettujen klustereiden käyttöä kaluston reitinoptimointiongelmissa, jotka sisältävät kapasiteettirajoitteita ja aikaikkunoita.

7 Yhteenveto

Tässä raportissa esitettiin lyhimmän polun laskentaa tehostava koostamistekniikka. Kirjallisuuskartoituksessa havaittiin, että vastaavanlaista tutkimusta ei ole tehty. Lyhimmän polun laskennan kiihdytystekniikoita on kuitenkin tutkittu paljon viime vuosina.

Esitetty koostamistekniikka muodostuu kahdesta eri vaiheesta: kohdepisteiden koostamisesta ja etäisyysmatriisin rakentamisesta. Menetelmän tehokkuus osoitettiin merkittäväksi testitapauksissa, joissa kohdepisteet sijaitsivat lähellä toisiaan. Lisäksi esitettiin uusi metriikka, koostamiskerroin, joka kuvaa koostamistekniikan tehokkuutta tutkittavassa tapauksessa.

Kiitokset

Tämän raportin tekemistä tuki Tekes TRANSOPT-projektin kautta. Tämä tuki on huomioitu suurella kiitollisuudella. Haluan kiittää ohjaajiani professori Olli Bräysyä (Agora Innoroad Laboratorio) ja tutkija Tuukka Purasta (Agora Innoroad Laboratorio) runsaasta palautteesta ja kommenteista, jotka auttoivat prosessin aikana.

Lähteet

[1] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993, sivut 93-133.

[2] H. Bast, S. Funke, P. Sanders, D. Schultes, *Fast Routing in Road Networks with Transit Nodes*, Science, Vol. 316, 2007, sivu 566.

[3] R. Bauer, D. Delling, P. Sanders, D. Schieferdecker, D. Schultes, D. Wagner, *Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm*, Lecture Notes in Computer Science, 2008, sivut 303–318.

[4] P.P. Chakrabarti, S. Ghose, S.C. Desarkar, *Heuristic Search through Islands*, Artificial Intelligence, Vol. 29, 1986, sivut 339–348.

[5] D. Delling, M. Holzer, K. Müller, F. Schulz, D. Wagner, *High-Performance Multi-Level Graphs*, 9th DIMACS Challenge on Shortest Paths, 2006.

[6] D. Delling, P. Sanders, D. Schultes, D. Wagner, *Highway Hierarchies Star*, 9th DIMACS Implementation Challenge, 2006.

[7] E.W. Dijkstra, *A Note on Two Problems in Connexion with Graphs*, Numerische Mathematik, Vol. 1, 1959, sivut 269–271.

[8] J.F. Dillenburg, P.C. Nelson, *Improving Search Efficiency using Possible Subgoals*, Mathematical and Computer Modelling, Vol. 22, 1995, sivut 397–414.

[9] R. Geisberger, P. Sanders, D. Schultes, D. Delling, *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks*, Lecture Notes in Computer Science, 2008, sivut 319–333.

[10] A.V. Goldberg, C. Harrelson, *Computing the Shortest Path: A Search meets Graph Theory*, Proceedings of the 16th annual ACM-SIAM symposium in Discrete Algorithms, 2005, sivut 156–165.

[11] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Better Landmarks within Reach*, Lecture Notes in Computer Science, Vol. 4525, 2007, sivut 38–51.

[12] A.V. Goldberg, H. Kaplan, R.F. Werneck, *Reach for A*: Efficient Point-to-Point Shortest Path Algorithms*, Workshop on Algorithm Engineering & Experiments, 2006, sivut 129–143.

[13] R. Gutman, *Reach-based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks*, Proceedings in 6th International Workshop on Algorithm Engineering, 2004, sivut 100–111.

[14] P.E. Hart, N.J. Nilsson, B. Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, System Science and Cybernetic, IEEE Transactions on, Vol. 4, 1968, sivut 100–107.

[15] M. Holzer, F. Schulz, D. Wagner, *Engineering Multi-Level Overlay Graphs for Shortest-Path Queries*, Workshop on Algorithm Engineering and Experiments, 2006, sivut 156–170.

[16] S. Irnich, *A Multi-Depot Pickup and Delivery Problem with a Single Hub and Heterogeneous Vehicles*, European Journal of Operational Research, Vol. 122, 2000, sivut 310–328.

[17] B. Kim, S. Kim, S. Sahoo, *Waste Collection Vehicle Routing Problem with Time Windows*, Computers & Operations Research, Vol. 33, 2006, sivut 3624-3642.

[18] G.A. Klunder, H.N. Post, *The Shortest Path Problem on Large-Scale Real-Road Networks*, Networks, Vol. 48, 2006, sivut 182–194.

[19] S. Knopp, P. Sanders, D. Schultes, F. Schulz, D. Wagner, *Computing Many-to-Many Shortest Paths using Highway Hierarchies*, In: Workshop on Algorithm Engineering and Experiments, 2007.

[20] U. Lauther, *Slow Preprocessing of Graphs for Extremely Fast Shortest Path Calculations*, Lecture at the Workshop on Computational Integer Programming at ZIB, 1997.

[21] F. Lawler, A. Rinnooy-Kan, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley, 1985.

[22] T.A.J. Nicholson, *Finding the Shortest Route between Two Points in a Network*, The Computer Journal, Vol. 9, 1996, sivut 275–280.

[23] T. Puranen, J. Brigatti, *Topology-based Optimal Road Network Pruning - A Method for Speeding up Shortest Path Computation*, Evolutionary Methods for Design, Optimization and Control, 2007, sivut 409-415.

[24] P. Sanders, D. Schultes, *Highway Hierarchies Haster Exact Shortest Path Queries*, Lecture Notes in Computer Science, Vol. 3669, 2005, sivut 568–579.

[25] P. Sanders, D. Schultes, *Engineering Highway Hierarchies*, Lecture Notes in Computer Science, Vol. 4168, 2006, sivut 804–816.

[26] P. Toth, D. Vigo, *The Vehicle Routing Problem*, SIAM, 1987.

[27] D. Wagner, T. Willhalm, *Geometric Speed-Up Techniques for Finding Shortest Paths in Large Sparse Graphs*, Lecture Notes in Computer Science, Vol. 2832, 2003, sivut 776–787.