Reitinhakualgoritmien toiminnan optimointi tieaineistolla

Oulun yliopisto
Tieto- ja sähkötekniikan tiedekunta
Tietojenkäsittelytieteiden tutkinto-ohjelma
Gradu -tutkielma
Olli Anttonen
5.6.2020

Tiivistelmä

Nopeimman reitin haku tieaineistolla on osa monien ihmisten arkipäivää esimerkiksi käytettäessä mobiililaitteita. Optimaalisen reitin haku kahden pisteen välillä on aihe, josta on olemassa paljon aiempaa tutkimustietoa. Klassisia algoritmeja, jotka ratkaisevat nopeimman reitin ongelman graafissa ja joita voidaan hyödyntää myös tieaineistolla ovat Dijkstran algoritmi ja A* -algoritmi. Uusimmissa tutkimuksissa on kehitetty nopeita algoritmeja, joille tarvittava reittiaineisto esikäsitellään nopeuden parantamiseksi. Tämän työn keskeisenä tutkimuskysymyksenä on selvittää, miten klassisia reitinhakualgoritmeja voidaan optimoida Suomen kattavalla Digiroad-tieaineistolla aiemmasta tutkimuksesta löytyvin menetelmin. Työssä tutkitaan tietorakenteiden optimointia tieaineistolle, keskinopeuden käyttöä heuristiikkana matka-ajan optimoinnissa sekä miten tieaineiston metadataa voidaan hyödyntää algoritmien optimoinnissa. Saatuja tuloksia verrataan esikäsittelyä hyödyntäviin algoritmeihin.

Kirjallisuuskatsaus sisältää käsittelyn klassisista reititysalgoritmeista, tieaineiston erityispiirteistä sekä miten esikäsittelyä hyödyntävien menetelmien kehitys on vaikuttanut tutkimukseen. Lisäksi käsitellään tietorakenteiden roolia reititysalgoritmien toiminnassa sekä millaisia tuloksia eri optimointimenetelmien yhdistelmillä on saatu. Tämän työn tutkimusmenetelmänä on suunnittelutiede (Design Science). Menetelmän avulla luotu artefakti lukee Digiroad-aineistoa ja toimii yhtenäisenä alustana kolmelle erilaisille koejärjestelylle, jotka vastaavat esitettyihin tutkimuskysymyksiin. Jokaisessa koejärjestelyssä kerättiin tietoa algoritmien toiminnassa suorituskyky ja laatumittareilla käyttäen viittäsataa satunnaisesti valittua pisteparia.

Tietorakenteiden ja algoritmien koejärjestelyn keskeisenä tuloksena havaittiin A* -algoritmin suoriutuvan 3,59 kertaa nopeammin kuin Dijkstran algoritmi Digiroad-aineistolla. Lisäksi tietorakenteiden käytännön toteutuksella on suuri vaikutus algoritmien nopeuteen. Binäärikeon toimintaa optimoimalla voitiin parantaa reitityksen nopeutta A* -algoritmilla 6,43 kertaisesti verrattuna perinteiseen binäärikekoon. Keskinopeutta käsittelevässä koejärjestelyssä tutkittiin A* -algoritmin heuristiikkaa ja millä arvioidulla keskinopeudella saadaan parhaat tulokset heuristiikassa linnuntietä hyödyntäen. Paras tasapaino reititysnopeuden ja optimaalisesta reitistä poikkeaman välillä saavutettiin arvoilla 70–90 km/h. Tällöin poikkeama oli 0,03–1,61 % optimaalisesta ratkaisusta reititysajan ollessa 3,54–14,84 kertaa parempi kuin referenssinä toimineessa Dijkstran algoritmissa. Kolmannessa koejärjestelyssä hyödynnettiin Digiroad-aineiston metatietoja luomalla niiden pohjalta erilaisia kaksitasoisia hierarkkisia algoritmeja käyttäen pohjana A* -algoritmia ja aiempien koejärjestelyjen tuloksia tavoitteena parantaa reititysaikoja. Tulokset voidaan jakaa kahteen ryhmään. Laadukkaimmilla algoritmeilla saavutettiin 14,25–18,27 kertainen parannus Dijkstran algoritmiin verrattuna poikkeaman optimaalisesta reitistä ollen 0,79–0,92 %. Nopeimmilla algoritmeilla saavutettiin 26,36–37,54 kertainen parannus Dijkstran algoritmiin, mutta poikkeama optimaalisesta reitistä oli tällöin 2,60–3,21 %. Tutkituilla algoritmeilla voidaan lähestyä nopeudeltaan yksinkertaisia esikäsittelyä hyödyntäviä algoritmeja, jos poikkeama optimaalisesta reitistä on hyväksyttävissä.

Asiasanat
Algoritmit, Dijkstran algoritmi, A* algoritmi, reititys, reitinhaku, optimointi, hierarkia, heuristiikka, keskinopeus, tieaineisto, Digiroad, Binäärikeko, Fibonaccin keko, Design Science, suunnittelutiede

Käsitteet

Käsite	**Selite**
Aineistotietokanta	Lähdeaineisto (esim. Digiroad) alkuperäisessä toimitusmuodossaan
Askel	Reititysalgoritmin alkeisoperaatio, jossa algoritmi hakee tietorakenteesta pienimmän avaimen omaavan arvon ja käsittelyn jälkeen merkitsee tämän loppuun käsitellyksi
Branch Pruning	Hakualueen rajaaminen määritellyn funktion avulla
Digiroad	Suomen kansallinen tietojärjestelmä, johon on koottu kattava data tie- ja katuverkostosta
Digiroad-aineisto	Digiroad tietojärjestelmästä vapaasti saatavilla oleva digitaalisessa muodossa oleva kopio
Heuristiikka	Funktio, joka pyrkii ohjaamaan algoritmin suoritusta kohti määriteltyä tavoitetta
Keskilinjageometria	Digiroad-aineistossa kahden risteyksen välistä tieosuutta kuvataan sen keskilinjan muodostaman pisteketjun avulla
Liikenne-elementti	Digiroad-aineistossa olevan tieosuuden pienin itsenäinen yksikkö. Graafiksi muunnettuna edustaa yhtä siihen kuuluvaa kaarta.
Tieosuus	Kts. Liikenne-elementti
Tielinkki	Kts. Liikenne-elementti

Sisällys

1. Johdanto

Graafi koostuu pisteistä ja niitä yhdistävistä kaarista. Kartassa olevat tiet ja risteykset voidaan kuvata graafin avulla, jolloin optimaalisen reitin laskentaan voidaan soveltaa graafeilla toimivia algoritmeja. Kahden pisteen välinen reitinhaku graafissa on yleinen ongelma, jota on tutkittu jo vuosikymmeniä ja jonka ratkaisemisessa on tänä aikana tehty merkittäviä edistysaskelia (Fu, Sun & Rilett, 2006). Paikannuksen ja digitaalisten karttojen kehityksen myötä tarve tehokkaille reitinhakumenetelmille on kasvanut. Tänä päivänä lähes jokaisella on käytössään mobiililaitteita, jotka hyödyntävät paikannusta ja reitinhakua tieaineistolla. Lisäksi vastaavia reitinhakuominaisuuksia voidaan hyödyntää simulaatioympäristöissä, kuten esimerkiksi erilaiset digitaalisia karttoja hyödyntävät liiketoimintasovellukset sekä pelit.

Viime aikoina on erityisesti tutkittu aineiston esikäsittelyä keinona tehostaa reitinhakua, ja tutkimustyö tällä osa-alueella on edennyt nopeasti (Bast ja muut, 2016; Sommer 2014). Esikäsittelyllä tarkoitetaan tieaineiston muokkaamista lisäämällä siihen tietoja ennen reititystä tarkoituksena nopeuttaa reitinhakua (Sommer, 2014). Esikäsittelyn perusteella algoritmit voidaan jakaa skaalalle, jossa toisessa laidassa ovat algoritmit, jotka eivät esikäsittele tieaineistoa (klassiset algoritmit) ja toisessa laidassa algoritmit, jotka laskevat kaikki mahdolliset reitit ennakkoon. Sommer (2014) viittaa artikkelinsa johdannossa esikäsittelyn käytön olevan sovelluskohtainen strategiavalinta, jossa ilman esikäsittelyä vältytään mahdollisesti raskaalta laskentaoperaatiolta, mutta vastapainona on kasvanut reitinhakuun käytettävä aika. Erityisesti muuttuvien aineistojen kohdalla klassisilla reitinhakualgoritmeilla on edelleen roolinsa.

Digiroad on suomalainen kansallinen tietojärjestelmä, joka sisältää tietoja Suomen tie- ja katuverkostosta sekä sen ominaisuuksista (Digiroad 2019a). Aineisto kattaa koko Suomen ja sen olemassaolo ja ylläpito ovat lakiin perustuvia (Laki tie- ja katuverkon tietojärjestelmästä, 2003). Aineiston avoimuus, kattavuus ja tarkkuus tekevät siitä sopivan tutkimuskäyttöön. Digiroad sisältää kattavan digitaalisen tieaineiston, jota on mahdollista hyödyntää muun muassa reititystarkoituksissa. Liikenne-elementit eli tieosuudet koostuvat niiden keskilinjojen muodostamien pisteketjujen avulla, jolloin teitä voidaan kuvata graafissa kaarina. Pelkän tieverkoston lisäksi Digiroad aineistosta saadaan luettua tieverkostoon liittyvää metadataa (ominaisuustietoa). Esimerkkeinä tällaisista ominaisuustiedoista ovat tien nopeusrajoitus, leveys, päällystetyyppi ja toiminnallinen luokka. Aineiston tietoja ylläpidetään Maanmittauslaitoksen, Liikenneviraston ja kuntien toimesta. Digiroad-aineisto sisältää noin 483 000 kilometriä tien keskilinjageometriaa, jonka keskimääräinen sijaintitarkkuus on 3 metriä. Digiroad-aineistosta kuvatut tiedot pohjautuvat Suomen liikenneviraston julkaisemiin julkisesti saatavilla oleviin aineistoihin (Digiroad 2019a, 2019b).

Digiroad on poikkeuksellinen aineisto avoimuutensa, kattavuutensa ja tarkkuutensa vuoksi. Aiemmassa tutkimuksessa on viitteitä siitä, että avoimet aineistot sisältävät puutteita esimerkiksi metadatan suhteen verrattuna kaupallisiin aineistoihin. Esimerkiksi artikkelissaan Delling, Goldberg, Pajor ja Werneck (2011) kertovat, että heidän tiedossaan ei ole avoimia aineistoja, jotka sisältävät tarkat kääntymissäännöt tieverkostolle. Tieaineiston merkityksestä algoritmien suunnittelussa Sommer (2014) toteaa, että lähdeaineiston valinnalla voi olla merkittävä vaikutus algoritmien suunnitteluprosessiin, koska tulokset eri vaihtoehtojen kokeilusta toimivat keskeisenä

palautteena algoritmien kehitysprosessissa. Sanders ja Schultes (2007) ovat tutkimuksessaan pitäneet todennäköisenä, että tieaineistojen saatavuuden parantuminen viime vuosina on mahdollistanut reititysalgoritmien nopean kehittymisen. Näistä syistä Digiroad on tutkimuksen kannalta mielenkiintoinen aineisto.

Tämän tutkielman tarkoituksena on selvittää kokeellisesti, miten klassisten reititysalgoritmien suorituskykyä voidaan parantaa hyödyntämällä koko Suomen kattavaa Digiroad-aineistoa ja siihen liittyvää metatietoa sekä olemassa olevaa tutkimustietoa erilaisista optimointimenetelmistä. Klassisia hyvin tunnettuja reititysalgoritmeja ovat esimerkiksi Dijkstran algoritmi (Dijkstra, 1959) ja A* -algoritmi Hart, Nilsson ja Raphael (1968). Optimointimenetelmillä tarkoitetaan algoritmien työn vähentämiseen tähtääviä keinoja, kuten esimerkiksi hakualueen rajaaminen ja käsiteltävien pisteiden vähentämiseen tarkoitetut menetelmät. Esikäsittelyä hyödyntävät optimointimenetelmät on otettu huomioon työn viitekehyksessä ja niihin liittyvää tutkimusta käsitellään osana kirjallisuuskatsausta. Työ keskittyy kuitenkin sellaisten optimointimenetelmien kehittämiseen, jotka eivät vaadi aineiston esikäsittelyä. Lisäksi aihe on rajattu pelkästään kahden pisteen väliseen reititykseen. Tutkittavilta algoritmeilta ja menetelmiltä ei lähtökohtaisesti vaadita matemaattista todistusta oikeellisuudesta eikä niiden ole välttämättä palautettava täysin oikeaa tulosta. Tästä huolimatta poikkeama optimaalisesta ratkaisusta on oltava kohtuullinen, sen on oltava mitattavissa ja laatu sekä soveltuvuus käyttökohteeseen arvioitavissa. Lisäksi algoritmien toiminnassa huomioidaan mahdollisuus optimoida reittiä joko lyhimmän etäisyyden tai matka-ajan mukaisesti. Koska nykyinen tutkimus painottuu vahvasti esikäsittelymenetelmiin, on myös perusteltua tarkastella käytännössä, onko vastaavia reitinhaun nopeushyötyjä mahdollista saavuttaa jatkokehittämällä klassisia algoritmeja.

Eräänä motivaationa tälle työlle ja yleisesti reitinhakualgoritmien tutkimukselle on ammatillinen mielenkiinto. Suunnittelin ja toteutin ennen vuotta 2010 reitinhakualgoritmeja ja digitaalisten karttojen ratkaisuja ajoneuvojen tietojärjestelmiin. Eräs haaste tässä käyttöympäristössä oli reititykseen käytetty aika erityisesti pitkillä reiteillä tilanteissa, jossa laskenta ja tallennuskapasiteetti olivat huomattavan rajallisia. Näiden kokemusten perusteella haluan tällä työllä tarjota tuloksia, jotka olisivat hyödynnettävissä vastaavissa käytännön sovelluksissa reititysaikojen optimointiin.

Tämän työn keskeinen ylätason tutkimuskysymys on, miten reititysalgoritmin tehokkuutta voidaan optimoida Digiroad-aineistolla huomioiden reittien laatu ja algoritmin toimintanopeus. Tähän kysymykseen vastataan kolmen alikysymyksen avulla. Ensimmäisenä kysymyksenä selvitetään millä tavoin reititysalgoritmien käyttämiä tietorakenteita voidaan optimoida tieaineistolle. Toisena kysymyksenä tarkastellaan mitä keskinopeutta voidaan käyttää reititysalgoritmin heuristisessa optimoinnissa Suomen tieaineistolla. Kolmantena kysymyksenä tutkitaan, miten reititysalgoritmin nopeutta voidaan parantaa hyödyntämällä Digiroad-aineiston sisältämää metadataa.

Työssä käytetään tutkimusmenetelmänä suunnittelutiedettä (design science). Menetelmän keskeiset periaatteet kuvataan artikkelin Hevner, March, Park ja Ram (2004) mukaisesti ja osana tutkimusmenetelmän esittelyä tätä työtä vertaillaan suunnittelutieteen ohjesääntöihin. Menetelmän mukaisesti osana työtä rakennetaan mittausartefakti, jolla koejärjestelyihin suunnitellut algoritmit testataan. Artefaktin tehtävänä on tarjota yhtenevä alusta, joka lukee koko Suomen Digiroad-aineiston tietokoneen muistiin ja jonka avulla kokeellisia algoritmeja voidaan suorittaa sekä mittaustuloksia tallentaa analysointia varten. Algoritmien suorituskyvystä tallennetaan

laatua ja suorituskykyä kuvaavat mittarit, jollaisia ovat esimerkiksi suoritusaika, käsiteltyjen pisteiden määrä ja reitin kokonaispituus.

Kirjallisuuskatsauksessa käydään läpi relevanttia olemassa olevaa tutkimusta. Keskeisiä lähteitä ovat artikkelit Bast ja muut (2016), Fu ja muut (2006) sekä Sommer (2014), jotka käsittelevät tutkimuksen nykytilaa. Aluksi käsitellään klassinen Dijkstran algoritmi sekä tieaineiston erityispiirteitä. Viitekehyksen rakentamiseksi käsitellään aineiston esikäsittelymenetelmiä. Reitinhaun optimointimenetelmät käsitellään jaettuna kolmeen kategoriaan: hakualuetta rajaavat menetelmät, hakuongelmaa pienentävät menetelmät ja käsiteltävien pisteiden vähentämiseen pohjautuvat menetelmät. Lisäksi käsitellään eri menetelmien yhdistelmiä. Tietorakenteiden osalta käsitellään keskeistä tutkimustietoa binäärikeosta ja Fibonaccin keosta. Näillä tietorakenteilla saatuja tutkimustuloksia käydään läpi erityisesti reitityskäyttöä peilaten.

Testattavan artefaktin ja koejärjestelyiden keskeiset ominaisuudet on kuvattu omassa luvussaan. Samassa yhteydessä on käsitelty tarkemmin Digiroad-aineistotietokannan rakennetta ja ominaisuuksia. Artefaktin keskeiset ominaisuudet ovat jaettavissa kolmeen osaan. Ensimmäisenä luetaan Digiroad-aineistoa ja muunnetaan se sopivaan graafimuotoon tietokoneen muistissa sekä suodatetaan siitä tarpeeton tieto. Koejärjestelyihin sisällytetään ainoastaan henkilöautolle ajettavat tiet, jotta aineisto olisi mahdollisimman yhtenäinen. Seuraavaksi valitaan tutkittavat pisteet algoritmien vertailuun huomioiden koejärjestelyn toistettavuus kaikille eri algoritmeille. Lopuksi suoritetaan tarvittavat kokeet halutuille algoritmeilla sekä tallennetaan mittausdata. Artefaktilla suoritetaan yhteensä kolme eri koejärjestelyä, jossa edellisten tuloksia hyödynnetään seuraavissa. Ensimmäisenä kokeena vertaillaan Dijkstran algoritmin ja A* -algoritmin toimintaa eri tietorakenteilla ja pyritään näin löytämään Digiroad-aineistolla parhaiten toimiva yhdistelmä. Toisessa koejärjestelyssä tutkitaan A* -algoritmin heuristiikan optimointia matka-ajan perusteella tarkoituksena löytää optimaalinen arvioitu keskinopeus, joka tarjoaa parhaan tasapainen laadun ja reititysnopeuden välillä. Kolmannessa koejärjestelyssä testataan erilaisia hierarkkisia reitinhakumenetelmiä pohjautuen A* -algoritmiin ja Digiroad-aineiston metadataan.

Mittaustulokset ja niiden evaluointi on tehty erillisessä luvussa. Tulokset käydään läpi jokaiselle koejärjestelylle erikseen. Kaikissa koejärjestelyissä on käytetty yhteenlaskettuja tuloksia 500 satunnaisen pisteparin reitityksestä. Verrokkialgoritmit tarjoavat tulosten osalta vertailupohjan aiempaan tutkimukseen, jolloin tulosten merkitystä voidaan peilata viitekehykseen. Keskustelu ja pohdinta luvussa on käsitelty tarkemmin työn merkitystä tutkimuksellisen viitekehyksen kautta sekä miten saadut tulokset sopivat siihen. Lisäksi luvussa on käsitelty työn rajoitteita, prosessin aikana opittuja asioita sekä jatkotutkimusaiheita.

Artefaktin lähdekoodit ovat saatavilla osoitteesta http://pathfinding.anttonen.info

2. Reitinhaun optimointi aiemmassa tutkimuksessa

Tässä tutkimuksessa tarkastellaan tilannetta, jossa reitinhakualgoritmi suorittaa reitinhakua staattisella tieaineistolla kahden pisteen välillä ja näin ollen olemassa olevaa tutkimustietoa on haettu tästä näkökulmasta. Yleisenä periaatteena tutkimustiedon läpikäynnissä on hyödynnetty tutkimustietokannoista löydettyjä muiden tutkijoiden tekemiä kirjallisuuskatsauksia. Näistä kirjallisuuskatsauksista on taas haettu keskeisiä alkuperäisiä lähteitä, jotka liittyvät tutkittavaan aihealueeseen. Kirjallisuuskatsauksien käytön tarkoituksena on hyödyntää keskeisten tutkijoiden näkemystä tällä hetkellä varsin laajaan tutkimuskenttään. Lisäksi artikkeleita on haettu ACM ja IEEE tietokannoista. Keskeisinä lähteinä klassisiin reitinhaun optimointimenetelmiin on käytetty artikkeleita Bast (2009), Fu ja muut (2006) sekä osaltaan uudempia esikäsittelymenetelmistä tehtyjä artikkeleita Sommer (2014) sekä Bast ja muut (2016).

Klassinen algoritmi, joka ratkaisee lyhimmän polun ongelman graafissa, on Dijkstran algoritmi (Dijkstra, 1959). Algoritmilla voidaan hakea lyhin polku kahden pisteen välillä tai alkupisteestä kaikkiin muihin graafin pisteisiin. Dijkstran algoritmi toimii pohjana suurelle osalle myöhempää tutkimusta, jota käsitellään seuraavissa kappaleissa. Tähän jälkeen tieaineiston eroja verrattuna perinteiseen teoreettiseen graafiin käydään läpi aiemman tutkimuksen perusteella.

Tieaineiston esikäsittely on keskeinen tutkimussuunta uusimmissa reititysalgoritmeissa kuten esimerkiksi artikkelit Sommer (2014) sekä Bast ja muut (2016) osoittavat. Esikäsittely tarkoittaa tieaineiston muokkaamista tai tiedon lisäämistä siihen ennen varsinaista reititystä, jonka tarkoituksena on nopeuttaa reitinhakua (Sommer, 2014). Tässä katsauksessa tarkastellaan esikäsittelyn suhdetta klassisiin algoritmeihin ja optimointimenetelmiin, jotka eivät hyödynnä esikäsittelyä. Näin muodostetaan tutkimustietoon pohjautuen ymmärrys siitä miten klassiset ja esikäsittelyä hyödyntävät menetelmät vertautuvat toisiinsa tehokkuuden ja laskentaresurssien osalta.

Reitinhaun optimointimenetelmiä tarkastellaan perustuen luokitukseen artikkelista Fu ja muut (2006). Nämä optimointimenetelmät on jaettu kategorioittain hakualuetta rajaaviin menetelmiin, hakuongelmaa pienentäviin menetelmiin ja käsiteltävien pisteiden vähentämiseen perustuviin menetelmiin. Menetelmien toimintaperiaatteiden lisäksi käsitellään tutkimustiedosta saatuja tuloksia menetelmien toimivuudesta ja tehokkuudesta. Lisäksi käsitellään tutkijoiden julkaisemia näkemyksiä eri optimointimenetelmiä yhdistämisestä toisiinsa.

Lisäksi tarkastellaan erilaisten tietorakenteiden merkitystä reititysalgoritmin toiminnalle aiemman tutkimuksen näkökulmasta. Keskeisiä tietorakenteita, joihin perehdytään tarkemmin, ovat binäärikeko (Williams, 1964) ja Fibonaccin keko (Fredman & Tarjan, 1984). Olemassa olevaa tutkimustietoa ja tietorakenteiden välisiä eroja suorituskyvyssä ja toteutuksessa tarkastellaan erityisesti reitityskäytössä tieaineistolla.

2.1 Klassinen Dijkstran algoritmi

Ehkäpä parhaiten tunnettu klassinen algoritmi, joka ratkaisee ongelman kahden pisteen välisestä reitityksestä graafissa, on Dijkstran algoritmi (Dijkstra, 1959). Algoritmi on toiminut pohjana kehitykselle reititysalgoritmien tutkimuksessa jo vuosikymmeniä (Bast, 2009) ja lisäksi sitä voidaan pitää myös perusratkaisuna silloin, kun halutaan löytää lyhin reitti yhdestä pisteestä kaikkiin muihin pisteisiin (Bast ja muut, 2016). Algoritmi toimii graafilla, joka sisältää joukon pisteitä V niihin liitettyjen kaarien joukon E avulla. Tällöin esimerkiksi Bollobás (1998, s. 1) esittää kirjassaan graafin G seuraavasti: $G = (V, E)$.

Dijkstran algoritmi on perusteiltaan yksinkertainen ja sen toiminta on hyvin kuvattua ja oikeellisuus matemaattisesti todistettu (Bast, 2009; Cormen, Leiserson, Rivest & Stein, 2009; Dijkstra 1959). Lyhyesti kuvattuna Dijkstran algoritmi aloittaa alkupisteestä ja tutkii polkuja seuraaviin pisteisiin ulospäin lyhimmästä aloittaen. Algoritmi asettaa tutkituille pisteille suorituksen aikana tietoa etäisyydestä alkupisteeseen ja lisäksi päivittää tiedon, jos se löytää paremmin reitin samaan pisteeseen suorituksen aikana. Keskeisenä ominaisuutena Dijkstran algoritmi ei erikseen pyri kohti loppupistettä, vaan se käy läpi kaikki pisteet etäisyysjärjestyksessä, kunnes se lopulta saavuttaa loppupisteen. Näin ollen, jos algoritmin suoritusta ei pysäytetä se löytää lyhimmän polun kaikkiin graafin pisteisiin tai vaihtoehtoisesti suoritus voidaan pysäyttää, kun loppupiste on löytynyt. Vaatimuksena algoritmin toiminnalle on kuitenkin se, että kaarien E pituus ei saa olla negatiivinen. Dijkstran algoritmin toiminta on kuvattu tarkemmin luvussa 4.

Lisäksi on huomioitava, että Dijkstran algoritmin suorituskyky riippuu merkittävästi tietorakenteesta, jota käytetään seuraavan pisteen valintaan algoritmin suorituksen aikana. Tietorakenteen tehtävänä on ylläpitää tietoa algoritmin löytämistä pisteistä sekä niiden etäisyydestä alkupisteeseen. Alkuperäisessä artikkelissaan Dijkstra (1959) ei tee vertailuja tai analyysiä tietorakenteiden välillä. Hän käyttää tietorakenteen kuvauksena sanaa ”set”, minkä voidaan ajatella tarkoittavan yksinkertaisesti tietorakennetta, jossa samaa tietoa ei ole tallennettu rakenteeseen kahteen kertaan. Monet muut tutkijat ovat kuitenkin vuosien varrella tehneet tutkimusta reitinhakualgoritmeihin sopivista tietorakenteista sekä vertailleet niiden toimivuutta eri ympäristöissä. Esimerkkinä voitaisiin mainita, vaikka Cherkassky, Goldberg ja Radzik (1996), Fredman ja Tarjan (1984) sekä Zhan ja Noon (1998). Erilaisia tietorakenteita sekä niiden välisiä eroja käydään läpi myöhemmin tässä luvussa.

2.2 Tieaineiston erityispiirteet

Käyttöympäristö on keskeinen seikka tutkittaessa reitinhakua. Tämä tutkimus pohjautuu keskeisesti tieaineiston käyttöön ja niinpä aiempaa tutkimusta tarkasteltaessa tieaineistolla saavutetut tutkimustulokset ovat tärkeässä osassa.

On mahdollista yksinkertaistaa reitinhaun määritelmä niin, että oletetaan tieaineiston muodostavan graafin, joka koostuu pisteistä ja niitä yhdistävistä kaarista, kuten aiemmassa kappaleessa on mainittu. Reitinhakua graafin sisällä voidaan tehdä pisteparien tai useiden pisteiden välillä. Tyypillinen ongelmanasettelu on esimerkiksi löytää lyhin polku kahden pisteen välillä. Tällöin optimaalinen lyhin polku pisteiden välillä on pisteitä yhdistävien kaarien pienin mahdollinen yhteenlaskettu etäisyys (Sommer, 2014). Vaihtoehtoinen ongelmanasettelu voi olla lyhimmän reitin hakeminen lähtöpisteestä kaikkiin mahdollisiin muihin pisteisiin graafissa. Klassinen esimerkki algoritmista, jolla molemmat ongelmat voidaan ratkaista, on Dijkstran algoritmi

(Dijkstra, 1959), jonka toiminta on kuvattu lyhyesti aiemmassa kappaleessa. Tämä tutkimus keskittyy erityisesti tilanteeseen, jossa halutaan hakea nopein reitti kahden pisteen välillä.

Reaalimaailmassa tieaineisto eroaa jossain määrin teoreettisesta graafista. Perusrakenteen osalta voidaan ajatella, että tieverkoston risteyskohdat ovat pisteitä ja kaaret teitä. Tällöin kaarien painona voidaan käyttää käyttötarkoituksen mukaan esimerkiksi tieosuuden pituutta tai matka-aikaa. Koordinaattien avulla voidaan yhdistää toisiinsa liittyvät kaaret ja pisteet. Lisäksi tieaineisto voi sisältää metatietoja, joita reititysalgoritmit voivat toiminnassaan hyödyntää, mutta joka ei muuten ole välttämätöntä graafin muodostuksen kannalta. Tällaisia metatietoja voivat olla esimerkiksi tien toiminnallinen luokka tai tunnelin korkeusrajoitus. Bast ja muut (2016) kertovat artikkelissaan, että tieverkosto ei muodosta tyypillistä kaksiulotteista graafia (jossa pisteillä on eri koordinaatit). He mainitsevat esimerkkinä tunnelit ja ylikulkuväylät, jotka rikkovat tätä periaatetta. Intuitiivisesti saattaisi olettaa, että tieverkostossa kaikilla pisteillä on erilaiset koordinaatit, mutta esimerkiksi Digiroad-aineistossa (Digiroad 2019a) on mahdollista löytää useita pisteitä, jotka on sijoitettu samaan kaksiulotteiseen koordinaattiin. Lisäksi jotkin rakennukset sisältävät rakenteita, kuten esimerkiksi parkkihallien spiraalimaiset kerrokset, jotka poikkeavat ainoastaan korkeuden osalta. Tämä seikka on otettava huomioon reitinhakualgoritmien suunnittelussa ja toiminnassa.

Zhan ja Noon (1998) kertovat artikkelissaan, että oikeassa tieaineistossa pisteiden ja kaarien välinen suhde on usein rajallinen. He kertovat artikkelissaan, että joissain tutkimuksissa satunnaisesti luodussa graafissa on saattanut olla kymmenkertainen suhdeluku kaarien ja pisteiden välillä. Heidän tutkimusaineistoissaan oikeassa tieverkostossa kaarien ja pisteiden suhdeluku on ollut 2,66–3,28 kaarta jokaista pistettä kohden. Tutkimuksessaan Cherkassky ja muut (1996) ovat myös todenneet, että algoritmien suorituskyky vaihtelee erityyppisillä aineistoilla.

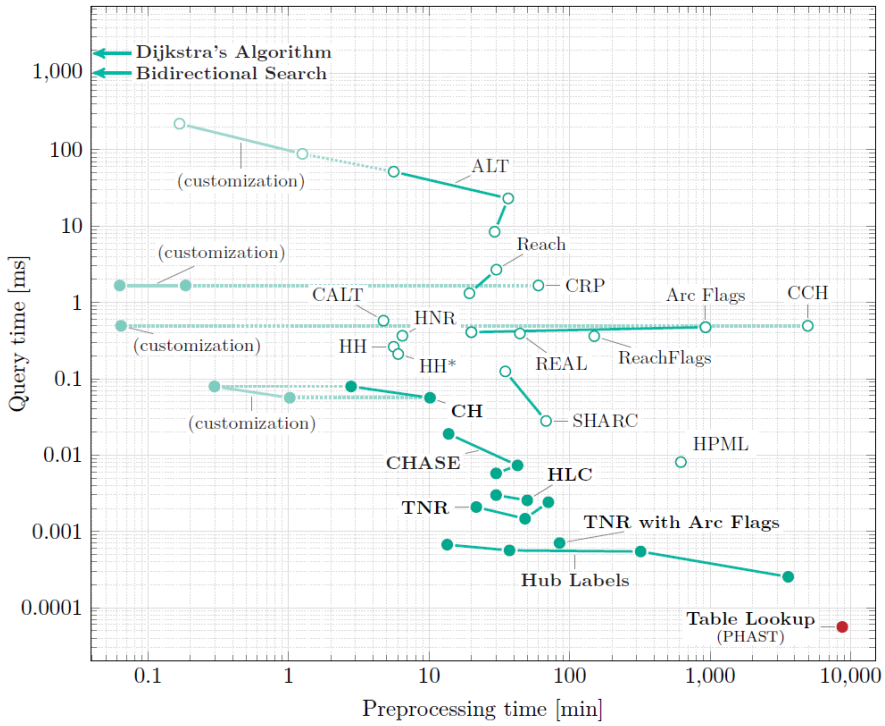
Muita keskeisiä eroja ovat esimerkiksi kääntymissäännöt, teiden yksisuuntaisuudet ja kulkurajoitukset kuten vaikkapa Sommer (2014) on huomannut vertaillessaan erilaisia ratkaisumalleja ongelmaan. Tällöin tieaineistolla toimivan algoritmin on pystyttävä ottamaan erilaiset tieliikenteen säännöt ja rajoitukset (esimerkiksi kevyen liikenteen väylä, painorajoitukset ja niin edelleen) huomioon omassa toiminnassaan. Lisäksi, jos halutaan rakentaa ohjelmistoa vaikkapa navigointikäyttöön, halutaan tyypillisesti myös huomioida matka-aika, jolloin kääntyminen tieosuudelta toiselle vaikuttaa matka-aikaan. Bast ja muut (2016) viittaavat artikkelissaan, että kääntymisaikojen huomioon ottaminen ja mahdollisuus päivittää nopeasti tieosuuksien kustannusarvoa (engl. cost function) ovat merkkejä tosielämän käyttöön sopivista reitinhakualgoritmista.

Artikkelissaan Bast ja muut (2016) kertovat, että riittävän pitkät reitinhaut tieaineistolla keskittyvät lopulta pieneen määrään keskeisiä tiesegmenttejä (esimerkkinä he antavat moottoritiet). Tutkijoiden mielestä on intuitiivista olettaa, että riittävän pitkässä reitinhaussa voidaan hakua rajoittaa niin, että reititysalgoritmi tutkii ainoastaan keskeisten tiesegmenttien välisiä polkuja. He huomauttavat kuitenkin, että näin toimittaessa ei voida matemaattisesti todistaa löydetyn reitin olevan juuri nopein mahdollinen reitti.

2.3 Reittiaineiston esikäsittely

Bastin ja muiden (2016) esittämän näkemyksen mukaan keskeinen ominaisuus, joka jakaa olemassa olevassa tutkimuksessa käsiteltyjä reititysalgoritmeja, on niiden kyky esikäsitellä graafin tietoa reitinhaun nopeuttamiseksi. Esikäsittely taas prosessina vaatii vaihtelevan määrän aikaa ennen varsinaista reitinhakua riippuen merkittävästi käytettävästä algoritmista. Kirjoittajien mukaan viimeaikainen tutkimus näyttää erityisesti keskittyneen erilaisten esikäsittelytapojen kehittämiseen ja nimenomaisesti reittikyselyiden nopeuttamiseen esikäsittelemällä graafia erinäisin tavoin ja tutkimus näyttää edelleen etenevän varsin nopealla tahdilla. Tämä tutkimussuunta on nähtävissä, kun verrataan aiempia kirjallisuuskatsauksia (Fu ja muut, 2006), joissa ei ole painotettu esikäsittelymenetelmiä uudempiin tutkimuksiin, joissa esikäsittely on keskeinen tutkimusalue (Bast ja muut, 2016; Sommer, 2014).

Artikkelin Bast ja muut (2016) mukaan uusien esikäsittelyä käyttävien reititysalgoritmien avulla voidaan hakea reittejä tieaineistolla mantereiden kokoisten alueiden sisällä jopa alle millisekunnin ajassa. He huomauttavat kuitenkin, että reititysongelma itsessään on vielä ratkaisematon siinä mielessä, että ei ole olemassa optimaalista ratkaisua kaikkiin käyttökohteisiin ja valittaessa sopivaa algoritmia on otettava huomioon eri vaihtoehtojen hyvät ja huonot puolet. Tällaisia valintoja ovat esimerkiksi hakuaika, esikäsittelyn vaatima aika, reittiverkon vaatima tilan käyttö ja algoritmin sopeutuvuus muutoksiin. Kirjoittajien mukaan reitin laatu on muuttumassa epäolennaiseksi kriteeriksi lähinnä sen takia, että yleisesti ottaen modernit algoritmit pystyvät tuottamaan todistetusti optimaalisia tuloksia. Bast ja muut (2016) esittävät kaavion avulla (kuva 1) vaikutuksia miten erilaiset esikäsittelyä hyödyntävät algoritmit toimivat suhteessa hakuaikaan ja tarvittavaan esikäsittelyaikaan.



Kuva 1. Reitinhaun vaatima aika suhteessa aineiston esikäsittelyn vaatimaan aikaan. Lähde: Bast ja muut (2016). Kuvaa käytetty julkaisijan Springer Nature luvalla[1].

[1] https://www.springernature.com/

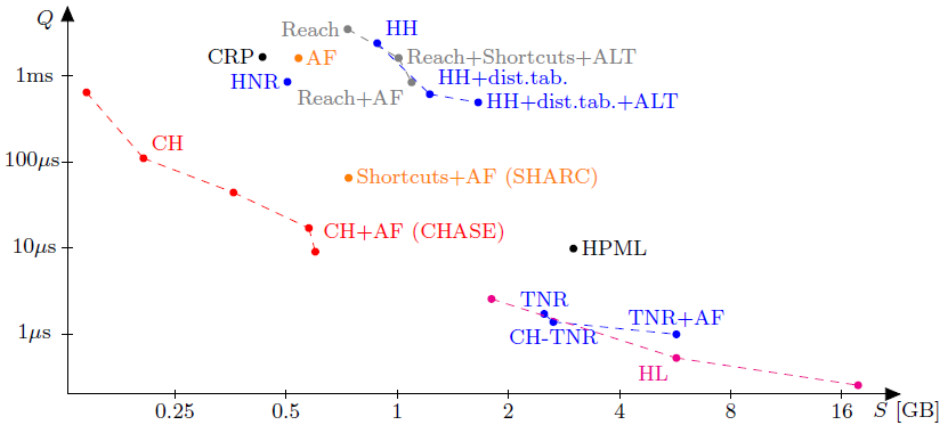
Kuvassa 1 esitettyjen tulosten perusteella havaitaan, että yleisesti mitä enemmän aikaa esikäsittelyyn käytetään, sitä nopeammin reittikyselyihin voidaan vastata. Bast ja muut (2016) huomauttavat kuitenkin, että kaikki algoritmit eivät ole suoraan verrannollisia toisiinsa. Lisäksi tutkijoiden olettamuksena esikäsittelyajoista on mainittu, että kaikki data mahtuu tietokoneen muistiin sekä esikäsittelyyn on saatavilla noin kertaluokkaa enemmän laskentatehoa kuin kyselyiden tekemiseen – he kutsuvat tätä olettamusta palvelinmalliksi. Lisäksi Bast ja muut (2016) mainitsevat erityisesti julkisen liikenteen reititysalgoritmit (bussit, raitiovaunut ja niin edelleen), joissa esikäsittelytekniikat eivät ole läheskään yhtä tehokkaita ja välttämättä täydellisiin reititystuloksiin ei aina päästä, koska algoritmien on otettava huomioon kellonaikakomponentti, vaikkakin algoritmien kehitys tällä alueella on ollut nopeaa.

Erilaisissa käyttöympäristöissä olevan rajallisen tallennuskapasiteetin kysymystä ovat tutkineet esimerkiksi Goldberg ja Werneck (2005) kehittämällä klassisen A* -reitinhakualgoritmin toimintaa niin, että se sisältää esikäsittelytoiminnallisuuksia, mutta käyttäen mobiililaitetta toimintaympäristönä. Heidän mallissaan esikäsitelty tieverkosto tallennetaan mobiililaitteen pysyvään muistiin, jolloin käyttömuistin (RAM) tarve pienenee merkittävästi. Kirjoittajien tulosten perusteella esikäsittelyn käyttäminen on mahdollista myös rajatun kapasiteetin laitteilla ainakin siinä tapauksessa, jos käytettävä algoritmi kehitetään tähän tarkoitukseen.

Toista lähestymistapaa muistikapasiteetin rajallisuuteen ovat tutkineet Efentakis, Pfoser ja Voisard (2011). He ovat kehittäneet menetelmän, jossa reitinhakualgoritmin mukana on erillinen komponentti, joka hallitsee datan lataamista massamuistista algoritmin suorituksen aikana (artikkelissaan tutkijat ovat hyödyntäneet perinteistä relaatiotietokantaa massamuistina). Tällöin erillistä aineiston esikäsittelyä ei vaadita ja hyvänä puolena reittiaineistoa voidaan päivittää taustalla reitinhakujen välissä käyttämättä laskenta-aikaa esikäsittelytietojen päivittämiseen ja tallentamiseen massamuistiin. Lähestymistavan haittapuolena on toisaalta se, että ensimmäiset reittihaut (tai reittihaut tieaineistoon, jotka eivät ole vielä muistissa) ovat hitaampia. Lähestymistapa siis soveltuu paremmin tilanteisiin, jossa suoritetaan paljon samankaltaisia reittihakuja.

Vertailun erilaisista reitinhakualgoritmeista on tehnyt Sommer (2014) tutkimalla artikkelissaan reittien hakuajan ja esikäsittelyn vaatiman tallennustilan välistä suhdetta. Kirjoittaja on selvittänyt, perustuen aiempaan tutkimukseen, miten esikäsittelyä hyödyntävät algoritmit käyttäytyvät suhteessa kyselyiden hakuaikaan ja esikäsittelytiedon vaatimaan tallennustilaan. Sommer (2014) on esittänyt artikkelissaan kaavion, joka on tarkasteltavissa kuvassa 2. Kuvassa käytetään logaritmista asteikkoa molemmilla akseleilla. Tulosten perusteella voidaan todeta, että esikäsittelyn vaatiman tallennustilan määrä nousee silloin, kun käytetään reittihakuihin nopeammin vastaavia algoritmeja.

Perustuen tutkijoiden Sommer (2014) sekä Bast ja muut (2016) raportoimiin tuloksiin reitinhakualgoritmit muodostavat esikäsittelyn osalta kompromissin, joka on määriteltävissä suhteina hakuajan, esikäsittelyyn vaadittavan ajan ja tallennustilan välillä. Lisäksi Sommer (2014) mainitsee artikkelissaan, että reitin laatu (engl. worst case accuracy) on myös yksi kompromissiin vaikuttava tekijä, jos reitinhakualgoritmi ei palauta todistetusti lyhintä reittiä. Bastin ja muiden (2016) tutkimuksen (kuva 1) perusteella eräs suhteellisen vähän esikäsittelyä hyödyntävä algoritmi on ALT (Goldberg & Harrelson, 2005) ja sen variaatiot. Tällöin ALT -algoritmin suorituskyky antaa viitteen siitä millaisia tuloksia ilman esikäsittelyä toimiva algoritmi voi saada suhteessa nopeaan esikäsittelyaikaan.



Kuva 2. Reitinhaun vaatima aika suhteessa aineiston esikäsittelyn vaatimaan tallennustilaan. Tieto on koostettu eri lähteistä ja yhdistetty samaan kaavioon. Lähde: Sommer (2014). Kuvaa käytetty julkaisijan Association for Computing Machinery (ACM) luvalla[2].

Artikkelissaan Goldberg ja Harrelson (2005) kuvaavat alkuperäiseen ALT -algoritmin, joka pohjautuu A* -algoritmiin (Hart ja muut, 1968), mutta johon on lisätty maamerkkeihin ja kolmiogeometriaan pohjautuvia ominaisuuksia, joiden avulla reittien hakua voidaan nopeuttaa. ALT on selkeästi esikäsittelyä hyödyntävä algoritmi, vaikka alkuperäinen A* -algoritmi itsessään ei esikäsittelyä hyödynnä. Keskeinen ominaisuus ALT -algoritmissa on maamerkkien (engl. landmark) valinta ja hyödyntäminen osana reitinhakua. Kartta-aineistosta valitaan tietty suhteellisen pieni määrä maamerkkejä ja kaikkien maamerkkien välille lasketaan optimaaliset lyhimmät reitit. Näitä reittejä hyödynnetään myöhemmin varsinaisessa reitinhaussa niin, että reitinhakualgoritmin läpikäymien pisteiden määrä vähenee merkittävästi ja seurauksena algoritmin suoritus nopeutuu. Artikkelissaan tutkijat mainitsevat lisäksi rajoittaneensa vertailussaan maamerkkien määrää, koska esikäsittelyn vaatima muistimäärä oli testilaitteistolla rajoittava tekijä.

Goldbergin ja Harrelsonin (2005) esittämien tulosten perusteella ALT -algoritmi suoriutuu erikokoisien tieaineistojen reitityksessä noin 15–30 kertaa nopeammin kuin perinteinen A* -algoritmi ja noin 6–16 kertaa nopeammin, kuin perinteinen Dijkstran algoritmi. Lisäksi tutkijat mainitsevat suorituskyvystä yleisesti artikkelinsa johdannossa niin, että hyvin toteutettu ALT -algoritmi on yhden tai useamman kertaluokkaa nopeampi kuin kaksisuuntainen Dijkstran -algoritmi. Ero suoritusnopeudessa kasvaa aineiston koon kasvaessa. Eräänä yksityiskohtana tutkijoiden raportoimien tulosten perusteella Dijkstran algoritmi suoriutui nopeammin kuin A* -algoritmi. Tutkijoiden artikkelissaan käyttämä aineisto on kuitenkin suurimmassakin kokeessa noin puolet Digiroad-aineiston koosta ja kuvaa maantieteellisesti huomattavasti suurempaa aluetta, jolloin tulokset eivät välttämättä ole täysin vertailukelpoisia.

Jatkokehitystyötä ALT -algoritmiin ovat suorittaneet Efentakis ja Pfoser (2013). Heidän tutkimuksensa keskeisenä kohteena on esikäsittelyajan parantaminen sekä algoritmin suoritusajan nopeuttaminen. He kertovat havaintona, että esikäsittelyvaihe on mahdollista suorittaa yhtäaikaisesti usealla prosessoriytimellä, koska esikäsittelyssä suoritetaan useita Dijkstran -algoritmin hakuja yhtäaikaisesti staattiselle aineistolle. Hyödyntämällä yhtäaikaista suoritusta ja muita optimointimenetelmiä, kuten

[2] Julkaisulupa hankittu organisaation Copyright Clearance Center, Inc kautta

esimerkiksi jatkokehitettyä maamerkkien valinta-algoritmia, tutkijat pystyivät vähentämään esikäsittelyajan noin 8–60 sekuntiin käyttäen 18 miljoonan pisteen aineistoa riippuen valittujen maamerkkien määrästä. Esikäsittelyn vaatima aika on saatu tutkijoiden kirjoitushetkellä käytettävissä olevalla tietokonelaitteistolla ja perustuen artikkelissa mainittuun kuvaukseen on luontevaa olettaa, että eri käyttöympäristössä tai erilaisella laitteistolla esikäsittelyn vaatima aika voi olla poikkeava. Tutkijoiden käyttämä aineisto on samaa kokoluokkaa kuin Digiroad-aineisto, mutta kattaa suuremman maantieteellisen alueen.

2.4 Reitinhaun optimointimenetelmät

Tässä kappaleessa käsitellään erilaisia tutkimustietokannoista löytyviä menetelmiä heuristisen reitinhaun optimoimiseksi, joilla pyritään nopeuttamaan algoritmien toimintaa. Heuristisilla menetelmillä tarkoitetaan menetelmiä, joiden avulla reitinhakualgoritmi pyrkii ohjautumaan tavalla tai toisella kohti tavoitettaan sen sijaan, että se tutkisi hakualuetta satunnaisesti, kunnes optimaalinen reitti löytyy. Tämä kappale keskittyy menetelmiin, jotka eivät vaadi tieaineiston esikäsittelyn käyttöä toiminnassaan. Esikäsittelyä hyödyntäviä menetelmiä on käsitelty edellisessä kappaleessa.

Optimointimenetelmät on kategorisoitu artikkelin Fu ja muut (2006) mukaisesti seuraavalla tavalla. Hakualuetta rajaavat menetelmät pyrkivät pienentämään ratkaistavaa ongelmaa algoritmien keinoin niin, että hakualueesta jätetään käsittelemättä pisteitä, jotka todennäköisesti eivät ole optimaalisella reitillä. Tällaisia menetelmiä ovat Branch Pruning ja A* -algoritmi. Hakuongelmaa pienentävät menetelmät taas pyrkivät jakamaan reitinhaun pienempiin osiin, jolloin reitinhaku nopeutuu. Tarkoituksena näissä menetelmissä on nopeuttaa ongelmanratkaisua jakamalla se pienempiin osiin. Näistä menetelmistä käsitellään reitinhakua kahdesta suunnasta ja välitavoitteiden käyttöä. Seuraavaksi käsitellään menetelmiä, jotka pyrkivät vähentämään läpikäytyjen pisteiden määrää. Keskeisenä menetelmänä on hierarkian hyödyntäminen osana reititysalgoritmin toimintaa. Lisäksi käsitellään, tutkimustietoon pohjautuen, mitä menetelmiä voidaan yhdistää toisiinsa ja saavuttaa sitä kautta tehokkuushyötyä.

2.4.1 Hakualuetta rajaavat menetelmät

Tässä kappaleessa kuvataan menetelmiä, joiden avulla voidaan optimoida reitinhakualgoritmin toimintaa rajaamalla hakualuetta pienemmäksi. Fu ja muut (2006) kertovat artikkelissaan, että algoritmit, jotka eivät hyödynnä tietoa tutkittavasta graafista joutuvat tutkimaan kaikki mahdolliset välipisteet (myös väärään suuntaan johtavat polut), ennen optimaalisen reitin löytymistä. Täten heidän mukaansa hakualetta rajaavien menetelmien tarkoituksena on hyödyntää informaatiota reitinhaun alku- ja loppupisteestä niin, että hakualuetta voidaan rajata pienemmäksi. Tutkijat mainitsevat keskeisiksi menetelmiksi Branch Pruning ja A* -algoritmin. Branch Pruning -menetelmä pyrkii rajaamaan hakualuetta pienemmäksi, kun taas A* -algoritmi pyrkii ohjaamaan reitinhakua heuristisen funktion avulla kohti loppupistettä.

Branch Pruning

Artikkelissaan Fu ja muut (2006) kuvaavat Branch Pruning -menetelmän toimintaa viitaten artikkeleihin Karimi (1996) ja Fu (1996). Menetelmän tarkoituksena on rajata hakualuetta funktion avulla, joka määrittää mitkä pisteet eivät todennäköisesti ole optimaalisella reitillä. He kirjoittavat esimerkkinä matka-ajan käyttämistä Branch Pruning funktiona arvioiden minimi ja maksimi matka-ajat. Tällöin jos

reitinhakualgoritmi huomaa tietyn pisteen olevan kauempana kohteesta kuin arvioitu maksimi matka-aika, voidaan päätellä, että piste ei kuulu optimaaliselle reitille ja se voidaan jättää huomiotta.

Fu ja muut (2006) kertovat, että Branch Pruning funktio luo eräällä tavalla ellipsin, joka kattaa alkupisteen, loppupisteen sekä jonkin verran ylimääräistä aluetta graafissa. Kirjoittajat kertovat Branch Pruning funktion haittapuolena olevan, että optimaalista reittiä ei välttämättä löydetä, jos sen muuttujien arvioinnissa tehdään virhe. Tämän haittapuolen potentiaaliseksi ratkaisuksi he tarjoavat funktion parametrien muuttamista algoritmin toiminnan aikana, jos algoritmi ei muuten pysty löytämään reittiä loppupisteeseen. Viitaten artikkeliin Fu (1996) tutkijat arvioivat Branch Pruning -menetelmän parantavan laskenta-aikaa noin 40–60 % verrattuna perinteiseen Dijkstran algoritmiin.

R. T. Honkanen (artikkelin luonnos vastaanotettu sähköpostitse, 2011) on kertonut ryhmänsä V. Pietikäinen, R. T. Honkanen ja J. Mielikäinen alustavista koetuloksista Branch Pruning -menetelmän toimivuudesta Digiroad -tieaineistolla. Hänen ilmoittamiensa tulosten perusteella Dijkstran algoritmin suorituksen aikana läpikäymien pisteiden määrä noin puolittuu menetelmää hyödyntämällä. Ryhmä toteutti Branch Pruning -menetelmän dynaamisesti niin, että hakualuetta laajennetaan askelittain, jos reittiä kohteeseen ei muuten löydy. Honkanen (2011) on julkaissut myös muuta reititysalgoritmeihin liittyvää tutkimusta.

A -algoritmi*

A* -algoritmi on klassinen reitinhakualgoritmi, joka löytää lyhimmät polun kahden pisteen välillä hyödyntäen heuristiikkaa, joka ohjaa algoritmin toimintaa kohti loppupistettä. Algoritmin kuvasivat artikkelissaan Hart, Nilsson ja Raphael (1968) ja se on suosittu ratkaisu lyhimmän polun löytämiseen ilman esikäsittelyä (Fu ja muut, 2006). A* -algoritmia ovat jatkokehittäneet muun muassa Goldberg ja Harrelson (2005) osana aiemmin kuvattua ALT -algoritmia.

Keskeinen ominaisuus A* -algoritmissa on sen sisältämä heuristiikkafunktio, joka pyrkii arvioimaan valittavia pisteitä algoritmin suorituksen aikana niin, että käsittelyssä suositaan pisteitä, jotka vievät algoritmia kohti loppupistettä. Toisin sanoen siis algoritmi ottaa huomioon etäisyyden lähtöpisteeseen sekä potentiaalifunktion, joka mittaa arvioitua etäisyyttä loppupisteeseen. Tällöin keskeistä algoritmin toiminnalle on sen sisältämän heuristiikkafunktion laatu, joka vaikuttaa merkittäviltä osin siihen, miten hyvin algoritmi suoriutuu (Bast, 2009; Fu ja muut, 2006). A* -algoritmi hakee todistetusti lyhimmän reitin niin kauan kuin heuristiikkafunktio aliarvioi tai tietää oikean etäisyyden loppupisteeseen (Hart ja muut, 1968). Tästä seurauksena esimerkiksi Bast (2009) mainitsee, että yksinkertaisena tapana hyödyntää heuristiikkaa tieaineistolla on linnuntiestä koostuvaa heuristiikka, koska tällöin yliarviointia ei pääse tapahtumaan ja toisaalta heuristiikan aliarviointi ei ole kovin suurta. Kahden koordinaatin välinen etäisyys on teknisesti yksinkertainen laskea. Lisäksi Bast (2009) mainitsee, että A* -algoritmi käyttäytyy kuten Dijkstran algoritmi silloin kuin heuristiikkafunktion arvo on nolla. Vastaavasti hän mainitsee, että teoreettisesti paras mahdollinen heuristiikka on sellainen, joka tietää tarkasti oikean etäisyyden kohteeseen, jolloin A* -algoritmi hakee optimaalisen reitin tutkimatta muita pisteitä, mutta hänen mukaansa tämä ei ole käytännössä mahdollista.

Bast ja muut (2016) toteavat artikkelissaan, että A* -algoritmin toimintaa on mahdollista nopeuttaa muokkaamalla heuristiikkafunktiota aggressiivisemmaksi, mutta tämä voi seurauksena tehdä löydetyistä reiteistä epäoptimaalisia. Sanders ja Schultes (2007) ovat todenneet, että tätä lähestymistapaa on usein sovellettu käytännön sovelluksissa. Jacob, Marathe ja Nagel (1999) ovat tutkineet A* -algoritmin heuristiikkafunktion toimintaa lisäämällä yliarviointikertoimen maantieteellisen etäisyyden lisäksi. He ovat havainneet, että näin voidaan saavuttaa merkittäviä parannuksia algoritmin suoritusnopeudessa niin, että käytännön reitit ovat kuitenkin lähellä optimaalista. Bast ja muut (2006) kertovat, että toinen mahdollinen optimointitapa on muokata A* -algoritmia niin, että se toimii kaksisuuntaisesti. Lisäksi he kertovat, että optimoitunakin A* -algoritmi, jossa hyödynnetään maantieteellisiä etäisyyksiä heuristiikkana, toimii hitaasti verrattuna muihin moderneihin ratkaisuihin.

Suorituskyvyn osalta Bast (2009) antaa artikkelissaan viitteen, että linnuntietä käytettäessä A* -algoritmin heuristiikka parantaa reitinhaun suorituskykyä tieaineistolla noin 2 tai 3 kertaisesti oletettavasti verrattuna Dijkstran -algoritmiin. Jacob ja muut (1999) kertovat artikkelissaan, että heidän koetulostensa perusteella A* -algoritmi suoriutuu noin 2 kertaa nopeammin kuin Dijkstran algoritmi. R. T. Honkanen (artikkelin luonnos, 2011) on kertonut ryhmänsä tuloksista, että A* -algoritmi käsittelee ainoastaan noin kolmasosan pisteitä tieaineistolla verrattuna perinteiseen Dijkstran algoritmiin ja noin 40 % vähemmän kuin Branch Pruning -menetelmä. Sommer (2014) kertoo A* -algoritmin olevan helppo toteuttaa ja tarjoavan ”kohtuullisen” nopeutuksen reitinhaussa. Liu (1997) vertaili artikkelissaan Dijkstran algoritmia A* -algoritmiin ja suurella kaupunkiaineistolla A* -algoritmi suoriutui reitinhausta noin puolessa ajassa Dijkstran algoritmiin verrattuna. Goldberg ja Harrelson (2005) raportoivat artikkelinsa tuloksissa A* -algoritmin toimivat noin puolet hitaammin kuin Dijkstran algoritmi.

2.4.2 Hakuongelmaa pienentävät menetelmät

Artikkelissaan Fu ja muut (2006) kategorisoivat hakuongelmaa pienentäviä menetelmiä. Heidän mukaansa on hyvin tunnettua, että yleisen hakuongelman ratkaisun vaatima laskenta-aika kasvaa suhteessa nopeammin kuin itse ongelman koko. He esittävät, että jakamalla ratkaistava ongelma pienempiin osiin voidaan ratkaisun vaatimaa laskenta-aikaa merkittäviltä osin vähentää.

Tässä kappaleessa käsitellään tutkimuksesta löytyviä menetelmiä hakuongelman pienentämiseen. Fu ja muut (2006) kertovat, että kaksisuuntainen haku pyrkii muuttamaan algoritmien toimintaa niin, että lyhintä polkua haetaan lähtöpisteestä loppupisteeseen ja toisin päin samaan aikaan. Välitavoitteiden käyttö taas pyrkii jakamaan reitin haun pienempiin osiin reitinhaun nopeuttamiseksi. Molemmista menetelmistä käydään läpi tutkimustietoon pohjautuvia mittaustuloksia.

Kaksisuuntainen haku

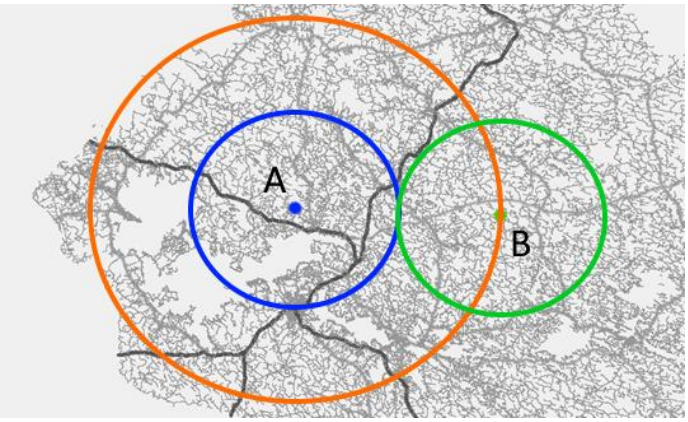
Kaksisuuntaisen haun tarkoituksena on optimoida reitinhakualgoritmin toimintaa hakemalla lyhintä reittiä lähteestä kohteeseen ja kohteesta lähteeseen yhtäaikaisesti. Sommer (2014) kertoo artikkelissaan kaksisuuntaisen haun antavan potentiaalisesti parempia tuloksia sekä mainitsee erityisesti, että kaksisuuntainen haku tekniikkana on sovellettavissa myös tieaineiston ulkopuolisiin käyttökohteisiin. Artikkelissaan Bast (2009) toteaa, että kaksisuuntainen haku itsessään ei tuo merkittävää parannusta reitinhaun suorituskykyyn, mutta se on tärkeä komponentti yhdisteltäessä sitä muihin nopeutusmenetelmiin. Samaa mieltä ovat myös Sanders ja Schultes (2007) ja he

kertovat tutkimuksessaan, että kaksisuuntainen haku on keskeinen osa luodessa tehokkaita hakutekniikoita.

Fu ja muut (2006) kuvaavat artikkelissaan, että keskeiset kysymykset kaksisuuntaisessa haussa ovat kuinka nämä kaksi hakuprosessia vuorottelevat ja toisena kysymyksenä on tieto siitä, milloin lyhin polku on löytynyt ja hakuprosessi voidaan lopettaa. Ensimmäisenä kaksisuuntaista hakua yhdistettynä lyhimmän polun hakuun graafista ehdotti artikkelissaan Dantzig (1960). Tätä ajatusta on myöhemmin hyödynnetty erityisesti Dijkstran algoritmin yhteydessä, mutta Goldberg ja Harrelson (2005) ja Sommer (2014) mukaan myös A* -algoritmia on mahdollista käyttää kaksisuuntaisesti viitaten alun perin Pohl (1971) kirjoittamaan artikkeliin aiheesta. Kaksisuuntaista A* -algoritmia ovat kehittäneet myös Ikeda ja muut (1994).

Kuvauksen kaksisuuntaisen haun toiminnasta Dijkstran algoritmin yhteydessä antaa artikkelissaan Bast (2009). Hänen mukaansa ajatuksena on yhtäaikaisesti hakea reittiä lähteestä kohteeseen ja takaisin kohteesta lähteeseen, kunnes nämä kaksi hakurintamaa kohtaavat toisensa. Tällöin hakuprosessi ylläpitää kahta tietorakennetta – toinen lähtöpisteestä alkavalle haulle ja toinen loppupisteestä alkavalle haulle. Jokaisella hakukierroksella laajennetaan graafista piste, jonka etäisyys on pienin ottaen kuitenkin huomioon etäisyydet molemmista hakuprosesseista. Tämä onnistuu vertaamalla molempien tietorakenteiden juuriarvoja ja valitsemalla käsittelyyn se kumman avain on pienempi. Prosessia jatketaan, kunnes yhdessä tietorakenteessa käsitelty piste saavutetaan myös toisessa tietorakenteessa (jolloin reitti koostuu tällöin molempien puolien summasta). Reitin optimaalisuus voidaan varmistaa vertaamalla löydettyjen polkujen kokonaispituuksia niin, että jatketaan hakua, kunnes löydetään reitti, jonka kokonaispituus on suurempi kuin edellinen löydetty reitti, jolloin tiedetään, että tämä edellinen reitti on todellakin optimaalinen.

Bast (2009) kuvaa artikkelissaan, miten tämä johtaa kokonaisuutena pienempään käsiteltyjen pisteiden määrään (esimerkki konseptista on kuvassa 3). Hän kuvaa Dijkstran algoritmin tutkimaa aluetta ympyränmuotoisena laajenevana alueena (jolla on säde r), jolloin kaksisuuntainen haku laajentaa kaksi pienempää aluetta (joilla molemmilla on säde r/2) ja käsiteltyjen kokonaispisteiden määrä ja hakuaika näin ollen puolittuvat. Sanders ja Schultes (2007) toteavat myös artikkelissaan kaksisuuntaisen Dijkstran algoritmin johtavan noin kaksinkertaiseen nopeutukseen perinteiseen Dijkstran algoritmiin verrattuna.



Kuva 3. Havainnekuva Dijkstran algoritmin laajentamasta hakualueesta Kainuun tieaineistolla, jossa haetaan reittiä pisteestä A pisteeseen B. Oranssi viiva kuvaa arvioitua laajennettua hakualuetta käytettäessä perinteistä Dijkstran algoritmia ja sininen ja vihreä ympyrä kuvaavat arvioitua laajennettua hakualuetta, kun käytössä on kaksisuuntainen Dijkstran algoritmi.

Sommer (2014), Goldberg ja Harrelson (2005) sekä Fu ja muut (2006) kuvaavat artikkeleissaan kaksisuuntaisen A* -algoritmin toteutuksen haasteita. He kertovat, että kaksisuuntainen A* -algoritmi on mahdollista toteuttaa, mutta toteutus on osaltaan monimutkainen ja ratkaisua voidaan lähestyä kahdella tavalla. Algoritmin päättymisehtoa voidaan muokata sopivalla tavalla tai vaihtoehtoisesti on käytettävä yhtenevää heuristiikkafunktiota (engl. consistent potential function) kaikille pisteille hakusuunnasta riippumatta. Yhtenevän heuristiikkafunktion etuna on, että kun hakurintamat kohtaavat voidaan olla varmoja siitä, että löydetty reitti on optimaalinen, toisin kuin päättymisehtoa muokkaavassa ratkaisutavassa. Vastaavasti taas haasteena on tällöin löytää optimaalinen heuristiikkafunktio, joka ei lisää hakuaikaa liian paljon.

Ensimmäinen lähestymistapa on muokata algoritmin lopetusehtoa, jonka osalta Sommer (2014) viittaa artikkeleihin Pohl (1971) ja Kwa (1989). Fu ja muut (2006) kertovat, että Pohlin (1971) mukaan heuristisina funktiona molemmissa hakusuunnissa olisi käytettävä alku- ja loppupisteitä, jolloin vältytään tilanteelta, että hakurintamat ohittaisivat toisensa kohtaamatta. Artikkelissaan Fu (1996) esittää, että A* -algoritmien suoritus voidaan lopettaa silloin, kun molemmat puolet ovat kohdanneet toisensa M kertaa, jolloin laskenta-ajan ja laadun välistä suhdetta voidaan ohjata muokkaamalla arvoa M.

Toisena vaihtoehtona artikkelissaan Ikeda ja muut (1994) ovat tutkineet A* -algoritmin kaksisuuntaista hakua ja verranneet kokeellisia tuloksia Dijkstran algoritmiin. He kehittivät A* -algoritmin heuristiikkaa niin, että heuristiikkafunktio on molempiin suuntaan yhtenevä. Ratkaisumallissa heuristiikkafunktio ottaa 50 % painoarvoilla kantaa käsiteltävän pisteen etäisyyteen sekä alku- ja loppupisteeseen. Tällöin algoritmin suoritusta ohjataan pois pisteistä, jotka ovat kaukana alku- ja loppupisteestä, mutta vastaavasti kirjoittajien mukaan heuristiikkafunktio on vähemmän optimaalinen kuin perinteisessä yhteen suuntaan hakevassa A* -algoritmissa. Ikeda ja muut (1994) toteavat, että tämä ratkaisumalli aiheuttaa enemmän tarvetta laskennalle algoritmin suorituksen aikana, mutta vastaavasti se vähentää reitin haun aikana algoritmin tutkimien pisteiden määrää. Yhtenevää heuristiikkafunktiota ovat jatkokehittäneet myös Goldberg ja Harrelson (2005) artikkelissaan, mutta he kertovat Ikedan ja muiden (1994) kehittämän funktion olevan hieman tehokkaampi testiaineistoilla.

Ikeda ja muut (1994) ovat tutkineet artikkelissaan eri tapoja kaksisuuntaisen hakualgoritmin toteutukseen vertailemalla hyödyntäen Japanin tieaineistoa ja Tokion lähialueita. Heidän tulostensa mukaan kaksisuuntainen Dijkstran algoritmi tutkii ainoastaan kolmasosan perinteisen Dijkstran algoritmin läpikäymistä pisteistä. Heidän toteuttamansa kaksisuuntainen A* -algoritmi vastaavasti tutkii ainoastaan 60 % kaksisuuntaisen Dijkstran algoritmin läpikäymistä pisteistä. Artikkelissaan Sanders ja Schultes (2007) kommentoivat, että kun A* -algoritmi toteutetaan kaksisuuntaisesti se johtaa noin 2–3 kertaa nopeampaan algoritmin suoritukseen verrattuna perinteiseen Dijkstran algoritmiin.

Goldberg ja Harrelson (2005) ovat tutkineet artikkelissaan eri variaatioita reititysalgoritmeista hyödyntäen tieaineistoa. He toteavat, että verrattuna perusmuodossaan Dijkstran algoritmiin, A* -algoritmi (käyttäen linnuntietä heuristiikkana) ei tuottanut hyötyä algoritmin ajonopeudessa. Heidän mukaansa tämä havainto ulottuu myös tilanteeseen, jossa näiden algoritmien kaksisuuntaisia versioita verrataan keskenään.

Välitavoitteet

Katsauksessaan Fu ja muut (2006) kertovat, että haettaessa lyhintä reittiä tieaineistolla välitavoite (engl. subgoal) voi olla mikä tahansa piste tai kaari, joka sijaitsee alkupisteen ja loppupisteen välisellä optimaalisella reitillä. Tällöin lyhimmän polun löytäminen voi tapahtua hakemalla erilliset reitit alkupisteestä välitavoitteeseen, loppupisteestä välitavoitteeseen sekä yhdistämällä nämä löydetyt reitit. Kirjoittajien mukaan välitavoitteista saavutettava tehokkuushyöty riippuu välitavoitteiden määrästä ja sijainnista graafissa. Tällöin mitä enemmän välitavoitteita käytetään ja mitä tasaisemmin ne sijaitsevat alku- ja loppupisteen välillä, sitä suurempi tehokkuusetu saavutetaan. Ratkaisun haittapuolena Fu ja muut (2006) mainitsevat, että välitavoitteita käytettäessä löydetty reitti ei välttämättä ole optimaalinen ja menetelmän käyttö riippuu siitä, onko tarvittavia välitavoitepisteitä käytettävissä. R. T. Honkanen (artikkelin luonnos, 2011) on todennut, että välitavoitteiden käyttö vaatii käytännössä esikäsittelyä, jotta välitavoitepisteet voidaan tunnistaa ennen varsinaisen reitinhaun aloittamista.

Välitavoitteiden käyttöä verraten A* -algoritmiin ovat tutkineet artikkelissaan Dillenburg ja Nelson (1995). Kokeellisessa ympäristössään he olettavat, että mahdolliset välitavoitteet ovat tiedossa etukäteen. Koejärjestelyssä tutkijat simuloivat kartta-aineistolla etukäteen erilaisia reittejä, joista he pystyivät valitsemaan useimmin käytetyt pisteet välitavoitteiksi. Tutkijat toteavat tuloksistaan, että haun tehokkuutta voidaan jonkin verran parantaa välitavoitteilla verrattuna perinteiseen A* -algoritmiin. Tutkijoiden tulostaulukon perusteella käsiteltyjen pisteiden määrä laski noin 20 % ja prosessoriajan tarve pieneni noin 30 %.

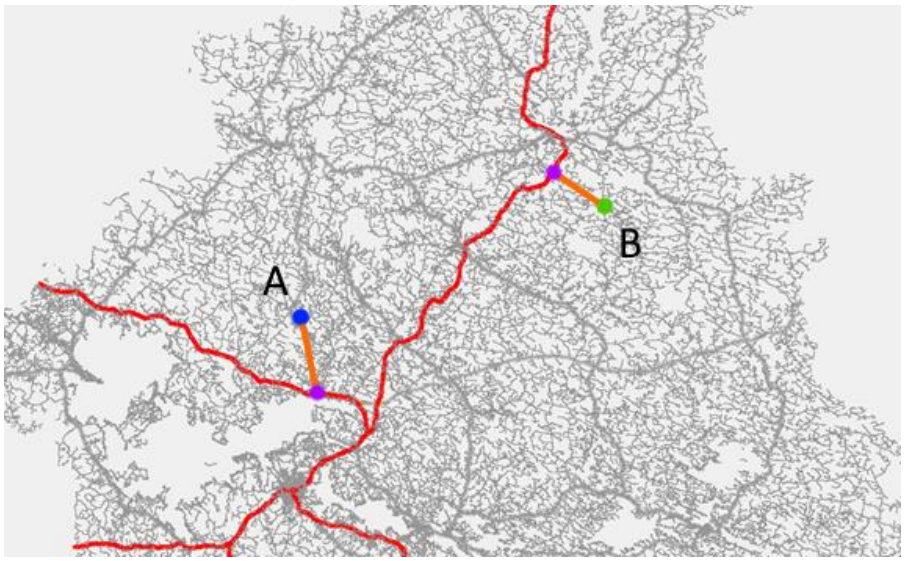
R. T. Honkanen (artikkelin luonnos, 2011) on testannut ryhmänsä kanssa välitavoitteiden käyttöä reitityksessä tieaineistolla. He sovelsivat työssään kaupunkien rajoilla olevia pisteitä välitavoitteina, jolloin saavutettu hyöty toteutui ainoastaan pidemmillä reititysmatkoilla. Kirjoittajien raportoimien tulosten perusteella välitavoitteiden käyttö yhdistettynä A* -algoritmiin vähensi algoritmin käsittelemien pisteiden määrää noin 10 kertaa pienemmäksi verrattuna perinteiseen A* -algoritmiin.

2.4.3 Käsiteltävien pisteiden vähentämiseen pohjautuvat menetelmät

Katsauksessaan Fu ja muut (2006) kertovat, että perinteisessä reitinhakualgoritmissa, kun piste valitaan käsittelyyn myös kaikki siihen kaarin liitetyt pisteet otettaan käsittelyyn riippumatta siitä, miten todennäköisesti ne ovat optimaalisella reitillä. Heidän mukaansa käsiteltävien pisteiden vähentämiseen pohjautuvat menetelmät systemaattisesti jättävät pisteitä käsittelemättä, jos niiden todennäköisyys olla optimaalisella reitillä on pieni tai niillä ei ole käytännön merkitystä reitinhaun kannalta. Fu ja muut (2006) kertovat, että tätä ideaa voidaan toteuttaa hyödyntämällä hierarkkista hakua, jonka perusajatuksena on keskittyä reitinhaussa korkean tason päätöksiin ensin ja jättää yksityiskohdat myöhempään tarkasteluun. Kuvassa 4 on esitetty kirjoittajien mukaan yksinkertainen malli reitinhausta jaettuna kahteen hierarkkiseen kerrokseen.

Car ja Frank (1994) ovat tutkineet artikkelissaan hierarkkista reitinhakua. He esittävät artikkelissaan keskeisiä kysymyksiä hierarkkisesta reitinhausta: kuinka voidaan muuntaa yhdellä tasolla esitetty verkko tasojen muodostamaksi hierarkiaksi, kuinka näiden tasojen välillä voidaan liikkua sekä kuinka voidaan taata ratkaisun oikeellisuus ja tehokkuus? He ehdottavat mallia, jossa reititysalgoritmi hakee ensin alku- ja loppupisteestä reitit pisteisiin, jotka johtavat seuraavalle hierarkian tasolle (havainne tästä konseptista on esitetty alla kuvassa 4). Fu ja muut (2006) toteavat kuitenkin tästä

lähestymistavasta, että lähimmät pisteet hierarkian tasolta toiselle eivät kuitenkaan välttämättä ole optimaalisella reitillä. Lisäksi he kertovat, että hierarkkisen reitinhaun ongelmana on, että jossain tapauksissa voi olla tarpeen siirtyä esimerkiksi valtatieltä toiselle käyttäen välissä pienempää tietä, jolloin ainoastaan valtateitä laskennassaan käyttävä hierakkinen haku ei löydä optimaalista polkua. Liu (1997) on tutkimuksessaan esittänyt ratkaisua, jossa valtatieltä kääntyvät pienemmät tiet luetaan mukaan valtatieksi hyvin lyhyeltä pituudelta, jolloin ainakin yksinkertaiset oikopolut voidaan huomioida.



Kuva 4. Havainnekuva hierarkkisesta hausta jaettuna kahteen kerrokseen Kainuun tieaineistolla. Pisteet A ja B esittävät alku- ja loppupistettä. Oranssit viivat esittävät lyhintä reittiä pisteisiin hierarkian seuraavalla tasolla. Punainen väri korostaa pääteitä, jotka muodostavat hierarkian toisen tason.

Bast ja muut (2016) kertovat artikkelissaan, että hierarkkiset hakumetodit pyrkivät hyödyntämään toiminnassaan tieverkostojen rakennetta. Heidän mukaansa riittävän pitkät reitit lopulta yhdistyvät pieneen määrään keskeisiä tärkeitä teitä, kuten esimerkiksi moottoritiet. Seurauksena tästä hakualgoritmille riittää, kun se tutkii ainoastaan näistä keskeisiä teitä silloin kun sen suoritus on kaukana lähtö- ja kohdepisteestä. He mainitsevat myös, että lähdeaineistossa olevat tiekategoriat ovat suosittu heuristiikka, mutta niiden haittapuolena on, että optimaalisen polun löytymistä ei voida taata. Sommer (2014) mainitsee tutkimuksessaan, että hierakkisille hakumenetelmille tieaineisto on tyypillinen käyttökohde ja ne usein soveltavat alkuperäisen lisäksi toista graafia reitinhaun nopeuttamiseen.

Liu (1997) on tutkinut tapaa hyödyntää hierarkkista reititystä suuren kaupungin tieaineistolla. Hän jakoi Singaporen kaupungin lohkoihin, joita jakavat valtatiet, jonka perusteella reitinhakuprosessia voitiin optimoida. Tällöin reitti saadaan hakemalla polku alkupisteestä valtatielle, loppupisteestä valtatielle ja hakemalla reitti näiden kahden valtatieliittymän välille, jolloin algoritmin ei tarvitse tutkia kuin kahden lohkon sisältöä toiminnassaan. Tutkimustuloksissa hierarkian käyttö jakamalla kaupunki lohkoihin paransi reitinhakuaikoja noin 3–4 kertaisesti verrattuna perinteiseen reititysmalliin, jossa koko kaupungin tieverkosto huomioidaan. Lisäksi tutkijan esittämien tulosten perusteella hierarkkinen haku lisäsi laskennallisia matka-aikoja keskimäärin 10,1 %, mutta huomioiden lyhyet valtateiltä kääntyvät tieosuudet valtateiksi matka-ajat lisääntyivät vain keskimäärin 0,4 %.

Tutkimuksessaan Sanders ja Schultes (2006) ovat luoneet menetelmän hyödyntää tieverkoston hierarkiaa osana reititysalgoritmia. Heidän menetelmänsä jättää reitinhaun aikana käsittelemättä pienen prioriteetin teitä käyttämällä esikäsiteltyä oikopolkujen verkostoa. Tällöin siirtymä alkuperäisestä tarkan aineiston sisältävästä graafista esikäsiteltyyn tehdään hyödyntämällä etäisyystietoa alku- ja loppupisteistä eli silloin kun algoritmin suoritus on riittävän kaukana, voidaan jättää osa aineistoa käsittelemättä. Reitityksessä heidän menetelmänsä hyödyntää kaksisuuntaista Dijkstran algoritmia.

R. T. Honkanen (artikkelin luonnos, 2011) on ryhmänsä kanssa kokeellisesti testannut hierarkian käyttöä reitityksessä tieaineistolla ilman esikäsittelyä. Heidän koejärjestelyssään hyödynnettiin kolmea hierarkiakerrosta ja ohjattiin reititysalgoritmia suosimaan valtateitä silloin, kun algoritmin käsittelemä piste sijaitsi yli 30 km päässä alku- tai loppupisteestä. Kirjoittajat toteuttivat koejärjestelyn A* -algoritmilla. Heidän tulostensa perusteella hierarkian käyttö vähensi algoritmin läpikäymien pisteiden määrää puolella verrattuna perinteiseen A* -algoritmiin. Samalla vertailulla algoritmin suoritusaika laski kuitenkin vain noin 35 %.

2.4.4 Optimointimenetelmien yhdistelmät

Bast ja muut (2016) kertovat artikkelissaan, että optimointimenetelmiä yhdistämällä voidaan saada tehokkuusetuja reititysalgoritmien toiminnassa silloin kun menetelmät täydentävät toisiaan. Lisäksi eri käyttökohteissa voidaan yhdistää myös esikäsittelyä hyödyntäviä menetelmiä perinteisiin menetelmiin. Artikkelissaan Wagner ja Willhalm (2007) tekevät yhteenvetoa eri menetelmien yhdistelmistä aiemman tutkimuksen perusteella. Heidän mukaansa kaikki yhdistelmät eivät tuo haluttua lopputulosta ja he mainitsevatkin, että oikean yhdistelmän valinta riippuu saatavissa olevasta aineistosta, keskusmuistin määrästä ja esikäsittelyyn käytettävissä olevasta ajasta.

Sanders ja Schultes (2007) toteavat, että kaksisuuntainen haku on menetelmä, joka voidaan yhdistää moniin muihin hakuoptimointeihin. He mainitsevat kaksisuuntaisen haun olevan keskeinen ainesosa kehittyneissä hakumenetelmissä. Holzer, Schulz, Wagner ja Willhalm (2005) toteavat artikkelissaan, että katuverkostoa kuvaavilla graafeilla kaksisuuntaisen haun käyttö johtaa lähes aina parannuksiin reititysnopeudessa verrattuna ratkaisuihin, jossa kaksisuuntaista hakua ei käytetä. Kaksisuuntaisen hakumenetelmän tutkimustuloksia on käsitelty tarkemmin luvussa 2.4.2.

Hierarkkiset hakumenetelmät ja kohdehakeutuvat menetelmät on mahdollista yhdistää hyvin tuloksin (Bast ja muut, 2016). Samaa mieltä ovat myös kokeidensa perusteella Holzer ja muut (2005). Klassinen esimerkki kohdehakautuvasta hakumenetelmästä on A* -algoritmi. Liu (1997) selvitti tutkimuksessaan hierarkkisessa haussa Dijkstran algoritmin ja kohdehakeutuvan A* -algoritmin välistä suorituskykyä ja hänen raportoimiensa tulosten perusteella A* -algoritmi suoriutui paremmin. Lisäksi R. T. Honkanen (artikkelin luonnos, 2011) on kertonut parannuksista hakuaikoihin yhdistämällä hierarkkisen menetelmän sekä A* -algoritmin.

Kohdehakeutuvat menetelmät toimivat hyvin yhdistettynä kaksisuuntaisiin hakumenetelmiin silloin, kun hyödynnetään yhtenevää heuristiikkafunktiota (Wagner & Willhalm, 2007). Yhtenevää heuristiikkafunktiota ovat tutkineet ja kokeellisesti testanneet esimerkiksi Ikeda ja muut (1994) ja heidän tulostensa perusteella kaksisuuntainen A* -algoritmi käsitteli reitinhaun aikana 80 % vähemmän pisteitä kuin perinteinen Dijkstran algoritmi. Kokeellisessa järjestelyssään Holzer ja muut (2005) ovat todenneet, että ilman esikäsittelyä toimivista vaihtoehdoista he havaitsivat kaksisuuntaisen kohdehakeutuvan menetelmän parhaaksi. Lisäksi Wagner ja Willhalm

(2007) ovat todenneet, että kaksisuuntainen haku ja hierarkkiset hakumenetelmät toimivat hyvin yhdessä. He ovat soveltaneet kaksisuuntaisuutta hierarkian korkeammilla tasoilla, jossa graafin pisteiden määrää on karsittu.

2.5 Reitinhaussa käytettävät tietorakenteet

Artikkeleiden Bast (2009) sekä Bast ja muut (2016) ja kirjan Cormen ja muut (2009) mukaan Dijkstran algoritmin teoreettinen suorituskyky on riippuvainen siinä käytetystä tietorakenteesta, jolla seuraavaksi käsiteltävä piste valitaan. Yleisimmin tutkimuksessa mainitut tietorakenteet ovat Fibonaccin keko (engl. Fibonacci Heap) ja binäärikeko (engl. Binary Heap), jotka mahdollistavat pienimmällä kustannusarvolla olevan pisteen hakemisen tietorakenteesta nopeasti (prioriteettijono) ja ovat näin toiminnalliselta konseptiltaan samankaltaisia. Tieaineistolla prioriteettijono sisältää tyypillisesti käsittelemättömiä risteyspisteitä, jolloin tietorakenne mahdollistaa pienimällä kustannuksella olevan tieosuuden hakemisen algoritmin käsittelyyn tehokkaasti. Tietorakenteiden ominaisuudet ilmenevät hyvin verratessa niitä teoreettisesti yksinkertaiseen järjestämättömään listarakenteeseen (R. T. Honkanen, artikkelin luonnos, 2011). Eroja teoreettisen suorituskyvyn ja käytännön toiminnan välillä ovat tutkineet esimerkiksi Cherkassky ja muut (1996), Goldberg ja Tarjan (1996), Zhan ja Noon (1998). Edellä mainittujen tutkijoiden mukaan Fibonaccin keko on suorituskyvyltään teoreettisesti parempi vaihtoehto tietorakenteeksi, mutta esimerkiksi artikkelissaan Goldberg ja Tarjan (1996) toteavat binäärikeon käytännön olosuhteissa tehokkaammaksi. Myös muita tietorakenteita on olemassa ja eri vaihtoehtojen suorituskykyä ovat tutkineet esimerkiksi Cherkassky ja muut (1996), Larkin, Sen ja Tarjan (2014) sekä Zhan ja Noon (1998). Seuraavissa kappaleissa kuvataan binäärikeon ja Fibonaccin keon keskeiset erot perustuen aiempaan tutkimukseen sekä selvitetään, miten niiden teoreettinen ja käytännön suorituskyky poikkeavat toisistaan reitinhakukäytössä.

2.5.1 Lista- ja kekorakenteet

Toteutustavaltaan yksinkertaisin mahdollinen tietorakenne on järjestämätön lista, jolloin reitinhakualgoritmin tallentamat pisteet eivät ole missään tietyssä järjestyksessä, vaan pisteiden järjestyksen määrää tiedon lisäysjärjestys. Listarakennetta käytettäessä tiedon lisääminen listarakenteeseen on nopeaa, koska olemassa olevaa järjestystä ei tarvitse muokata, mutta haittapuolena pienimmän kustannusarvon omaavan pisteen hakeminen pois on työlästä listan koon kasvaessa. Listan toteutustavan mukaan on mahdollista, että algoritmin on listaa käyttäessään huolehdittava myös siitä, että samaa tietoa ei lisätä tietorakenteeseen kahdesti. R. T. Honkasen (artikkelin luonnos, 2011) ryhmän raportoimien tulosten perusteella listarakenne on käytännössä huono ratkaisu reitinhakualgoritmin tietorakenteeksi verrattuna Fibonaccin kekoon tai binäärikekoon.

Fredman ja Tarjan (1984) kuvaavat artikkelissaan kekorakennetta (engl. heap tai priority queue) seuraavasti: Keko on abstrakti tietorakenne, joka koostuu arvoista, joilla jokaisella on määrätty avain. He kuvaavat kekorakenteen sisältävän vähintään seuraavat operaatiot: keon luonti, uuden arvon lisäys määrätyllä avaimella, pienimmän avaimen määräämän arvon löytäminen sekä pienimmän avaimen määräämän arvon poistaminen. Lisäksi he pitävät seuraavia operaatioita hyödyllisinä: kahden keon yhdistäminen yhdeksi keoksi, keon sisältämän määrätyn avaimen pienentäminen sekä määrätyn arvon poistaminen keosta. He huomauttavat, että kahdessa viimeisessä tapauksessa haluttu avaimen tai arvon paikka keossa on tunnettava ennalta. Tutkijat painottavat myös, että kekorakenne ei tue tehokasta arvon hakua rakenteesta.

Kekorakenteesta on olemassa useita erilaisia variantteja, joita on kehitetty vuosien varrella. Reititysalgoritmien näkökulmasta kekorakenteen tekee mielenkiintoiseksi se, että keolla voidaan toteuttaa tehokkaasti näissä algoritmeissa tarvittava prioriteettijono. Yhden varhaisimmista kekorakenteen kuvauksista kirjoitti Williams (1964). Hän kuvaa artikkelissaan, kuinka kekorakennetta voidaan hyödyntää lajittelualgoritmissa. Tähän algoritmiin alun perin suunniteltua tietorakennetta kutsutaan yleisesti binäärikeoksi. Jatkotutkimusta kekorakenteesta on tehnyt artikkelissaan Vuillemin (1978). Hänen artikkelissaan esitellään binomikeko (engl. binomial heap), joka mahdollistaa usean yhtäaikaisen juuriarvon olemassaolon sekä yhdistämisen kekorakenteessa.

Reitityskäytössä eräs moderni ja tutkijoiden arvostama kekototeutus on Fibonaccin keko, jonka esittelivät Fredman ja Tarjan (1984). Heidän mukaansa Fibonaccin keko suunniteltiin käytettäväksi Dijkstran algoritmin suorituksessa. Fibonaccin keko pohjautuu osin artikkelin Vuillemin (1978) aiempaan tutkimukseen ja se parantaa kekototeutuksen teoreettisia suoritusaikoja. Toinen usein tutkimuksessa mainittu kekorakenne on binäärikeko, joka on teoreettiselta aikakompleksisuudeltaan hitaampi kuin Fibonaccin keko (Taulukko 1), mutta joka on joidenkin tutkijoiden mukaan helpompi toteuttaa ja toimii käytännössä alhaisemmilla suoritusajoilla käytettäessä Dijkstran algoritmia (Goldberg & Tarjan, 1996; Barbehenn, 1998).

Taulukko 1. Binäärikeon ja Fibonaccin keon operaatioiden aikakompleksisuuden teoreettista vertailua perustuen kirjaan Cormen ja muut (2009). Muuttuja n kuvaa kekorakenteessa olevien arvojen määrää.

Operaatio	**Binäärikeko (hitain tapaus)**	**Fibonaccin keko (amortisoitu)**
Keon luonti	$O(1)$	$O(1)$
Arvon lisäys	$O(\log n)$	$O(1)$
Pienimmän avaimen haku	$O(1)$	$O(1)$
Pienimmän avaimen poisto	$O(\log n)$	$O(\log n)$
Keon yhdistäminen toiseen	$O(n)$	$O(1)$
Avaimen pienentäminen	$O(\log n)$	$O(1)$
Arvon poisto	$O(\log n)$	$O(\log n)$

Cormen ja muut (2009) toteavat kirjassaan, että Fibonaccin keot ovat parhaimmillaan silloin, kun poisto-operaatioiden (engl. delete) sekä pienimmän avaimen poistojen (engl. extract-min) määrä on pienempi verrattuna muihin keko-operaatioihin. He kertovat myös, että suurimmassa osassa sovelluksia Fibonaccin keot eivät ole yhtä houkuttelevia kuin binäärikeot niiden ohjelmointiin liittyvien monimutkaisuuksien takia. Poikkeuksena mainitaan tilanteet, joissa sovellus käsittelee suuria datamääriä tai käyttökohde soveltuu hyvin Fibonaccin keon ominaisuuksiin.

Binäärikeko

Ensimmäisen kuvauksen binäärikeon toiminnasta on tehnyt Williams (1964) artikkelissaan osana lajittelualgoritmia. Lisäksi Cormen ja muut (2009) ovat kirjassaan kuvanneet binäärikeon ominaisuuksia ja toimintaa tarkemmin. Heidän mukaansa binäärikeko noudattaa kekojärjestystä, jossa joko pienimmän tai suurimman avaimen omaava arvo on aina keon juuriarvo. Valinta näiden välillä voidaan tehdä käyttökohteen perusteella. Keskeinen binäärikeon ominaisuus on myös se, että jokaisella arvolla voi

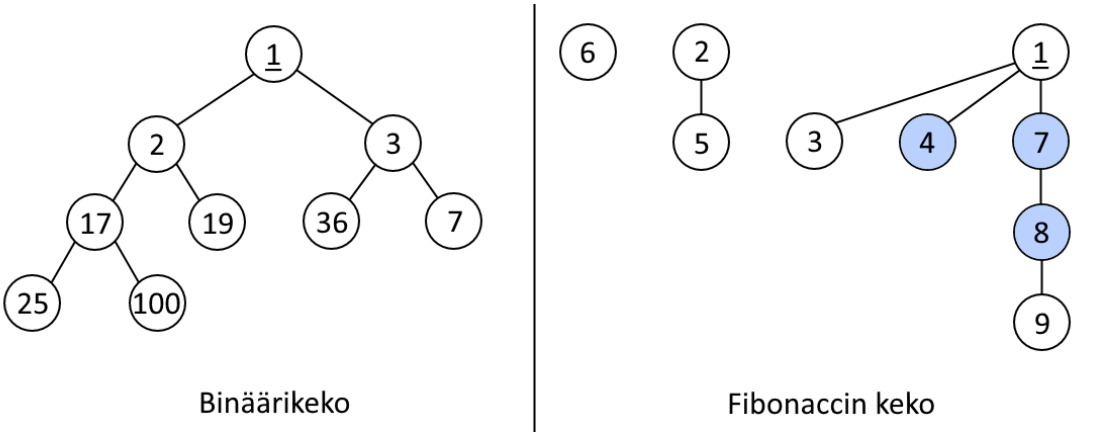
olla korkeintaan kaksi lapsiarvoa. Lisäksi binäärikeko sisältää aina korkeintaan yhden juuriarvon. Tyypillinen binäärikeko on esitetty kuvassa 5.

Cormen ja muut (2001) kertovat kirjassaan, että suurin osa binäärikeon operaatioista tapahtuvat pahimmassa tapauksessa ajassa $O(\log N)$, jossa N kuvaa keossa olevien arvojen määrää. Tästä poiketen nopeampia operaatioita ovat keon luonti ja juuriarvon avaimen hakeminen, jotka tapahtuvat vakioajassa $O(1)$. Lisäksi kaksi binäärikekoa voidaan liittää toisiinsa ajassa $O(N)$. He osoittavat, että Dijkstran algoritmin suoritusaika toteutettuna binäärikeolla on $O((V+E)\log V)$, jossa V kuvaa graafissa olevien pisteiden määrää ja E kaarien määrää. Vertailukohtana alkuperäisen Dijkstran algoritmin suoritusaika listarakenteella on $O(V^2)$, kuten ovat esittäneet Barbehenn (1998) sekä Cormen ja muut (2009).

Fibonaccin keko

Artikkelin Fredman ja Tarjan (1984) mukaan teoreettisesti tarkasteltuna Fibonaccin keko suoriutuu kaikista operaatioista amortisoidussa vakioajassa paitsi pienimmän arvon poisto ja arvon poisto, jotka tapahtuvat amortisoidussa ajassa $O(\log N)$. Nämä ovat teoreettisesti parempia arvoja kuin edellä mainitussa binäärikeossa. Cormen ja muut (2009) korostavat erityisesti Fibonaccin keossa ominaisuutta, joka sallii kahden kekorakenteen liittämisen toisiinsa amortisoidussa vakioajassa, joka heidän mukaansa on merkittävä ero suorituskyvyssä verrattuna binäärikekoon. Lisäksi he analysoivat kirjassaan Dijkstran algoritmin suoritusaikaa Fibonaccin keolla. He osoittavat suoritusajan olevan $O(E + V\log V)$, joka on parempi kuin binäärikeolla.

Fibonaccin keon tarkan toiminnan ovat kuvanneet artikkelissaan Fredman ja Tarjan (1984). Tutkijat kertovat, että Fibonaccin keko, kuten muutkin kekorakenteet, noudattavat niin sanottua kekojärjestystä: Oletetaan, että *X* on mikä tahansa keossa oleva arvo. Tällöin *X*:n avain ei voi olla pienempi kuin sen isännällä (hierarkiassa ylempänä oleva avain) olettaen, että *X*:llä on isäntä. Tämä takaa sen, että keon ylimpänä arvona on aina arvo, jonka avain on pienin. Tämä järjestys voidaan tarpeen mukaan myös kääntää toisinpäin, jolloin keon ylimmäksi arvoksi saadaan arvo, jonka avain on suurin. Tyypillinen Fibonaccin keko on esitetty kuvassa 5. Aiemmin esitellyt reititysalgoritmit hyödyntävät tyypillisesti tietorakenteita tallentamalla niihin suorituksensa aikana käsittelemättömien pisteiden etäisyyksiä tehokasta pienimmän arvon hakua varten. Tietorakenne sisältää tällöin osajoukon varsinaisesta tieaineistoa kuvaavasta graafista.



Kuva 5. Tyypillisen binäärikeon ja Fibonaccin keon rakenne. Alleviivatut arvot osoittavat kekojen juuriarvoja. Fibonaccin keon merkatut arvot on osoitettu sinisellä.

Toisin kuin monissa muissa kekorakenteissa, Fibonaccin keko ei aseta rajoituksia puurakenteiden määrälle tai koostumukselle. Toisin sanoen Fibonaccin keko voi koostua useammasta puusta ja jopa niin, että keko sisältää pelkästään puiden juuriarvoja. Yllä olevassa kuvassa 5 on esitetty tyypillinen binäärikeko ja Fibonaccin keko, joka koostuu kolmesta puurakenteesta. Artikkelin Fredman ja Tarjan (1984) mukaan Fibonaccin kekoa voidaan kuvata ”laiskaksi” kekorakenteeksi sillä perusteella, että tiettyjen operaatioiden nopeus saavutetaan viivästyttämällä kekorakenteen ylläpito-operaatioita myöhemmäksi (esimerkkinä arvon lisäys kekorakenteeseen). Kirjoittajien mukaan tämä vaikuttaa osaltaan myös teoreettisiin suoritusaikoihin niin, että joissain tapauksissa operaatioiden suorittaminen kestää pidempään kuin edellä annettu amortisoitu aika.

2.5.2 Tietorakenteiden vertailua reitityskäytössä

Tietorakenteiden käytännön suorituskyvystä Bast ja muut (2016) kertovat artikkelissaan, että eri binäärikeon toteutukset toimivat usein paremmin viitaten tuloksiin artikkelista Cherkassky ja muut (1996). Kyseisen artikkelin tuloksissa tutkijat lausuvatkin Dijkstran algoritmia käytettäessä binäärikeon[3] toimivan kaikissa heidän testeissään nopeammin kuin Fibonaccin keko. He toteavat myös yksinkertaisen listarakenteella toimivan ”naiivin” Dijkstran algoritmin toimivan vain pienellä määrällä pisteitä.

Goldberg ja Tarjan (1996) ovat tutkineet artikkelissaan Dijkstran algoritmin käytännön suorituskykyä, ja he ovat rakentaneet matemaattisen mallin, jolla voidaan ennustaa algoritmin suorittamien avaimen pienennysoperaatioiden (engl. decrease key) määrä. He ovat empiirisesti huomanneet, että näitä operaatioita suoritetaan suhteellisen harvoin. He myös mainitsevat Fibonaccin keon pienimmän avaimen poisto-operaation olevan käytännössä hitaampi kuin binäärikeossa, koska tietorakenne on toteutukseltaan monimutkaisempi. Uuden arvon lisääminen ja avaimen pienentäminen ovat operaatioita, joissa tutkijoiden mukaan Fibonaccin keko on tehokkaampi, kun keon koko kasvaa. Pohjaten näihin havaintoihin he tarjoavat selityksen miksi artikkelin Cherkassky ja muut (1996) tulokset viittaavat Fibonaccin keon suoriutuvan huonommin käytännössä kuin sen teoriassa pitäisi. He mainitsevat myös omien kokeellisten tulostensa viittaavan siihen, että Fibonaccin keko suoriutuu paremmin tiheillä graafeilla (jotka sisältävät runsaasti pisteiden välisiä yhteyksiä).

Artikkelissaan Zhan ja Noon (1998) tutkivat useiden eri tietorakenteiden käytännön nopeutta käyttäen pohja-aineistona Yhdysvaltojen eri osavaltioiden tieverkostoja. Muilta osin he kertovat testinsä pohjautuvan artikkelin Cherkassky ja muut (1996) koejärjestelyyn, poistaen kuitenkin sellaiset algoritmit, jotka eivät sovi käytettäväksi oikealla tieverkostolla. Kirjoittajien käyttämä aineisto oli jaettuna kahteen osaan: päätieverkoston aineisto ja tarkempi tieverkoston aineisto. Erona aineistoilla on pääasiassa se, että tarkempi aineisto sisältää enemmän pisteitä ja kaaria. Pieni aineisto sisälsi datan kymmenestä osavaltiosta, joista jokaisessa oli mukana 523–2878 pistettä. Suuri aineisto taas sisälsi samoista osavaltioista 35793–92792 pistettä. Tutkijoiden tuloksista voidaan tarkastella muun muassa kokonaissuoritusajat Dijkstran algoritmin toteutuksille eri tietorakenteilla.

---

[3] Cherkassky, Goldberg ja Radzik (1996) testasivat monilapsista kekorakennetta (engl. k-ary heap) niin, että jokaisella arvolla voi olla 3 lapsiarvoa.

Tutkijoiden Zhan ja Noon (1998) suorittamien kokeiden tulosten perusteella pienellä aineistolla listarakenne suoriutui Dijkstran algoritmilla paremmin kuin kekopohjaiset rakenteet, mutta suurella aineistolla listarakenne oli yli 10 kertaa hitaampi. Kekopohjaisista toteutuksista Fibonaccin keko oli molemmilla aineistoilla hieman hitaampi kuin binäärikeko. Pienellä aineistolla siis listarakenne oli nopein, binäärikeko toinen ja Fibonaccin keko hitain (suoritusajat: 75,09 ms, 77,47 ms ja 124,63 ms). Suuremmalla aineistolla binäärikeko oli nopein, Fibonaccin keko toinen ja listarakenne viimeinen (suoritusajat: 7,97 s, 12,73 s, 71,21 s). Näiden tulosten perusteella tutkijat suosittelevat välttämään listarakenteen käyttöä. Jacob ja muut (1999) ovat tutkineet tietorakenteita Dijkstran algoritmin ja A* -algoritmin kanssa. He suosittelevat koetulostensa perusteella yksinkertaisen binäärikeon käyttöä Dijkstran algoritmin yhteydessä tieaineistolla.

R. T. Honkanen (artikkelin luonnos, 2011) on ryhmänsä kanssa suorittanut kokeita käytännön tieaineistolla ja heidän tulostensa mukaan binäärikeko suoriutuu jossain määrin paremmin kuin Fibonaccin keko sekä Dijkstran, että A* -algoritmilla. Heidän työssään on käytetty pohja-aineistona Digiroad-aineistotietokantaa. Heidän esittämiensä tulosten perusteella listarakenteen suoritusaika on noin kaksinkertainen verrattuna Fibonaccin kekoon jo siinä vaiheessa, kun aineistosta on käyty läpi 10 000 pistettä. Lisäksi Dijkstran algoritmin suoritusaika järjestämättömillä listoilla kasvaa käytännössä noin $O(n^2)$, kun pisteiden määrä lisääntyy. Kirjoittajien mukaan kekopohjaiset tietorakenteet tarjoavat huomattavan parannuksen suoritusaikoihin. He ovat käyttäneet työssään suurempaa aineistoa kuin aiemmin esitetyssä artikkelissa Zhan ja Noon (1998). Aineisto on kattanut noin 2,2 miljoonaa pistettä, kun vastaavasti Zhanin ja Noonin (1998) käyttämä suurempi aineisto kattaa yhdestä osavaltiosta alle 0,1 miljoonaa pistettä.

Tietorakenteiden käytännön suorituskyvyn ja teoreettisen suorituskyvyn eroja eri työkuormilla ovat tutkineet Larkin ja muut (2014). He löysivät tutkimuksessaan vahvan korrelaation tietorakenteen suorituskyvyn ja tietokoneen prosessorin L1-välimuistin tehokkuuden välillä niin, että työkuorman suoritusaika nousee mitä vähemmän tietorakenteen dataa L1-välimuisti pystyy tallentamaan. Lisäksi he tulivat siihen tulokseen, että yksinkertaisemmat tietorakenteet toimivat käytännön oloissa nopeammin, mutta työkuormalla on merkittävä vaikutus tietorakenteen nopeuteen eikä ole olemassa yhtä oikeaa tietorakennetta kaikkiin käyttökohteisiin. Vastaavan havainnon prosessorin välimuistin merkityksestä ovat tehneet aiemmin myös Jacob ja muut (1999). He ovat havainneet, että suurin osa prosessin suorittamista käskyistä liittyy käytettyyn tietorakenteeseen ja algoritmin suoritusajasta merkittävä osa kuluu siihen, kun prosessori odottaa tietoa keskusmuistista. Aihetta ovat kommentoineet myös Sanders ja Schultes (2007) ja he toteavat artikkelissaan, että suurilla tieaineistoilla valitun tietorakenteen merkitys vähenee, koska välimuistin viiveet aiheuttavat pullonkaulan ja toisaalta suurempia nopeushyötyjä on saatavilla hyödyntämällä muita nopeutustekniikoita, jotka pitävät tietorakenteen koon pienempänä.

Artikkelissaan Larkin ja muut (2014) tutkivat työkuormana eri tietorakenteiden toimintaa Dijkstran algoritmilla. He testasivat kaksi erilaista varianttia binäärikeosta, joiden toteutus tietokoneen muistissa oli ohjelmoitu toisistaan poikkeavasti. Ensimmäinen variantti (engl. explicit) oli toteutettu käyttämällä tietokoneen kekomuistista (engl. heap) varattuja arvoja ja osoittimia ja toinen variantti (engl. implicit) oli toteutettu asettamalla puurakenne yksittäisen muistitaulukon (engl. array) sisälle hyödyntäen osoittimia puun ominaisuuksien toteutuksessa. Lisäksi näistä binäärikekorakenteista oli käytössä eri variantit, joissa sallittujen lapsiarvojen maksimimäärä oli 2, 4, 8 tai 16. Vertailukohtana he testasivat myös muita

tietorakenteita, kuten binomikeko ja Fibonaccin keko. Eri variantit ja niiden suhteelliset suoritusajat on esitetty taulukossa 2.

Taulukko 2. Tietorakenteiden suhteelliset suoritusajat Yhdysvaltain tieaineistolla käytettäessä Dijkstran algoritmia. Tiedot on koostettu artikkelin Larkin ja muut (2014) tulosten perusteella.

Tietorakenteen toteutustapa	Suoritusaika (suhteellinen)
Keko-4-lapsiarvoa-T2	1,00
Keko-8-lapsiarvoa-T2	1,07
Keko-2-lapsiarvoa-T2	1,17
Keko-16-lapsiarvoa-T2	1,37
Binomikeko	2,37
Fibonaccin keko	3,15
Keko-4-lapsiarvoa-T1	3,39
Keko-2-lapsiarvoa-T1	3,84
Keko-8-lapsiarvoa-T1	4,20
Keko-16-lapsiarvoa-T1	5,94

Tutkijat kertovat tuloksistaan, jotka ovat koostettuna yllä olevassa taulukossa, että reitinhaussa Yhdysvaltain tieaineistolla toinen binäärikeon variantti neljällä mahdollisella lapsiarvolla (engl. implicit_4) suoriutui testistä pienimmässä ajassa, Fibonaccin keon ollessa 3,15 kertaa hitaampi ja ensimmäinen variantti (engl. explicit_4) ollessa 3,39 kertaa hitaampi. Tulostaulukon perusteella eri binäärikeon variantien väliset erot olivat joitain kymmeniä prosentteja paitsi 16-lapsiarvon toteutukset, jotka olivat selvästi hitaimpia muihin verrattuna.

3. Tutkimusmenetelmä

Artikkelissaan Hevner ja muut (2004) kertovat, että organisaatiot kehittävät tietojärjestelmiä oman tehokkuutensa ja tuottavuutensa parantamiseksi. He kuvaavat kaksi tietojärjestelmätieteelle keskeistä paradigmaa: käyttäytymistiede (Behavioral science) kehittää ja tutkii ihmisten tai organisaatioiden toimintaa kuvaavia teorioita, kun taas suunnittelutiede (Design science) keskittyy uusien ja innovatiivisten artefaktien luomiseen, joiden avulla ihmisten ja organisaatioiden toimintaa voidaan kehittää. Hevner ja muut (2004) pitävät näitä kahta paradigmaa toisiaan täydentävinä, sillä tieteellistä tutkimusta tulisi arvioida myös sen käytännön vaikutusten kautta.

Tässä tutkielmassa hyödynnetään Design science -menetelmää reitinhakualgoritmien vertailussa ja optimoinnissa Suomen tieaineistolla. Käytettävä tutkimusmenetelmä ja sen keskeiset periaatteet kuvataan tässä luvussa perustuen Hevner ja muut (2004) sekä Hevner (2007) lähteisiin. Tutkimusmenetelmän soveltuvuus algoritmeja ja kartta-aineistoa sisältävän artefaktin (mittausohjelma) toimintaan käydään läpi Hevner ja muut (2004) seitsemän ohjesäännön mukaisesti perustellen miten tutkimus täyttää tutkimusmenetelmän sille asettaman vaatimukset. Lisäksi käydään läpi keskeiset tutkimuskysymykset ja peilataan niitä suunnittelutieteen keskeisiin periaatteisiin.

3.1 Suunnittelutiede

Artikkelin Hevner ja muut (2004) mukaan suunnittelutiede (Design Science) on lähtökohtaisesti ongelmanratkaisumenetelmä. Menetelmällä luodaan artefaktien kautta innovaatioita, jotka sisältävät uusia ideoita, teknisiä kyvykkyyksiä, käytäntöjä ja tuotteita, mutta menetelmä pohjautuu kuitenkin tieteellisiin teorioihin ja niiden laajentamiseen, testaukseen sekä hyödyntämiseen käytännössä. Menetelmä on siis käytännönläheisempi kuin puhtaasti teoreettinen tutkimus.

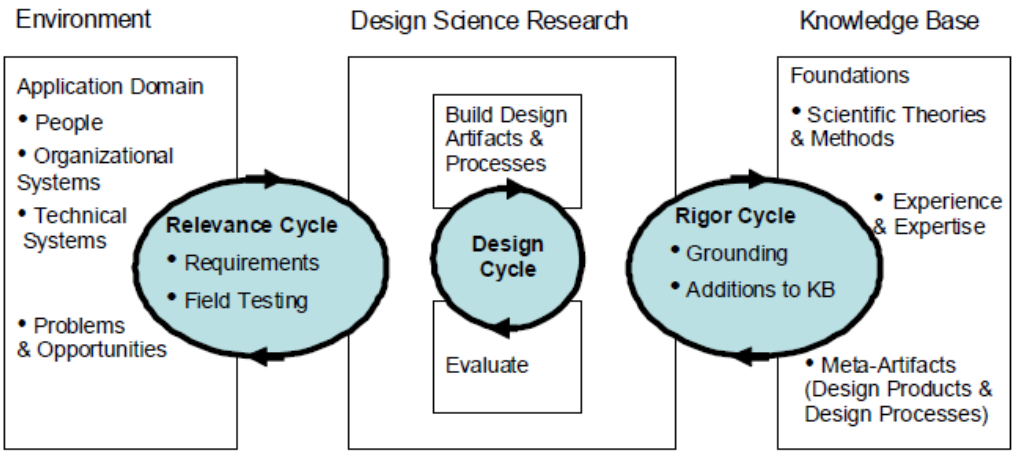
Design Science -menetelmässä keskeisessä osassa on artefakti, jota pyritään kehittämään tai parantamaan ja jonka toimintaa voidaan kvantitatiivisesti mitata sekä verrata erilaisiin vaihtoehtoihin. Vastaavasti käyttäytymistiede voi osaltaan tutkia miten artefakti vaikuttaa toimintaympäristöönsä. Esimerkiksi IT-ympäristössä artefaktina on monesti tietojärjestelmä tai sen osa, jolloin on tyypillistä tutkia esimerkiksi tietojärjestelmän käyttökelpoisuutta, vaikutusta sen käyttäjiin tai järjestelmän vaikutusta organisaatioon. Hevner ja muut (2004) argumentoivatkin, että suunnittelutiede ja käyttäytymistiede ovat toisiinsa kiinteästi yhteydessä ja tutkimusmenetelmät täydentävät toisiaan.

Suunnittelutieteen tuottama innovatiivinen artefakti on seurausta suunnitteluprosessin aikana suoritetuista aktiviteeteista (Hevner ja muut, 2004). Prosessin aikana tutkija kehittää sekä rakennusprosessia että artefaktia. Artefaktia evaluoimalla voidaan ymmärtää lisää sen toiminnasta ja suorituskyvystä sekä laajentaa ymmärrystä ongelma-alueesta, joka ollaan ratkaisemassa ja näin ollen muodostuu rakentamisesta (engl. build) ja evaluoinnista (engl. evaluate) koostuva kehä, jonka loppupisteenä syntyy lopullinen artefakti (Hevner 2007, Hevner ja muut, 2004). Suunniteltujen artefaktien evaluoinnissa voidaan hyödyntää tieteellistä ymmärrystä luonnonlaeista ja käyttäytymisteorioista vastaavasti kuten muidenkin objektien kanssa. Lopputuloksena tavoitellaan

toimintaympäristöönsä sopivaa ja käyttökelpoista artefaktia, jolloin myös artefaktin evaluointia on suoritettava tästä näkökulmasta.

Tutkimuksen näkökulmasta suunnittelutiede ja siihen liittyvä rakentamisen ja evaluoinnin kehä toimivat osana viitekehystä (Hevner ja muut, 2004), jossa on otettava huomioon toimintaympäristö sekä olemassa oleva tieto ja teoriat. Yleisesti ottaen organisaatiolla on vaatimuksia tai tarpeita, jotka toimivat syötteenä suunnittelutieteen prosessille. Prosessin lopputuloksena tuotetaan toimintaympäristönsä sopiva artefakti. Näin suunnittelutiede tuottaa ympäristöönsä relevantteja tuloksia. Vastaavasti suunnittelutiede hyödyntää prosessissaan olemassa olevaa tieteellistä teoriaa, malleja, toimintatapoja ja viitekehyksiä. Näin voidaan taata suunnittelutieteen täsmällisyys ja perusteet olemassa olevan tutkimukseen pohjautuen. Suunnittelutiede myös omalta osaltaan tuo lisäyksiä ja täydennyksiä tieteelliseen tutkimukseen, jota muut voivat hyödyntää.

Hevner (2007) kuvaa tarkemmin suunnittelutieteen prosessin kolmena syklinä: relevanssisykli (engl. relevance), suunnittelusykli (engl. design) ja täsmällisyyssykli (engl. rigor). Nämä prosessit on esitetty tarkemmin kuvassa 6. Kuvatut syklit ovat linjassa aiemman tutkimuksen (Hevner ja muut, 2004) kanssa tarkentaen miten artefaktin suunnittelu on yhteydessä toimintaympäristöön ja tieteelliseen tutkimukseen.



Kuva 6. Design Science -tutkimusmenetelmän keskeiset prosessit. Lähde: Hevner (2007).

Relevanssisyklistä Hevner (2007) mainitsee, että hyvä suunnittelutieteen tutkimus usein alkaa tunnistamalla mahdollisuuksia ja ongelmia oikeissa käyttöympäristöissä. Hän viittaa myös Iivarin (2007) huomioon siitä, että osa suunnittelutiedettä on uusien potentiaalisten kehityskohteiden tunnistamista ennen kuin varsinaista ongelmaa on ehditty tunnistaa. Artikkelin Hevner (2007) mukaan relevanssisykli asettaa vaatimukset ja käyttökontekstin suunniteltavalle artefaktille, mutta lisäksi se asettaa vaatimukset siitä millainen artefakti on lopputuloksena hyväksyttävissä.

Suunnittelutiede on vahvasti sidonnainen olemassa olevan tutkimuksen viitekehykseen ja teoriaan. Hevner (2007) muistuttaa, että olemassa oleva tutkimustieto sisältää myös kokemuksia ja asiantuntemusta sovellusalueesta sekä siihen liittyvistä artefakteista ja prosesseista. Täsmällisyyssykli varmistaa aikaisemman tutkimustiedon pohjaksi uudelle tutkimukselle ja tukee näin innovointia. Keskeistä on tutkijan oikeaoppinen tieteellisten teorioiden valinta ja hyödyntäminen artefaktin rakentamisessa ja evaluoinnissa. Huomattavaa on kuitenkin, että sekä Hevner (2007) ja Iivari (2007) ovat yhtä mieltä,

että suunnittelutieteen tutkimuksessa on jätettävä tilaa tutkijan luovuudelle uusien artefaktien suunnittelussa. Tieteelliset kontribuutiot ovat keskeisiä tuloksia muiden tutkijoiden näkökulmasta ja vastaavasti käyttökelpoiset tulokset ja parannukset ovat keskeisiä tietojärjestelmiä hyödyntävien sidosryhmien näkökulmasta (Hevner, 2007).

Hevner ja muut (2004) sekä Hevner (2007) mukaisesti on tärkeää erottaa suunnittelutiede ja rutiininomainen kehitystyö tai järjestelmäkehitys toisistaan. He mainitsevat rutiininomaisen kehitystyön olevan enemmän olemassa olevan teorian ja tutkimustiedon hyödyntämistä organisaatioiden kohtaamiin ongelmiin. Vastaavasti suunnittelutiede kehittää uniikkeja tai innovatiivisia tapoja ratkaista ongelmia paremmin kuin olemassa olevilla menetelmillä. Olennaisena piirteenä suunnittelutieteen ratkaisumalli tuo selvästi uuden kontribuution tieteelliseen tutkimukseen.

3.2 Tutkimuskysymykset ja ohjesäännöt

Hevner ja muut (2004) määrittämät keskeiset suunnittelutieteen ohjesäännöt on esitetty kuvassa 7. Kirjoittajat eivät kuitenkaan lähtökohtaisesti edellytä kaikkien ohjesääntöjen täsmällistä noudattamista tutkimuksessa, mutta korostavat perusteltua lähestymistä, jossa tutkimusta on arvioitu peilaten jokaiseen ohjesääntöön. Alla testattavaa artefaktia ja siihen liittyvää tutkimusprosessia verrataan näihin ohjesääntöihin.

Table 1. Design-Science Research Guidelines

Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Kuva 7. Design Science -tutkimusmenetelmän seitsemän ohjesääntöä. Lähde: Hevner ja muut (2004).

1. Suunnittelu artefaktina

Artikkelissaan Hevner ja muut (2004) määrittelevät, että suunnittelutieteen tuloksena syntyy tarkoituksenmukainen IT artefakti, joka ratkaisee jonkin organisaationaalisen ongelman. Artefakti itsessään voi olla jonkinlainen järjestelmä, malli, metodi tai muu vastaava luomus. Kirjoittajien määritelmä artefaktista ei kuitenkaan tarkoita, että tutkimuksen tuloksena syntyvä artefakti olisi kaikissa tapauksissa itsessään täydellinen tietojärjestelmä. Heidän mukaansa validi artefakti voi olla esimerkiksi työkalu, joka parantaa varsinaisen käytännön tietojärjestelmän luontiprosessia. Tutkimuksen näkökulmasta on kuitenkin tärkeää, että artefakti jollain tapaa luo uutta tieteellistä tietoa, joka laajentaa olemassa olevaa tutkimuspohjaa.

Tutkimus suoritetaan mittausohjelman avulla, joka toimii tutkimusmenetelmän näkökulmasta artefaktina. Mittausohjelma lukee sisään Digiroad-kartta-aineistoa, valitsee aineistosta pistepareja, suorittaa pistepareille reitityksen tutkittavien algoritmien avulla sekä tuottaa mittausdataa, joka auttaa vastaamaan tutkimuskysymyksiin. Teknisesti mittausohjelma on muokattavissa hyödyntämään erilaisia kartta-aineistoja sekä suorittamaan erilaisia reititysalgoritmeja muokkaamatta ohjelman perusrakennetta, joka tekee siitä sopivan eri vaihtoehtojen iterointiin sekä vertailukelpoisten tulosten tuottamiseen. Artefaktin tarkoituksena on luoda algoritmi, joka on mitattavissa paremmaksi kuin olemassa oleva algoritmi. Aineisto ja reititysalgoritmit voisivat myös toimia alijärjestelmänä jossain muussa tietojärjestelmässä, joka tällaista toimintoa tarvitsee. Tällöin kyseinen artefakti olisi toiminnassa ympäristössä, jossa sitä käytetään organisaation ja siinä toimivien ihmisten toimesta, jolloin mittausdataa ja saatuja tuloksia voidaan arvioida myös tässä kontekstissa. Tämän tutkimuksen näkökulmasta mittausohjelma on itsenäisesti suoritettava ohjelma algoritmien evaluointiin tieaineistolla, joka täyttää artikkelin Hevner ja muut (2004) asettaman määritelmän artefaktista.

2. Ongelman relevanssi toimintaympäristössä

Suunnittelutieteellisen tutkimuksen tavoitteena on rakentaa teknologisia ratkaisuita tärkeisiin ja relevantteihin organisaatioiden ongelmiin (Hevner, 2004). Ongelmanratkaisu voidaan yksinkertaistaa nykytilan ja tavoitetilan väliseksi eroksi. Niinpä keskeinen väline tutkimuksen kannalta on artefakti, jota iteroidaan järjestelmän parantamiseksi ja näin tavoitetilan saavuttamiseksi.

Optimaalinen reitinhaku graafissa on ollut tutkimustyön kohteena jo useita vuosikymmeniä. Katsauksia aiheeseen on saatavilla esimerkiksi artikkeleista Bast ja muut (2016), Sommer (2014) sekä Fu ja muut (2006). Internet, GPS ja nykyiset älylaitteet tarkoittavat, että digitaaliset kartat ovat yhä useamman käyttäjän ulottuvilla. Kehitysaskeleet reitinhaun optimoinnissa mahdollistavat karttojen käytön entistä pienemmillä laitteilla sekä aiempaa nopeammin tai vaihtoehtoisesti suurempien aineistojen tai käytännön ongelmien ratkaisun. Vastaavasti avoin data (esimerkiksi Digiroad 2019a) saattaa entistä suuremmat kartta-aineistot kaikkien ulottuville. Tehokkaampien ja monipuolisempien reitinhakualgoritmien saatavuus voisi edelleen jatkaa tätä kehitystä sekä tarjota alustan uusille tietojärjestelmille, joka tekee aiheeseen liittyvän algoritmitutkimuksen tärkeäksi. Kaikkia mahdollisia käyttökohteita reitinhaulle kartta-aineistolla ei ole välttämättä vielä kuviteltu, mutta esimerkiksi entistä pienemmät tietotekniset laitteet voisivat tarjota uusia mahdollisuuksia, joita eri organisaatiot voisivat hyödyntää tietojärjestelmien avulla. Vastaavasti kehitysaskeleet reitinhaun nopeudessa voisivat johtaa parempaan käyttökokemukseen samalla säästäen

laskentatehoa muihin käyttötarkoituksiin. Näin arvioiden tämä tutkimus täyttää kriteerin relevantista ongelmasta.

3. Artefaktin evaluointi

Artikkelissaan Hevner ja muut (2004) kertovat, että artefaktin käyttökelpoisuus, laatu ja tehokkuus on pystyttävä osoittamaan hyvin suunniteltujen evaluointimenetelmien avulla. Tämä tutkimuksen keskeiset tutkimuskysymykset liittyvät reititysalgoritmien tehokkuuteen ja laatuun. Keskeisenä tutkimuskysymyksenä halutaan selvittää, miten reititysalgoritmin tehokkuutta voidaan optimoida Digiroad-aineistolla huomioiden reittien laatu ja algoritmin toimintanopeus. Tähän laajempaan kysymykseen vastataan alla esitettyjen alikysymysten avulla luomalla artefakti, jonka avulla suoritetaan kokeita.

- Millä tavoin reititysalgoritmien käyttämiä tietorakenteita voidaan optimoida tieaineistolle?
- Mitä keskinopeutta voidaan käyttää reititysalgoritmin heuristisessa optimoinnissa Suomen tieaineistolla?
- Paljonko reititysalgoritmin nopeutta voidaan parantaa hyödyntämällä Digiroad-aineiston metadataa?

Artefaktin avulla suoritetaan erillinen koejärjestely jokaiselle alikysymykselle niin, että saadut tulokset tukevat seuraavia koejärjestelyitä. Tarkan mittausdatan kerääminen artefaktista on keskeistä, jotta esitettyihin tutkimuskysymyksiin voidaan vastata. Algoritmien evaluointi pohjautuu siihen, että verrokkialgoritmeja voidaan mitata suhteessa uusiin algoritmeihin käyttäen samaa aineistoa ja reitityspisteitä. Käytettyjä mittareita ovat esimerkiksi algoritmin suorittamien askelten määrä, suoritusaika ja haetun reitin eroavaisuus verrokkialgoritmista (mittarit on kuvattu tarkemmin luvussa 3.3). Näin saatu data on vertailukelpoista ja sen pohjalta voidaan tehdä johtopäätöksiä uuden algoritmin toiminnasta täyttäen vaatimuksen artefaktin evaluointimenetelmistä.

4. Tutkimustulokset

Suunnittelutiede pyrkii laajentamaan olemassa olevaa tieteellistä tutkimustietoa. Hevner ja muut (2004) kertovat, että tutkimustietoa voidaan laajentaa kolmella tavalla. Artefakti laajentaa tutkimustietoa, jos se ratkaisee aiemmin ratkaisemattoman ongelman. Tutkimus voi myös tuottaa uusia malleja ja metodeja, jotka laajentavat tieteellisen tutkimuksen pohjaa. Lisäksi suunnittelutiede voi tuottaa uusia metodologioita tai metriikoita, jotka ovat merkittäviä suunnittelutieteen tutkimukselle.

Tämä tutkimus pyrkii laajentamaan tieteellistä pohjaa tuottamalla uuden algoritmin perustuen olemassa olevaan tieteelliseen tutkimukseen ja vertaamalla sitä verrokkialgoritmeihin laajan tieaineiston kontekstissa. Tulosten näkökulmasta, vaikka tavoitteena onkin rakentaa algoritmi, joka toimii mittareilla paremmin kuin verrokkinsa, myös päinvastainen tulos on hyväksyttävä, koska sen avulla saadaan uutta tietoa eri lähestymistapojen toiminnasta Digiroad-aineistolla sekä voidaan osaltaan vahvistaa olemassa olevien tutkimustuloksien paikkaansa pitävyyttä. Tältä osin tehty työ siis täyttää asetetun vaatimuksen kontribuutiosta tieteelliseen tutkimukseen.

5. Tutkimuksen täsmällisyys

Artikkelissaan Hevner ja muut (2004) kertovat, että suunnittelutiede vaatii täsmällisten metodien ja arviointikriteerien käyttöä. He painottavat erityisesti artefaktin rakennusvaiheessa huomiota sen käyttökelpoisuuteen ja yleiskäyttöisyyteen. Hyvään

täsmällisyyteen päästään soveltamalla asianmukaisesti teoreettista viitekehystä ja tutkimusmenetelmiä. Lisäksi artefaktin suorituskykyä mittaaviin arvoihin on kiinnitettävä huomiota. On keskeistä ymmärtää, miksi rakennettu artefakti toimii tai ei toimi, jotta tätä tietoa voidaan myöhemmin hyödyntää uusien artefaktien suunnittelussa.

Lähtökohtaisesti testattavan artefaktin tuottamat tulokset riippuvat kolmesta syötteestä: Annetusta aineistosta, valituista mittauspisteistä aineiston sisällä sekä testattavasta algoritmista. Aineisto sekä mittauspiteet pysyvät tutkimuksen aikana samoina, jolloin ainoa muuttuva tekijä on testattava algoritmi, jolloin saadut tulokset ovat verrannollisia. Algoritmeja verratessa hyödynnetään hyvin tunnettuja verrokkialgoritmeja (Dijkstra, A*), joiden avulla tuloksia voidaan arvioida laajemmassa viitekehyksessä. Esimerkiksi Bast ja muut (2016) kuvaavat artikkelissaan Dijkstran algoritmia standardiratkaisuna ja A* -algoritmia klassisena lopputulokseen ohjautuvana algoritmina, jolloin nämä valinnat verrokkialgoritmeiksi ovat luontevia vertailukohtia. Mittareina käytetään arvoja, joiden vertaaminen keskenään on matemaattisesti yksinkertaista eikä ihmisen tekemä syöte vaikuta lopputuloksiin. Matemaattista todistusta algoritmien oikeellisuudesta ei kuitenkaan käytetä lukuun ottamatta verrokkialgoritmeista annettuja todistuksia (Cormen ja muut 2009; Hart ja muut, 1968). Testauksen tuloksia analysoidaan erillisessä luvussa, jonka tarkoituksena on käydä läpi saadut mittaustuloset sekä ymmärtää syitä artefaktin käyttäytymiselle. Tällöin työ täyttää vaatimuksen suunnittelutieteen täsmällisyydestä.

6. Suunnitteluprosessi

Suunnittelutiede on sovellettaessa lähtökohtaisesti iteratiivinen hakuprosessi (Hevner ja muut, 2004). Tietojärjestelmissä on usein mahdotonta määrittää täysin parasta tai optimaalista ratkaisua käytännön ongelmiin. Mahdollisten ratkaisujen määrä voi olla suuri tai täysin optimaalinen ratkaisu mahdoton määrittää. Heuristiset hakumenetelmät tähtäävät käyttökelpoisten ja hyvien ratkaisuiden tuottamiseen. Tästä näkökulmasta suunnittelutieteessä käytettävän artefaktin suunnittelu tähtää nimenomaisesti toimivan ja tehokkaan ratkaisun hakemiseen iteroimalla erilaisten vaihtoehtojen välillä parantaen artefaktia vaiheittain, kunnes hyvä ratkaisu on löytynyt. Ratkaisun toimivuus toimintaympäristössään on myös pystyttävä selittämään. Lisäksi hyvä ratkaisu on pystyttävä mittaamaan esimerkiksi todistamalla, että ratkaisu on lähellä optimaalista tai vertailemalla ratkaisua muihin vastaaviin.

Tutkimuksessa toteutettavan artefaktin voidaan ajatella muodostavan yksittäisen osan todellisen tietojärjestelmän kokonaisuudesta. Tällä ratkaisulla on pyritty luomaan mahdollisimman vertailukelpoinen tutkimustulos, jota voidaan hyödyntää muiden tietojärjestelmien suunnittelussa. Artefakti on suunniteltu niin, että sen avulla voidaan vertailla vaihtoehtoisia algoritmeja käyttäen samaa lähdeaineistoa ja näin saadaan matemaattisesti vertailukelpoisia tuloksia. Hyvä ratkaisu voidaan näin osoittaa selvästi vertailemalla tuloksia keskenään mittareiden avulla ja toteamalla onko uusi testattava algoritmi parempi kuin vertailualgoritmi tai aiempi iteraatio. Lisäksi iteraation avulla syntynyttä tietoa voidaan itsessään hyödyntää seuraavien järjestelmien suunnittelussa, riippumatta siitä saadaanko tuloksena verrokkialgoritmia parempaa lopputulosta. Algoritmien toiminta pyritään selittämään mahdollisimman tarkasti, jotta tutkimustulokset ovat toistettavissa ja algoritmeja voidaan hyödyntää muissa tietojärjestelmissä. Suunnitteluprosessi täyttää näin sille asetetut vaatimukset.

7. Tulosten kommunikointi

Hevner ja muut (2004) painottavat suunnittelutieteen tutkimuksen kommunikoinnissa sidosryhmien ymmärrystä. Teknologisesti orientoituneet lukijat tarvitsevat riittävät tiedot tutkimuksesta, jotta tulokset ovat toistettavissa ja näin hyödynnettävissä uudelleen sekä käytettävissä tulevaisuuden tutkimuksen pohjana. Organisaation johto taas tarvitsee oikeat tiedot päätöksenteon pohjaksi, jotta se voi tehdä oikeita päätöksiä organisaation resurssien käytöstä ja tietojärjestelmän sopivuudesta organisaation käyttöön. Tutkimus on lisäksi syytä kirjoittaa kohdeyleisö huomioiden.

Tutkimuksessa toteutettavan artefaktin tarkoituksena on vastata esitettyihin tutkimuskysymyksiin, jotka ovat luonteeltaan algoritmisia ja näin tutkimus on lähempänä teknologisesti orientoitunutta lähestymistapaa. Tulokset voivat kuitenkin olla merkityksellisiä myös liitettynä loppukäyttäjää lähempänä oleviin tietojärjestelmäkokonaisuuksiin ja sitä kautta tuloksilla voi olla merkitystä organisaation johdon kannalta. Esimerkiksi reitinhakua hyödyntävän tietojärjestelmän käyttökokemus voi parantua tehokkaammalla reitinhaulla. Suunniteltu artefakti kuvataan tässä tutkimuksessa mahdollisimman yksityiskohtaisesti, jotta tietoja voidaan hyödyntää myöhemmissä käyttökohteissa. Lisäksi mittaustulokset esitetään yksinkertaisesti osana yhteenvetoa (sekä kattavasti omassa luvussaan), jotta organisaation johdon on helppo ymmärtää tutkimuksen mahdollinen potentiaali omassa organisaatiossaan. Näistä näkökulmista tarkastellen tutkimus täyttää vaatimukset tulosten kommunikoinnista.

3.3 Algoritmien suorituskykymittarit

Edellä kuvattujen ohjesääntöjen mukaan algoritmien mittauksessa on käytettävä täsmällisiä arviointikriteereitä. Alla on esitelty kaikkien algoritmien toiminnassa käytettävät suorituskyky- ja laatumittarit. Mittarit perustuvat soveltuvilta osin aiempaan tutkimustietoon, joita muut tutkijat ovat hyödyntäneet omissa tutkimuksissaan. Mittausmetodeista on oleellista huomioida, että tämän tutkimuksen algoritmeilta ei odoteta matemaattista oikeellisuutta, jolloin on lisäksi perustelua mitata miten algoritmin tulokset poikkeavat varmasti oikeasta tuloksesta. Sanders ja Schultes (2007) suosittelevat artikkelissaan koejärjestelyä, jossa tuloksia verrataan yksinkertaisiin ja varmasti toimiviin ratkaisuihin, jolloin poikkeamat voidaan tunnistaa. Tähän tutkimuksen koejärjestelyissä sovelletaan samaa periaatetta vertaamalla saatuja tuloksia yksinkertaisiin verrokkialgoritmeihin.

Mittausmenetelmä perustuu satunnaisten pisteparien hyödyntämiseen, jolloin sama määrä reittejä haetaan eri verrokkialgoritmeilla. Pisteparien määrällä voidaan vähentää satunnaisen virheen vaikutusta suoritusajoissa ja lopputuloksissa hyödynnetään pääsääntöisesti kokonaisaikoja eli yhteenlaskettuja mittareita kaikille pistepareille. Verrokkialgoritmien välillä kaikki pisteparit pidetään samoina, jolloin erot mittaustuloksissa syntyvät algoritmien toiminnasta eikä pisteparien valintaprosessi näin vaikuta lopputuloksiin. Tarkempi prosessi pisteparien valinnasta ja mittausdatan tallennuksesta on kuvattu luvussa 4.

Algoritmien suorituskykyä kuvaavat mittarit ovat suoritusaika ja askelten määrä. Ensimmäinen kuvaa sitä miten kauan algoritmilla kestää suorittaa reitinhaku mitattuna millisekunneissa ja toinen kuvaa algoritmin käsittelemien pisteiden määrää. Käsitelty piste on siis algoritmin tietorakenteesta hakema pienimmän avaimen omaava arvo, jonka algoritmi käsittelyn jälkeen merkitsee loppuun käsitellyksi. Tällöin algoritmi on tehokkaampi mitä pienempi sen suoritusaika on ja mitä vähemmän askelia sen tarvitsee suorittaa. Keskeisenä erona näiden mittareiden välillä on niiden laitteistoriippuvuus.

Suoritusaika riippuu algoritmia suorittavan tietokoneen laskentatehosta ja muista teknisistä ominaisuuksista, kun taas vastaavasti askelten määrä riippuu ainoastaan algoritmista itsestään. Tällöin nämä kaksi mittaria yhdessä antavat kuvan algoritmien suoritusnopeudesta verrattuna toisiinsa. Näitä mittareita on hyödynnetty aiemmassa tutkimuksessa muun muassa artikkeleissa Cherkassky ja muut (1996), Ikeda ja muut (1994), Larkin ja muut (2014) sekä Zhan ja Noon (1998).

Algoritmin laatua kuvataan reitin kokonaispituudella ja matkustusajalla. Laadun näkökulmasta on keskeistä, että laskettua reittiä voidaan verrata parhaaseen mahdolliseen tulokseen, jolloin poikkeama voidaan mitata ja tunnistaa. Tästä syystä algoritmeja verrataan aina verrokkialgoritmiin, joka tuottaa optimaalisen tuloksen, jolloin testattavien algoritmien laadun ja suorituskyvyn suhdetta voidaan verrata. Reitin kokonaispituus ja matka-aika on otettu huomioon erillisinä mittareinaan, koska käytännöllinen sovellus hyötyy enemmän matka-ajasta mittarina, mutta verrokkina kokonaispituus on yksinkertaisempi mittari, jota vastaan algoritmeja voidaan verrata. Algoritmin keskimääräistä poikkeamaa voidaan arvioida usean pisteparin yhteenlaskettujen reittien pituuksien ja matka-aikojen erotuksella verrokkialgoritmiin nähden. Aiemmassa tutkimuksessa reitin pituuden poikkeamaa referenssistä hyödyntävät artikkelissaan esimerkiksi Jagadeesh, Srikanthan ja Quek (2002).

4. Testattava artefakti

Artefaktin tarkoituksena on tarjota mittausalusta, jonka avulla voidaan suorittaa kokeita reititysalgoritmeille ja tietorakenteille Digiroad-tieaineistolla. Nämä kokeet antavat vastaukset esitettyihin tutkimuskysymyksiin. Tässä luvussa on käsitelty Digiroad-aineistoa, mittausohjelman rakennetta ja toimintaa sekä tutkittavien algoritmien ja tietorakenteiden yhdistelmiä jaettuna kolmeen eri koejärjestelyyn. Digiroad-aineisto on koko Suomen kattava digitaalinen aineisto, joka sisältää tieverkoston yksityiskohtaiset tiedot ja siihen liitetyn metadatan.

Mittausohjelma on itsenäinen ajettava sovellus, jonka toiminnoiksi testattavat algoritmit ja tietorakenteet on ohjelmoitu. Sen keskeisinä toimintoina ovat Digiroad-aineiston luku tietokoneen muistiin annetuista lähdetiedostoista, testattavien pisteparien valinta sekä yhtenäisen alustan tarjoaminen eri algoritmien suoritukselle. Mittausohjelma sisältää myös mekanismit satunnaisten virheiden minimointiin ja mittausdatan tallennukseen. Mittausohjelma käyttää vain keskeistä osaa Digiroad-aineistosta, jota tarvitaan algoritmien ja tietorakenteiden testaukseen.

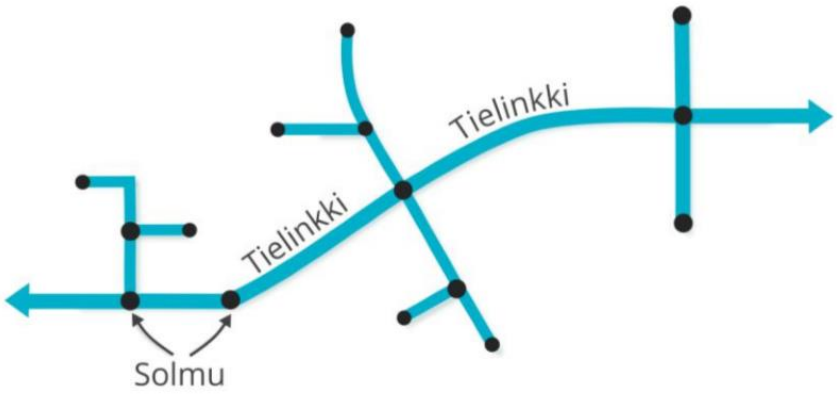
Artefaktin suorittamat kokeet on jaettu kolmeen koejärjestelyyn. Nämä koejärjestelyt tuottavat tuloksenaan tietoa, jota seuraava koejärjestely vuorostaan voi hyödyntää. Keskeiset algoritmit, jota koejärjestelyissä käytetään ovat Dijkstran algoritmi ja A* -algoritmi. Tietorakenteista tutkitaan binäärikekoa ja sen jatkokehitettyä versiota. Kokeiden tulokset on esitetty luvussa Evaluointi ja -tulokset.

Ensimmäisessä koejärjestelyssä pyritään selvittämään miten käytetty tietorakenne vaikuttaa reititysalgoritmin toimintaan ja miten tietorakennetta voidaan optimoida. Verrokkialgoritmina käytetään järjestämätöntä listaa, johon binäärikekoa verrataan keskeisillä algoritmeilla. Lisäksi testataan parannettua versiota binäärikeosta. Toisessa koejärjestelyssä testataan A* -algoritmin heuristiikkaa eri keskinopeuksilla Suomen tieaineistolla. Käytännön sovelluksen näkökulmasta reitin laskenta matka-ajan perusteella (etäisyyden sijasta) on usein perusteltua ja A* -algoritmin suosittu heuristiikka on hyödyntää laskennassa linnuntietä. Laskettaessa tätä heuristiikkaa matka-ajalle on kuitenkin tiedettävä arvioitu keskinopeus kohteeseen. Koejärjestelyn tarkoituksena on tutkia miten eri keskinopeudet vaikuttavat reitityksen laatuun ja suorituskykyyn.

Kolmannen koejärjestelyn tarkoituksena on parantaa reititysalgoritmin suorituskykyä hyödyntämällä hierarkkista reitinhakua. Kokeessa hyödynnetään aiemmista koejärjestelyistä saatuja tuloksia tietorakenteiden, algoritmien ja keskinopeuksien osalta. Testattava hierarkkinen algoritmi hyödyntää Digiroad-aineiston metadataa ja pyrkii sen avulla vähentämään algoritmin tekemää työn määrää, jotta reitinhaku nopeutuu. Koejärjestelyssä suoritetaan testit usealle eri variaatiolle algoritmista optimaalisen ratkaisun löytämiseksi.

4.1 Digiroad-aineistotietokanta

Digiroad on tietojärjestelmä, joka sisältää tiedot koko Suomen tie- ja katuverkon keskilinjageometrista sekä siihen liitetyistä ominaisuustiedoista (metatieto). Tärkein Digiroad-aineistosta saatava tieto on väylän keskilinjageometria, joka kuvaa tien reittiä alku- ja loppupisteiden välillä seuraten nimensä mukaisesti tien keskiviivaa. Digiroad tietolajien kuvaus määrittelee keskilinjageometrian seuraavasti: ”Digiroadin keskilinjageometrian muodostavat teiden, katujen, kevyen liikenteen väylien, rautateiden ja lauttayhteyksien keskilinjojen sijaintia kuvaavat murtoviivat” (Digiroad, 2019b, s. 5). Kuvassa 8 on esitetty havainne tielinkeistä ja niitä yhdistävistä solmuista (pisteistä). Tieverkostoon liittyvää metatietoa on kiinnitetty liikenne-elementteihin ja lineaarisesti referoituihin ominaisuustietoihin. Tällaisia metatietoja ovat esimerkiksi nopeusrajoitus, tien toiminnallinen luokka ja bussipysäkin sijainti. Näiden metatietojen kuvaus perustuu Digiroad tietolajien kuvaukseen (Digiroad, 2019b).

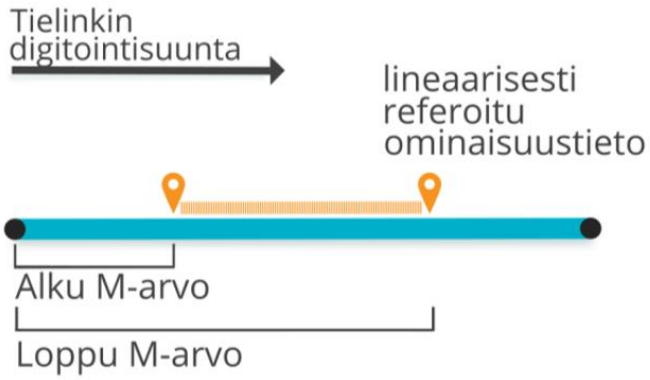


Kuva 8. Konseptikuva Digiroad-aineiston Liikenne-elementeistä ja päätepisteistä. Keskilinjageometria muodostuu tielinkkiä kuvaavien sinisten kaarien sijaintitiedoista. Liikenne-elementit (tielinkit) on merkattu sinisellä sekä pisteet mustalla. Lähde: Liikennevirasto, Digiroad-aineiston tietolajien kuvaus.

Liikenne-elementti on lohko, joka määrittelee keskilinjageometriasta muodostuvan väylän. Toisin sanoen liikenne-elementti on keskilinjageometrian pienin itsenäinen yksikkö – se katkeaa aina väylän liittymäkohdissa. Liikenne-elementti voi edustaa esimerkiksi maantietä, katua, yksityistietä, lauttayhteyttä tai monia muita väylätyyppejä. Tie-elementti on yhteisnimitys nimenomaisesti tieverkostoa esittäville väylätyypeille (tie-elementti ei voi olla rautatie tai lauttaelementti). Liikenne-elementin keskilinjageometria sisältää kaksi keskeistä ominaisuutta: pisteet, joista geometria muodostuu, sekä digitointisuunta eli geometriapisteiden järjestys. Digitointisuunta, eli järjestys, johon keskilinjageometrian pisteet on järjestetty, on merkittävä esimerkiksi silloin kun tie-elementti on yksisuuntainen tai siihen on liitetty kääntymismääräys. Muut liikenne-elementtiin liittyvät tiedot on merkitty siihen liittyvinä ominaisuustietoina (joihin viitataan yleisesti metadatana).

Lineaarisesti referoitu ominaisuustieto koostuu keskilinjageometriasta ja sen tarkoituksena on sitoa yhteen toisiinsa liittyvät tiedot tielinkissä. Referoidun ominaisuustiedon avulla voidaan osoittaa pistemäisiä paikkoja tai viivamaisia ominaisuuksia tielinkin sisällä. Yksinkertainen esimerkki pistemäisestä tiedosta on bussipysäkin sijainti tielinkissä. Viivamaisella ominaisuustiedolla voidaan esimerkiksi ilmaista, että annettu nopeusrajoitus on voimassa puoleen väliin tielinkin pituutta, jonka jälkeen nopeusrajoitus muuttuu toiseksi. Tällöin itse tielinkin geometria ei katkea

useampaan osaan, vaikka tien jokin keskeinen ominaisuus muuttuu. Kuten liikenne-elementeilläkin, myös referoiduilla ominaisuustiedoilla on digitointisuunta, jolloin esimerkiksi bussipysäkin sijainti oikealla kaistalla voidaan ilmaista yksiselitteisesti. Referoitujen ominaisuustietojen määrittely tapahtuu suhdelukujen avulla (mitta-arvo eli m-arvo), joiden perusteella voidaan yksilöidä sijainti tielinkissä. Tällaista suhdeluvuilla määriteltyä aluetta kutsutaan nimellä segmentti. Esimerkiksi tielinkin keskiosassa voi olla päällystettyä tietä, jolloin tämä tieto on merkittynä vain osalle tielinkistä – toisin sanoen päällystetty tie muodostaa segmentin tielinkin m-arvojen välillä. Esimerkiksi tiesegmentin alkupisteen suhdeluku voi olla 50, loppupisteen 100 ja päällystetty tie sijaita välillä 60–75. Havainnekuva lineaarisesti referoiduista ominaisuustiedoista on esitetty kuvassa 9.

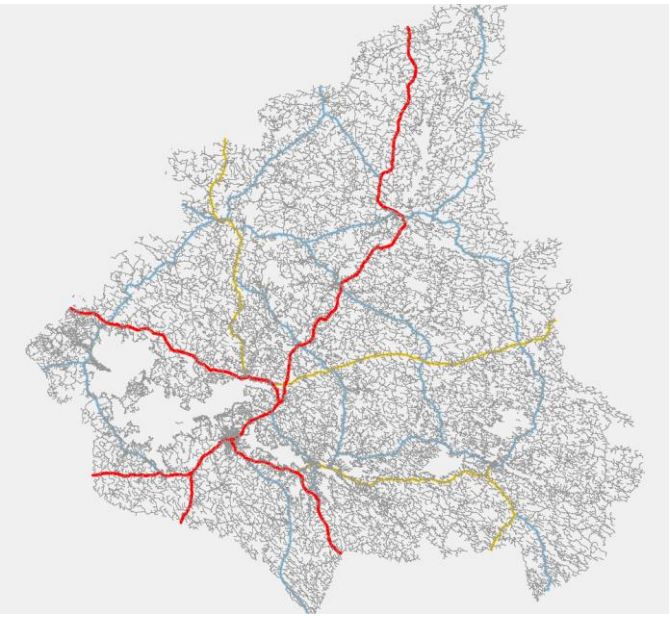


Kuva 9. Esimerkkikuva yksittäisestä tielinkistä, joka sisältää lineaarisesti referoitua ominaisuustietoa. M-arvot ilmaisevat ominaisuustiedon alku- ja loppupistettä suhteessa tielinkkiin. Lähde: Liikennevirasto, Digiroad-aineiston tietolajien kuvaus.

Digiroad-aineisto on saatavilla kahdessa pääasiallisessa muodossa: R ja K. Digiroad R on rakennettu niin, että liikenne-elementit on yhdistetty lineaarisesti referoiduiksi ominaisuustiedoiksi ja näille elementeille on dynaamisesti liitetty ominaisuustietoja. Vastaavasti taas Digiroad K -toimitusmuodossa liikenne-elementit toimitetaan ilman yhdistämistä. Tällöin ominaisuustiedot on merkitty jokaiselle liikenne-elementille erikseen ja viivamaisen ominaisuustiedon muuttuessa myös tielinkki on katkaistu. Toimitusmuodon valinta riippuu pääsääntöisesti loppukäyttäjän käyttötarkoituksesta aineistolle. Tässä tutkimuksessa on käytetty Digiroad K -toimitusmuotoa, jolloin aineistoa luettaessa tielinkit päättyvät aina risteyksiin sekä viivamaisten ominaisuustietojen muutoksiin.

Toiminnallinen luokka määrittää tien liikenteellistä tärkeyttä. Esimerkiksi kansalliset valtatiet on merkitty omalla toiminnallisella luokituksellaan, joka erottaa ne muista teistä kuten vaikkapa kantatie tai seututie. Tätä tietoa voidaan hyödyntää reitityksessä esimerkiksi pitkien matkojen osalta niin, että reititysalgoritmi ohjaisi mahdollisimman pitkän reitin toiminnalliselta luokaltaan tärkeälle tielle. Lisäksi reititysalgoritmin suorituksen aikana voitaisiin jättää käsittelemättä joitain tieosuuksia, jotka todennäköisesti eivät ole optimaalisella reitillä perustuen toiminnalliseen luokkaan. Kuvassa 10 on esitetty Kainuun alueen tieverkosto, jossa tärkeimmät toiminnalliset luokat on korostettu väreillä. Toinen reitityskäytössä potentiaalisesti hyödyllinen tieto on liikenne-elementin kuntakoodi, jonka vertailu ohjelmallisesti on nopeampaa kuin etäisyyden laskenta toiseen pisteeseen ja jota voidaan hyödyntää hierarkkisen rakenteen luomisessa. Tiedon avulla voidaan päätellä tilanteita, jolloin reitinhaku on kaukana alku- ja loppupisteestä, jolloin voidaan muokata reititysalgoritmin toimintaa vastaavasti kuten toiminnallisenkin luokan osalta. Lisäksi nopeusrajoitus on keskeinen tieto, joka

määrittää miten nopeasti tieosuudella voidaan ajaa. Tämän tiedon käyttäminen on oleellista, kun haetaan matka-ajaltaan optimaalista reittiä tai halutaan esittää reittejä, jotka vastaavat reaalimaailmaa.



Kuva 10. Esimerkkikuva Digiroad-aineistosta, jossa Kainuun tieverkoston toiminnallisia luokkia on korostettu. Luokan 1 tiet (valtatiet) on merkitty punaisella, luokan 2 tiet (kantatiet) on merkitty keltaisella ja luokan 3 tiet (seututiet) on merkitty sinisellä. Tieaineisto Digiroad (2019a), joka on käsitelty Easy GIS .NET ohjelmalla.

Huomioitava seikka aineiston tielinkeistä on se, että vaikka ne tyypillisesti alkavat ja päättyvät toisen tielinkin (esimerkiksi risteys) alku- tai loppupisteisiin näin ei kuitenkin ole kaikissa tapauksessa. Tielinkki katkeaa lisäksi tilanteissa, joissa tielinkki sisältää viivamaista metatietoa ja joka muuttuu tielinkin alku- ja loppupisteiden välillä. Esimerkki tällaisesta muuttuvasta tiedosta on toiminnallinen luokka.

Digiroad-aineiston kaikkien toimitusmuotojen koordinaattijärjestelmänä on käytössä EUREF-FIN koordinaattijärjestelmä ja UTM-projektioon perustuva ETRS-TM35FIN -projektio (EPSG: 3067). Koordinaatit ilmoitetaan metreinä ja sijaintitietojen korkeusjärjestelmä on N60. EUREF-FIN koordinaatiston ero yleisesti satelliittipaikannuksessa käytettävään WGS84 koordinaatistoon on alle metri, jolloin ero useimmissa käyttökohteissa on merkityksetön (Digiroad, 2019b). Digiroad-aineiston koordinaatit ovat suoraan hyödynnettävissä esimerkiksi reitinhakualgoritmeissa ilman koordinaattimuunnosta, johtuen metripohjaisesta tasokoordinaatistosta.

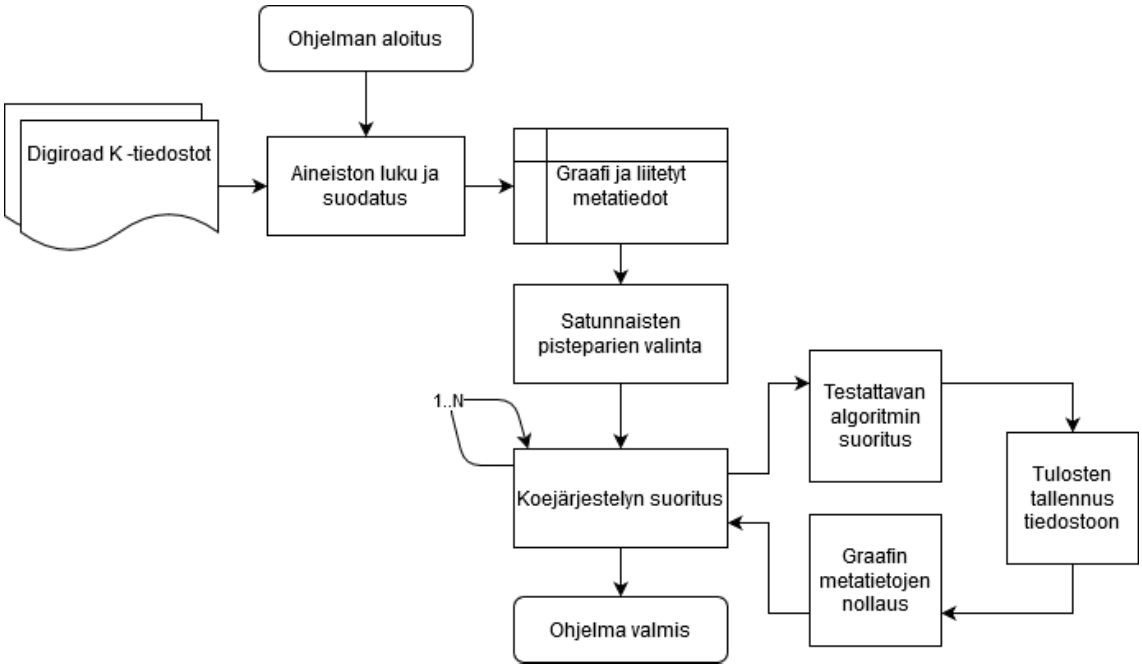
Liikennevirran suunta kertoo, onko tieosuus mahdollista ajaa molempiin suuntiin vai onko kyse yksisuuntaisesta tiestä. Navigointikäytössä tämä tieto on oleellinen reitin oikeellisuuden takaamiseksi. Digiroad-aineisto voi sisältää yksisuuntaisia teitä niin, että kulkusuunta on rajoitettu digitointisuunnan mukaisesti tai sitä vastaan, jolloin metadatan attribuutti ilmaisee, kummin päin rajoitus on voimassa.

Tielinkin tyyppi erottaa tieosuudet toisistaan. Tyyppitiedolla mahdollistetaan esimerkiksi autotien erottaminen kevyenliikenteen väylästä, joka on oleellinen tieto navigointikäytössä. Silta, alikulku ja tunnelit on kuvattu omalla metatiedollaan, joka on merkityksellistä, kun esimerkiksi risteyksiä on useammassa tasossa. Näin voidaan erottaa eri tasojen risteykset toisistaan.

Digiroad-aineisto sisältää myös osoitetiedot, joita voitaisiin käyttää osoitehakujen työkaluna. Lisäksi aineisto sisältää myös pisteisiin sidottuja tietoja, kuten esimerkiksi julkisen liikenteen pysäkkien sijaintitiedot. Vaikka nämä tiedot ovatkin keskeisiä itse aineiston kannalta ne eivät ole merkittäviä tässä mittausohjelmassa tutkittavien algoritmien näkökulmasta.

4.2 Mittausohjelman rakenne ja toiminta

Tässä kappaleessa on kuvattu mittausohjelman eli artefaktin keskeiset toiminnot. Mittausohjelman tarkoituksena on luoda alusta eri algoritmien ja tietorakenteiden testaamiseen Digiroad-aineistolla. Artefakti edustaa myös mahdollista käyttöympäristöä käytännön sovelluksen alijärjestelmänä. Mittausohjelma voidaan jakaa kolmeen toiminnalliseen osaan, jotka koostuvat aineiston luvusta graafiksi, pisteparien valinnasta ja ajan mittauksesta sekä varsinaisten algoritmien suorituksesta ja tulosten tallennuksesta. Ohjelman osien toiminta on esitetty korkealla tasolla kuvassa 11.



Kuva 11. Havainnekuva artefaktin rakenteesta, joka osoittaa miten eri moduulit liittyvät toisiinsa.

Ensimmäinen osa on Digiroad-tieaineiston datan luku tiedostoista tietokoneen muistiin, jossa se muodostaa graafin, jota reititysalgoritmit voivat käyttää. Muistiin ladataan ainoastaan henkilöautolla ajettavat ja tavoitettavat tiet Suomen tieaineistosta. Graafi koostuu pisteistä sekä niitä yhdistävistä kaarista sisältäen reititysalgoritmien tarvitseman metadatan. Graafi organisoidaan tietokoneen muistiin niin, että algoritmit voivat käyttää sitä viiveettömästi eli yhteen pisteeseen liitetyt kaaret ja niiden kautta seuraavat pisteet ovat haettavissa vakioajassa. Toisena osana on testattavien satunnaisten pisteparien valinta algoritmien käyttöön sekä ajan mittausrakenne. Näiden avulla pyritään vähentämään satunnaisten virheiden vaikutusta mittaustuloksiin sekä tekemään ohjelman suorituksesta toistettava. Kolmantena osana on algoritmien toiminta mittausohjelmassa sekä mittausdatan tallennus tiedostoon. Algoritmien suorituksessa pyritään varmistamaan, että kaikkia algoritmeja käsitellään identtisesti sekä ettei edellisen algoritmin suoritus vaikuta seuraavaan algoritmiin.

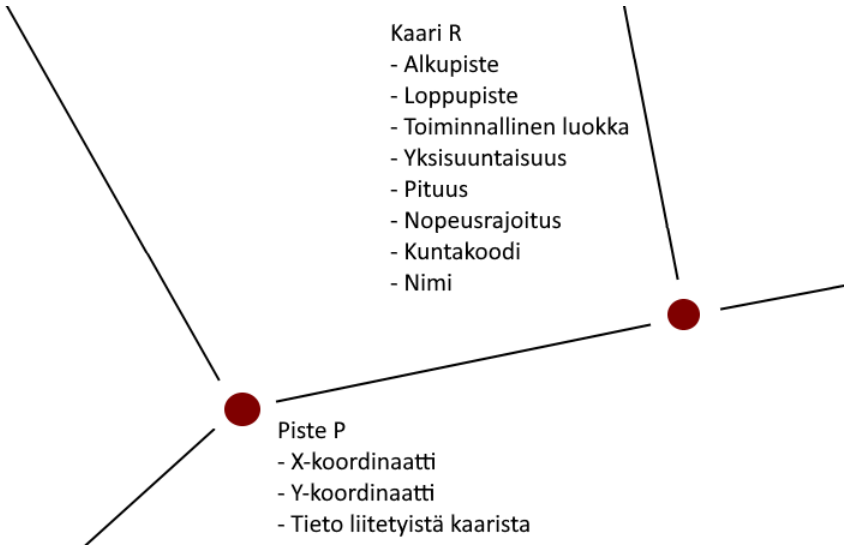
4.2.1 Datan luku ja suodatus

Datan lukuprosessin tarkoituksena on muodostaa Digiroad-aineistosta tietokoneen muistiin pisteistä ja kaarista koostuva graafi, jota voidaan käyttää reititysalgoritmin toiminnassa ja joka sisältää tarvittavat metatiedot optimaalisen reitin löytämiseen. Digiroad-aineisto on toimitusmuodossaan kuvattu kuten tietokannan taulut eli se sisältää useita tauluja, jotka koostuvat riveistä ja sarakkeista. Yksi tieosuus voi koostua yhdestä tai useammasta rivistä tietokannassa. Syynä useampien rivien käyttöön samalla tieosuudella on esimerkiksi nopeusrajoitus, joka muuttuu tien keskivälissä. Digiroad-aineiston sisäiset yksilöivät tunnisteet linkittävät eri taulut toisiinsa. Lukuoperaatiossa Digiroad-aineisto luetaan yksi maakunta kerrallaan sekä tuodaan se olioista koostuvaan formaattiin niin, että eri oliot on linkattu toisiinsa. Tällöin algoritmin lukiessa olioita se voi hakea yhdestä pisteestä tai kaaresta siihen yhteydessä olevat toiset kaaret ja pisteet ilman, että alkuperäiseen binääritiedostoon on tarpeen tehdä hakuja. Toisin sanoen graafin pisteeseen S liitetyt muut pisteet on mahdollista lukea vakioajassa, jolloin hakuaika tai sen vaihtelut eivät vaikuta suorituskykyyn tai mittauksiin. Lisäksi lukuoperaatiossa sama piste luetaan aineistosta useita kertoja (yhden tieosuuden päättymispiste on yleensä myös seuraavan tieosuuden alkupiste) ja prosessin aikana nämä samaa koordinaattia osoittavat pisteet yhdistetään.

Alla kuvattavan käsittelyprosessin jälkeen koko Suomen tieverkko luettuna mittausohjelman muistiin graafiksi sisältää 2 631 714 kaarta ja 2 475 695 pistettä. Tällöin jokaisella pisteellä on keskimäärin 1,06 liitettyä kaarta, joka on suhteellisen pieni suhdeluku verrattuna aiempaan tutkimukseen (Goldberg & Harrelson, 2005; Jacob ja muut, 1999; Zhan & Noon, 1998). Tämä voi osaltaan selittyä Suomen maantieteellä sekä sillä, että määrällisesti suurin osa tieosuuksista on pieniä tontille johtavia yksityisteitä. Lisäksi Digiroad-aineiston K-toimitusmuodossa tieosuuksia on katkottu metatietojen mukaisesti osiin eli kaikki tieosuudet eivät pääty risteyksiin. Hallinnollisten luokkien väliset suhteet on kuvattu tarkemmin luvun 4.3.4 taulukossa 4.

Mittausohjelman ohjelmointikielenä käytetään yleisesti tunnettua Microsoftin C# kieltä ja sen määritystä 5.0 (Microsoft, 2012). Ohjelmakoodi on kehitetty Microsoft Visual Studio 2012 -ohjelmistolla. Binääritiedon luku väliaikaiseen tietokoneen muistissa olevaan formaattiin (joka mallintaa alla olevia tietokanta tauluja) on toteutettu Easy GIS .NET (2019) -kirjastolla. Kirjasto lukee alkuperäisen binääridatan rivi kerrallaan ja tarjoaa koodista rajapinnan tietojen lukemiseen. Data luetaan alkuperäisistä tiedostoista vain yhden kerran ja muodostettua graafia hyödynnetään useita kertoja mittausohjelman ajon aikana niin, ettei alkuperäinen luettu data muutu lukuprosessin valmistumisen jälkeen. Koko Suomen kaikkien maakuntien aineiston luku kestää joitain minuutteja riippuen mittausohjelmaa suorittavan tietokoneen nopeudesta.

Digiroad-tieaineisto luetaan Easy GIS .NET (2019) kirjastolla lukuprosessin aikana kahdesti. Ensimmäisellä läpikäynnillä muodostetaan muistiin väliaikainen rakenne, joka sisältää avainarvo -pareina tieosuuden yksilöllisen tunnisteen, nopeusrajoituksen ja tieosuuden pituustiedot. Tämä lukuprosessi tehdään, koska osa tieosuuksista on jaettu usealle riville, jolloin nopeusrajoituksesta on tarpeen laskea painotettu keskiarvo sekä pituudesta yhteenlaskettu pituus koko tieosuudelle ottaen huomioon kaikki samaa tieosuutta koskevat tiedot. Toisella lukukerralla haetaan tieosuuden alku- ja loppupisteet, muut metatiedot sekä välimuistista keskimääräinen nopeusrajoitus ja tieosuuden yhteenlaskettu pituus. Lukuprosessin lopputuloksena saadaan aineistosta seuraavat tiedot: Tieosuuden alku- ja loppupiste, keskimääräinen nopeusrajoitus, tieosuuden yhteenlaskettu pituus, hallinnollinen luokka, tien kuntakoodi sekä yksisuuntaisuussäännöt. Nämä tiedot on esitetty kuvassa 12.



Kuva 12. Digiroad-aineistosta mittausohjelmaan luettavat tiedot sekä pisteiden ja kaarien väliset linkitystiedot.

Aineiston luvun yhteydessä siitä suodatetaan osa materiaalista pois. Lähtökohtana on lukea kaikki sellaiset tieosuudet, jotka ovat ajettavissa tavallisella henkilöautolla, koska näin saatu tieverkosto on maantieteellisesti kattavin mahdollinen. Toisin sanoen kevyen liikenteen väylät ja esimerkiksi lauttareitit jätetään suodatusvaiheessa pois. Teknisesti olisi mahdollista myös ottaa kaikki tieosuudet mukaan ja suodattaa dynaamisesti algoritmin suorituksen aikana tarpeettomat tieosuudet pois riippuen millaisilla parametreilla algoritmia ajetaan. Tämä jätetään kuitenkin mahdolliseksi jatkokehitysaiheeksi. Lisäksi tieverkostossa on olemassa tilanteita, jossa saman alku- ja loppupisteen välissä on reititysalgoritmin näkökulmasta useampia vaihtoehtoisia tieosuuksia. Näistä merkittävin esimerkki ovat eri valtateiden varrella olevat lyhyet lepopaikat tai muut vastaavat tienvarressa olevat palvelut. Aineiston lukuvaiheessa nämä suodatetaan pois, koska reititysalgoritmit eivät tyypillisesti pysty käsittelemään useita kaaria samojen pisteiden välillä. Suodatuksessa varmistetaan, että tärkeämmän toiminnallisen luokan tieosuus jää mukaan aineistoon.

Mittausohjelman reititysalgoritmit suoritetaan aina kahden pisteen välillä. Algoritmin suoritus ei ota huomioon mahdollista tarkempaa paikkaa tieosuuden sisällä, kuten esimerkiksi vaikkapa talonumeroa, vaikka Digiroad-aineisto sisältääkin talonumerotiedot. Johtuen tästä rajauksesta nousevat esille aineistossa esiintyvät tieosuudet, jotka alkavat ja päättyvät samaan pisteeseen. Tällaisia tieosuuksia ovat esimerkiksi ympyrän tai silmukan muotoiset ajoneuvon kääntöpaikat, jotka tyypillisesti sijaitsisivat päällystämättömien teiden päässä tai joissain harvoissa tapauksessa myös kaupungeissa. Tieaineiston luvussa tällaiset tieosuudet suodatetaan pois, koska nämä tieosuudet liittyvät aina toisiin tieosuuksiin (eli kyseinen koordinaatti on jo mukana graafissa), jolloin vaikutusta aineiston käyttöön reititysalgoritmin näkökulmasta ei ole. Kääntöpaikkojen käsittely olisi tarpeen ainoastaan, jos reititystä tehtäisiin muihin kuin tieosuuden alku- tai loppukoordinaatteihin.

Toinen tieverkostoille tyypillinen ominaisuus ovat yksisuuntaisesti ajettavat tieosuudet. Kuvattuna asiaa graafin G näkökulmasta, jos pisteestä P_1 on olemassa kaari pisteeseen P_2 se ei välttämättä tarkoita, että kaarta on olemassa toiseen suuntaan. Digiroad-aineisto sisältää tiedon tieosuuden yksisuuntaisuudesta niin, että tieosuus on joko kaksisuuntainen, se on yksisuuntainen digitointisuunnan mukaisesti tai yksisuuntainen digitointisuuntaa vastaan. Mittausohjelma sisältää kuitenkin vain tiedon siitä onko tieosuus yksisuuntainen vai kaksisuuntainen. Tieto luetaan aineistosta sekä käsitellään

niin, että digitointisuuntaa vastaan olevassa yksisuuntaisuudessa digitointisuunta käännetään vastakkaiseksi ja kyseinen tieosuus merkataan yksisuuntaiseksi, jolloin yksisuuntaisuus on aina tieosuuden alkupisteestä loppupisteeseen. Yksisuuntainen tieosuus otetaan huomioon algoritmien toiminnassa niin, että luettaessa pistettä P graafista palautetaan ainoastaan kaksisuuntaiset sekä digitointisuunnan mukaiset mukaan sallitut yksisuuntaiset tieosuudet.

Reitinhakualgoritmien näkökulmasta kääntymismääräyksiä voidaan pitää vastaavanlaisena ominaisuutena kuin yksisuuntaisia tieosuuksia. Algoritmin suorituksen laajentaessa pisteen P kääntymismääräyksellä rajattu kaari on jätettävä huomiotta, kun palautetaan graafista pisteeseen P kaarilla liitettyjä muita pisteitä. Digiroad-aineisto sisältää tarkat kääntymismääräystiedot, mutta niitä ei ole erikseen tuotu mukaan mittausohjelman graafiin ja tämän käsittelyn implementointi ja vaikutusten arviointi jätetään jatkokehitysaiheeksi.

Lisäksi aineistosta suodatetaan pois sellaiset pisteet, jotka eivät ole tavoitettavissa henkilöautolla ajettaessa seuraavasti. Oletetaan että tieaineisto luetaan graafiin G ja graafissa olevia pisteitä kuvataan P_n. Algoritmin suorituksen näkökulmasta teemme oletuksen, että pisteestä P_1 on aina olemassa mahdollinen reitti pisteeseen P_x. Tämän oletuksen täyttämiseksi graafissa olevista pisteistä suodatetaan pois sellaiset pisteet, jotka eivät ole tavoitettavissa alkupisteestä P_a. Alkupiste on satunnaisesti valittu hallinnollista luokkaa 1 (valtatie) edustava piste tieaineistossa, joka on valittu keskeiseltä osalta Suomen tieverkostoa. Suodatus tehdään suorittamalla Dijkstran algoritmi ilman päättymissääntöä sekä merkkaamalla kaikki pisteet, jotka algoritmi käy läpi. Tällöin algoritmin suorituksen jälkeen voidaan poistaa graafista sellaiset pisteet, joita ei ole merkattu. Tämä käsittely suoritetaan ennen varsinaisia mittauksia eikä sen suoritusaika vaikuta mittausohjelman tuloksiin, vaan sen ainoa funktio on muodostaa yhtenäinen graafi. Esimerkki suodattamalla poistettavasta aineistosta ovat esimerkiksi saarissa sijaitsevat yksityistiet, jotka eivät ole tavoitettavissa henkilöautolla muusta tieverkostosta tai ne ovat saavutettavissa ainoastaan keinoin (esimerkiksi lautta) jotka ovat aiemmin suodatettu pois.

Reitityskäytössä erityistä tarvetta muuntaa koordinaatteja alkuperäisestä EUREF-FIN koordinaatistosta toiseen ei ole. Geometrian kannalta tärkeimmät aineistosta saatavat tiedot ovat liikenne-elementin alku- ja loppupiste, kyseisen elementin (tieosuuden) pituus sekä sen nopeusrajoitus ja lisäksi kytkentä muihin tieosuuksiin. Tällöin ei ole tarvetta aineistoa hyödyntäessä laskea uudestaan tieosuuden pituutta alkuperäistä koordinaattiketjua seuraten. Toisin sanoen koordinaateilla ei siis ole tarpeen suorittaa laskentaa eikä yhdistää muihin aineiston ulkopuolisiin tietoihin, jolloin pisteen koordinaatit toimivat reititysalgoritmin näkökulmasta uniikkeina tunnisteina. Tilanne muuttuisi, jos esimerkiksi reitityksen yhteyteen rakennettaisiin sovellusta, joka vaikkapa hyödyntäisi GPS-paikannusta, jolloin koordinaattien muunnos joko aineiston tai GPS-paikantimen (WGS84 -koordinaatisto) osalta tulisi tarpeelliseksi.

Eräs piirre Digiroad-aineistossa on se, ettei kaikille tieosuuksille (tyypillisesti pienet yksityistiet) ole määritelty tarkkaa pituutta tai nopeusrajoitusta. Tästä syystä tieosuuksien pituudet muodostetaan datan luvun yhteydessä laskemalla ne ohjelmallisesti keskilinjageometrian perusteella, jolloin saadaan tarkka tien pituus metreinä. Lisäksi niiden tieosuuksien kohdalla, joihin ei ole merkitty nopeusrajoitusta, käytetään yleisrajoituksena 50 km / h. Tätä valinta perustuu Tiehallinnon (2009) julkaisemaan suunnitteluohjeistukseen taajama-alueen yleisrajoituksesta.

4.2.2 Pisteparien valinta ja ajan mittaus

Tutkimustulosten paikkansapitävyyden kannalta on tärkeää, että eri algoritmeja voidaan verrata toisiinsa täsmällisesti. Tällöin artefaktin on varmistettava, että käytettävä graafi ja graafista valitut pisteet reititykselle ovat täsmälleen samat ja kellonajan mittaus suoritetaan tarkasti. Lisäksi halutaan varmistaa, että pistepareja valitessa ei luoda tahattomasti painotuksia, jotka vääristäisivät mittaustuloksia ja suoritusaikaa mitattaessa on tärkeää valita metodi, joka on mittaustarkkuudeltaan riittävä. Yleisesti ottaen algoritmien suorituksessa satunnaisten mittausvirheiden vaikutus tuloksiin halutaan minimoida. Tällaiset virheet voivat johtua vaikkapa käytetyn ohjelmointikielen toiminnasta, käyttöjärjestelmästä tai tietokoneen suoritusnopeuden vaihtelusta. Esimerkiksi suorittimen vaikutusta reititysalgoritmien toimintaan ovat käsitelleet artikkelissaan Larkin ja muut (2014). Tällöin on järkevää mitata useita satunnaisia pistepareja, jolloin pienet satunnaiset vaihtelut eivät vaikuta kokonaisuuden kannalta lopputuloksiin merkittävästi. Vastaavaa lähestymistapaa ovat käyttäneet artikkelissaan Delling ja muut (2011), jossa he valitsevat arviointiin aineistosta suuren määrän satunnaisia alku- ja loppupisteitä.

Datan lukuvaiheen yhteydessä artefakti säilyttää viittauksen kaikkiin graafiin kuuluviin pisteisiin. Tällöin indeksinumeroa käyttämällä voidaan hakea mikä tahansa haluttu piste. Yhtä reittihakua varten tarvitaan luonnollisesti kaksi pistettä, joiden välille reitti haetaan. Edellä mainittujen vaatimusten perusteella halutaan hakea satunnaisia ohjelmallisesti arvottuja pisteitä, jolloin valinnassa ei ole tahattomia ihmisestä johtuvia painotuksia. Toisaalta pisteiden valinnan halutaan olevan toistettavissa, jolloin tulokset voidaan tarvittaessa tarkistaa myöhemmin. Näistä syistä pisteet valitaan hyödyntämällä C# -kielen vakiokirjaston *System.Random* -luokkaa (Microsoft, 2019a), joka tarjoaa mahdollisuuden hakea pseudosatunnaisia lukuja käyttämällä siemenarvoa. Tällöin aina haettaessa uutta pistettä tarkistetaan tallennettujen pisteiden kokonaismäärä ja haetaan *Random* -luokan avulla pseudosatunnainen numero nollan ja pisteiden määrän välillä. Haetut pisteet ovat samoja ohjelman suoritusten välillä, koska ohjelma käyttää aina samaa siemenarvoa. Haettua numeroa käytetään indeksinä, jonka avulla haetaan graafista tallennettu piste. Prosessi toistetaan niin monesti kuin pistepareja tarvitaan, jolloin kahdesta peräkkäisestä pisteestä muodostuu aina yksi alku- ja loppupisteen muodostama pistepari.

Tyypillinen mittari algoritmien suorituskyvylle on suoritusajan mittaaminen millisekunneissa. Tämä on myös tyypillinen tapa mitata reititysalgoritmeja, kuten luvussa 3.3 on käsitelty. Artefakti on suunniteltu mittaamaan ainoastaan algoritmien suoritusaikaa, jolloin esimerkiksi ohjelman käynnistys ja datan luku eivät sisälly mitattuihin aikoihin. Suorituksen aikana algoritmit kirjoittavat tietoja suorituksen tilasta graafiin, jolloin suorituksen loputtua nämä tiedot on nollattava ennen seuraavan algoritmin suoritusta tai reitin hakua. Tätä nollausprosessia ei myöskään lasketa mukaan suoritusaikoihin. Näin pyritään varmistamaan mittausten vertailukelpoisuus keskenään. Ajan mittaaminen tapahtuu hyödyntämällä C# -kielen vakiokirjaston *System.Diagnostics.Stopwatch* -luokkaa (Microsoft, 2019b). Tämä luokka mahdollistaa korkean tarkkuuden mittauksen sekä toteutuneen suoritusajan hakemisen millisekunneissa. Tällöin omaa koodia suoritusajan laskemiseen kellonaikojen perusteella ei tarvita. Dokumentaation perusteella *Stopwatch* -luokan tarjoaman aikamääreen tarkkuus riippuu laitteistosta ja käyttöjärjestelmästä missä ohjelmaa ajetaan. Tästä syystä artefaktin koodi tarkistaa ohjelman käynnistyksen yhteydessä, että korkean aikatarkkuuden tila on päällä.

4.2.3 Algoritmien suoritus ja mittausdatan tallennus

Algoritmien suoritus mittausohjelmassa perustuu siihen, että mitattavia algoritmeja suoritetaan yksi kerrallaan. Myös algoritmit on ohjelmoitu yksisäikeisesti, jolloin niistä saatava data on näin vertailukelpoista myös suoritusajan osalta. Käytettävät Dijkstran -algoritmi ja A* -algoritmi tallentavat suorituksensa aikana metadataa graafin pisteisiin. Tällaisia kenttiä ovat esimerkiksi algoritmin mittaama matka-aika sekä heuristiikka-arvot. Tästä syystä graafissa on erillisiä metadatakenttiä, jotka on asetettu jokaiselle pisteelle muuttujiksi. Ennen algoritmin suoritusta nämä muuttujat on nollattava, jotta ne eivät vaikuta seuraavan algoritmin toimintaan. Mittausohjelmassa on kaikille algoritmeille yhteinen suoritusfunktio, jonka tehtävä on huolehtia metadatakenttien nollauksesta ja vasta sen jälkeen ajaa algoritmi. Tällöin kaikkien algoritmien osalta metadata nollataan samalla tavalla. Suoritusajan mittaus tapahtuu vastaavasti itse algoritmin reititysfunktio alussa. Tällöin valmisteleva koodi, mukaan luettuna metadatan nollaus, suoritetaan ennen reititysalgoritmin ajan mittauksen alkua, jolloin se ei vaikuta lopputuloksiin. Vastaavasti ajan mittaus pysäytetään reititysfunktion lopussa. Tällöin siis tuloksien tallennus tai muut toimenpiteet eivät vaikuta ajan mittaukseen.

Kaikkien algoritmien toiminnasta kirjoitetaan keskeiset mittaustulokset tekstitiedostoon. Jokaisesta yksittäisestä reitistä tallennetaan mahdollista myöhempää tarkastelua varten alkupiste, loppupiste sekä kaikki luvussa 3.3 kuvatut mittaustiedot. Samalla mittausohjelma pitää kirjaa kaikkien mitattavien pisteparien kumulatiivisista mittaustiedoista. Näitä kumulatiivisia tietoja käytetään mittaustulosten esittämiseen. Tietojen tallentamiseen käytetään avoimen lähdekoodin NLog -kirjastoa (NLog, 2020). Suoritus tai väliaikatietoja ei kuitenkaan tallenneta algoritmien suorituksen aikana vaan vasta suorituksen jälkeen, jolloin voidaan varmistaa, että tallennusprosessi ei vaikuta mittaustuloksiin.

Kaikki testit on ajettu tietokoneella, joka on varustettu AMD Ryzen 7 3700X -suorittimella ja jossa on käytettävissä 16 gigatavua 3200 Mhz muistia kahdessa muistikanavassa. Suorittimessa on yhteensä 36,5MB välimuistia jaettuna L1-L3 tasoille. Testitietokoneen käyttöjärjestelmä on Windows 10 versio 1809, jossa on kirjoitushetkellä asennettu uusimmat päivitykset huhtikuulta 2020. Käytössä oleva .NET -ajoympäristö on versiossa 4.7.2. Mittausohjelma on käännetty Visual Studio 2012 versiolla Release -konfiguraatioon ja ajettu itsenäisenä suoritettavana ohjelmana komentoriviltä. Kaikki koejärjestelyt on suoritettu käyttäen 500 satunnaisesti valittua pisteparia. Tämä määrä pistepareja on valittu tasapainoksi kokeiden suoritukseen tarvittavan ajan sekä tilastollisesti merkittävän määrän välillä niin, että satunnaisten virheiden määrä lopputuloksissa olisi mahdollisimman pieni.

4.3 Tutkittavat algoritmit ja tietorakenteet

Tässä kappaleessa käydään läpi tutkitut algoritmit, tietorakenteet ja optimointimenetelmät. Artefakti on jaettu kolmeen erityyppiseen kokeeseen, jotka iteraatioina suoritettuna tukevat toisiaan ja tuottavat vastaukset asetettuihin tutkimuskysymyksiin. Ensimmäinen koejärjestely testaa iteratiivisesti eri tietorakenteita Dijkstran ja A* -algoritmeilla tarkistaen tulokset yksinkertaisempia toteutuksia vasten. Näin voidaan selvittää millä tavoin reititysalgoritmien käyttämiä tietorakenteita voidaan optimoida tieaineistolle. Tästä kokeesta saatujen tulosten perusteella käytetään optimaalisinta tietorakennetta muissa kokeissa. Toisessa koejärjestelyssä testataan arvioituja keskinopeuksia Suomen tieaineistolla A* -algoritmin heuristiikaksi. Tulosten perusteella voidaan valita oikea tasapaino suorituskyvyn ja reittien laadun välillä. Tätä tulosta käytetään hyödyksi viimeisessä koejärjestelyssä. Kolmantena kokeena luodaan

Digiroad-aineiston metatietojen perusteella kahteen tasoon pohjautuva hierarkkinen reititysalgoritmi, jonka toimivuutta, suorituskykyä ja laatua verrataan aiempiin algoritmeihin.

4.3.1 Reititysalgoritmien toteutus

Artefakti käyttää kaikissa koejärjestelyissä reititysalgoritmeina Dijkstran algoritmia ja A* -algoritmia. Molemmat näistä ovat hyvin tunnettuja algoritmeja, joista Dijkstran algoritmi on toteutukseltaan yksinkertaisempi. Molempien algoritmien toiminta on hyvin tunnettua ja niitä on kuvattu esimerkiksi lähteissä Cormen ja muut (2009) sekä Hart ja muut (1968). Tässä kappaleessa on kuvattu keskeisiä käytännön toteutukseen liittyviä toimintaperiaatteita, joihin alkuperäiset lähteet eivät ota kantaa. Tällaisia ovat yksisuuntaisten tieosuuksien huomioiminen, A* -algoritmin suljetun listan toteutustapa sekä pisteiden ja kaarien yhtäläisyyden tarkistaminen.

Digiroad-aineisto sisältää tietoa yksisuuntaisesti ajettavista tieosuuksista. Nämä säännöt on otettava huomioon, jotta reititysalgoritmin hakema reitti olisi pätevä myös käytännön sovelluksessa. Tieaineisto on tallennettu tietokoneen muistiin graafiksi, joka koostuu pisteistä ja kaarista. Kaarille on tallennettu attribuutti, joka kuvaa onko tieosuutta mahdollista ajaa kaksisuuntaisesti vai digitointisuuntaan alkupisteestä loppupisteeseen yksisuuntaisesti. Osana reititysalgoritmin toimintaa sekä Dijkstran algoritmi, että A* -algoritmi tutkivat käsiteltävän pisteen naapureita eli niitä pisteitä, jotka ovat tieosuudella liitettynä käsiteltävään pisteeseen. Molempien algoritmien toteutuksessa on luotu suodatin, jonka tehtävänä on ”piilottaa” algoritmilta pisteet, joihin matkustaminen käsiteltävältä pisteeltä rikkoisi tieosuudelle määriteltyä yksisuuntaisuussääntöä. Toisin sanoen jokaisesta liitetystä tieosuudesta tarkistetaan yksisuuntaisuus ja johtaisiko tieosuuden kulkeminen säännön rikkomiseen ja näin ollessa jätetään piste käsittelemättä. Tämä tarkistus osaltaan lisää algoritmin suoritusaikaa jokaiselle tutkitulle pisteelle, mutta käytännöllisen sovelluksen näkökulmasta se on suoritettava. Tarkistus suoritetaan samalla tavalla kaikille tieosuuksille vakioajassa ja sen toteutus koostuu yksinkertaisesti metodikutsusta ja ehtolauseista ollen näin yksinkertainen.

A* -algoritmi poikkeaa Dijkstran algoritmista tietorakenteiden osalta niin, että se vaatii toisen tietorakenteen käyttöä käsiteltyjen tai ”suljettujen” pisteiden seurantaan. Tyypillisesti tätä rakennetta nimitetään suljetuksi listaksi. Tällöin tietorakenteen tehtävä on ainoastaan tarkistaa, onko annettu piste tietorakenteessa sekä lisätä käsitellyt pisteet siihen. Tässä mittausohjelmassa suljettu lista on toteutettu käyttämällä ylimääräistä metadatamuuttujaa jokaiselle pisteelle, joka poistaa tarpeen ylläpitää erillistä suljettua listaa. Algoritmin alussa tämä *boolean* -tyyppinen muuttuja on asetettu jokaiselle pisteelle tilaan *false*. Muuttuja asetetaan arvoon *true* aina, kun kyseinen piste on valittu A* -algoritmin toimesta käsiteltäväksi pisteeksi ja se on käsitelty loppuun eli optimaalinen reitti kyseiseen pisteeseen on selvillä. Kun algoritmi tutkii käsiteltävän pisteen kaarin liitettyjä muita pisteitä, se voi jättää suljetuksi merkatut pisteet käsittelemättä. Hyötynä tästä lähestymistavasta on se, että pisteen merkkaaminen suljetuksi ja tilan tarkastaminen tapahtuu vakioajassa eikä erillisen tietorakenteen ylläpitämiselle ole tarvetta. Haittapuolena taas metadatan nollaamisesta on huolehdittava aina algoritmin suorituksen jälkeen. Mittausohjelmassa on funktio, jonka tehtävä on nollata tämä metadata graafin kaikilta pisteiltä aina ennen algoritmin suoritusta.

Datan lukuprosessin aikana sekä reititysalgoritmien suorituksessa on tarpeen vertailla kahta graafin pistettä toisiinsa sekä todeta onko kyseessä sama piste. Vastaava toiminta on tarpeen myös kaarien osalta. Vertailu on toteutettu käyttämällä pisteen osalta sen X ja Y koordinaatteja sekä kaaren osalta sen alku ja loppupisteen koordinaatteja. Lisäksi datan lukuprosessin aikana ja satunnaisten pisteiden valintaprosessin aikana pisteitä tallennetaan hajautustauluun. Tällöin pisteiden vertailu perustuu hajautuslukuun, jonka laskemiseksi käytetään alla esiteltyä funktiota.

```
public override int GetHashCode()
{
    unchecked
    {
        int hash = 17;
        hash = hash * 23 + this.X.GetHashCode();
        hash = hash * 23 + this.Y.GetHashCode();
        return hash;
    }
}
```

Funktiossa otetaan ensin pois käytöstä C# -kielen suojaus luvun ylivuodolle, joka estäisi hajautusluvun pysymisen mahdollisimman laadukkaana. Tämän jälkeen alustetaan muuttuja valitulla alkuluvulla, jonka jälkeen lisätään siihen molempien koordinaattien hajautusluvut luottaen ohjelmointikielen omaan toteutukseen, mutta kertomalla ne ensin alkuluvulla. Alkulukujen käytön tarkoituksena on vähentää mahdollisia hajautuslukujen törmäyksiä, jossa kahdelle toisistaan poikkeavalle koordinaatille annettaisiin sama hajautusluku. Alkulukujen käyttöä hajautuslukujen muodostamisessa ovat käsitelleet kirjassaan Cormen ja muut (2009).

4.3.2 Tietorakenteiden optimointi eri algoritmeilla

Aiemman tutkimuksen perusteella tietorakenteen toteutuksella on suuri merkitys reititysalgoritmin toimintanopeuteen (aihetta on käsitelty tarkemmin luvussa 2.5). Tässä koejärjestelyssä tutkitaan miten aiemmassa tutkimuksessa hyväksi havaitut tietorakenteet vaikuttavat tunnettujen reititysalgoritmien toimintanopeuteen sekä testataan käytännön kehitysaskelta parhaaksi tunnistettuun tietorakenteeseen. Tulosten perusteella voidaan vertailla tietorakenteiden ja algoritmien nopeutta erityisesti Suomen tieaineistolla hyödynnettäessä Digiroad-aineistoa. Koejärjestely on toteutettu iteratiivisesti niin, että ensimmäisenä testataan mahdollisimman yksinkertaista toteutusta, jonka avulla saadaan haettua oikea reitti. Tällöin artikkelin Sanders ja Schultes (2007) mukaisesti yksinkertaisen ja varmasti oikean toteutuksen tuloksia voidaan verrata optimaalisempiin ratkaisuihin myös varmistamalla, että kaikki tulokset ovat identtisiä, koska tietorakenteen ei kuulu vaikuttaa laskettujen reittien laatuun. Seuraavaksi laajennetaan koejärjestelyä monimutkaisemmilla elementeillä, joiden oletetaan parantavan ratkaisun suorituskykyä. Näitä iteraatioita ovat Dijkstran algoritmi binäärikeolla, Dijkstran algoritmi edelleen kehitetyllä binäärikeolla, A* -algoritmi binäärikeolla sekä A* -algoritmi edelleen kehitetyllä binäärikeolla. Nämä yhdistelmät on listattu taulukossa 3 ja iteraatioiden toteutuksen yksityiskohdat on kuvattu myöhemmin tässä luvussa. Koejärjestelyn tulosten perusteella voidaan valita paras algoritmin ja tietorakenteen yhdistelmä, jota voidaan hyödyntää seuraavissa kokeissa.

Taulukko 3. Testattavat tietorakenne ja algoritmiyhdistelmät

Algoritmi	Tietorakenne
Dijkstran algoritmi	Listarakenne (järjestämätön)
Dijkstran algoritmi	Binäärikeko
Dijkstran algoritmi	Parannettu binäärikeko
A* -algoritmi	Binäärikeko
A* -algoritmi	Parannettu binäärikeko

Binäärikeko on valittu tutkittavaksi tietorakenteeksi, koska aiemmat tutkimustulokset tukevat sen käyttöä tehokkaana, mutta toisaalta suhteellisen yksinkertaisena tietorakenteena. Fibonaccin keko on teoreettisesti tehokkaampi tietorakenne, mutta aiemmin esiteltyjen tutkimustulosten valossa teoreettinen tehokkuus ei johda käytännössä parempaan suorituskykyyn (tuloksia on käsitelty tarkemmin luvussa 2.5.2). Koejärjestelyssä käytetty binäärikeko on luotu hyödyntäen taulukkopohjaista toteutustapaa, koska Larkin ja muut (2014) suorittamien kokeiden perusteella sen suorituskyky on parempi kuin suoriin osoittimiin perustuva toteutustapa. Keskeisenä rajoitteena binäärikeko ei tue tehokkaalla tavalla tarkistusta onko annettu piste tallennettu kekoon ja tätä toimintoa käytetään sekä Dijkstran algoritmin ja A* -algoritmin toteutuksessa. Alla on esitelty jatkokehitetty binäärikeon variantti tähän koejärjestelyyn, jonka tehtävä on pääasiallisesti nopeuttaa tätä tarkastusprosessia.

Tietorakenteita testataan käyttämällä reititysalgoritmeina Dijkstran algoritmia ja A* -algoritmia. Aiemmin esitellyissä tutkimustuloksissa A* -algoritmilla on pystytty saavuttamaan merkittäviä nopeushyötyjä verrattuna Dijkstran algoritmiin. Lisäksi tutkimustulosten perusteella A* -algoritmin heuristiikalla on vaikutuksia siihen, paljonko dataa tietorakenteeseen tallennetaan ja sitä kautta myös tietorakenteen suorituskykyyn. Dijkstran algoritmi taas on toteutukseltaan yksinkertaisempi ja soveltuu näin hyvin verrokkialgoritmiksi. Molempien algoritmien toiminta pohjautuu aiemmassa tutkimuksessa esitettyihin kuvauksiin huomioiden kuitenkin käytännön toteutuksen valinnat, jotka on kuvattu luvussa 4.3.1. Lisäksi A* -algoritmin heuristiikka tässä kokeessa on toteutettu laskemalla linnuntie loppupisteeseen käyttämällä maantieteellistä etäisyyttä. Tällöin voidaan varmistaa, että A* -algoritmi palauttaa aina oikean tuloksen. Käytännön sovelluksen näkökulmasta maantieteellistä etäisyyttä parempi mittari on matka-aika ja tätä heuristiikkaa kehitetään seuraavissa kokeissa.

Binäärikeon käytännön toteutus vastaa yleisesti tunnettuja menetelmiä, kuten on käsitelty esimerkiksi lähteessä Cormen ja muut (2009) ja sen aikakompleksisuus on esitetty luvussa 2.5.1. Binäärikeko on ohjelmoitu erikseen tätä artefaktia varten, koska valmista toteutusta ei ole olemassa vakiokirjastossa. Binäärikeon toimintaa ja operaatioita on syytä tarkastella lähemmin reititysalgoritmin näkökulmasta. Molemmissa reititysalgoritmeissa kekorakenne luodaan kerran, joka on vakioajassa tapahtuva operaatio. Molemmat algoritmit poistavat jokaisella suoritusaskeleellaan pienimmällä avaimella olevan arvon keosta, joka tapahtuu suhteellisen tehokkaasti ajassa $O(\log N)$, jossa N kuvaa keossa olevien elementtien määrää. Löydetyt uudet pisteet voidaan myös lisätä kekorakenteeseen ajassa $O(\log N)$. Lisäksi molemmat algoritmit joutuvat tutkimaan käsiteltävään pisteeseen liitetyt muut pisteet jokaisella suoritusaskeleella. Tämä tarkoittaa jokaisen liitetyn pisteen osalta tarkistusta onko piste jo olemassa kekorakenteessa. Käytännössä tarkistus johtaa aikakompleksisuuteen $O(N)$ jokaiselle liitetylle pisteelle. Pienenä optimointina Dijkstran algoritmi voi jättää

tarkistuksen tekemättä, jos löydetty piste on uusi eikä sillä ole asetettua etäisyystietoa ja vastaavasti A* -algoritmi voi jättää tarkistuksen tekemättä, jos piste on jo suljettu. Joissain tapauksissa on tarve päivittää jo löydetyn pisteen avainta, joka johtaa käytännössä kahteen toimenpiteeseen keon sisällä: Ensin on löydettävä kekorakenteen taulukosta indeksi olemassa olevalle arvolle ja sen jälkeen päivitettävä arvon sijainti keossa, jolloin käytännön aikakompleksisuudeksi muodostuu $O(N + \log N)$. Ero teoreettiseen arvoon muodostuu siitä, että teoreettisessa mallissa päivitettävän arvon indeksi kekorakenteen taulukossa oletetaan olevan tiedossa, kuten voidaan nähdä esimerkiksi lähteestä Cormen ja muut (2009) s. 164.

Parannetun binäärikeon tarkoituksena on nopeuttaa kekorakenteessa kahta keskeistä toimintoa. Ensimmäinen on toteuttaa nopea tarkistus siitä, onko annettu arvo jo olemassa kekorakenteessa. Toinen on nopeuttaa olemassa olevan avaimen päivitysprosessia. Molemmat on toteutettu lisäämällä ylimääräinen tietorakenne, joka pitää kirjaa kekorakenteen arvoista sekä niiden indekseistä kekorakenteen taulukossa. Tämä lähestymistapa ei sinänsä ole uusi, vaan vastaavaan ideaan viittaavat lyhyesti artikkelissaan esimerkiksi Larkin ja muut (2014). Tietorakenne on toteutettu käyttämällä vakiokirjaston *System.Collections.Generic.Dictionary* -luokkaa (Microsoft, 2019c). Käytännössä kyseessä on hajautustaulukko, joka tallentaa avainarvopareja, jotka tässä tapauksessa ovat piste ja sen indeksi kekorakenteen taulukossa. Aikakompleksisuudeltaan hajautustauluun lisäys, poisto ja tarkistusoperaatio tapahtuvat vakioajassa, pois lukien harvaan tapahtuvat tilanteet, jossa taulukon kokoa kasvatetaan, jolloin tämä operaatio tapahtuu ajassa $O(N)$. Hyödyntäen ylimääräistä tietorakennetta sekä Dijkstran, että A* -algoritmin suorittama tarkistus pisteen olemassaolosta kekorakenteessa putoaa aikakompleksisuudeltaan vakioajaksi. Lisäksi avaimen päivityksen aikakompleksisuus parantuu luokkaan $O(\log N)$, koska tarvittavaa indeksiä ei tarvitse enää etsiä kekorakenteesta. Haittapuolena ylimääräisestä tietorakenteesta varsinaisen kekorakenteen lisäksi on lisääntynyt suoritusaika lähes kaikkien operaatioiden kohdalla, koska molemmat tietorakenteet on pidettävä ajan tasalla. Tämän lisätyön vaikutusta voidaan todentaa mittaamalla, miten suorituskyky muuttuu verrattuna tavalliseen binäärikekoon.

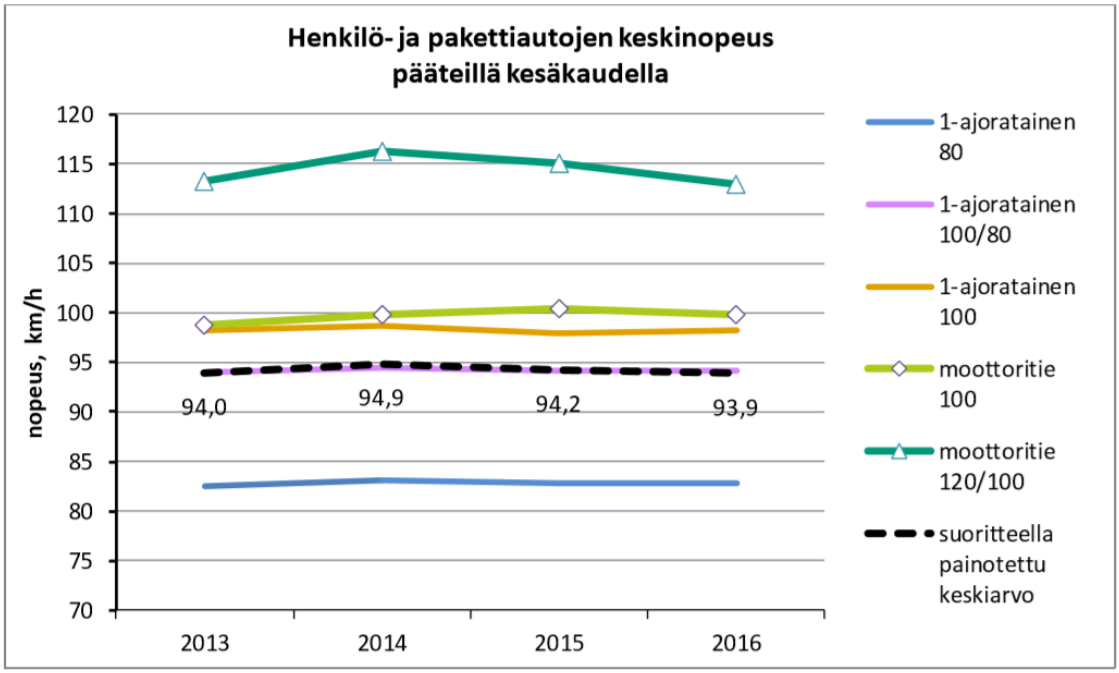
4.3.3 Keskinopeuden hyödyntäminen heuristiikkana

Käytännöllisen reitityssovelluksen näkökulmasta on usein perusteltua optimoida reittejä pituuden sijasta matka-aikaan pohjautuen. Tällöin arvioitaessa tieosuuden kustannusta reititysalgoritmin on otettava huomioon etäisyyden lisäksi myös nopeusrajoitus ja näin muodostuva matka-aika. Dijkstran algoritmin näkökulmasta keskeinen toiminta ei juurikaan muutu. A* -algoritmi taas hyödyntää toiminnassaan heuristiikkaa ja luvussa 2 läpikäydyn aiemmin tutkimuksen perusteelle linnuntien käyttäminen heuristiikkana on yleistä, koska näin voidaan taata, että algoritmi tuottaa varmasti oikean reitin. Käytettäessä matka-aikaa reitin optimoinnissa on myös heuristiikkaa muutettava vastaavasti. Käytettäessä heuristiikkana ajonopeutta tieosuudella on myös kyettävä arvioimaan millä keskinopeudella reittiä ajetaan. Lisäksi keskinopeutta käytettäessä on hankalampi varmistaa, että löydetty reitti on varmasti optimaalisin, koska arvio keskinopeudesta on voinut johtaa siihen, että A* -algoritmi ei enää välttämättä palauta varmasti oikeaa tulosta. Toisaalta keskinopeuden arviointi mahdollisimman lähelle toteumaa nopeuttaa A* -algoritmin toimintaa. Käytännön sovellus ei lisäksi välttämättä aina edellytä täysin oikeaa vastausta, vaan riittävän hyvä reitti saattaa olla hyväksyttävissä. Tämän koejärjestelyn tarkoituksena on löytää Suomen tieaineistolle kesäolosuhteissa sopiva keskituntinopeus, jonka avulla voidaan löytää tasapaino A* -algoritmin tuottamien reittien laadun ja nopeuden välille.

A* -algoritmin käyttämä matka-aikaan pohjautuva heuristiikkafunktio on esitetty alla. Funktio laskee kahden pisteen välisen etäisyyden kilometreinä tunnissa ja jakaa sen arvioidulla matka-ajalla. Etäisyys on tarpeen jakaa tuhannella, koska käytetty tasokoordinaatistojärjestelmä käyttää yksikkönä metrejä, kun taas vastaavasti nopeusrajoitus ja tätä kautta myös arvioitu matkanopeus ilmoitetaan kilometreinä tunnissa. Koodin perusteella voidaan myös nähdä arvioidun matkanopeuden olevan käänteisessä suhteessa heuristiikkaan. Tällöin suurempi arvioitu matkanopeus johtaa laadukkaampaan reittiin, koska mahdollisen heuristisen yliarvioinnin vaikutus reitin laatuun pienenee.

```
public static double GetHeuristicTravelTime(Point current, Point
target, double travelspeed)
{
    double difX = target.X - current.X;
    double difY = target.Y - current.Y;
    double euclidean = Math.Sqrt((difX * difX) + (difY * difY));
    return (euclidean / 1000) / travelspeed;
}
```

Liikenneviraston (2017) julkaiseman selvityksen mukaan kaikkien autojen keskituntinopeus maanteillä vuonna 2016 oli 93 km/h. Tämän tutkimuksen näkökulmasta keskeisempi painotus on pelkästään henkilöautojen keskinopeus kesäkaudella, joka on esitetty kuvassa 13 vuosien 2013 ja 2016 välillä. Kuvasta voidaan huomata, että keskituntinopeus on suhteellisen muuttumaton vuosien välillä, mutta toisaalta siitä voi myös huomata vaihteluvälin olevan suhteellisen suurta riippuen tieluokasta (82–116 km/h). Tätä vaihteluväliä voidaan käyttää tämän koejärjestelyn näkökulmasta käyttökelpoisena lähtöarvona eli vähintään vaihteluvälillä olevat keskinopeusvaihtoehdot kannattaa tutkia.



Kuva 13. Liikenneviraston tuottama selvitys henkilö- ja pakettiautojen keskinopeuksista keskeisillä tietyhmillä vuonna 2013–2016. Lähde: Liikennevirasto (2017).

Perustuen edellä esitettyyn Liikenneviraston (2017) selvitykseen voidaan johtaa tutkittava keskinopeusväli. Realistisesti nopeuden ylärajana 120 km/h on maksimiarvo, jonka pitäisi tuottaa laadultaan paras mahdollinen reitti. Myös käytännön sovelluksessa liikennerajoitusten ja toteutuneiden keskinopeuksien pohjalta on epätodennäköistä, että keskinopeus voisi olla tätä suurempi, vaikka reitin alku ja loppupiste sijaitsisivat moottoriteillä. Alarajan määrittäminen on haasteellisempaa, koska liikenteeseen liittyvien seikkojen lisäksi jossain tilanteissa saatetaan haluta määritellä alaraja pieneksi A* -algoritmin toiminnan nopeuttamiseksi. Tästä syystä alarajaksi on määritelty suhteellisen pieni nopeus 50 km/h. Näiden rajojen sisällä on tutkittu kaikki keskinopeusyhdistelmät välillä 50–120 km/h käyttäen 10 km/h askelväliä, jolloin tutkittavia yhdistelmiä on yhteensä 8 kpl. Näin saatujen tulosten pitäisi olla käyttökelpoisia käytännön sovelluksen optimoinnissa nopeuden ja laadun suhteeseen.

Tämä koejärjestely on toteutettu hyödyntämällä A* -algoritmia. Heuristiikan käyttäminen edellä kuvatulla tavalla ei ole oleellista Dijkstran algoritmin toiminnalle, koska Dijkstran algoritmi ei pyri hakeutumaan kohti loppupistettä kuten A* -algoritmi tekee. Dijkstran algoritmia käytetään verrokkialgoritmina, joka palauttaa aina oikean tuloksen, jolloin testattavan keskinopeuden vaikutusten reitin laatuun ja reitityksen nopeuteen voidaan testata. Tietorakenteena kaikissa vaihtoehdoissa on käytetty parannettua binäärikekoa, joka on esitelty tarkemmin edellisen koejärjestelyn kuvauksessa. Tämä tietorakenne on valittu, koska perustuen tuloksiin tietorakenteiden optimoinnista, parannettu binäärikeko on tehokkain tietorakenne molemmille algoritmeille. Lisäksi koejärjestelyn tulokset ovat vertailukelpoisia silloin, kun kaikkia testattavia vaihtoehtoja ajetaan samalla tietorakenteella.

4.3.4 Hierarkkinen optimointi tieaineiston metadatalla

Digiroad-aineisto sisältää metatietoa Suomen tieverkostosta, jota voidaan potentiaalisesti hyödyntää reititysalgoritmin toiminnan nopeuttamisessa. Keskeinen metatieto on tieosuuden toiminnallinen luokka. Digiroad-aineiston (2019b) kuvauksen perusteella toiminnallinen luokka ilmaisee liikenneväylän liikenteellistä tärkeyttä. Tämä taas on potentiaalisesti merkittävä tieto hierarkkisen reitinhaun toteutuksessa. Luvussa 2.4.3 esiteltyjen tutkimustulosten näkökulmasta katsottuna toiminnallinen luokka voisi siten olla väline hierarkkisen reitinhaun kautta reititysnopeuden parantamiseen. Taulukossa 4 on esitetty koko Suomen Digiroad-aineistosta luetut tieosuudet jaettuna eri toiminnallisiin luokkiin, prosenttiosuudet eri luokista sekä lyhyt kuvaus toiminnallisen luokan tarkoituksesta perustuen Digiroad (2019b) tietolajien kuvaukseen. Tarkastelemalla taulukkoa voidaan huomata, että ylivoimaisesti suurin osa tieosuuksista on alempia toiminnallisia luokkia. Tällöin hierarkkista optimointimenetelmää käytettäessä voitaisiin jättää suurin osa tielinkeistä tutkimatta ja saavuttaa näin tehokkuutta reititysalgoritmin toimintaan.

Toinen keskeinen kysymys hierarkkisessa haussa on eri hierarkian tasojen toteutus sekä siirtymät tasojen välillä. Yksi potentiaalinen ratkaisu on hyödyntää Digiroad-aineiston metatietoa kuntakoodin muodossa. Kuntakoodi on jokaisella tieosuudella oleva numero, jota voidaan verrata esimerkiksi alku- ja loppupisteen kuntakoodiin. Tästä voidaan päätellä, onko reititysprosessi lähellä vai kaukana alku- tai loppupisteestä. Toinen mahdollisuus on hyödyntää etäisyystietoa reititysalgoritmin käsiteltävästä pisteestä alku- ja loppupisteeseen, joka ei ole riippuvainen aineiston metadatasta. Tässä koejärjestelyssä on tavoitteena rakentaa kuntakoodista, etäisyystiedosta sekä toiminnallisesta luokasta hierarkkinen reititysalgoritmi ja tutkia miten tämä algoritmi suoriutuu verrattuna verrokkialgoritmeihin nopeuden ja reittien laadun suhteen. Koejärjestelyssä hyödynnetään tuloksia kahdesta aiemmasta kokeesta liittyen

tietorakenteisiin sekä A* -algoritmin heuristiikan optimointiin arvioidun keskinopeuden avulla.

Taulukko 4. Digiroad-aineiston (2019b) henkilöautolla ajettavat toiminnalliset luokat sekä mittausohjemaan koko Suomen tieaineistosta ajetut graafin pisteiden määrät ja prosenttiosuudet.

Toiminnallinen luokka	Tieosuudet	Prosenttiosuus
Luokka 1: Valtakunnallinen tai seudullinen pääkatu	47769	1,82 %
Luokka 2: Kantatie tai seudullinen pääkatu	25956	0,99 %
Luokka 3: Seututie tai alueellinen pääkatu	102104	3,88 %
Luokka 4: Yhdystie tai kokoojakatu	339316	12,89 %
Luokka 5: Liityntäkatu tai tärkeä yksityistie	334250	12,70 %
Luokka 6: Muu yksityistie	1782319	67,72 %

Hierarkiatasojen luonnissa voidaan käyttää apuna eri hallinnollisten luokkien prosenttiosuuksia. Näin huomataan, että sulkemalla pois kaksi alinta luokkaa (luokat 5 ja 6) voidaan potentiaalisesti jättää käsittelemättä 80,43 % tieosuuksista. Lisäksi tarkastelemalla Digiroad-aineiston tietolajien kuvausta (Digiroad, 2019b) voidaan päätellä, että nämä kaksi tasoa eivät todennäköisesti sisälly kahden pisteen välisiin reitteihin muuten kuin lähellä alku- ja loppupistettä. Tietolajien kuvauksessa todetaan, että ”liityntäkadulla on välitön yhteys tontille tai rakennuspaikalle.” (s.14). Houkuttelevaa on myös sulkea yhdystiet (luokka 4) pois hierarkian ylemmältä tasolta, koska tällöin voidaan sulkea pois 93,32 % tieosuuksista. Liu (1997) on tutkimuksessaan havainnut lyhyiden oikopolkujen vaikutuksen reitityksessä kaupunkiolosuhteissa. Tällä perusteella on kuviteltavissa, että yhdystietä käytetään lyhyenä oikopolkuna osana pidempää reittiä, vaikka teknisesti kokoojakadulla ei saisi olla tällaista liikennettä (Digiroad 2019b). Kaksitasoisessa hierarkiassa olisi näiden lukujen valossa perusteltua tehdä jako tasojen välille niin, että luokat 1–4 tai 1–3 sisältyvät korkeammalle tasolle, jota käytetään algoritmin ollessa kaukana alku- ja loppupisteestä. Useamman kuin kahden tason käyttö ei välttämättä ole perusteltua tieosuuksien lukumäärän perustella, mutta voi potentiaalisesti olla tulevaisuuden jatkotutkimuskohde.

Hierarkian toiminnan kannalta on keskeistä luoda mekanismi, joka tekee päätöksen, onko algoritmin käsittelemä piste lähellä vai kaukana alku- tai loppupisteestä. Tässä koejärjestelyssä on käytössä kaksi potentiaalista tietoa: tieaineiston metadatassa oleva kuntakoodi, joka on asetettu jokaiselle tieosuudelle sekä tasokoordinaatistosta laskettu etäisyys alku- ja loppupisteeseen. Etäisyys kahden pisteen koordinaattien välillä on mahdollista laskea käyttämällä yksinkertaista trigonometriaa, mutta se lisää kuitenkin jokaisella askeleella suoritettavan työn määrää. Kuntakoodi taas on vastaavasti nopea tarkastaa koodissa yksinkertaisella ehtolauseella. Pelkästään kuntakoodin käyttäminen ehtona hierarkian tasojen välillä ei kuitenkaan välttämättä ole riittävää. On helppo kuvitella varsin realistinen skenaario, jossa reitityksen alku- tai loppupiste sijaitsee aivan kahden kunnan rajalla. Tällöin, jos hierarkia estää pienempien tieosuuksien käsittelyn, voi optimaalinen reitti jäädä löytymättä, vaikka kohdepiste sijaitsisi varsin lähellä kunnan keskustaa. Toinen mahdollinen skenaario on pidempi yksityistieosuus (esimerkiksi mökkitie) joka kulkee kahden kunnan rajan yli, jolloin optimaalinen reitti jäisi löytymättä, jos yksityistiet pudotettaisiin algoritmin käsittelystä pelkästään kuntakoodin perusteella. Tällöin on perusteltua, että hierarkiassa hyödynnetään ainakin etäisyyttä kohdepisteisiin kriteerinä hierarkian tasojen välillä siirtymiselle.

Avoin kysymys tämän koejärjestelyn näkökulmasta on miten suuri etäisyys käsiteltävällä pisteellä pitäisi olla alku- tai loppupisteeseen, jotta hierarkian tasoa voitaisiin vaihtaa. Yhtenä lähtökohtana voitaisiin pitää Suomen kuntien keskimääräistä maantieteellistä kokoa. Kuntaliiton (2019) julkaiseman tiedon mukaan Suomen kuntien pinta-alojen mediaani on 753 km^2, joka muutettuna ympyrän halkaisijaksi on 30,96 km. Julkaisun mukaan kuntien pinta-alojen vaihteluväli on kuitenkin erittäin suurta (6–17 334 km^2). Lisäksi kuntakoodin käyttö yhdessä etäisyyden kanssa voi tarkoittaa sitä, että on riittävää hyödyntää pienempää etäisyyttä ratkaisemaan aiemmin kuvattu haaste kuntakoodin toiminnassa, kun kohdepiste on lähellä kunnan rajaa. Koejärjestelyn näkökulmasta optimaalinen etäisyys on se, joka ei johda tilanteisiin, jossa reitti jää kokonaan löytymättä tai reitin pituus oleellisesti pitenee optimoinnin seurauksena. Tällä perusteella koejärjestelyssä on testattu etäisyyksiä 30 km, 15 km ja 7,5 km.

Koejärjestelyssä testataan taulukossa 5 määritellyt yhdistelmät hierarkkisesta reitinhakuprosessista. Kaikki algoritmit käyttävät laskennassa matka-aikaa ja tietorakenteena käytetään parannettua binäärikekoa, jolloin tulokset ovat verrannollisia keskenään. Lähtötason vertailukohteena on Dijkstran algoritmi parannetulla binäärikeolla, joka antaa varmasti oikeat tulokset kaikille reiteille, mutta toimii oletettavasti hitaammin kuin muut vaihtoehdot. Testattavissa vaihtoehdoissa käytetään A* -algoritmia, koska se on osoittautunut nopeammaksi aiempien koetulosten perusteella. A* -algoritmin osalta vertailukohtana on versio, joka hyödyntää 90 km/h arvioitua keskinopeutta heuristiikassaan. Tämä keskinopeus antaa hyvän tasapainon nopeuden ja laadun välille. Samaa keskinopeutta hyödynnetään myös kaikissa hierarkiaa käyttävissä vaihtoehdoissa. Hierarkian käyttöä reitityksen optimoinnissa testataan käyttämällä muuttujina toiminnallista luokkaa sekä etäisyyttä alku- ja loppupisteeseen. Kuntakoodi (KK) toimii ylimääräisenä muuttujana niin, että kaikissa vaihtoehdoissa tarkempi hierarkian taso on käytössä algoritmin tutkiessa pistettä, joka on samassa kunnassa alku- tai loppupisteen kanssa. Kutakin hierarkiaa on testattu etäisyyden lisäksi sekä kuntakoodilla että ilman sitä.

Taulukko 5. Koejärjestelyssä testattavat algoritmi- ja hierarkiavaihtoehdot eriteltynä niissä olevien eroavaisuuksien mukaisesti. Kaikki hierarkiat on testattu käyttämällä kuntakoodia tasojen siirtymänä sekä ilman sitä.

Vaihtoehto	Algoritmi	Hierarkian tasot	Ylemmän tason toiminnalliset luokat	Tason siirtymä
Dijkstra verrokki	Dijkstra	Yksi taso	Kaikki	-
A* verrokki	A*	Yksi taso	Kaikki	-
Hierarkia 1	A*	Kaksi tasoa	Tasot 1-4	7,5 km
Hierarkia 2	A*	Kaksi tasoa	Tasot 1-4	15 km
Hierarkia 3	A*	Kaksi tasoa	Tasot 1-4	30 km
Hierarkia 4	A*	Kaksi tasoa	Tasot 1-3	7,5 km
Hierarkia 5	A*	Kaksi tasoa	Tasot 1-3	15 km
Hierarkia 6	A*	Kaksi tasoa	Tasot 1-3	30 km

Koejärjestelyn tuloksia rajoittaa eri muuttujista koostuvien vaihtoehtojen potentiaalisesti suuri määrä. Etäisyysmuuttujalle on haasteellista määrittää sopivia vaihtoehtoja ilman testituloksia. Koejärjestelyyn on sisällytetty kohtuullinen määrä vaihtoehtoja, mutta on mahdollista, että suorituskyvyltään optimaalisimmat vaihtoehdot ovat testattavien vaihtoehtojen ulkopuolella. On myös mahdollista, että osa tutkittavista yhdistelmistä on hylättävä joukosta, koska ne eivät löydä reittiä testattavalle pisteparille.

5. Evaluointi ja tulokset

Tässä luvussa esitellään aiemmin kuvatun artefaktin ja koejärjestelyjen avulla saadut mittaustulokset. Mittaustulokset on esitetty ja analysoitu jokaiselle koejärjestelylle erikseen. Ensimmäisenä käydään läpi tietorakenteiden optimointiin tähtäävän koejärjestelyn tulokset. Toisena esitellään arvioidun keskinopeuden hyödyntämiseen heuristiikkana suunnitellun koejärjestelyn tulokset. Kolmantena arvioidaan hierarkkiseen optimointiin tieaineistolla tarkoitetun koejärjestelyn tulokset.

5.1 Tietorakenteiden vertailu

Ensimmäisessä koejärjestelyssä tutkittiin erilaisten tietorakenteiden toimintaa Dijkstran ja A* -algoritmeilla optimoidessa lyhintä reittiä. A* -algoritmin heuristiikkana käytettiin maantieteellistä etäisyyttä linnuntietä pitkin kohdepisteeseen. Testattuja satunnaisia pistepareja oli käytössä 500 kpl. Algoritmien laatu on kaikilla vaihtoehdoilla sama eli ne palauttavat identtiset reitit. Kaikkien pisteparien yhteenlaskettu reitti oli pituudeltaan 177 923,79 km ja yksittäisen reitin keskimääräinen pituus oli 355,85 km. Parhaalla yhdistelmällä eli A* -algoritmilla ja parannetulla binäärikeolla reitin laskenta-aika oli keskimäärin 402 millisekuntia. Algoritmikohtaiset tulokset on esitetty taulukossa 6.

Taulukko 6. Tulokset eri tietorakenteiden vertailusta Dijkstran ja A* -algoritmeilla. Yhteenlaskettu aika on kokonaisaika kaikille 500 pisteparille. Askelien määrä ilmaisee algoritmin käsittelemien pisteiden kokonaismäärän.

Algoritmi	**Tietorakenne**	**Yhteenlaskettu aika (sekuntia)**	**Askelien määrä (kpl)**	**Askelia sekunnissa (kpl/s)**
Dijkstran algoritmi	Listarakenne (järjestämätön)	6549,60	624745744	95387
Dijkstran algoritmi	Perinteinen Binäärikeko	4636,45	624745744	134747
Dijkstran algoritmi	Parannettu binäärikeko	720,97	624745744	866535
A* -algoritmi	Perinteinen Binäärikeko	1068,54	163410648	152929
A* -algoritmi	Parannettu binäärikeko	200,88	163410648	813474

Molemmat algoritmit käsittelevät jokaisella askeleella yhden pisteen tietorakenteesta ja merkkaavat sen loppuun käsitellyksi. Tuloksista voidaan todeta, että Dijkstran algoritmi tarvitsee samojen reittien laskentaan 3,82 kertaa enemmän askelia, kuin A* -algoritmi. Lisäksi askelien määrää sekunnissa tarkastelemalla huomataan, että A* -algoritmin askeleen suorittaminen kestää 6,52 % pidempään kuin Dijkstran algoritmilla käytettäessä parannettua binäärikekoa. Tämä on osaltaan odotettua, koska A* -algoritmi käyttää toiminnassaan heuristiikkaa, joka vaatii ylimääräistä laskenta-aikaa verrattuna Dijkstran algoritmiin. Verrattaessa algoritmien yhteenlaskettuja suoritusaikoja binäärikekoa käytettäessä A* -algoritmi suoriutuu 4,34 kertaa nopeammin ja parannettua binäärikekoa käytettäessä 3,59 kertaa nopeammin kuin Dijkstran algoritmi.

Tietorakenteiden vertailussa järjestämätön listarakenne palauttaa hitaimmat tulokset, joka on referenssinä toimivalta tietorakenteelta odotettu tulos. Binäärikekojen toteutuksen välillä suorituskykyero on kuitenkin merkittävä. Parannettu binäärikeko mahdollistaa vakioajassa tarkistuksen onko annettu elementti tietorakenteessa (Contains) sekä ajassa $O(\log N)$ avaimen päivityksen (UpdateKey), joita molemmat algoritmit hyödyntävät. Dijkstran algoritmilla parannettu binäärikeko oli 6,43 kertaa nopeampi kuin perinteinen binäärikeko. Vastaavasti A* -algoritmilla parannettu binäärikeko oli 5,32 kertaa nopeampi kuin perinteinen binäärikeko. Päivitysoperaation parannuksen vaikutusta ei suoraan pystytä erottamaan koetuloksista.

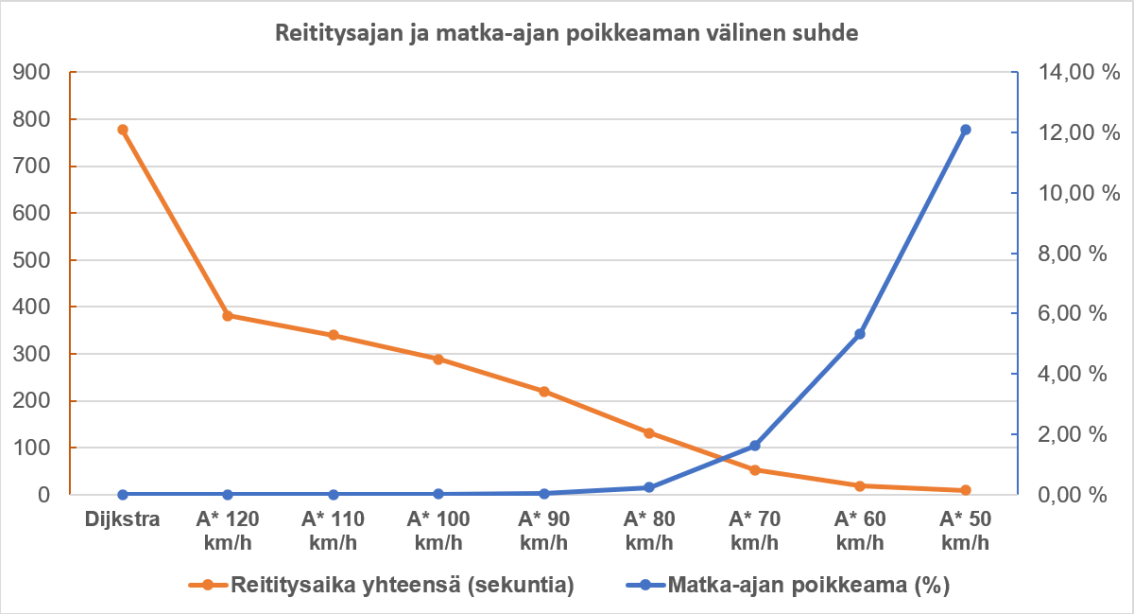
5.2 Keskinopeuden vaikutuksen vertailu

Koejärjestelyssä testattiin A* -algoritmin optimointia matka-ajan laskennassa käytettäessä eri keskinopeusoletuksia heuristiikassa. Verrokkialgoritmina toimi Dijkstran algoritmi, joka palauttaa aina optimaalisen tuloksen, koska se ei käytä heuristiikkafunktiota. A* -algoritmin heuristiikkafunktion tarkoituksena on ohjata algoritmin suoritus kohti asetettua kohdepistettä. Kaikissa testatuissa vaihtoehdoissa heuristiikkana on käytetty linnuntietä kohdepisteeseen ja optimoitaessa reittiä matka-ajan mukaisesti on otettava huomioon teiden pituuden lisäksi myös oletettu matkanopeus. A* -algoritmi palauttaa optimaalisen tuloksen, jos heuristiikkafunktion arvioima matka-aika on pienempi tai yhtä kuin optimaalinen matka-aika. Vastaavasti A* -algoritmin suoritus nopeutuu yliarvioimalla matka-aikaa, mutta tällöin palautettu tulos voi poiketa optimaalisesta ratkaisusta. Koejärjestelyn avulla on pyritty selvittämään poikkeaman ja nopeusparannuksen välistä suhdetta algoritmin käytännön tehokkuuden parantamiseksi. Suuremman matkanopeuden käyttö matka-aikaa arvioitaessa johtaa laadultaan parempaan tulokseen ja pienempi matkanopeus parantaa nopeutta, koska matkanopeus vaikuttaa käänteisesti matka-aikaan (laskentafunktio on esitelty luvussa 4.3.3). Tietorakenteena kaikissa vaihtoehdoissa on käytetty parannettua binäärikekoa. Koejärjestelyn tulokset on saatu laskemalla yhteen 500 satunnaisen pisteparin muodostamat reitit ja niiden tiedot, kuten edellisessäkin kokeessa. Pisteparit eivät kuitenkaan ole yhteneviä eri kokeiden välillä. Koejärjestelyn tulokset on esitetty taulukossa 7.

Taulukko 7. Tulokset eri keskinopeuksien vertailusta. Dijkstran algoritmi edustaa verrokkia, joka tuottaa aina laadultaan optimaalisen tuloksen.

Algoritmi	Matka-aika (tuntia)	Matka-ajan poikkeama (%)	Reittien pituus yhteensä (km)	Reititysaika yhteensä (sekuntia)	Askelien määrä (kpl)
Dijkstra	2077,99	0,00 %	184794,99	776,77	624951811
A* (120 km / h)	2077,99	0,00 %	184794,99	381,16	297572744
A* (110 km / h)	2077,99	0,00 %	184795,06	339,39	264517071
A* (100 km / h)	2078,14	0,01 %	184686,52	288,48	222992751
A* (90 km / h)	2078,58	0,03 %	184577,12	219,62	169031372
A* (80 km / h)	2082,96	0,24 %	184113,89	131,49	100479438
A* (70 km / h)	2111,44	1,61 %	183967,38	52,34	40418121
A* (60 km / h)	2188,97	5,34 %	185141,62	18,26	14149958
A* (50 km / h)	2329,22	12,09 %	188971,83	9,36	6901831

Dijkstran algoritmi ja A* -algoritmi käyttäen 120 km/h keskinopeutta heuristiikkana tuottavat keskenään vastaavat optimaaliset reitit, mutta A* -algoritmi on tehtävässään noin 2,04 kertaa nopeampi. Tämä kerroin on merkittävästi huonompi kuin optimoidessa maantieteellisen etäisyyden mukaan (kuten edellisessä kokeessa). Todennäköinen syy tälle on, että A* -algoritmin heuristiikan on aliarvioitava matka-aikaa enemmän, jotta reitti pysyy oikeana kaikissa mahdollisissa tilanteissa. Pienentämällä arviota keskinopeudesta päästään tarkempiin arvioihin toteumasta ja sitä kautta myös A* -algoritmin reititysnopeus paranee merkittävästi, mutta virhe optimaaliseen reittiin verrattuna kasvaa. Tämän suhteen voi nähdä kuvasta 14, joka osoittaa miten A* -algoritmin tuottamien reittien poikkeama optimaalisesta reitistä kasvaa eri keskinopeusarvoilla. Verrattuna Dijkstran algoritmiin vasta 80 km/h keskinopeusoletuksella päästään parempaan tulokseen reititysajan suhteen kuin maantieteellistä etäisyyttä käytettäessä, jolloin A* -algoritmi on 5,9 kertaa nopeampi, mutta tällöin matka-ajan poikkeama optimaalisesta reitistä on 0,24 %. Reittien pituus testattujen vaihtoehtojen välillä vaihtelee odotetusti eri keskinopeuksien välillä, koska maantieteellinen pituus ei ole itsessään optimointikriteeri. Tällöin matka-ajaltaan pidempi reitti voi olla myös maantieteelliseltä etäisyydeltään lyhyempi.



Kuva 14. Havainnekuva keskinopeuden vertailusta, joka esittää kokonaisreititysajan suhdetta poikkeamaan optimaalisesta matka-ajasta eri algoritmeilla.

Tarkastelemalla askelten määrää suhteessa reititysaikaan voidaan tehdä havaintoja algoritmien suoritusnopeudesta. Verrokkina toimiva Dijkstran algoritmi suoritti 804552 askelta sekunnissa, kun taas keskiarvo kaikkien A* -algoritmin keskinopeuksien välillä oli 768926 askelta sekunnissa, jolloin keskimäärin A* -algoritmien nopeus askelina oli 4,43 % hitaampi. Hitain A* -algoritmin versio oli 50 km/h keskinopeus ja nopein 120 km/h, jolloin 120 km/h oli askelien nopeudessa mitattuna 5,88 % nopeampi. Erot muiden keskinopeusluokkien välillä olivat pienempiä. Havainnoista voidaan päätellä, että algoritmien suoritusnopeudessa on jonkin verran eroja, mutta ne rajoittuvat kokonaisuutena yksittäisten prosenttien päähän toisistaan. Havainnot kuitenkin vahvistavat edellisen kokeen tuloksia siitä, että Dijkstran algoritmin askeleen suoritusnopeus on hieman A* -algoritmia nopeampaa.

Optimaalisen keskinopeuden valinta A* -algoritmin heuristiikaksi riippuu lähtökohtaisesti käytännön sovelluksen asettamista vaatimuksista. Näiden koetulosten perusteella arvioidut keskinopeudet välillä 70–90 km/h näyttävät tuottavan parhaan tasapainon A* -algoritmin reititysnopeuden ja laadun välillä, kun reittiä optimoidaan matka-ajan perusteella. Vaihtoehdoista 90 km/h tuottaa paremman laadun ja 70 km/h paremman reititysnopeuden. Reititysnopeuden kasvaessa reitin laatu heikkenee ja sitä kautta matka-aika lisääntyy 0,03–1,61 % verrattuna täysin optimaaliseen ratkaisuun. Vastaavasti reititysaika vähenee 3,54–14,84 kertaisesti. A* -algoritmi voi tuottaa myös optimaalisia reittejä, mutta tällöin käytettävä keskinopeus on valittava huolellisesti, jotta yliarviointia ei tapahdu. Käytettäessä 120 km/h keskinopeusarviota reititysnopeus on silti 2,04 nopeampi kuin Dijkstran algoritmissa, joka tekee A* -algoritmista houkuttelevan myös vaadittaessa optimaalisia reittejä.

5.3 Hierarkkinen optimointi

Kolmannessa koejärjestelyssä tutkittiin hierarkiaa hyödyntävää A* -algoritmiin pohjautuvaa reititysalgoritmia eri parametreillä optimoidessa matka-aikaa. Dijkstran algoritmi ja A* -algoritmi 90 km/h keskinopeusheuristiikalla toimivat verrokkeina. Testattuja satunnaisia pistepareja oli käytössä 500 kpl. Dijkstran algoritmi on ainoa, joka palauttaa taatusti oikeat tulokset ja verrokki A* -algoritmi on valittu aiemman kokeen perusteella hyvää laadun ja nopeuden tasapainoa edustavana vaihtoehtona. Samaa 90 km/h keskinopeusheuristiikkaa on käytetty myös kaikissa testatuissa hierarkioissa. Koetulokset on esitetty alla olevassa taulukossa.

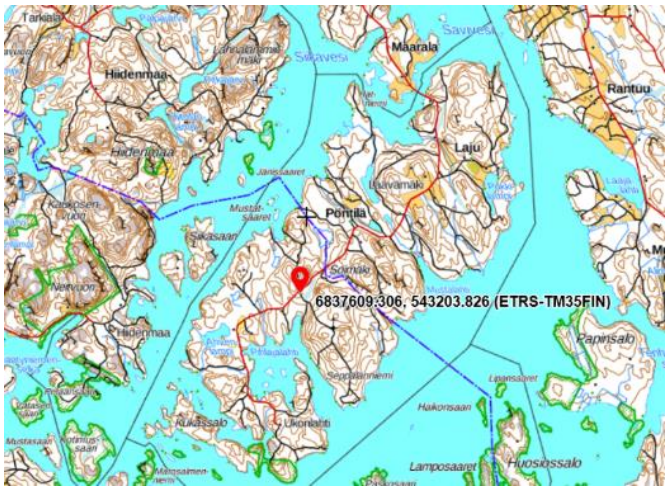
Taulukko 8. Hierarkiakoejärjestelyn tulokset. Kuntakoodia ilmaisee lyhenne KK. Verrokkialgoritmeilla Dijkstra ja A* 90 km/h ei ole hierarkiatasoja. Viiva Matka-ajassa ilmaisee, että tutkittu hierarkia ei löytänyt reittiä kaikkien pisteparien välille.

Algoritmi	Ylemmän tason toiminnalliset luokat	Tason siirtymä	Matka-aika (tuntia)	Matka-ajan poikkeama (%)	Reititysaika yhteensä (sekuntia)	Reittien pituus yhteensä (km)	Askelien määrä (kpl)
Dijkstra	-	-	2077,99	0,00 %	772,13	184794,99	624951811
A* 90 km / h	-	-	2078,58	0,03 %	218,72	184577,12	169031372
Hierarkia 1 KK	Tasot 1-4	KK + 7,5 km	2098,54	0,99 %	44,68	186329,71	41327029
Hierarkia 1	Tasot 1-4	7,5 km	-	-	-	-	-
Hierarkia 2 KK	Tasot 1-4	KK + 15 km	2096,56	0,89 %	46,01	186180,03	42455403
Hierarkia 2	Tasot 1-4	15 km	2097,05	0,92 %	42,26	186279,63	39231812
Hierarkia 3 KK	Tasot 1-4	KK + 30 km	2094,49	0,79 %	54,19	185966,11	48796091
Hierarkia 3	Tasot 1-4	30 km	2094,59	0,80 %	52,62	185999,44	47559178
Hierarkia 4 KK	Tasot 1-3	KK + 7,5 km	-	-	-	-	-
Hierarkia 4	Tasot 1-3	7,5 km	-	-	-	-	-
Hierarkia 5 KK	Tasot 1-3	KK + 15 km	2144,63	3,21 %	20,57	190442,60	21012864
Hierarkia 5	Tasot 1-3	15 km	-	-	-	-	-
Hierarkia 6 KK	Tasot 1-3	KK + 30 km	2131,92	2,60 %	29,29	189152,81	28423092
Hierarkia 6	Tasot 1-3	30 km	-	-	-	-	-

Koejärjestelyssä seitsemän tutkitusta kahdestatoista hierarkiavaihtoehdosta löysi reitin kaikkien annettujen pisteparien välille ja loput hylättiin, koska ne eivät löytäneet vähintään yhtä reittiä tutkittujen pisteparien välille. Seitsemän onnistuneen vaihtoehdon osalta vain kahdessa matka-ajan poikkeama ylitti 1 % rajan. Kuntakoodia lisäparametrina käyttävistä vaihtoehdoista viisi kuudesta löysi kaikki reitit, mutta ilman kuntakoodia vain kaksi kuudesta löysi kaikki reitit. Toiminnallisista luokista katsottuna ne hierarkiavaihtoehdot, jotka rajasivat vähemmän pisteitä (1–4) toimiessaan ylemmällä

tasolla löysivät kaikki reitit yhtä poikkeusta lukuun ottamatta. Aggressiivisemmin pisteitä karsivat vaihtoehdot (1–3) löysivät kaikki reitit vain kahdessa tapauksessa kuudesta. Kokonaisuutena parhaiten kaikki reitit löysivät kuntakoodia käyttävät hierarkiat, joiden ylempi toiminnallinen taso kattoi luokat 1–4.

Reittien visuaalinen tarkastelu paljastaa syitä sille miksi jotkin yhdistelmät eivät ole löytäneet kaikkia reittejä. Tyypillisen syy reitin puuttumiselle on joko alku- tai loppupisteen sijainti lähellä kunnan rajaa, tieyhteyksien kulkeminen pisteeseen toisen kunnan alueen kautta ja tieyhteyksien toiminnallinen luokka. Esimerkkinä tällaisesta tilanteesta on esitetty kuvassa 15, jossa kartalle on merkitty saarella sijaitseva reitin alkupiste. Tässä tilanteessa saari on jaettu kahden kunnan alueelle ja ainoa pääsy pisteeseen kulkee toisen kunnan kautta suhteellisen pienen toiminnallisen luokan tien kautta. Tästä tarkastelusta voidaan päätellä, että tason siirtymä, kuntakoodin käyttö ja etäisyysparametri vaikuttavat osaltaan siihen, että kaikkia reittejä ei löydetä. Vastaavasti taas kaikki reitit löytyvät silloin, kun nämä parametrit ovat sopivan sallivat.



Kuva 15. Havainnekuva eräästä reitin päätepisteestä saarella, johon kaikki tutkitut hierarkiat eivät löytäneet reittiä. Kunnan raja on esitetty kuvassa sinisellä katkoviivalla.

Reititysnopeuden osalta verrokkina toiminut Dijkstran algoritmi suoritti kaikki reititykset yhteenlasketussa ajassa 772,13 s. A* -algoritmi ilman hierarkiaa suoriutui 3,53 kertaa nopeammin, joka on odotettu tulos perustuen havaintoihin edellisistä kokeista. Testatuista hierarkioista reititysajaltaan parhaiten suoriutui hierarkia 5 kuntakoodilla, joka oli 37,53 kertaa Dijkstran algoritmia nopeampi ja 10,63 kertaa nopeampi kuin A* -algoritmi, mutta 3,21% poikkeamalla oikeasta tuloksesta. Hitain testatuista hierarkioista oli hierarkia 3 kuntakoodilla, joka suoriutui 14,25 kertaa nopeammin kuin Dijkstran algoritmi ja 4,04 kertaa A* -algoritmia nopeammin. Pienimmän poikkeaman testatuista hierarkioista tuotti hierarkia 3 kuntakoodilla, jonka poikkeama oli 0,79%, joka tarkoittaa 16,5 tunnin lisäystä optimaaliseen 2077,99 tunnin matka-aikaan. Molemmat variaatiot hierarkioista 2–3 tuottivat samaa luokkaa olevan poikkeaman ja reititysajan.

Hierarkiat 2 ja 3 tuottivat tulokset sekä kuntakoodilla että ilman. Kuntakoodin käyttäminen pienensi poikkeamaa 0,01–0,04 prosenttiyksikköä, mutta kasvatti reititysaikaa 3–9 %. Toisaalta etäisyyden kasvattaminen tason siirtymässä pienensi poikkeamaa 0,10–0,12 prosenttiyksikköä, jolloin reititysaika kasvoi 17,55–24,51 %. Näiden tuloksien suhde tarjoaa viittauksen siihen, että kuntakoodin käyttäminen ei välttämättä ole tehokkain tapa pienentää poikkeamaa tai että kuntakoodia mekanismina olisi jatkokehitettävä poikkeaman vähentämiseksi.

Dijkstran algoritmin laskemien reittien kokonaispituus oli 184794,99 km, jolloin yhden reitin keskimääräinen pituus on 369,59 km ja reititysaika keskimääräiselle reitille 1 544 millisekuntia. Vastaavasti A* -algoritmille keskimääräinen reititysaika oli 437 millisekuntia. Tutkituissa hierarkiavaihtoehdoissa keskimääräisen reitin reititysaika oli 41–108 millisekuntia, jossa hierarkia 5 kuntakoodilla oli nopein ja hierarkia 3 kuntakoodilla hitain. Vertailukohtana käytettävyyteen Nielsenin (1993) mukaan 100 millisekuntia on maksimi raja-arvo, jossa sovelluksen käyttäjä kokee, että sovelluksen vasteaika on välitön.

6. Keskustelu ja pohdintaa

Tämän työn keskeisenä tutkimuskysymyksenä on ollut etsiä keinoja reititysalgoritmin tehokkuuden optimointiin Digiroad-aineistolla ottaen huomioon laskettujen reittien laatu ja algoritmin toimintanopeus. Ylätason tutkimuskysymykseen vastataan kolmen alikysymyksen avulla. Alikysymyksiin liittyy yhteensä kolme erillistä koejärjestelyä, joiden tulokset on käsitelty luvussa 5. Tässä luvussa tarkastellaan miten nämä tulokset sopivat aiemman tutkimuksen viitekehykseen, mitä jatkotutkimusaiheita tulosten perusteella voidaan suositella ja mitä havaintoja työn suorittamisen aikana on tehty sekä rajoitteita sillä on.

Ensimmäisen tutkimuskysymyksen tarkoituksena on selvittää millä tavoin reititysalgoritmien tietorakenteita voidaan optimoida tieaineistolle. Suoritettujen askelten osalta aiemmin todettiin, että Dijkstran algoritmi tarvitsee samojen reittien laskentaan 3,82 kertaa enemmän askelia, kuin A* -algoritmi. Ikeda ja muut (1994) ovat artikkelissaan raportoineet 2,57 kertaisesta askelien erotuksesta samojen algoritmien välillä, mutta heidän tuloksensa on saatu Tokion metropolialueella, kun taas tässä koejärjestelyssä on käytetty koko Suomen aineistoa. Tietorakenteiden koejärjestelyn tuloksissa käytiin läpi, että yhteenlaskettujen suoritusaikojen osalta binäärikekoa käytettäessä A* -algoritmi suoriutui 4,34 kertaa nopeammin ja parannettua binäärikekoa käytettäessä 3,59 kertaa nopeammin kuin Dijkstran algoritmi. Koejärjestelyn tulokset ovat jossain määrin parempia kuin aiemmassa tutkimuksessa keskimäärin raportoidut tulokset. Bast (2009) kertoo artikkelissaan yleisellä tasolla A* -algoritmin tarjoavan noin 2–3 kertaisen nopeutuksen. Liu (1997) kertoo tuloksissaan noin kaksinkertaisesta nopeutuksesta. Samaan tulokseen ovat päätyneet kokeissaan myös Jacob ja muut (1999). R. T. Honkanen (artikkelin luonnos, 2011) on kertonut noin 3,4 kertaisesta parannuksesta algoritmien välillä A* -algoritmin hyödyksi. Vastakohtana taas Goldberg ja Harrelson (2005) ovat raportoineet artikkelissaan A* -algoritmin toimivan noin puolet hitaammin kuin Dijkstran algoritmi.

Vaihtelevia tuloksia voi osaltaan selittää Holzerin ja muiden (2005) havainto siitä, että kohdehakeutuvien algoritmien suorituskyky on vahvasti riippuvainen käytetyn graafin rakenteesta. He kertovat osaltaan alle kaksinkertaisesta parannuksesta tieaineistolla. Tässä koejärjestelyssä saadut tulokset puoltavat vahvasti A* -algoritmin käyttöä Dijkstran algoritmin sijasta Digiroad-aineistolla. Tulosta voi osaltaan selittää Digiroad-aineiston suhteellisen pieni kaarien ja pisteiden välinen suhdeluku, joka on 1,06 (aineistoa ja sen toimitusmuotoa on käsitelty tarkemmin luvussa 4.1) ja tältä osin suhdeluku on pienempi kuin useimmissa tutkimusaineistoissa erityisesti silloin kun aiemmassa tutkimuksessa on käytetty aineistona kaupunkialueita (Jacob ja muut, 1999; Jagadeesh ja muut, 2002).

Tietorakenteita koskevien tulosten osalta nousee esille havainto siitä, miten tärkeää tietorakenteen käytännön toteutus on sen teoreettisen suorituskyvyn lisäksi. Useimmat tutkimusartikkelit keskittyvät tietorakenteiden teoreettisten ominaisuuksien optimointiin ja käytännön toteutusvalinnat jäävät vähemmälle huomiolle. Binäärikeon toiminnan osalta tyypillinen oletus näyttää olevan, että tarvittavien keon indeksien säilytys toteutetaan kekorakenteen ulkopuolella, jotta osa keko-operaatioista voidaan suorittaa tehokkaammin (Cormen ja muut, 2009). Larkin ja muut (2014) viittaavat artikkelissaan myös lyhyesti ylimääräiseen tasoon (level of indirection) indeksitiedon säilyttämiseksi, joka nostaa päivitysoperaation tehokkuutta. Näin katsottuna parannetun binäärikeon ylimääräinen tietorakenne, joka tallentaa indeksitietoja binäärikeosta, on tutkimustietoon peilautuen oikea valinta ja esitetyt tulokset tukevat tätä päätelmää. Suorituskyvyn suurta parannusta selittänee se, että molemmat testatut algoritmit suorittavat jokaiselle käsitellylle pistelle tarkistusoperaation kekorakenteesta.

Toisella tutkimuskysymyksellä haluttiin selvittää millä tavoin arvioitua keskinopeutta voidaan hyödyntää A* -algoritmin heuristiikan optimoinnissa Suomen tieaineistolla. Tulosten perusteella kesäajonopeuksilla arvioidut keskinopeudet välillä 70–90 km/h näyttävät tuottavan parhaan tasapainon reititysnopeuden ja reittien laadun välillä. Optimaalista reittiä haettaessa tulokset ovat keskimäärin hyvin linjassa aiemman tutkimuksen kanssa. Yleisin havainto siitä, että A* -algoritmi suoriutuu noin kaksi kertaa nopeammin kuin Dijkstran algoritmi (Bast, 2009; Jacob ja muut, 1999) näyttää olevan linjassa kokeen tulosten kanssa, kun A* -algoritmin halutaan tuottavan aina optimaalisia reittejä. Jacob ja muut (1999) ovat tehneet kokeita A* -algoritmin heuristiikkafunktiolla yliarvioiden kertoimen avulla maantieteellistä reittiä. He kertovat merkittävistä parannuksista algoritmin nopeudessa silloin, kun heuristiikkafunktiota muutetaan aggressiivisemmaksi. Tutkijoiden raportoimien tulosten mukaan A* -algoritmin toiminta on parhaimmillaan lähes 40 kertaa nopeampaa 16 % poikkeamalla optimaalisesta reitistä. Tässä koejärjestelyssä on saavutettu vastaavasti 40,72 kertainen parannus 12,09 % poikkeamalla (50 km/h) verrattuna oikean reitin aina tuottavaan vaihtoehtoon (120 km/h). Lisäksi Jacob ja muut (1999) ovat havainneet, että virheen määrä reitissä nousee hitaammin kuin saatava nopeushyöty, jolloin mittauksilla on selvitettävissä optimaalinen nopeuden ja laadun välinen suhde. Tämän koejärjestelyn tulokset tukevat tätä päätelmää.

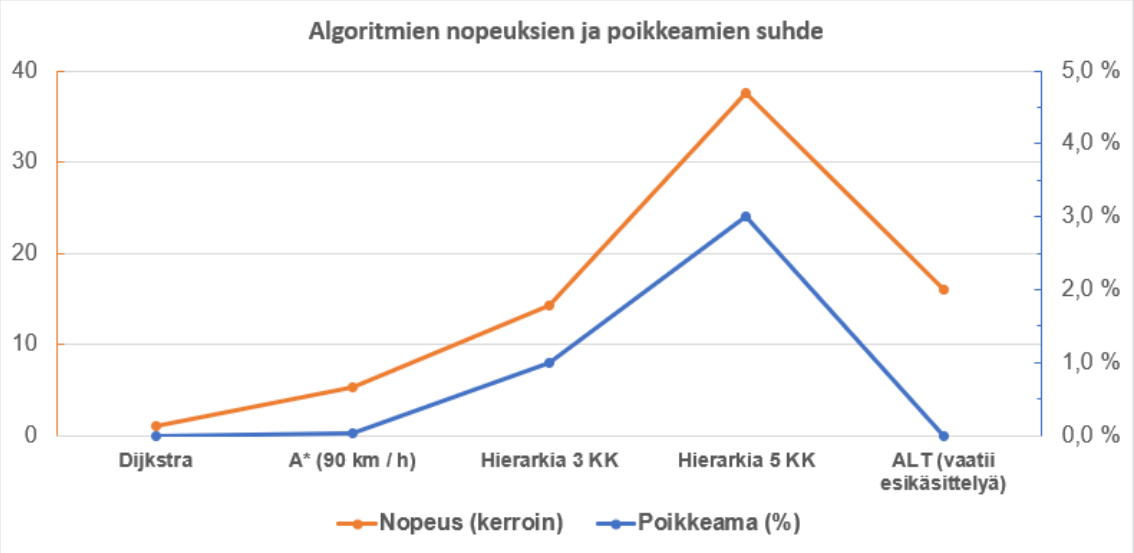
Kolmantena tutkimuskysymyksenä selvitettiin reititysalgoritmin nopeuden optimointia Digiroad-aineiston sisältämän metadatan avulla. Koejärjestelyssä hyödynnettiin metatietoja luomalla erilaisia kaksitasoisia hierarkkisia algoritmeja käyttäen pohjana A* -algoritmia. Laadukkaimmilla algoritmeilla saavutettiin 14,25–18,27 kertainen parannus Dijkstran algoritmiin verrattuna poikkeaman optimaalisesta reitistä ollen 0,79–0,92 %. Nopeimmilla algoritmeilla päästiin 26,36–37,54 kertaiseen parannukseen Dijkstran algoritmiin verrattuna, mutta poikkeama optimaalisesta reitistä oli tällöin 2,60–3,21 %.

Jagadeesh ja muut (2002) ovat testanneet artikkelissaan esikäsittelyä hyödyntävää algoritmia Singaporen kaupungin tieaineistolla. He käyttävät vertailukohtana A* -algoritmia ja kertovat, että heidän luomansa hierarkiaa soveltava algoritmi saavuttaa 52 kertaisen parannuksen reititysnopeudessa poikkeaman ollessa keskimäärin 3,31 %. Vaikka tuloksia on haasteellista verrata aineiston erilaisuuden vuoksi (suurkaupunki ja koko maa) viittaavat tulokset kuitenkin siihen, että esikäsittelyprosessilla on saavutettavissa nopeushyötyä, jota tämän koejärjestelyn hierarkkiset algoritmit eivät pysty tarjoamaan.

Goldberg ja Harrelson (2005) ovat tutkineet artikkelissaan esikäsittelyyn pohjautuvaa A* -algoritmin varianttia (ALT). Verratessa koejärjestelyn tuloksia heidän työhönsä on otettava huomioon, että he ovat raportoineet aineistollaan Dijkstran algoritmin toimivan noin kaksi kertaa nopeammin kuin A* -algoritmi, joka poikkeaa muiden tutkimusten tuloksista. Goldberg ja Harrelson (2005) ovat raportoineet tuloksensa käytetyn matka-ajan ja maantieteellisen etäisyyden mukaisesti. Kirjoittajien suurimmalla tieaineistolla (M_{11}) suorittamien kokeiden perusteella heidän algoritminsa (AL variantti) suoriutui noin 16,02 kertaa nopeammin kuin yksisuuntainen Dijkstran algoritmi sekä 28,98 kertaa nopeammin kuin A* -algoritmi matka-aikaa optimoidessa. Kaksisuuntaiseen Dijkstran algoritmiin verrattuna heidän algoritminsa oli 13,08 kertaa nopeampi. Kun tutkijoiden kaikki eri aineistokoot otetaan huomioon, voidaan tuloksista todeta ALT -algoritmin olevan keskiarvona 12,21 kertaa nopeampi kuin Dijkstran algoritmi. Verrattuna tämän koejärjestelyn edellä esitettyihin tuloksiin Dijkstan algoritmiin ja A* -algoritmiin molemmilla lähestymistavoilla voidaan päästä vastaaviin tuloksiin. Merkittävänä erona tulosten välillä Goldbergin ja Harrelsonin (2005) esikäsittelyä hyödyntävällä algoritmilla saadaan varmasti oikeat tulokset, kun taas tässä koejärjestelyssä tutkittu hierarkkinen algoritmi poikkeaa oikeasta tuloksesta 0,79–3,21 %, mutta ei vaadi aineiston esikäsittelyä.

Kokonaisuutena koejärjestelyn tuloksia voi suhteuttaa Bastin ja muiden (2016) esittämään kaavioon eri algoritmien nopeuden suhteista esikäsittelyn vaatimaan aikaan, joka on esitetty aiemmin luvun 2.3 kuvassa 1. Tässä kokeessa testatuilla hierarkkisilla menetelmillä voidaan parantaa A* -algoritmin reitinhakunopeutta niin, että niillä voidaan tavoitella parhaassa tapauksessa esikäsittelyä käyttävän ALT -algoritmin mahdollistamaa suorituskykyä. Esikäsittelystä luopumisen haittapuolena on vastaavasti poikkeaman hyväksyminen täysin optimaalisesta reitistä. Kohtuulliseen nopeusparannukseen on mahdollista päästä myös suhteellisen pienellä poikkeamalla. Toisaalta ALT -algoritmin esikäsittelyn vaatima aika on suhteellisen pieni ja siitä on mahdollista tehdä varsin nopea suhteellisen suurellakin aineistolla kuten ovat osoittaneet Efentakis ja Pfoser (2013).

Arvioitaessa reititysalgoritmien toimintaa kokonaisuutena on otettava huomioon reititysnopeuden lisäksi poikkeama optimaalisesta reitistä ja esikäsittelyn vaatima aika. Lisäksi esikäsittelyä hyödyntävien menetelmien osalta on otettava huomioon reitinhaun nopeuden ja esikäsittelyn vaatiman tallennustilan tarve. Näistä suhteista ovat kirjoittaneet aiemmin esimerkiksi Bast ja muut (2016) sekä Sommer (2014). Optimaalinen algoritmi olisi nopea, tuottaisi aina oikean tuloksen eikä vaatisi aikaa eikä tallennustilaa esikäsittelylle, mutta käytännössä kuitenkin sopiva algoritmi on valittava käyttökohteeseensa eri vaihtoehtojen joukosta sen kompromissit huomioiden. Uusin tieteellinen tutkimus on keskittynyt kehittämään menetelmiä, joka tuottavat oikean reitin nopeasti, mutta hyväksyen esikäsittelyn vaatiman ajan ja tallennustilan kompromissin. Tässä tutkielmassa on osoitettu, että klassisten reititysalgoritmien toimintaa on mahdollista nopeuttaa ja siten lähestyä yksikertaisempien esikäsittelymenetelmien (esimerkiksi ALT) saavuttamaa suorituskykyä hyväksymällä poikkeama optimaalisesta reitistä. Reititysnopeuden ja poikkeaman suhteen näitä tuloksia on esitetty kuvassa 16. Tulokset myös osaltaan tukevat vallitsevaa käsitystä siitä, että esikäsittelyä hyödyntävillä algoritmeilla voidaan saavuttaa huomattavasti nopeampia reititysaikoja, kuten esimerkiksi Bast ja muut (2016) ovat kirjoittaneet.



Kuva 16. Nopeuden ja poikkeaman välinen suhde verrokkialgoritmeilla, kahdella hierarkkisella algoritmilla sekä esikäsittelyä käyttävällä ALT-algoritmilla. Nopeuskertoimet ovat suhteita Dijkstran algoritmin suoritusaikoihin. ALT-algoritmin tiedot pohjautuvat artikkeliin Goldberg ja Harrelson (2005).

Verratessa tuloksia klassisista algoritmeista voidaan huomata, että optimoidessa matka-aikaa A* -algoritmi 90 km/h keskinopeusheuristiikalla tuottaa selvän nopeushyödyn verrattuna Dijkstran algoritmiin käytännössä hyvin pienellä haittapuolella poikkeamassa. Kuvassa 16 esitetty nopeuskerroin on verrannollinen Dijkstran algoritmin nopeuteen ja ALT-algoritmin tiedot on päätelty Goldbergin ja Harrelsonin (2005) julkaisemista tuloksista, jotka on saatu eri tieaineistolla. ALT-algoritmi pystyy kuitenkin tuottamaan varmasti oikeat reitit ilman poikkeamaa esikäsittelyä hyödyntämällä. Lisäksi on syytä huomioida, että reititysnopeudeltaan parempia esikäsittelyä hyödyntäviä algoritmeja on olemassa (Bast ja muut, 2016). Tässä tutkielmassa testattujen hierarkkisten algoritmien versiot 3 kuntakoodilla ja 5 kuntakoodilla on valittu esitettäväksi, koska ensimmäinen tarjoaa testatuista vaihtoehdoista pienimmän poikkeaman ja jälkimmäinen suurimman nopeuden. Muut testatut hierarkkiset algoritmit asettuvat silloin näiden ääripäiden väliin. Tulosten perustella voidaan todeta, että testatuilla hierarkkisilla menetelmillä voidaan saavuttaa selvästi parempia reititysnopeuksia kuin klassisilla algoritmeilla, jos niiden haittapuolena oleva poikkeama on hyväksyttävä.

Klassisiin reititysalgoritmeihin liittyvää tutkimusta on julkaistu runsaasti viime vuosikymmeninä eikä lähdemateriaalista ole puutetta. Osaltaan kirjallisuuskatsausta olisi mahdollista parantaa tekemällä siitä enemmän systemaattinen. Kirjallisuuskatsauksen keskeisinä lähteinä ovat eri tutkijoiden kirjoittamat kirjallisuuskatsaukset aihealueesta. Tällöin on mahdollista, että kirjallisuuskatsauksia tehneiden tutkijoiden omat näkemykset ovat voineet vaikuttaa osaltaan läpikäytyihin artikkeleihin ja sitä kautta tuloksiin. Valittu lähestymistapa on kuitenkin ollut luonteva valinta ottaen huomioon työmäärän kokonaisuutena.

Suosituksen tekemistä parhaasta algoritmista olisi helpottanut todellinen sovellus, jonka alijärjestelmäksi artefakti olisi suunniteltu, jolloin suosituksessa olisi voitu ottaa huomioon todellinen käyttöympäristö. Toisaalta esitettyjen tulosten perusteella on mahdollista evaluoida, olisiko jokin testattu ratkaisu sopiva eri käyttökohteisiin. Keskeinen kysymys tätä pohdittaessa onkin nopeuden lisäksi myös esikäsittelyajan ja sallitun poikkeaman välinen suhde, johon esitetyt tulokset antavat yhden näkökulman.

Esitettyjen tulosten rajoitteena on osaltaan tilastollisen analyysin puute, joka olisi voinut tarjota enemmän tietoa tutkittujen algoritmien toiminnasta sekä jatkokehityskohteita.

Digiroad-aineisto on laajuudeltaan erittäin kattava, mutta sen julkaisumuoto on haastava ohjelmallisen toiminnan näkökulmasta sen taulupohjaisen lähestymistavan vuoksi. Lukukoodia aineiston lukemiseksi graafimuotoon tarvittiin odotuksia enemmän ja toisaalta lukeminen vie jonkin verran laskenta-aikaa. Käytännön sovelluksen näkökulmasta olisi luontevaa kehittää erityyppinen nopeammin luettava muoto, joka sisältäisi vain käyttökohteeseen tarvittavan aineiston. Lisäksi aineistossa on jonkin verran virheitä, jotka on käsiteltävä ohjelmallisesti lukuvaiheessa, vaikka aineisto yleiseltä laadultaan onkin hyvällä tasolla.

Kirjallisuuskatsauksen ja saatujen tulosten valossa tietorakenteiden merkitys reititysalgoritmin toiminnalle on keskeistä. Aihetta on tutkittu aiemmin paljon ja tietorakenteiden tutkimuksessa on tehty merkittäviä parannuksia vuosikymmenten aikana, josta esimerkkinä artikkeli Cherkassky ja muut (1996). Useissa tutkimuksissa on keskitytty kehittämään yksittäisiä ominaisuuksia, kuten vaikkapa minimiarvon nopeaa poistoa kekorakenteesta, mutta lisäksi tutkimuksessa näyttää myös olevan oletuksia esimerkiksi liittyen kekorakenteen indeksien säilytykseen ja käsittelyyn keon ulkopuolella, joiden vaikutus tietorakenteen tehokkuudelle on suuri. Suhteellisen yksinkertainen kekorakenne näyttää tulosten valossa kuitenkin olevan edelleen hyvin toimiva ratkaisu ja eräs mielenkiintoinen jatkokehitysaskel olisikin ollut testata kekorakennetta, joka sallii useamman kuin kahden lapsiarvon käytön, joka voisi mahdollisesti parantaa algoritmien reititysnopeutta.

Koejärjestelyissä kehitetyt hierarkkiset algoritmit ovat toteutukseltaan suhteellisen yksinkertaisia ja helppoja toteuttaa. Tämä tekee niistä houkuttelevia käytännön sovelluksen toteutuksen näkökulmasta olettaen, että poikkeamat optimaalisesta reitistä ovat hyväksyttävissä. Niiden heikkoutena on kuitenkin tietyillä parametreillä riski siitä, että haettua reittiä ei löydy. Tätä riskiä olisi mahdollista pienentää tai poistaa luomalla algoritmiin mekanismi, joka tunnistaisi nämä tilanteet ja tarvittaessa muuttaisi parametrejä ajon aikana. Esimerkkeinä voisi olla vaikkapa etäisyysparametrin kasvattaminen reitin puuttuessa tai kuntarajojen tunnistaminen alku- tai loppupisteen lähellä, jolloin algoritmi voisi huomioida läheisen kuntakoodin alemman tason hierarkiaksi. Tuloksista ei käy suoraan ilmi, miten reittien poikkeama on jakautunut eri pisteparien välille, mutta on todennäköistä, että se keskittyy pienelle määrälle reittejä, jolloin kuvattu tunnistusmekanismi voisi oleellisesti vähentää poikkeamaa ja siten reittien laatua.

Jatkokehitysaskeleena olisi mielenkiintoista yhdistää kaksisuuntainen haku toteutettuihin hierarkkisiin algoritmeihin ja tutkia millaisia hyötyjä näin voitaisiin saavuttaa. Aiemmat tutkimustulokset viittaavat kaksisuuntaisuuden tuovan nopeusparannusta reititysaikoihin (Ikeda ja muut, 1994; Sanders & Schultes, 2007). Vastakohtana taas kaksisuuntaisuuden toteuttaminen lisää algoritmien toteutuksen monimutkaisuutta, josta syystä tämä askel on jätetty toteuttamatta tässä työssä. Lisäksi erillisenä jatkotutkimusaiheena olisi mahdollista tutkia miten eri suorittimien välimuisti vaikuttaa algoritmien suorituskykyyn moderneilla tietokoneilla. Tietokonelaitteistot ovat kehittyneet viime vuosina huomattavasti ja artefaktin toteutuksen aikana syntyi viitteitä siitä, että kaikki suorituskyvyn erot kahdella eri tietokoneella eivät selity pelkästään suorittimen kellotaajuuden ja suoritettujen käskyjen määrällä kellojaksossa (instructions per clock) välillä. Tätä viitettä puoltaa myös osaltaan aiempi tutkimus, joista esimerkkeinä artikkelit Jacob ja muut (1999) sekä Larkin ja muut (2014).

7. Yhteenveto

Reititysalgoritmien käyttämien tietorakenteiden optimointia tutkittiin Dijkstran algoritmilla ja A* -algoritmilla. Verratessa algoritmeja Digiroad-aineistolla A* -algoritmi suoriutui etäisyyttä optimoidessa 3,59 kertaa nopeammin kuin Dijkstran algoritmi käytettäessä tietorakenteena parannettua binäärikekoa. Hierarkkisten algoritmien koejärjestelyssä matka-aikaa optimoidessa ero A* -algoritmin hyväksi oli 3,53 -kertainen huomioiden 0,03% poikkeama optimaalisesta reitistä. Tietorakenteella on merkittävä vaikutus algoritmien nopeuteen. Binäärikeolla saavutettiin huomattava nopeusparannus verraten referenssinä toimineeseen listarakenteeseen. Binäärikeon toteutustavalla on merkittävä vaikutus sen suorituskykyyn. A* -algoritmilla parannettu binäärikeko, joka toteuttaa lisätietorakenteella vakioajassa elementtien tarkistuksen ja avaimen päivityksen, suoriutui 5,32 kertaa nopeammin kuin tavallinen binäärikeko.

A* -algoritmi hyödyntää toiminnassaan heuristista funktiota, joka ohjaa sitä kohdepisteeseen. Heuristisen funktion on otettava huomioon oletettu keskinopeus, kun reittiä optimoidaan matka-aikaan pohjautuen. Koejärjestelyn avulla tutkittiin millä keskinopeudella saadaan paras tasapaino algoritmin tuottamien reittien laadun ja reititysnopeuden välillä. Digiroad-aineistolla kesäajonopeuksilla suoritetun kokeen perusteella arvioidut keskinopeudet välillä 70–90 km/h näyttävät tuottavan parhaan tasapainon nopeuden ja laadun välillä siten, että 90 km/h tuottaa laadukkaamman reitin ja 70 km/h nopeamman reititysajan. Reititysnopeuden kasvaessa reitin myös poikkeama kasvaa ja sitä kautta matka-aika lisääntyy 0,03–1,61 % verrattuna optimaaliseen ratkaisuun. Vastaavasti reititysaika vähenee 3,54–14,84 -kertaisesti. Valinta keskinopeuksien välillä on tehtävä käyttökohteen vaatimusten mukaan.

Digiroad-aineiston sisältämän toiminnallisen luokan ja kuntakoodin perusteella luotiin kokeellisia algoritmeja, joiden avulla tutkittiin kaksitasoista hierarkiaa reititysnopeuden parantamiseksi. Laadultaan parhaat tulokset saavutettiin käyttämällä ylemmällä tasolla luokkia 1–4 ja 15–30 km etäisyyttä alku- ja loppupisteestä tason siirtymänä. Kuntakoodin käyttö siirtymässä paransi laatua vain hieman. Verrattuna Dijkstran algoritmiin nämä hierarkkiset algoritmit olivat 14,25–18,27 kertaa nopeampia ja verrattuna A* -algoritmiin 90 km/h keskinopeusheuristiikalla 4,04–5,17 kertaa nopeampia poikkeaman optimaalisesta reitistä ollen 0,79–0,92 %. Nopeimmat tulokset saavutettiin ylemmän hierarkiatason sisältäen toiminnalliset luokat 1–3 ja tason siirtymän ollessa 15–30 km etäisyys sekä kuntakoodin vaihdos alku- ja loppupisteestä. Tällöin verrattuna Dijkstran algoritmiin saavutettiin 26,36–37,54 kertainen parannus ja A* -algoritmiin verrattuna 7,47–10,63 -kertainen parannus, mutta poikkeama optimaalisesta reitistä oli tällöin 2,60–3,21 %.

Sovellusten näkökulmasta saatuja tuloksia voidaan soveltaa reitinhakutoimintojen nopeuttamiseksi. Dijkstran ja A* -algoritmit ovat yksinkertaisia toteuttaa eivätkä vaadi aineiston esikäsittelyä. Oikean tietorakenteen, keskinopeusheuristiikan ja hierarkian avulla A* -algoritmista on mahdollista saada suhteellisen yksinkertaisilla muutoksilla huomattavasti nopeampi ja näin parantaa sovelluksen käyttökokemusta. Algoritmeja on myös mahdollista mukauttaa käyttökohteeseensa nopeuden ja laadun suhteen edellä mainittujen parametrien avulla. Esikäsittelyä hyödyntävillä menetelmillä on mahdollista saavuttaa vielä parempia reititysnopeuksia ja optimaalisia reititystuloksia, jos esikäsittelyn vaatima aika ja tallennustila ovat hyväksyttäviä käyttökohteessa.

Lähteet

Barbehenn, M. (1998). A Note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. *IEEE Transactions on Computers*, 47(2), 263.

Bast, H. (2009). Car or public transport – two worlds. In Albers, S., Alt, H., & Näher, S. (Toim.), *Efficient Algorithms* (pp. 355-367). Berlin: Springer.

Bast, H., Delling, D., Goldberg, A., Müller-Hannemann, M., Pajor, T., Sanders, P., et al. (2016). Route planning in transportation networks. In L. Kliemann & P. Sanders (Toim.), *Algorithm engineering* (pp. 19-80). Cham: Springer.

Bollobás, B. (1998). *Modern Graph Theory*. New York: Springer-Verlag.

Car, A., & Frank, A. (1994). General principles of hierarchical spatial reasoning - the case of wayfinding. *Proceedings of the 6th International Symposium on Spatial Data Handling*, *2*, 646-664.

Cherkassky, B. V., Goldberg, A. V., & Radzik, T. (1996). Shortest paths algorithms: theory and experimental evaluation. *Mathematical Programming*, 73(2), 129-174.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms (3rd. ed).* MIT Press.

Dantzig, G. B. (1960). On the shortest route through a network. *Management Science*, 6(2), 187-190.

Delling, D., Goldberg, A. V., Pajor, T., & Werneck, R. F. (2011). Customizable route planning. *Proceedings of the 10th International Symposium on Experimental Algorithms*, 376-387.

Digiroad (2019a). *Digiroad Kansallinen tie- ja katuverkon tietojärjestelmä*. Lainattu 30.11.2019, saatavilla: https://vayla.fi/avoindata/digiroad

Digiroad (2019b). *Digiroad tietolajien kuvaus*. Lainattu 30.11.2019, saatavilla: https://vayla.fi/avoindata/digiroad/dokumentit

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik,* 1(1), 269-271.

Dillenburg, J. F., & Nelson, P. C. (1995). Improving search efficiency using possible subgoals. *Mathematical and computer modelling*, 22(4-7), 397-414.

Easy GIS .NET (2019). *Easy GIS .NET 4.5 API Reference*. Lainattu 8.2.2019, saatavilla: https://www.easygisdotnet.com/api/ReferenceAPI.aspx

Efentakis, A. & Pfoser, D. (2013). Optimizing landmark-based routing and preprocessing. *Proceedings of the Sixth ACM SIGSPATIAL International Workshop on Computational Transportation Science (IWCTS '13)*, 25-30.

Efentakis, A., Pfoser, D., & Voisard, A. (2011). Efficient data management in support of shortest-path computation. *Proceedings of the 4th ACM SIGSPATIAL International Workshop on Computational Transportation Science*, 28-33.

Fredman M. L., & Tarjan, R. E. (1984). Fibonacci heaps and their uses in improved network optimization algorithms. *Proceedings of the 25th Annual Symposium on Foundations of Computer Science*, 338-346.

Fu, L. (1996). Real-time vehicle routing and scheduling in dynamic and stochastic traffic networks. *Unpublished Ph.D. Dissertation. University of Alberta, Edmonton.*

Fu, L., Sun, D., & Rilett, L.R. (2006). Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, *33*, 3324-3343.

Goldberg, A., & Harrelson, C. (2005). Computing the shortest path: A* search meets graph theory. *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms (SODA '05)*, 156-165.

Goldberg, A., & Tarjan, R. (1996). Expected performance of Dijkstra's shortest path algorithm, Technical Report TR-530-96, *NEC Research Institute Report, Princeton University*.

Goldberg, A., & Werneck, R. (2005). Computing point-to-point shortest paths from external memory. *Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05)*, 26-40.

Hart, P., E., Nilsson, N., J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions of Systems Science and Cybernetics*, 4(2), 100-107.

Hevner, A. R., March, S., T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, 28(1), 75-105.

Hevner, A. R. (2007). A three cycle view of design science research. *Scandinavian Journal of Information Systems*, 19(2), 87-92.

Holzer, M., Schulz, F., Wagner, D., & Willhalm, T. (2005). Combining speed-up techniques for shortest-path computations. *Journal of Experimental Algorithmics (JEA)*, 10, 2.5-es.

Honkanen, R. (2011). Matrix Based Calculation of All-Pairs Shortest Paths on the GPU. In Computer science I like - proceedings of miniconference on 4.11.2011 (Toim. Martti Penttonen). *Publications of the University of Eastern Finland. Reports and Studies in Forestry and Natural Sciences*, 6, 87-100.

Iivari, J. (2007). A paradigmatic analysis of information systems as a design science. *Scandinavian Journal of Information Systems*, 19(2), 2007.

Ikeda, T., Hsu, M., Imai, H., Nishimura, S., Shimoura, H., Hashimoto, T., et al. (1994). A fast algorithm for finding better routes by AI search techniques. *Proceedings of the 1994 Vehicle Navigation and Information Systems Conference (VNIS'94)*, 291-296.

Jacob, R., Marathe, M., & Nagel, K. (1999). A computational study of routing algorithms for realistic transportation networks. *Journal of Experimental Algorithmics (JEA)*, 4, 6-es.

Jagadeesh, G. R., Srikanthan, T., & Quek, K. H. (2002). Heuristic techniques for accelerating hierarchical routing on road networks. *IEEE Transactions on intelligent transportation systems*, 3(4), 301-309.

Karimi, H. A. (1996). Real-time optimal-route computation: A Heuristic Approach. *ITS Journal - Intelligent Transportation Systems Journal*, 2(3), 111-127.

Kuntaliitto (2019). *Kuntien pinta-alat ja asukastiheydet*. Lainattu 12.4.2020, saatavilla: https://www.kuntaliitto.fi/tilastot-ja-julkaisut/kaupunkien-ja-kuntien-lukumaarat-ja-vaestotiedot/kuntien-pinta-alat-ja-asukastiheydet

Kwa, J. (1989). BS∗: An admissible bi-directional staged heuristic search algorithm, *Artificial Intelligence*, 38(1), 95-109.

Laki tie- ja katuverkon tietojärjestelmästä (2003). *28.11.2003/991*. Lainattu 8.2.2019, saatavilla: http://www.finlex.fi/fi/laki/ajantasa/2003/20030991

Larkin, D. H., Sen, S., & Tarjan, R. E. (2014). A back-to-basics empirical study of priority queues. *Proceedings of the Sixteenth Workshop on Algorithm Engineering and Experiments (ALENEX)*, 61-72.

Liikennevirasto (2017). Autojen nopeudet maanteillä vuonna 2016. Lainattu 10.3.2020, saatavilla: https://julkaisut.vayla.fi/pdf8/lts_2017-23_autojen_nopeudet_web.pdf

Liu, B. (1997). Route finding by using knowledge about the road network. *IEEE transactions on systems, man, and cybernetics - Part A: Systems and humans*, 27(4), 436-448.

Microsoft (2012). *C# Language Specification 5.0*. Lainattu 16.2.2019, saatavilla: https://www.microsoft.com/en-us/download/details.aspx?id=7029

Microsoft (2019a). *Random class (System)*. Lainattu 22.3.2020, saatavilla: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8

Microsoft (2019b). *Stopwatch class (System.Diagnostics)*. Lainattu 23.3.2020, saatavilla: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=netframework-4.8

Microsoft (2019c). *Dictionary class (System.Collections.Generic.Dictionary)*. Lainattu 5.4.2020, saatavilla: https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?view=netframework-4.8

Nielsen, J. (1994). *Usability engineering*. San Francisco: Morgan Kaufmann.

NLog (2020). *Flexible & free open-source logging for .NET*. Lainattu 24.3.2020, saatavilla: https://nlog-project.org/

Pohl, I. S. (1971). Bi-directional search. *Machine Intelligence*, 6, 127-140.

Sanders, P., Schultes, D. (2006). Engineering highway hierarchies. *Proceedings of the 14th European Symposium on Algorithms (ESA 2006)*, 804-816.

Sanders, P., & Schultes, D. (2007). Engineering fast route planning algorithms. *Proceedings of the Sixth International Workshop on Experimental Algorithms*, 22-36.

Sommer, C. (2014). Shortest-path queries in static networks. *ACM Computing Surveys (CSUR)*, 46(4), 45.

Tiehallinto (2009). *Nopeusrajoitukset*. Lainattu 10.3.2020, saatavilla: https://julkaisut.vayla.fi/thohje/pdf/2100063-v-09-nopeusrajotukset.pdf

Vuillemin, J. (1978). A data structure for manipulating priority queues. *Communications of the ACM*, 21(4), 309-315.

Wagner, D., & Willhalm, T. (2007). Speed-up techniques for shortest-path computations. *Annual Symposium on Theoretical Aspects of Computer Science (STACS 2007)*, 23-36.

Williams, J. W. J. (1964). Algorithm 232 – heapsort. *Communications of the ACM*, 7(6), 347-348.

Zhan, F. B., & Noon, C. E. (1998). Shortest path algorithms: an evaluation using real road networks. *Transportation Science*, 32(1), 65-73.