Pro gradu -tutkielma
Maantiede
Geoinformatiikka

VEKTORI- JA RASTERIAINEISTOJEN YHDISTÄMINEN KUSTANNUSPINNAKSI REITINETSINTÄÄ VARTEN

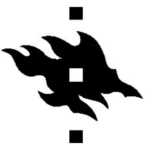
Elias Annila

2018

Ohjaaja:
Petteri Muukkonen

HELSINGIN YLIOPISTO
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
GEOTIETEIDEN JA MAANTIETEEN OSASTO
MAANTIEDE

PL 64 (Gustaf Hällströmin katu 2)
00014 Helsingin yliopisto



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta/Osasto Fakultet/Sektion – Faculty	Laitos/Institution– Department
Matemaattis-luonnontieteellinen tiedekunta	Geotieteiden ja maantieteen osasto

Tekijä/Författare – Author

Elias Annila

Työn nimi / Arbetets titel – Title

Vektori- ja rasteriaineistojen yhdistäminen kustannuspinnaksi reitinetsintää varten

Oppiaine /Läroämne – Subject

Maantiede (geoinfomatiikka)

Työn laji/Arbetets art – Level	Aika/Datum – Month and year	Sivumäärä/ Sidoantal – Number of pages
Pro gradu	Joulukuu 2018	69s

Tiivistelmä/Referat – Abstract

Alhaisimman kustannuksen reitin etsintä on paikkatietoanalyysi, jolla pyritään selvittämään edullisin mahdollinen kustannus kustannuspinnan kohteiden välillä. Perinteisesti paikkatieto-ohjelmissa analyysi on toteutettu siten, että kustannuspinta mallinnetaan rasterina, jossa jokainen solun arvo kuvaa kustannusta liikkua kyseisen solun alueella.

Kustannuspinnan mallintamiselle on rasterin lisäksi olemassa useita muitakin vaihtoehtoja, kuten kustannusviivat ja polygonit. Eri tavat mallintaa kustannuspintaa soveltuvat käytettäväksi erilaisten aineistojen kanssa, mutta mikään tavoista ei sovellu hyvin käytettäväksi kaiken tyyppisten aineistojen kanssa.

Tässä työssä esitetään menetelmä, jolla eri aineistotyyppejä voidaan yhdistää yhdeksi kustannuspinnaksi ja muodostaa siitä yhtenäinen verkko reitinetsintää varten. Esitelty menetelmä hyödyntää kuudentoista naapurin menetelmää rasteriaineistoissa ja näkyvyysverkkoon perustuvaa menetelmää aluemaisten vektorikohteiden osalta. Työssä esitellään kuinka erityyppisten aineistojen yhdistäminen yhdeksi verkoksi tehdään teoriatasolla, selvitetään kuinka tämä verkonmuodostus saadaan tehtyä tehokkaasti käytännön sovelluksessa ja testataan algoritmin toimintaa käytännön tapaustutkimuksen kautta.

Kehitetty menetelmä todettiin tapaustutkimuksessa käyttökelpoiseksi ja sen havaittiin mahdollistavan entistä monimutkaisempien kulkukustannukseen vaikuttavien ilmiöiden mallintamisen käyttäen tavanomaisia paikkatietoaineistoja. Kehityskohteita havaittiin liittyen algoritmin tehokkuuteen, tarkkuuteen ja algoritmin vaatiman kustannuspinnan muodostamisen helppouteen. Useisiin havaituista kehityskohteista esitetään mahdollisia ratkaisuvaihtoehtoja, joiden käytännön toteutus jätettiin kuitenkin jatkotutkimuksen kohteeksi.

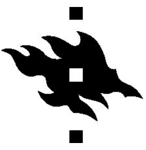
Avainsanat – Nyckelord – Keywords

Reitinoptimointi, kustannuspinta, alhaisimman kustannuksen reitinetsintä, näkyvyysverkko, GIS

Säilytyspaikka – Förvaringställe – Where deposited

E-thesis

Muita tietoja – Övriga uppgifter – Additional information



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Tiedekunta/Osasto Fakultet/Sektion – Faculty	Laitos/Institution– Department
Faculty of Science	Department of Geosciences and Geography

Tekijä/Författare – Author

Elias Annila

Työn nimi / Arbetets titel – Title

Combining vector and raster data into a cost surface for cost path analysis

Oppiaine /Läroämne – Subject

Geoinformatics

Työn laji/Arbetets art – Level	Aika/Datum – Month and year	Sivumäärä/ Sidoantal – Number of pages
Master's thesis	December 2018	69p.

Tiivistelmä/Referat – Abstract

Searching for a least cost path is an analysis which is used to find out a path that minimizes cost of travelling through a cost surface between given start and end points. Traditionally in GIS context the cost surface has been modelled using a raster layer in which value of each cell represents the cost of travelling through the cell.

There exists several other options besides a raster layer for modelling the cost surface. For example polygonal areas or linear features may be used. Different methods for modelling the cost surface have their own merits and each is particularly well suited for specific application. None of the methods however excel at representing all kinds of features that may be found on a cost surface.

In this thesis a method for combining different types of datasets into a composite cost surface is presented as well as a method for creating a navigable graph from this composite cost surface. The presented method utilizes sixteen connected raster approach on raster datasets and takes a visibility graph based approach when dealing with polygonal cost areas. The method for combining different datasets is discussed on theoretical level as well as in terms of creating an efficient practical application. The application is then tested through a case study to assess its usability.

In the case study the method for combining different types of data into one cost surface was found out to be a viable idea and that it allows for modelling increasingly complex cost related phenomenon using typical GIS data. Potential for further development was found related to efficiency and accuracy of the used algorithm as well as related to the ease of creating a navigable cost surface. Possible solutions are discussed to many of these development points, but practical applications of these are left for further research.

Avainsanat – Nyckelord – Keywords

path optimization, least cost path, cost surface, visibility graph, GIS

Säilytyspaikka – Förvaringställe – Where deposited

E-thesis

Muita tietoja – Övriga uppgifter – Additional information

Sisällysluettelo

1. Johdanto **2**

2. Teoriatausta **4**

2.1 Reitin etsinnän teoriaa 4

2.2 Kustannuspinnan muodot ja verkonmuodostus 6

2.3 Reitinetsintä paikkatietosovelluksissa 14

3. Aineistot ja menetelmät **15**

3.1 Komposiittikustannuspinta 15

3.2 Verkkojen yhdistäminen 16

3.3 Toteutettu reitinetsintäsovellus 20

3.4 Tietorakenteet 20

3.5 Syöte 23

3.6 Kustannuspinnan lukeminen 24

3.6.1 Polygonit 24

3.6.2 Rasterit 27

3.6.3 Viivat 29

3.6.4 Pisteet 32

3.7 Käsiteltävän solmun valinta 32

3.8 Solmun naapurien löytäminen 34

3.9 Tapaustutkimus maastonavigoinnista 41

3.10 Aineiston esittely 42

3.11 Kustannuspinnan muodostaminen 43

3.12 Tuotettujen reittien arviointi 48

4 Tulokset **50**

4.1 Reitinetsintään käytetty aika 50

4.2 Tuotettujen reittien laadun arviointi 50

5. Johtopäätökset ja keskustelu **56**

5.1 Keskustelu tapaustutkimuksen tuloksista 56

5.2 Kustannuspinnan muodostamisen haasteet 57

5.3 Mahdollisia jatkokehityksen kohteita 58

5.3.1 Kustannusalueiden spatiaali-indeksointi 58

5.3.2 Rasteritason ja määrittelypolygonin solmujen välisten kaarten määrittäminen 59

5.3.3 Jatkuvan kustannusrasterin muodostaminen 60

5.3.4 Tuotettujen reittien korjaaminen jälkikäteen 62

5.4 Toteutetun algoritmin hyödyllisyys 63

5.5 Vaihtoehtoisia lähestymistapoja kustannuspinnan mallintamiseen 65

5.6 Loppusanat 66

KIRJALLISUUS **67**

LIITTEET **69**

1. Johdanto

Paikkatieto-ohjelmistoissa on tarjolla useita erityyppisiä työkaluja alhaisimman kustannuksen reitin etsimistä varten. Suurin osa työkaluista on tarkoitettu reitin etsimiseen rasterimuotoisesta kustannuspinnasta, mutta myös vektoripohjaisia algoritmeja on tarjolla. Eri menetelmien paremmuudesta on julkaistu useampia tutkimuksia vaihtelevin lopputuloksin (Bemmelen ym. 1993, Antikainen 2009, Annila 2016). Yhteistä näille tutkimuksille on kuitenkin ollut, että niissä on havaittu sekä rasteri- että vektoripohjaisten menetelmien sisältävän omat heikkoutensa ja vahvuutensa.

Eri reitinetsintämenetelmien hyvät ja huonot puolet tulevat esiin eri tavoin riippuen käytettävän aineiston tyypistä. Esimerkiksi rasterimenetelmien liikkumissuuntien rajoittuneisuudesta johtuvat virheet korostuvat kustannusalueiden ollessa suuria ja homogeenisiä (Antikainen 2009), kun taas vektoripohjaiset menetelmät ovat erityisen tehottomia liukuvasti muuttuvien rasterista johdettujen kustannuspintojen kanssa.

Myös lähtöaineiston muuntaminen analyysin vaatimaan muotoon tuottaa omat ongelmansa ja virhelähteensä. Esimerkiksi tieverkostoa rasteroidessa täytyy viivamaisten kohteiden ympärille laskea jokin bufferi (Rothley 2005, Dean, Vaishnavi & Neeraj 2016), mikä voi olla ongelmallista tiheän viivaverkoston kanssa työskennellessä. Lisäksi, jos viivakohteelle on esimerkiksi määritelty eri kulkukustannukset eri suuntiin kuljettaessa, on ilmeistä, ettei aineiston muuttaminen rasteri- tai polygonimuotoon ole täysin yksinkertaista.

Tehtiin reitinetsintä sitten viivojen, rasterin tai polygoniaineiston pohjalta, ovat käytettävät reitinetsintäalgoritmit kuitenkin hyvin samanlaisia. Kaikissa menetelmissä muodostetaan ainakin konseptuaalisesti solmuista ja niitä yhdistävistä kaarista koostuva verkko, jossa varsinainen reitinetsintä tehdään (Antikainen 2009). Ainoa ero eri aineistotyyppien kanssa käytettävien menetelmien välillä on, kuinka kustannuspinnan kohteet määrittelevät luotavan verkon solmut ja kaaret. Kun kustannuspinnasta laadittava verkko on saatu määriteltyä, itse reitinetsintä tehdään aineiston tyypistä riippumatta käyttäen täysin samoja algoritmeja.

Koska eri aineistotyypit kuvaavat selvästi parhaiten erityyppisiä aineistoja, tuntuisi järkevältä pyrkiä kuvaamaan jokainen kustannuspinnan osa sitä parhaiten kuvaavalla aineistotyypillä. Tiet kuvataan viivoina, järvet polygoneina, korkeusvaihtelut rastereina ja niin edelleen. Kun eri aineistojen kanssa käytettävät analyysimenetelmät vielä ovat pohjimmiltaan samanlaisia, ei ole mitään estettä sille, ettei kustannuspinta voisi koostua samaan aikaan useammasta erityyppisestä aineistosta, jotka yhdistetään vasta reitinetsintäalgoritmin toimesta yhdeksi yhtenäiseksi verkoksi. Aiemmissa eri aineistotyyppejä yhdistävissä tutkimuksissa on keskitytty lähinnä viiva- ja rasteriaineistojen yhdistämiseen (Choi, Um & Park 2014, Dean, Vaishnavi & Neeraj 2016).

Tämän työn tavoitteena onkin esitellä algoritmi, joka mahdollistaa rasteri-, viiva- ja polygoniaineiston yhdistämisen yhdeksi kustannuspinnaksi (kuva 1), siten että reitinetsintä etenee loogisesti kustannuspinnan osasta toiseen. Työssä esitellään algoritmin toiminta teoriatasolla ja testataan sen soveltuvuutta käytännön reitinoptimointiin metsämaastossa pienen tapaustutkimuksen avulla.

Kuva 1. Esimerkki komposiittikustannuspinnasta, jossa yhdistyvät viiva-, rasteri- ja polygonkohteet.

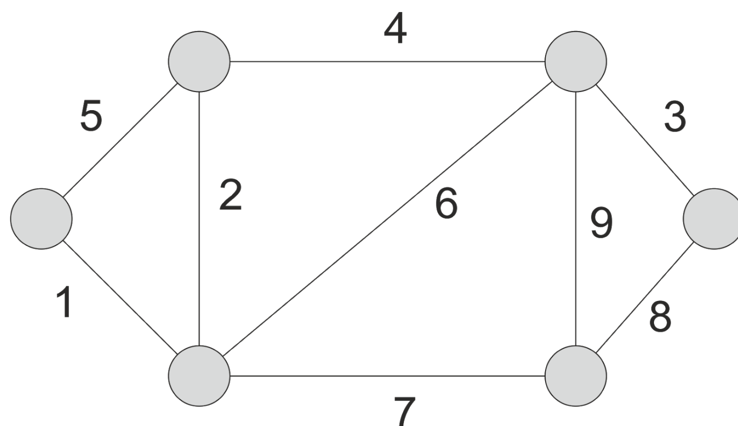
2. Teoriatausta

2.1 Reitin etsinnän teoriaa

Verkkoteoriassa verkko tarkoittaa solmuista ja niitä yhdistävistä kaarista muodostuvaa kokonaisuutta (kuva 2). Kaaret voivat olla suunnattuja tai suuntaamattomia ja kaariin voidaan liittää jokin kaaripaino. Jonkin verkossa kuljetun polun kustannus muodostuu polun varrelle jäävien kaarten painojen summasta. (Bondy & Murty 2008)

Verkossa kuljettujen polkujen kustannusten optimointi on yleinen ongelma, josta on olemassa useita erilaisia variaatioita kuten alhaisimman kustannuksen polun etsintä yhdestä solmusta muihin solmuihin, kauppamatkustajan ongelma tai alhaisimman kustannuksen polkujen etsintä kaikkien verkon solmujen välillä.

Verkossa tapahtuvaa reitinoptimointia käytetään lukuisiin eri käyttötarkoituksiin, jotka vaihtelevat tekoälyn päätöksenteosta tietoliikenneverkkojen toimintaan. Paikkatietosovelluksissa alhaisimman kustannuksen reittejä selvitetään esimerkiksi liittyen maastonavigointiin (Etula & Antikainen 2014), infrastruktuurin suunnitteluun (Feldman ym. 1995, Iqbal, Sattar & Nawaz 2006, Bagli, Geneletti & Orsi 2011) tai vaikkapa eläinten liikkeiden mallintamiseen (Desrochers ym. 2011).



Kuva 2. Painotettu suuntaamaton verkko.

Yhdestä solmusta muihin painotetun verkon solmuihin tapahtuvassa reitinoptimoinnissa yleisimmin käytetty menetelmä lienee Dijkstran algoritmi (Dijkstra 1959). Algoritmin ideana on ottaa vuoronperään käsittelyyn solmu, johon saapumisen kustannus lähtösolmusta on pienin jäljellä olevista solmuista. Jokaisen käsiteltävän solmun kohdalla päivitetään kustannus kyseisen solmun käsittelemättömiin naapureihin siten, että jos kustannus saapua naapuriin käsiteltävän solmun kautta on alhaisempi kuin alhaisin tähän asti löydetty kustannus saapua naapurisolmuun päivitetään solmuun saapumisen kustannus.

Dijkstran algoritmille on olemassa joitakin vaihtoehtoja. Mikäli emme ole kiinnostuneita kuin reiteistä tiettyihin maalipisteisiin voidaan käyttää A* algoritmia, jossa reitinetsintärintamaa ohjataan maalipisteitä kohti heuristiikan avulla (Hart, Nilsson & Raphael 1972). A* on Dijkstran algoritmia tehokkaampi vaihtoehto erityisesti tilanteissa, joissa maalipisteet ovat sijoittuneet selvästi johonkin suuntaan lähtöpisteestä.

A* ja Dijkstran algoritmi ovat pohjimmiltaan hyvin samanlaisia deterministisiä algoritmeja. Reitinetsintään voidaan kuitenkin ottaa myös hyvin erilainen lähestymistapa, jossa ei pyritä löytämään absoluuttisesti oikeaa reittiä, vaan tehokkaasti hyvä arvio parhaasta reitistä. Näiden menetelmien etuna on, että ne pystyvät löytämään suhteellisen nopeasti hyvän reitin tilanteissa, joissa kaikkein parasta reittiä etsivät algoritmit ovat liian hitaita. Yksi esimerkki tällaisesta satunnaisalgoritmista on ns. *Ant colony* menetelmä (Dorigo & Stützle 2003), joka simuloi muurahaisten tekemää feromoneihin perustuvaa reitinoptimointia.

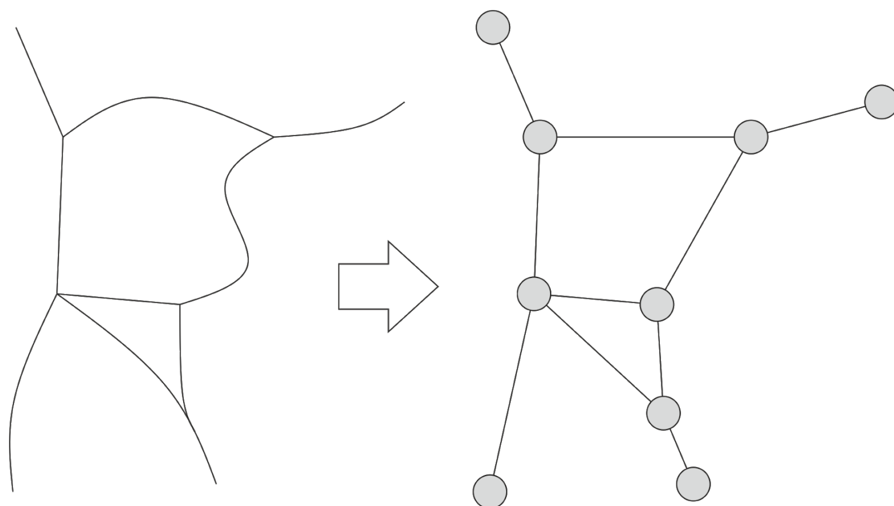
Suuria aineistoja käsitteleviä reitinetsintäsovelluksissa ei kuitenkaan useinkaan käytetä yksinkertaisia Dijkstran tai A* -algoritmeja edes silloin kun tarvitaan parasta mahdollista reittiä. Syynä tähän on se, että kustannuspinnan esikäsittelyllä voidaan huomattavasti nopeuttaa kahden pisteen välisen reitin selvittämiseen tarvittavaa aikaa. Se, millä tavalla esikäsittely tehdään, vaihtelee eri algoritmien välillä. Vaihtoehtoja on esimerkiksi verkon jakaminen osiin (Goldberg & Harrelson 2005) tai reittien laskenta tiettyjen avainsolmujen välille (Gutman 2004). Yhteistä menetelmille on, että niissä tehdään ensin kertaalleen aikaa vievä esikäsittelyvaihe, minkä jälkeen yksittäisten pisteparien väliset kyselyt saadaan tehtyä nopeasti. Usein käytännön sovelluksissa kyselyitä eri pisteiden välille tulee paljon, joten esikäsittelyyn käytettävä aika ei ole kovin merkityksellistä. Yleisesti käytettyjä esikäsittelyä

hyödyntäviä algoritmeja ovat esimerkiksi ALT (Goldberg & Harrelson 2005), Arc-Flags (Gutman 2004) ja Reach Based Pruning (Goldberg, Kaplan & F Werneck 2009).

2.2 Kustannuspinnan muodot ja verkonmuodostus

Riippumatta käytettävästä reitinetsintäalgoritmista reitinoptimointi liittyen johonkin käytännön ongelmaan toteutetaan siten, että eri vaihtoehdot reitin kulkemiselle mallinnetaan verkkona, minkä jälkeen reitin optimointi tehdään verkossa käyttäen haluttua algoritmia. Esimerkiksi tieverkosta voidaan muodostaa verkko siten, että jokainen verkon solmu kuvaa risteystä ja niitä yhdistävät tieosuudet kaaria (kuva 3), joiden painot kuvaavat tieosuuden kulkemisen kustannusta (Sadeghi-Niaraki ym. 2011). Käytännössä tieverkon mallintaminen ei kuitenkaan usein ole näin yksinkertaista johtuen esimerkiksi yksisuuntaisista teistä tai rajoituksista sallituissa kääntymissuunnissa risteyksissä (Speičys & Jensen 2008).

Reitinoptimointia voidaan haluta tehdä myös ennalta määrittelemättömän verkoston ulkopuolella. Tällöin ei ole ilmiselvää kuinka verkon solmut ja niitä yhdistävät kaaret tulisi määritellä. Esimerkki tällaisesta reitinoptimoinnista on maastossa jalan liikkumisen mallintaminen (Antikainen 2009). Maastossa kulkuvauhti riippuu eri tekijöistä kuten korkeusvaihteluista, maastonpeitteestä jne. Reitin ei kuitenkaan tarvitse kulkea minkään ennalta määrättyjen pisteiden kautta samaan tapaan kuin tieverkossa liikuttaessa vaan se voi edetä mihin tahansa suuntaan määritellyllä kustannuspinnalla.

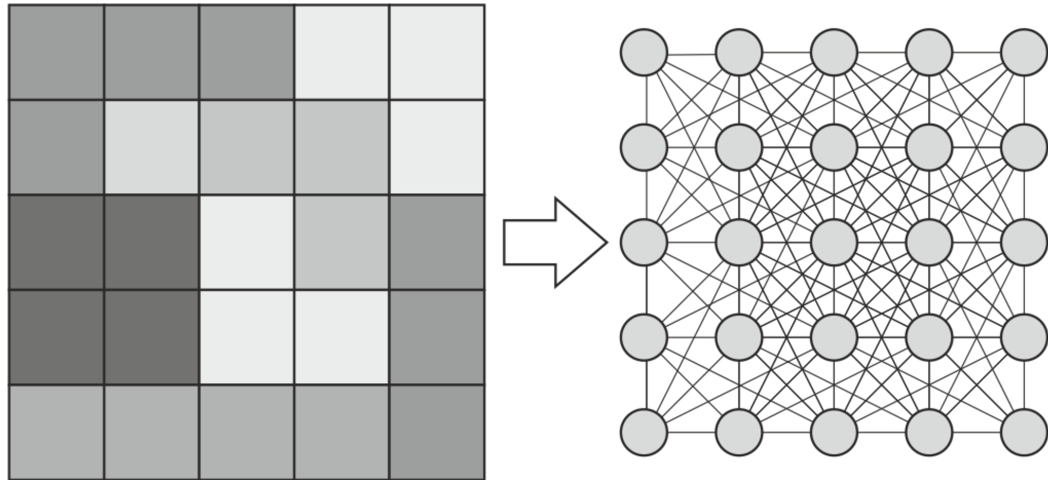


Kuva 3. Tieverkostosta reitinetsintää varten muodostettava verkko. Risteykset kuvataan solmuina ja niiden väliset tieosuudet kaarina.

Yleisimmin paikkatietoanalyyseissä käytetty kustannuspinnan muoto on kustannusrasteri. Se löytyy yleisimmistä kustannusetäisyyden laskentaa tukevista paikkatieto-ohjelmistoista (mm. Arcmap, Grass, SAGA). Kustannusrasterista voidaan muodostaa verkko reitinetsintää varten usealla tapaa.

Yleisimmin käytetty menetelmä on kuvata jokainen rasterin solu solmuna ja lisätä solmusta kaaret sitä ympäröiviin kahdeksaan solmuun (Latombe 1991, Douglas 1994). Tällöin kustannuksena naapureiden välillä käytetään solujen kustannusarvojen keskiarvoa, joka painotetaan vaaka- ja pystysuuntiin liikuttaessa solun koolla c ja diagonaalisesti liikuttaessa arvolla $c \cdot \sqrt{2}$ johtuen erisuuntaisten kaarien eri pituuksista. Menetelmän etuna on sen yksinkertaisuus, mutta ongelmana on kuitenkin rajoittuneista liikkumissuunnista johtuva virhe laskettavissa poluissa (Goodchild 1977, Bemmelen ym. 1993, Douglas 1994, Annila 2016), minkä vuoksi onkin kehitetty muita vaihtoehtoisia menetelmiä kuvata rasteria verkkona.

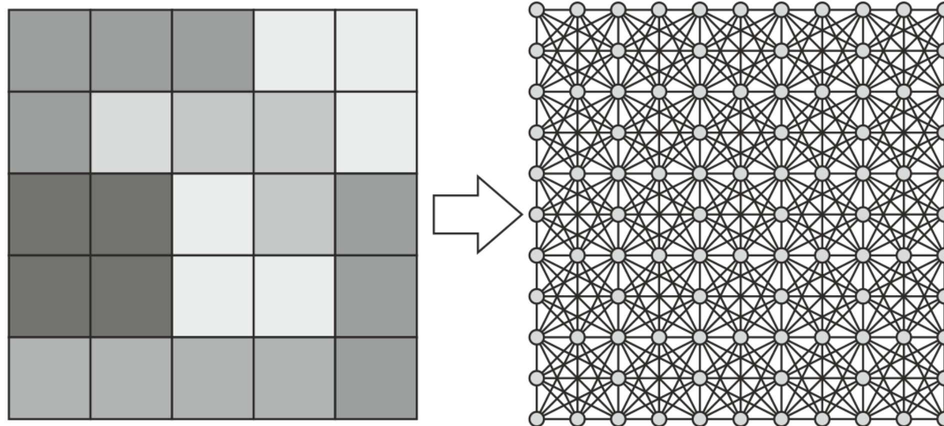
Naapuruston kokoa voidaan kasvattaa kuuteentoista tai suurempiinkin naapurustoihin (kuva 4), rajoittuneista liikkumissuunnista johtuvan virheen vähentämiseksi (Bemmelen ym. 1993, Xu & Lathrop 1995). Kun naapureiksi aletaan ottaa myös soluja, jotka eivät ole suoraan käsiteltävän solun vieressä, täytyy kaaripainoa laskettaessa ottaa huomioon myös väliin jäävien solujen kustannukset. Suuremmilla naapurustoilla myös laskenta-aika alkaa kasvaa johtuen kasvavasta kaarien määrästä verkossa. Käytettäessä tällaista yksinkertaista naapurustoa kuusitoista naapuria onkin todettu hyväksi kompromissiksi tehokkuuden ja tarkkuuden välillä (Bemmelen ym. 1993). Naapurustoa on myös mahdollista kasvattaa epäsymmetrisesti, jos on jo olemassa jonkinlainen arvio siitä, kuinka reitinetsintä tulee käyttäytymään (Xu & Lathrop 1995).



Kuva 4. Kustannusrasterista kuudentoista naapurin menetelmällä muodostettu verkko.

Toinen tapa muodostaa verkko rasterista on asettaa verkon solmut rasterin keskipisteiden sijaan niiden reunoille (kuva 5). Tällä extended raster menetelmällä voidaan naapuruston fyysistä kokoa kasvattamatta lisätä reitinetsintään haluttu määrä kulkusuuntia (Bemmelen ym. 1993). Koska solmut sijaitsevat rasterien reunoilla, muodostetaan jokaisesta solmusta kaaret saman rasterin solun alueella sijaitseviin solmuihin. Jokainen kaari siis kulkee vain yhden rasterin solun alueella ja näin kaaripaino määräytyy yksinkertaisesti rasterin solun kustannusarvon ja kaaren pituuden tulona. Jos kaari kulkee kahden solun rajaa pitkin, saa se painokseen solujen kustannuksista alhaisemman (Bemmelen ym. 1993). Koska solmujen säännöllisestä sijoittelusta johtuen kaarien pituuksia on vain rajallinen määrä, voidaan kaarien pituudet laskea etukäteen samaan tapaan kuin yksinkertaista naapurustoa käytettäessä.

Yhteistä kaikille yllä kuvatuille rasterimenetelmille on, että jokainen rasterin solu aiheuttaa verkossa uusia solmuja. Tämä on ongelmallista, sillä paikkatietoaineistoissa rasterit voivat olla suuria, mikä johtaa suureen määrään solmuja verkossa (Antikainen 2013). Verkon kasvaminen taas lisää tarvetta laskenta-ajalle ja muistille. Jokaisen solun ei kuitenkaan periaatteessa tarvitsisi aiheuttaa uutta solmua verkkoon. Verkon solmut ovat kohtia, joissa reitinetsintä voi haarautua eri suuntiin, mutta optimaalinen alhaisimman kustannuksen reitti yhdestä pisteestä muihin pinnan pisteisiin haarautuu ainoastaan kustannusalueiden rajoilla (de Berg ym. 2008). Näin ollen laajemman kustannusalueen sisään jäävien solujen ei ole reitinetsinnän kannalta välttämätöntä aiheuttaa uusia solmuja verkkoon.



Kuva 5. Extended raster menetelmässä solmut sijoitetaan rasterin solujen keskipisteiden sijaan reunoille. Näin jokainen verkon kaari kulkee yhden solun alueella.

Tämän ongelman välttämiseksi on olemassa eri strategioita yksinkertaistaa rasterista luotavaa verkkoa, esimerkiksi Quadtree- (Samet 1984) ja HPA*-menetelmä (Botea, Müller & Schaeffer 2004). HPA*:n ideana on jakaa rasteri neliskulmaisiin blokkeihin, joiden reunoille määritetään kohdat, joiden kautta blokkien välillä voidaan liikkua. Tämän jälkeen lasketaan optimaalinen reitti kunkin blokin sisällä ja lopullinen reitti kahden pisteen välillä saadaan näitä blokkien sisäisiä reittejä yhdistelemällä.

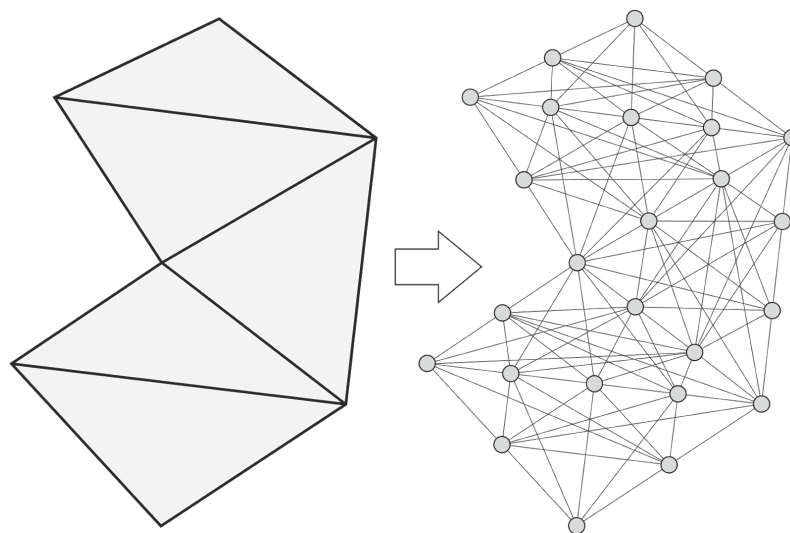
Quadtree-menetelmä on erityisen tehokas silloin kun käsiteltävä kustannuspinta koostuu laajemmista homogeenisistä alueista. Quadtree-menetelmässä rasterin sisällä yksittäiset solut korvataan suuremmilla nelikulmaisilla alueilla aina, jos isomman alueen sisälle jää vain yhtä kustannusarvoa (Samet 1984). Tässä työssä esitettävän kustannuspintoja yhdistävän menetelmän kannalta quadtree-menetelmä ei kuitenkaan ole kovinkaan kiinnostava, sillä jos kustannuspinta ei ole erityisen heterogeeninen, on polygonimuotoinen kustannuspinta luonteva vaihtoehto pinnan esittämiselle rasterin sijaan.

Toinen yleisesti paikkatieto-ohjelmistoista löytyvä kustannuspinnan muoto on viivamaisista kohteista koostuva verkko. Esimerkiksi tieverkosto. Tällaisessa verkostossa kustannus lasketaan tyypillisesti risteyksien välille siten, että jokaiselle viivasegmentille on ilmoitettu sitä pitkin liikkumisen kustannus, ja risteysten väliseksi kustannukseksi tulee niiden välisten segmenttien yhteenlaskettu pituus kerrottuna segmenttien alueella käytettävällä kustannuksella (Sadeghi-Niaraki ym. 2011). Liikkumiskustannus voi olla joko sama tai eri

kulkusuunnasta riippuen, ja viivoja pitkin etenemisen lisäksi kustannus voi tulla viivaverkostoon liittymisestä tai viivan ylittämisestä (Dean, Vaishnavi & Neeraj 2016). Mikäli halutaan saada selville kustannuksen saavuttava polku pelkän kustannuksen lisäksi, täytyy myös risteyksien välille jäävien osuuksien muodot tallentaa jotenkin.

Kolmas kustannuspinnan muoto on polygoneista koostuva luokiteltu kustannuspinta. Reitinetsintäongelmaa tällaisessa pinnassa kutsutaan tyypillisesti *weighted region problem* nimellä (Mitchell & Papadimitriou 1991). Tapoja muodostaa verkko polygoneista koostuvasta pinnassa on useita (Antikainen 2009). Verkon muodostamiseen liittyy kaksi keskeistä tekijää: Mitkä kustannuspolygonien osat kuvaavat solmuja, ja kuinka näiden solmujen väliset kaaret määritellään (Antikainen 2009)?

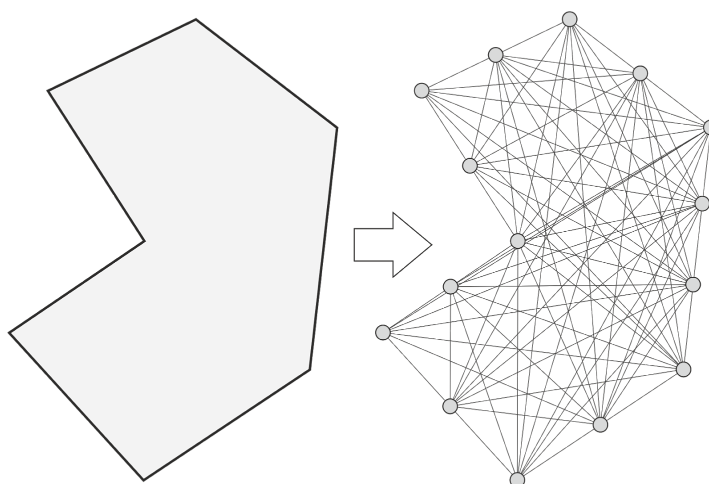
Solmujen osalta vaihtoehtoina on sijoittaa solmut polygonien sisälle tai niiden reunoille. Koska alhaisimman kustannuksen polku ei ikinä tee mutkaa homogeenisen alueen sisällä (de Berg ym. 2008), on polygonien reunat järkevä vaihtoehto solmujen sijoittelulle. Kuitenkin on myös olemassa reitinetsintämenetelmiä, joissa solmut sijoitetaan polygonin sisälle esimerkiksi varsinaisista polygoneista laskettujen Voronoipolygonien reunoille tai kolmioidun polygonin kolmioiden keskipisteisiin. Nämä menetelmät tuottavat nopean verkonmuodostuksen, tehokkaan reitinetsinnän ja reittejä, jotka ovat kaukana polygonin reunoista, mutta eivät alhaisimman mahdollisen kustannuksen polkuja (Giesbrecht 2004).



Kuva 6. Polygonista kolmioinnin pohjalta muodostettu verkko. Menetelmä johtaa solmuihin homogeenisen kustannusalueen sisällä, mikä ei ole optimaalista.

Mikäli solmut sijoitetaan polygonin reunoille, on kaarien muodostukseen käytettäväksi menetelmäksi vielä useita vaihtoehtoja. Yksi varsin yksinkertainen menetelmä on jakaa polygoni kuperiin osiin, esimerkiksi kolmioihin, jolloin jokainen saman osan sisään jäävä solmu voidaan yhdistää muihin osan solmuihin (kuva 6) (Perez Ruy-Díaz & Safar Sideq 2004). Jokaisen osan sisään voidaan luoda solmuja mielivaltaisella tavalla, esimerkiksi lisäämällä niitä luotujen kolmioiden reunoille.

Tämän menetelmän etuna on sen yksinkertaisuus, mutta mikäli reunoille laskettavien solmujen tiheyttä kasvatetaan, mikä onkin tarpeen tarkkojen polkujen selvittämiseksi, kasvaa myös kaarien määrä verkossa (Antikainen 2009). Mikäli solmuja ei lisätä riittävän tiheästi, syntyy laskettuihin reitteihin mutkia kuperien kustannusalueiden rajalla vaikka kustannus rajan molemmin puolin olisi sama, mikä johtaa varsin epätyydyttäviin polkuihin (Antikainen 2009). Toteutetussa reitinetsintäsovelluksessa onkin päädytty käyttämään hieman monimutkaisempaa näkyvyyteen perustuvaa menetelmää ja pyritty toteuttamaan näkyvyyden määrittäminen mahdollisimman tehokkaasti.



Kuva 7. Polygonista muodostettu näkyvyysverkko. Näkyvyysverkossa solmuja syntyy vain polygonin kulmiin ja reunoille, mikä takaa, ettei homogeenisen kustannusalueen sisään synny mutkia.

Polygonin reunoille sijoitetut solmut yhdistetään näkyvyyteen perustuvassa menetelmässä siten, että jokainen kustannuspolygonin reunoilla tai sisällä oleva solmu saa naapurikseen

kaikki ne solmut, jotka voidaan yhdistää käsiteltävänä olevaan solmuun suoralla janalla ilman, että jana leikkaa polygonin reunoja. Syntynyttä verkkoa kutsutaan näkyvyysverkoksi (kuva 7) (de Berg ym. 2008). Kustannus kullekin verkon kaarelle on tällöin kaaren pituus kerrottuna polygonin kustannuksella.

Näkyvyyteen perustuvan menetelmän etuna on, että se mahdollista vapaamuotoisten homogeenisten kustannuspolygonien käyttämisen kuperien polygonien sijaan. Menetelmä myös takaa, että polku on optimaalinen jokaisen yhtenäisen kustannusalueen sisällä (de Berg ym. 2008), joten se toimii erityisen hyvin tilanteissa, joissa eri kustannusluokkia on vähän ja ne muodostavat suuria yhtenäisiä alueita.

Näkyvyysverkon muodostamista pidetään usein laskentaresursseiltaan raskaana menetelmänä, mutta näkyvyysverkon muodostamiseen on olemassa myös tehokkaita algoritmeja. Koko polygonin kattava näkyvyysverkko voidaan muodostaa yksinkertaisille polygoneille ajassa $O(n + k)$ (Hershberger 1989) ja polygoneille, joissa on reikiä, ajassa $O(k + n \log n)$ (Pocchiola & Vegter 1996), missä k, kuvaa näkyvyysverkon kaarien määrää ja n polygonin solmujen määrää. Vaikka näkyvyysverkossa voikin suurimmillaan olla $n \cdot (n - 1)$ kaarta, on tarvittava solmujen määrä huomattavasti pienempi, kuin jos vastaava kustannuspinta haluttaisiin mallintaa esimerkiksi rasterimenetelmillä. Tämän takia näkyvyyteen perustuvat menetelmät voivat käytännön toteutuksissa olla laskentaresurssien tarpeeltaan täysin kilpailukykyisiä rasteripohjaisten menetelmien kanssa (Annila 2016).

Näkyvyysverkon muodostamisen suurin ongelma onkin se, että näkyvyysverkon tehokkaasti muodostavat algoritmit ovat huomattavasti monimutkaisempia toteuttaa, kuin esimerkiksi aiemmin esitellyt rasteri- ja vektoripohjaiset verkonmuodostusmenetelmät (Antikainen 2009). Yksi tapa yksinkertaistaa näkyvyyteen perustuvan menetelmän toimintaa on olla muodostamatta kokonaista näkyvyysverkkoa.

Reitinetsinnän toimimiseksi riittää etsiä yhdestä solmusta näkyvät naapurit vasta siinä vaiheessa kun solmu otetaan käsittelyyn esimerkiksi Dijkstran algoritmin toimesta (Dijkstra 1959). Näin koko verkkoa ei tarvitse välttämättä muodostaa, jos reitinetsintä ei etene kaikkiin

polygonin solmuihin. Suurimmat edut tässä menetelmässä ovat kuitenkin, ettei kaaria tarvitse tallentaa missään vaiheessa ja se että yhdestä solmusta näkyvien solmujen selvittäminen voidaan tehdä varsin yksinkertaisella algoritmilla (Lee & Preparata 1984, Annila 2016).

Muodostettiin verkko sitten näkyvyyteen tai kuperiin osiin perustuen, pitää kustannusalueiden rajoja ylittää myös muualla kuin polygonien kulmapisteissä, joten polygonien reunoille lisätään pisteitä (kuva 7). Pisteitä voidaan lisätä usealla eri strategialla, yksinkertaisimmillaan lisätään vakiomäärä pisteitä jokaiselle segmentille tai lisätään pisteet siten, että minkä tahansa kahden solmun väli muodostuu enintään halutun mittaiseksi. Solmuja voidaan kuitenkin lisätä tarkoituksella eri määrä eri alueille. Esimerkiksi reitin tarkka optimointi kahden polygonin rajalla on sitä tärkeämpää, mitä suurempi kustannus polygoneilla on. Tämän vuoksi voisi olla järkevää käyttää kustannukseen sidottua funktiota solmujen määrän päättämiseen. Myös strategioita, joissa solmuja sijoitetaan tiheämmin segmenttien päätyihin kuin keskelle, on ehdotettu (Mata & Mitchell 1997, Aleksandrov ym. 1998) algoritmeille, joissa kustannuspolygonit on jaettu ensin kolmioihin.

Näiden suorille osuuksille laskettavien reunapisteiden optimaalinen sijainti voidaan myös ratkaista perustuen Snellin taittumisyhtälöön (Lösch 1954, Mitchell & Papadimitriou 1991, Bemmelen ym. 1993). Kun tämä optimointimenetelmä yhdistetään polygonin jakamiseen kolmioihin, saadaan ratkaistua todellinen alhaisimman kustannuksen polku kustannuspinnassa. Menetelmä on kuitenkin laskennallisesti raskas, mikä tekee siitä huonosti käytännön toteutuksiin soveltuvan (Bemmelen ym. 1993).

Sen lisäksi, että polygonit voivat määrittää niiden alueella liikkumisen kustannuksen, voivat ne määritellä myös erillisen polygonin reunaa pitkin liikkumisen kustannuksen (Antikainen 2009). Reunaa pitkin kuljettaessa kustannus kahden polygonin rajalla määräytyy reunaa pitkin kulkemisen kustannus sen mukaan mikä vierekkäisten polygonien reunojen ja sisäosien kustannuksista on alhaisin (Antikainen 2009).

2.3 Reitinetsintä paikkatietosovelluksissa

Reitinetsinttään vektori- ja rasterimuotoisista paikkatietoaineistoista on kehitetty työkaluja ainakin 90-luvulta lähtien (Bemmelen ym. 1993, Douglas 1994). Suurin osa tutkimuksesta on

keskittynyt kehittämään menetelmiä yhtä aineistotyyppiä käytettäessä, ja niin vektori- kuin rasteriaineistoille onkin kehitetty useita erilaisia algoritmeja alhaisimman kustannuksen polun löytämiseksi.

Myös joitakin rasteriaineistoja viivakohteisiin yhdisteleviä menetelmiä on kehitetty (Choi, Um & Park 2014, Dean, Vaishnavi & Neeraj 2016), mutta nämä menetelmät ovat keskittyneet rasteripohjaisten menetelmien yhdistämiseen vain joko viivamaisten esteiden, tai viivamaisten kulkureittien kanssa. Näissä tutkimuksissa on myös käytetty viivojen ja rasterin solujen yhdistämiseen yksinkertaista menetelmää, jossa viivan solmusta voidaan siirtyä ainoastaan sen rasterin solun, jossa viivan solmu sijaitsee, keskipisteeseen.

Kuitenkin varsinaisissa paikkatieto-ohjelmistoissa ei tyypillisesti ole tukea muille kuin rasteripohjaisille reitinetsintämenetelmille. Myös rasteripohjaisten menetelmien toteutuksissa on eroja, ja esimerkiksi ESRI:n suosittu ArcMap -ohjelmisto tukee vain kahdeksansuuntaista reitinetsintää. Monipuolisempien reitinetsintätyökalujen huono saatavuus on ongelmallista, sillä se johtaa tilanteeseen, jossa alkeellisten työkalujen käytöstä tulee ikään kuin normaalitilanne, eikä uskoakseni suurella osalla paikkatieto-ohjelmistojen käyttäjistä ole tietoa kehittyneempien analyysimenetelmien olemassaolosta.

3. Aineistot ja menetelmät

3.1 Komposiittikustannuspinta

Mikään yllä mainituista tavoista kuvata kustannuspintaa ei sovellu hyvin kaiken tyyppisille aineistoille. Jos esimerkiksi mallinnettavassa ongelmassa kustannuspinta koostuu vaihtelevan kustannuksen maastosta ja sitä halkovasta tieverkostosta, jossa käytetään kulkusuunnasta riippuvaa kustannusta, ei mikään yllä olevista menetelmistä tarjoa hyvää ratkaisua. Perinteisissä rasterimenetelmissä jouduttaisiin ongelmiin kulkusuunnasta riippuvan kustannuksen kanssa, vaikka myös kulkusuunnasta riippuvan kustannuksen mallintaminen rasterissa on mahdollista (Collischonn & Pilar 2000). Toisaalta vektorimenetelmillä heterogeeninen kustannuspinta tuottaisi ongelmia tehokkaan laskennan kanssa.

Näiden ongelmien välttämiseksi voidaan kustannuspinta muodostaa useasta erityyppisestä karttatasosta. Tällöin kustannuspinnassa saadaan mallinnettua reaalimaailman erityyppiset kohteet niitä parhaiten kuvaavilla aineistotyypeillä. Koska kustannuspinta koostuu useasta erityyppisestä ja päällekkäisestä tasosta, jokaiselle kustannuspinnan pisteelle ei ole enää määritelty selkeää yksiselitteistä kustannusta, vaan kustannus jossakin pisteessä voi riippua muistakin tekijöistä kuin pisteen sijainnista. Esimerkiksi, jos kustannuspinnassa on viivamaisia kohteita, voi viivan kohdalla laskettava kustannus määräytyä joko viivaa pitkin kulkemisen kustannuksesta, tai viivaa ympäröivän polygonin kulkukustannuksesta.

Jos jokaiselle kustannuspinnan pisteelle, joka ei sijaitse minkään viivamaisen kohteen tai kustannuspolygonin rajan päällä, halutaan määrittää yksiselitteinen kustannus, on nähdäkseni tarpeen määritellä, etteivät kustannusalueet saa leikata toisiaan. Tämä mahdollistaa sen, että jokaiselle kustannuspinnan pisteelle voidaan selvittää, mitä kustannusaluetta pitkin siihen pääsee, selvittämällä kustannusalue, jonka sisällä se sijaitsee. Viivat ja pisteet voivat kuitenkin leikata aluemaisia kohteita vapaasti, sillä tässä työssä toteutetussa algoritmissa nämä kohteet eivät vaikuta liikkumiseen kustannuspolygonien tai rastereiden sisällä.

Koska rasteriaineistot ovat luonteeltaan nelikulmaisia, on käytännössä mahdotonta muodostaa rastereita siten, etteivät ne olisi päällekkäin epäsäännöllisen muotoisten kustannuspolygonien

kanssa. Tästä syystä jokaiselle rasteripinnalle annetaan määrittelypolygoni, joka kertoo, millä alueella kyseistä rasteria käytetään kustannuksen ilmaisuun. Tähän määrittelypolygoniin pätee toteutetussa algoritmissa samat rajoitukset kuin muihinkin polygoneihin, eli se ei saa mennä päällekkäin muiden polygonien kanssa. Itse rasterin tulee peittää koko sen määrittelypolygoni, mutta muutoin kehitetty menetelmä ei aseta rajoituksia rasterin koon tai muodon suhteen.

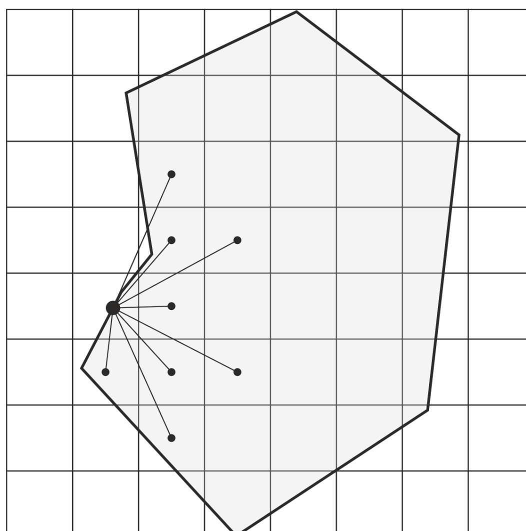
3.2 Verkkojen yhdistäminen

Kun eri kustannuspinnan osat esitetään erityyppisinä aineistoina ja niistä muodostetaan verkko eri menetelmin, on tarpeen yhdistää verkot, siten että siirtymät eri verkon osien välillä ovat loogisia ja tuottavat hyvän arvion alhaisimman kustannuksen polusta yhdistetyssä kustannuspinnassa. Tähän päästään määrittelemällä kolmen tyyppisiä siirtymiä:

1. rasteri - polygoni
2. rasteri - viiva
3. polygoni - viiva.

Kaikki siirtymät muodostavat syntyvässä verkossa suuntaamattomia kaaria, joten siirtymä esimerkiksi rasterista polygoniin on samanlainen kuin siirtymä polygonista rasteriin.

Rasterin ja polygonin välinen siirtymä tehdään hyödyntäen rasterin määrittelypolygonia. Tämän määrittelypolygonin ja sen naapuripolygonien välillä siirtyminen tapahtuu yhteisten reunasolmujen kautta samaan tapaan kuin minkä tahansa muidenkin viereisten polygonien tapauksessa. Määrittelypolygonin sisällä käytetään kuitenkin näkyvyyteen perustuvan verkon sijaan rasteripinnasta muodostettavaa verkkoa.

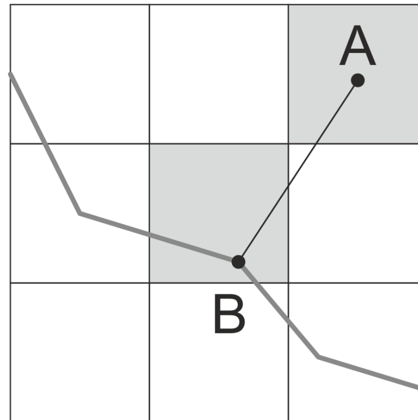


Kuva 8. Rasterin määrittelypolygonin solmusta sitä ympäröiviin rasterin soluihin muodostettavat kaaret.

Rasterin solujen keskipisteisiin sijoitettavat solmut eivät kuitenkaan osu tasan päällekkäin rasterin määrittelypolygonin solmujen kanssa. Määrittelypolygonin solmuista muodostetaan kaaret rasterin soluihin siten, että selvitetään, minkä rasterin solun alueella polygonin reunasolmu on. Tämän jälkeen reunasolmusta lisätään kaaret niihin rasterin soluihin, joiden keskipiste on polygonin sisällä ja jotka ovat sen rasterin solun naapureita, jonka sisällä reunasolmu sijaitsee (kuva 8). Kaarten kustannukset määritetään samoin kuin normaalisti rasterin solujen välillä.
Tämä johtaa pieneen epätarkkuuteen, sillä määrittelypolygonin solmu ei ole tarkasti solun keskipisteessä, joten ei ole selvää kuinka suuren osan matkasta kaari kulkee minkäkin solun alueella (kuva 9). Tämän takia on hieman virheellistä käyttää solujen kustannusten keskiarvoa, kuten tehdään normaalisti rasterin solujen välillä, varsinkin kun on mahdollista, että kaari kulkee myös solun alueella, jonka kustannusta ei oteta lainkaan keskiarvossa huomioon. Tarkan kustannuksen laskeminen jokaiselle kaarelle on kuitenkin huomattavasti hitaampaa, koska tällöin tulee selvittää, miten suuri osuus kaaresta kulkee minkäkin solun alueella.

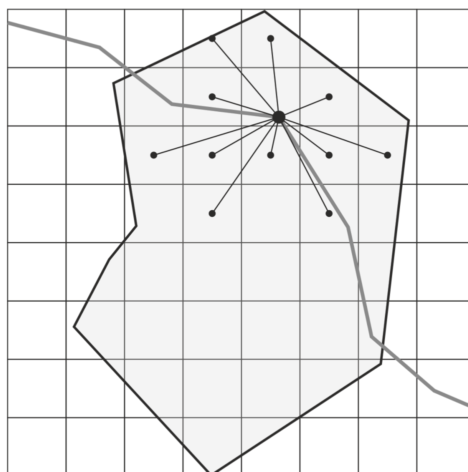
Toinen virhelähde rasterin ja polygonin väliseen siirtymään aiheutuu siitä, että määrittelypolygonin reunasolmun naapurisolun keskipiste voi sijaita määrittelypolygonin sisällä, mutta reunasolmua ja sen naapuria yhdistävä kaari ei ole koko matkaltaan

määrittelypolygonin sisällä (kuva 8). Tämä virhe voidaan poistaa tarkastamalla jokaisen kaaren kohdalla leikkaako se määrittelypolygonia ja lisäämällä vain ne kaaret, jotka eivät leikkaa. Sitä, kuinka tämä tarkastus voitaisiin tehdä mahdollisimman tehokkaasti, käsitellään tarkemmin kappaleessa 5.3.2.



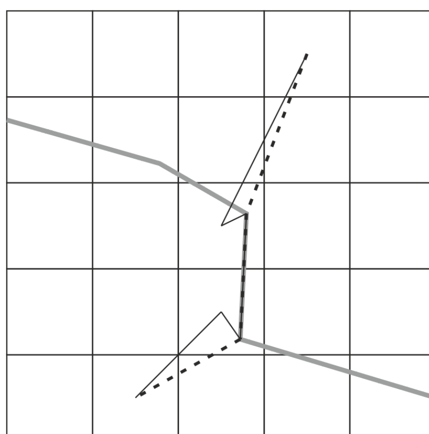
Kuva 9. Rasterin solun keskipisteessä sijaitsevan solmun A ja viivakohteen solmun B kaaren kustannus muodostuu tummennettujen solujen kustannusten keskiarvona. Tämä on kuitenkin virheellistä, sillä kaari kulkee myös kolmannen solun kautta.

Molemmat yllä kuvatuista virhelähteistä ovat pieniä, jos rasterin solukoko on pieni suhteessa polygonien yksityiskohtien kokoon, mutta kuten myöhemmin tapaustutkimuksen kohdalla huomattiin voi virhe kuitenkin olla merkittävä. Molempien virheiden korjaamista voitaisiin helpottaa huomattavasti käyttämällä laajemman naapuruston sijaan extended raster menetelmää, sillä tämä lyhentäisi rasterin solujen välille muodostuvia kaaria, mikä taas tekisi epätodennäköisemmäksi sen, että ne ylipäätänsä ylittäisivät polygonin reunoja. Lisäksi koska extended raster menetelmässä kaaret kulkevat vain yhden rasterin solun alueella, olisi jokaisen reunasolmuun rasterin solmusta kulkevan kaaren tarkan kustannuksen laskeminen huomattavasti helpompaa ja nopeampaa.



Kuva 10. Viivakohteen solmusta muodostetaan kaaret niihin rasterin soluihin, jotka ovat viivasolmun sisältävän solun naapureita.

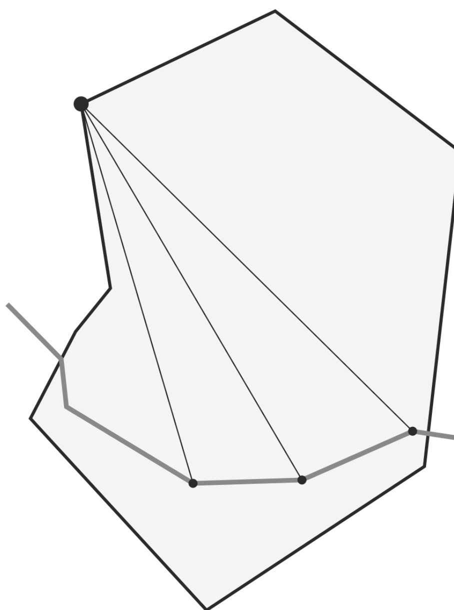
Jotta viivakohteista muodostettavaan verkkoon voitaisiin liittyä ja niistä poistua, muuallakin kuin risteyksien kohdalla lisätään viivoista muodostettavaan verkkoon solmuja kaikkiin mutkapisteisiin ja suorille osuuksille vähintään käyttäjän määrittelemän maksimietäisyyden välein. Tämän jälkeen rasterin ja viivakohteiden välinen siirtymä tehdään samoin kuin rasterin ja sen määrittelypolygonin reunasolmujen välinen siirtymä (kuva 10). Erona on kuitenkin se, että viivan ylittämiseen ei liity mitään kustannusta, joten ei tarvitse tarkistaa leikkaako kaari viivakohdetta. Toisaalta kaikkien virheiden eliminoimiseksi olisi kuitenkin tarpeen varmistua siitä, että viivapisteen ja rasterin solun välinen kaari ei leikkaa rasterin määrittelypolygonia.



Kuva 11. Jos rasterin solun sisällä oleva viivakohteen (harmaa viiva) solmu yhdistetään vain sen sisältävän solun keskipisteen naapuriksi, aiheuttaa se epäluonnollisia mutkia polkuun (yhtenäinen tumma viiva). Jos myös sisältävän solun naapurit lisätään viivakohteen naapureiksi, vältytään mutkilta (tumma katkoviiva), mutta kaaren kustannuksen määrittäminen monimutkaistuu.

Vaihtoehtoinen yksinkertaisempi tapa yhdistää rasteri- ja vektorikohteiden solmut toisiinsa olisi vain yksinkertaisesti lisätä kaari jokaisesta vektorikohteen solmusta sen solun keskipisteeseen, jonka alueella vektorikohteen solmu sijaitsee (Choi, Um & Park 2014). Tämä menetelmä johtaa kuitenkin siihen, että reitteihin syntyy viivojen ja rastereiden rajoilla epäluonnollisia teräviä mutkia (kuva 11).

Viivasolmujen ja normaalin kustannuspolygonin solmujen välinen siirtymä tehdään siten, että polygonin sisään jäävistä viivan solmuista lisätään kaaret niihin polygonin reunasolmuihin, jotka ovat nähtävissä viivasolmusta (kuva 12). Samoin toimitaan myös polygonin sisään jäävien maalipisteiden kanssa.



Kuva 12. Polygonin ja viivakohteen solmujen väliset kaaret määritetään näkyvyyteen perustuen.

3.3 Toteutettu reitinetsintäsovellus

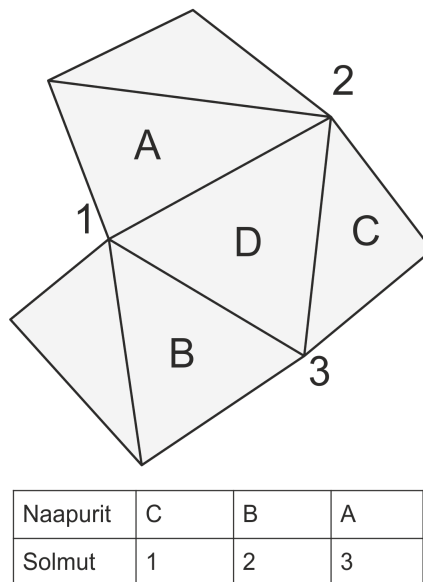
Teoriaosuudessa esitellyn kustannuspintoja yhdistävän algoritmin testaamiseksi laadittiin algoritmin toteuttava reitinetsintäsovellus. Sovellus ratkaisee alhaisimman kustannuksen polun ongelman yhdestä pisteestä useaan maalipisteeseen kustannuspinnassa, joka koostuu rasteri-, polygoni- ja viiva-aineistoista. Toteutetussa sovelluksessa valittiin rasterialueiden kohdalla käytettäväksi kuudentoista naapurin menetelmä ja polygonimuotoisten kustannusalueiden kohdalla näkyvyyteen perustuva menetelmä verkon muodostamiseen.

Varsinainen reitinetsintä tehtiin käyttäen Dijkstran algoritmia (Dijkstra 1959). Sovellus ohjelmoitiin käyttäen C++ ohjelmointikieltä, ja ohjelman lähdekoodi löytyy liitteestä 1.

3.4 Tietorakenteet

Algoritmin on tarpeen pitää kirjaa jokaiseen solmuun liittyvistä tiedoista. Jokaiselle solmulle tallennetaan sen koordinaatit, saavuttamisen kustannus lähtösolmusta, solmusta saavutettavat suorat naapurit, sekä tieto mihin polygoneihin ja kolmioihin solmu kuuluu. Solmun suorat naapurit ovat solmun naapureita, jotka selvitetään syötettä luettaessa ja pidetään tallennettuna koko ohjelman suorituksen ajan. Suorat naapurit koostuvat viiva ja rasterikohteiden perusteella muodostetuista kaarista. Tietoa polygoneista ja kolmioista, joihin solmu kuuluu, tarvitaan näkyvyyden perusteella selvitettävien naapureiden löytämiseksi.

Algoritmin toiminnassa samoissa koordinaateissa sijaitsevia pisteitä käsitellään yhtenä solmuna. Koska eri polygonien samaa pistettä kuvaavat kulmapisteet saattavat olla hieman eri kohdassa johtuen liukuluvuilla laskemisen epätarkkuudesta tulkitaan kaksi solmua samaksi solmuksi jos ne sijaitsevat alle 0.0001 karttayksikön etäisyydellä toisistaan. Metrisessä koordinaattijärjestelmässä tämä tarkoittaa 0.1 mm etäisyyttä.



Naapurit	C	B	A
Solmut	1	2	3

Kuva 13. Kolmio-tietorakenteeseen tallennettavat solmut ja naapurikolmiot kolmiolle D. Naapurikolmiot (C,B,A) ja solmut (1,2,3) tallennetaan kolmen mittaisiin taulukoihin, siten että samasta taulukon indeksistä löytyy aina solmu, ja sitä vastakkaisen sivun takana oleva naapurikolmio.

Kolmio tietorakennetta tarvitaan näkyvyyden perusteella selvitettävien naapureiden löytämiseen. Jokainen polygoni jaetaan kolmioihin solmujen välisten näkyvyyksien selvittämiseksi. Jokaisesta kolmiosta tallennetaan sen kulmapisteet, kolmion naapurikolmiot ja kolmion sisälle jäävät solmut. Kolmion kulmasolmut ja naapurikolmiot on tallennettu kolmen mittaisiin taulukoihin (kuva 13), siten että solmua vastakkaisen sivun takaa löytyvä naapuri on tallennettuna solmun indeksiä vastaavaan indeksiin naapuritaulukossa. Tämä mahdollistaa solmua vastakkaisen naapurin löytämisen tehokkaasti, mikä on tärkeää suppiloalgoritmin nopean toiminnan kannalta. Mikäli solmua vastaava naapuri puuttuu, tarkoittaa se kolmion solmua vastaavan sivun olevan polygonin reuna. Kolmion sisälle jäävät solmut voivat olla lähtö- tai maalisolmuja tai viivakohteiden solmuja.

Koska polygonit kolmioidaan vasta algoritmin edetessä, täytyy solmut, joista polygonit koostuvat, tallentaa syötteen lukemisen yhteydessä. Lisäksi jokaisesta polygonista, joka on rasterin määrittelypolygoni, täytyy tallentaa syötteen lukemisen yhteydessä määrittelypolygonin sisältämä rasteri. Rasteriin pitää päästä käsiksi, mikäli viivakohteita luettaessa löydetään viivoja, jotka kulkevat rasterin sisällä.

Syötteen lukemisen aikana on usein tarpeellista selvittää minkä polygonin sisällä maalipiste tai viivakohteen piste sijaitsee tai leikkaako viivakohteen jana jonkin polygonin reunaa. Näiden kyselyiden nopeuttamiseksi jokaisesta polygonista tallennetaan myös polygonin solmujen pienimmät ja suurimmat x- ja y-koordinaatit, eli polygonin *bounding box*. Kyselyitä voisi luultavasti vielä nopeuttaa lisää toteuttamalla jonkinlainen polygonien spatiaali-indeksointi.

Kaikki algoritmin käsittelemät solmut tallennetaan hajautustauluun käyttäen niiden x- ja y-koordinaateista muodostettua paria avaimena (James 2018). Näin voidaan vakioajassa kysyä onko jossakin koordinaatissa jo olemassa solmua, minkä avulla varmistutaan ettei mihinkään koordinaatteihin pääse syntymään päällekkäisiä solmuja. Jos solmu tulee esiin uudestaan esimerkiksi toista polygonia lukiessa, lisätään kyseinen solmu vain kuulumaan uuteen polygoniin.

3.5 Syöte

Algoritmin syöte koostuu seuraavista karttatasoista:

1. Polygonimuotoiseseta kustannuspinnasta
2. Viivoista koostuvasta kustannuspinnasta
3. Joukosta kustannusrastereita.
4. Lähtöpisteestä
5. Maalipisteistä

Kustannuspinnan eri osien ei tarvitse olla yhtenäiset vaan osa polygoneista voi olla saavutettavissa ainoastaan esimerkiksi viivoja pitkin. Alueet, joille ei ole määritelty mitään kustannuspinnan osaa tulkitaan alueiksi, joille reitinetsintä ei saa edetä missään tapauksessa. Kustannuspinta voi hyvin sisältää myös alueita, jotka eivät ole mitenkään saavutettavissa lähtöpisteestä, mutta jos tämä tiedetään jo kustannuspintaa luotaessa, on näiden alueiden lisääminen syötteeseen turhaa.

Polygonikustannuspinta sisältää normaaleja kustannuspolygoneja sekä yhden tai useamman määrittelypolygonin jokaiselle rasteritasolle. Kaikki polygonit annetaan kuitenkin algoritmille yhdessä tiedostossa. Kustannuspolygonien kustannukset on määritelty attribuuttitaulussa kahteen sarakkeeseen, joista toinen kertoo kustannuksen liikkua polygonin sisällä ja toinen kustannuksen kulkea polygonin reunaa pitkin. Polygonitason kohteiden tulee olla yksittäisiä polygoneja, eikä niin sanottuja *multipart-kohteita* saa olla.

Rastereiden määrittelypolygonit sisältävät attribuuttitaulussa polygonin alueella käytettävän rasteritiedoston tiedostopolun. Rasterikarttatasojen soluissa määritelty kustannus on voimassa ainoastaan sen määrittelypolygonin alueella. Määrittelypolygonien reunoja pitkin kulkemiselle ei ole tässä toteutuksessa määritelty erillistä kustannusta, vaikka sellainenkin olisi toki toteutettavissa.

Viivatasossa jokaiselle kustannusviivalle on määritelty attribuuttitauluun kustannus kulkea viivaa eteen- ja taaksepäin. Kulkusuuntaa määräytyy viivaa piirrettäessä, joskin se on mahdollista kääntää ympäri jälkeenpäin. Samoin kuin polygonien tapauksessa, myös

viivatason kohteiden tulee koostua vain yhdestä viivasta, eikä multipart-kohteita saa olla. Se ettei multipart-kohteita saa esiintyä syötteessä, johtuu vain toteutetun työkalun ohjelmoinnista, eikä multipart-kohteiden käsittelylle ole mitään periaatteellista estettä algoritmin toiminnassa.

Mikäli viivakohteet risteävät keskenään siten, että viivalta voi kääntyä toiselle viivalle, tulee viivojen risteyskohdassa olla solmu molemmissa viivakohteissa. Mikäli solmua ei ole tulkitaan risteys siten, että viivakohteesta toiseen ei voi siirtyä suoraan kulkematta ympäröivän kustannuspinnan kautta. Tällaisella viivojen risteyksellä voidaan mallintaa esimerkiksi alikulkutunnelia. Toteutettu algoritmi ei tue monimutkaisempia risteyksiä, joissa osa kääntymissuunnista on sallittuja ja osa ei. Tosielämän tieverkoissa tällaiset rajoitukset kuitenkin ovat yleisiä, ja myös niiden mallintamiseen on kehitetty toimivia menetelmiä (Speičys & Jensen 2008).

Lähtö- ja maalipisteet eivät sisällä mitään attribuuttitietoa, vaan ne sisältävät ainoastaan tiedot pisteiden sijainneista. Lähtöpiste on vain yksi piste, ja maalipistetaso voi sisältää mielivaltaisen määrän pisteitä. Lähtö- ja maalipisteiden tulee sijaita jonkin kustannus- tai määrittelypolygonin sisällä tai tasan viivakohteiden kulmapisteessä, jotta ne olisivat löydettävissä. Ilmeinen puute onkin tilanne, jossa maalipiste sijaitsee tasan viivalla, mutta ei sen kulmapisteessä. Tämän korjaamiseksi olisi viivoja luettaessa jokaisen viivasegmentin kohdalla tarkistettava onko jokin pistemäisistä kohteista tasan viivan päällä.

3.6 Kustannuspinnan lukeminen

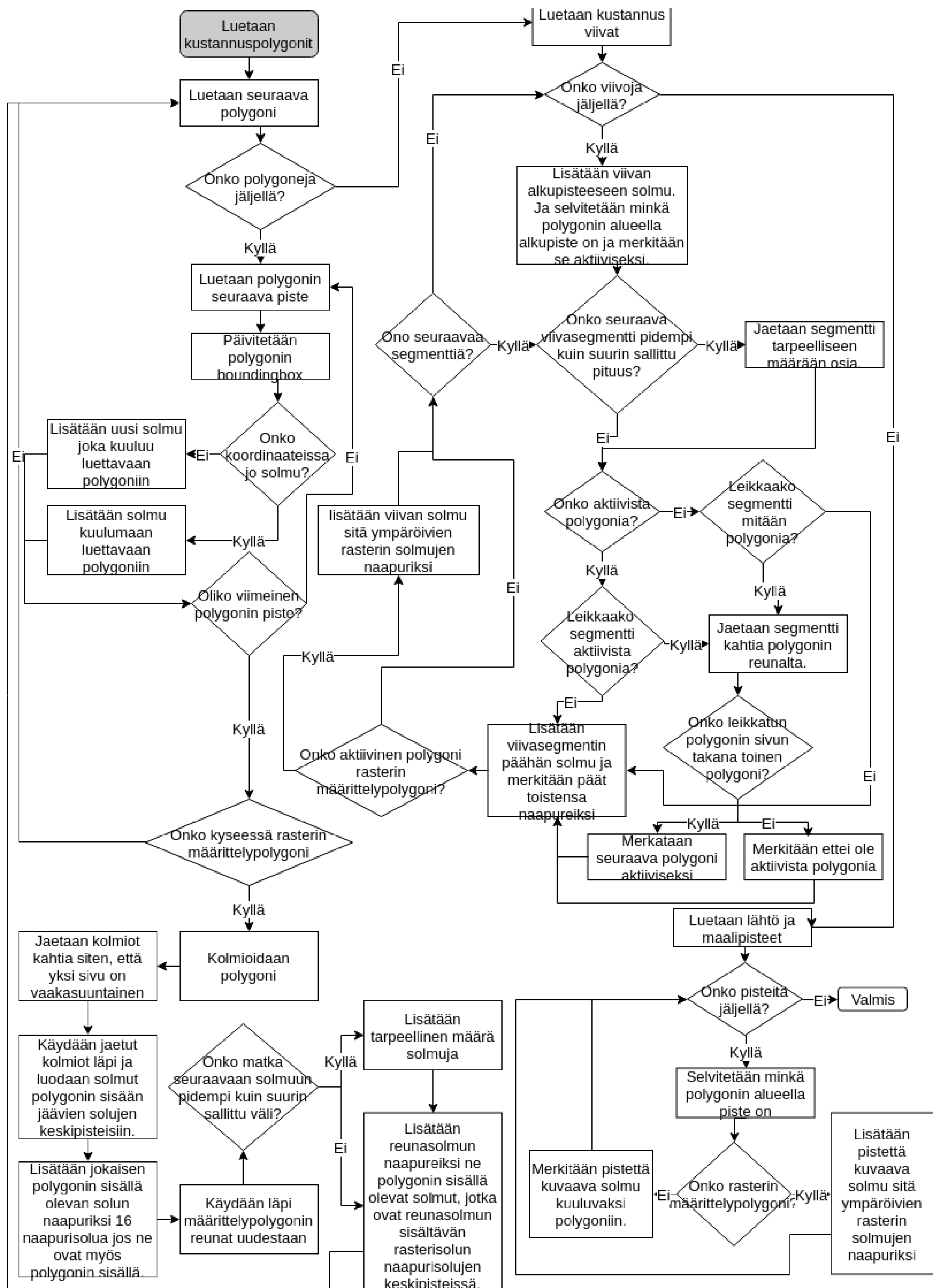
3.6.1 Polygonit

Kehitetty algoritmi lukee ensin kustannuspolygonit, sitten rasterit ja lopuksi viivat ja pisteet. Näin tehdään, koska viivojen, ja pisteiden kohdalla tulee tietää, minkä kustannuspolygonin tai rasterin alueella piste tai viiva sijaitsee. Koko syötteen lukemisprosessi on kuvattu alla olevissa kappaleissa syötteen tyyppi kerrallaan ja lopuksi yhteenvetona vuokaaviossa (kuva 14).

Polygonit luetaan solmu kerrallaan. Solmut tallennetaan hajautustauluun ja merkitään

kuuluvaksi luettavaan polygoniin. Jos polygonin solmuja lukiessa huomataan, että jokin solmuista on jo olemassa, tarkoittaa tämä sitä, että se on luotu jo aiemmin toisen polygonin yhteydessä. Tällaisessa tilanteessa ei luoda uutta solmua vaan lisätään jo olemassa olevaan solmuun merkintä, että se kuuluu polygoniin, jota oltiin lukemassa. Kun jokainen kustannuspinnan polygoni on luettu, on jokaiselle polygoniin kuuluvalle solmulle tallennettu tieto mihin polygoneihin se kuuluu. Tämä mahdollistaa reitinetsinnän siirtymisen polygonista toiseen polygonin reunat muodostavien solmujen kautta.
Kolmiointia tai siitä laskettavia näkyvyyteen perustuvia naapureita ei siis vielä lasketa tässä vaiheessa, vaan tämä tehdään myöhemmässä vaiheessa, mikäli reitinetsintä saapuu kyseisen polygonin alueelle. Kolmioinnille ei ole tarvetta vielä tässä vaiheessa, mutta sitä ei myöskään voida tehdä ennen kaikkien viivojen ja maalipisteiden lukemista, sillä näitä kohteita lukiessa voi polygonin sisään jäädä uusia pisteitä, jotka pitää ottaa huomioon kolmioinnissa.

Myöskään pisteitä, jotka sijoittuvat polygonin reunojen suorille osuuksille ei luoda vielä tässä vaiheessa, vaan vasta polygonia kolmioitaessa. Polygonin vierekkäiset reunasolmut lisätään toistensa suoriksi naapureiksi vasta kun reitinetsintä saapuu kyseiseen polygoniin ja polygoni kolmioidaan. Syötteen lukuvaiheessa polygonin reunoja pitkin kulkemisen kustannus ainoastaan tallennetaan muistiin kolmiointivaihetta varten. Tämä johtuu siitä, että reunoille laskettavat lisäpisteet pitää saada mukaan reunoja pitkin liikkumisen mallintavaan suorien naapureiden ketjuun, ja nämä lisäpisteet luodaan vasta polygonia kolmioitaessa, jolloin ne saadaan helposti lisättyä kuuluvaksi oikeisiin kolmioihin.



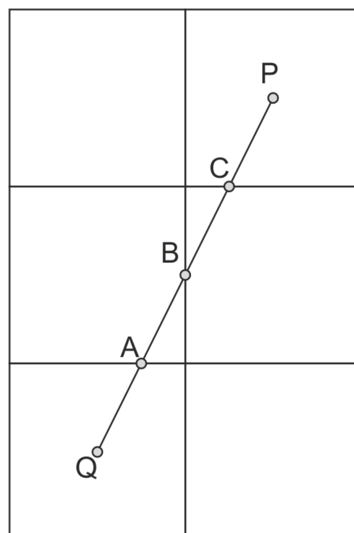
Kuva 14. Syötteen lukeminen vuokaaviona.

3.6.2 Rasterit

Mikäli polygonin kustannusrasterista kertovaan muuttujaan oli tallennettu arvo, luetaan rasteri seuraavasti: Määrittelypolygonin reunat käydään läpi, ja jokaiselle reunapisteelle lasketaan, minkä solun alueella se sijaitsee. Polygonin reunapisteen suoriksi naapureiksi lisätään sitä ympäröivät kuusitoista solua, mikäli näiden solujen keskipisteet sijaitsevat määrittelypolygonin alueella. Mikäli polygonin sivu on pidempi kuin käyttäjän määrittelemä maksimipituus lisätään sivulle tarvittava määrä solmuja ja käsitellään nämä lisäsolmut samaan tapaan kuin muutkin solmut.

Muiden rasterisolujen keskipisteisiin luodaan solmut, mikäli ne sijaitsevat määrittelypolygonin sisällä. Solmuille lisätään suoriksi naapureiksi sitä ympäröivien kuudentoista rasterin solun keskipisteisiin luodut solmut, mikäli nämä olivat rasterin määrittelypolygonin sisällä. Kunkin naapurin kustannus lasketaan solun alueella kuljetun matkan ja kustannuksen mukaan. Kaikki kaaren alueelle jäävät rasterin solut kattavat yhtä suuren osan kaaren pituudesta, joten suorien naapureiden kustannukseksi merkitään niiden solujen kustannusten keskiarvo, joiden yli kaari kulkee. Pää ja väli-ilmansuuntiin kulkeville kahdeksalle naapurille tämä tarkoittaa kahden solun keskiarvoa ja näiden välisiin suuntiin jääville ulommille naapurille neljän solun keskiarvona (kuva 15).

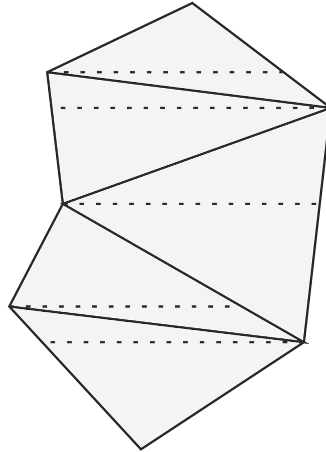
Naapurien kustannuksia ei painoteta kaaren pituuden mukaan, sillä kustannus painotetaan naapurien välisellä etäisyydellä myöhemmässä vaiheessa kaikille verkon kaarille. Tämä ei ole täysin optimaalista, sillä rasterin solujen väliset etäisyydet ovat yhtä suuria joten on turhaa laskea niitä useaan kertaan. Kuitenkin algoritmi on yksinkertaisempi toteuttaa näin, koska yhdistettäessä rasterin määrittelypolygonin reunoja rasterin solujen keskipisteissä oleviin solmuihin, eivät solmujen väliset etäisyydet ole vakioita ja ne pitäisi joka tapauksessa laskea erikseen. Nyt toteutetulla menetelmällä kaikki solmut voidaan käsitellä samaan tapaan.



Kuva 15. Solmujen P ja Q välisellä janalla pisteen (x, y) koordinaatit noudattavat kaavaa $y = 2x + y_Q - 2x_Q$. Janan tiedetään ylittävän solujen rajoja kohdissa $A = (x_A, y_Q + c \div 2), B = (x_Q + c \div 2,\ y_B), C = (x_C, y_Q + 3c \div 2)$.
Kun ratkaistaan puuttuvat x- ja y- koordinaatit saadaan ylityskohdat ja päätepisteet $Q = (x_Q, y_Q),\ A = (x_Q + c \div 4, y_Q + c \div 2),\ B = (x_Q + 2c \div 4,\ y_Q + 2c \div 2),$ $C = (x_Q + 3c \div 4, y_Q + 3c \div 2),\ P = (x_Q + 4c \div 4,\ y_Q + 4c \div 2)$.
Tarkastelemalla x- ja y-koordinaattien eroja kunkin kohdan välillä voidaan havaita x-koordinaatin kasvavan aina $c \div 4$ ja y-koordinaatin $c \div 2$ pisteiden välillä. Näin ollen myös pisteiden välisten etäisyyksien on oltava samansuuruisia.

Rasterin lisäämisen tehokkuuden kannalta on siis keskeistä saada selvitettyä nopeasti mitkä rasterin solujen keskipisteet ovat polygonin sisällä. Polygonin värittäminen on paljon tutkittu ongelma liittyen tietokonegrafiikkaan ja siihen on kehitetty useita tehokkaita algoritmeja (Theoharis ym. 2008). Värittämisessä on kyse samasta ongelmasta, eli sen selvittämisestä mitkä pikselit, tai tässä tapauksessa rasterin solut, jäävät polygonin sisään.

Tässä sovelluksessa rasterointi tehdään jakamalla polygoni kolmioihin käyttäen Seidelin algoritmia samaan tapaan kuin muitakin kustannuspinnan polygoneja kolmioitaessa. Tämän jälkeen jokainen syntynyt kolmio jaetaan vielä kahtia siten, että syntyy kaksi uutta kolmiota, joiden yksi sivu on vaakasuuntainen. Tämä saadaan tehtyä valitsemalla kolmion kärjistä pystysuunnassa keskimmäinen ja piirtämällä siitä jana vastakkaiselle kolmion kyljelle (kuva 16).



Kuva 16. Polygonin sisään jäävien rasterin solujen määrittämistä varten tehtävä kolmiointi. Syntyvässä kolmioinnissa, jokaisella kolmiolla on yksi vaakasuuntainen kylki, mikä helpottaa kolmion sisään jäävien solujen tarkistamista rivi kerrallaan.

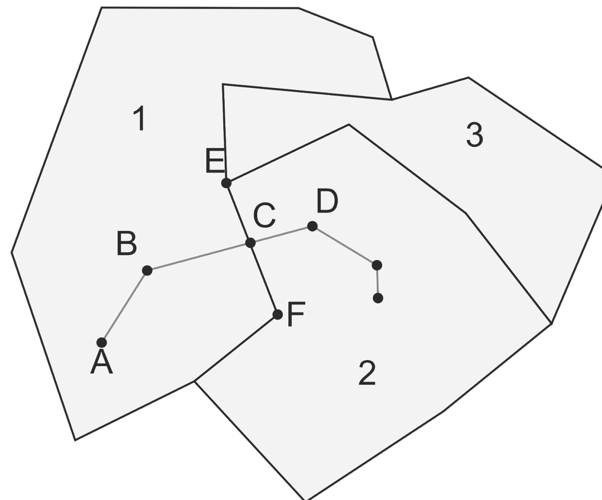
Kun nämä lopulliset kolmiot on saatu luotua, käydään jokainen kolmio läpi vaakasuuntaisesta sivusta kärkeä kohti edeten rasterin rivi kerrallaan ja merkitään kolmion vinojen kylkien väliin jäävät solut kuuluvaksi polygoniin (Theoharis ym. 2008). Kirjaa siitä, minkä solujen väliin kolmion sisään jäävä osuus jää, pidetään seuraavasti: Uudelle riville siirryttäessä tarkistetaan onko edellisen rivin alkupisteen kohdalla oleva uuden rivin solun keskipiste kolmion vasemmanpuoleisen vinon kyljen oikealla puolella. Mikäli on, tiedetään solun olevan uuden rivin aloituspiste. Mikäli ei ole, siirrytään yksi solu oikealle ja toistetaan testi. Näin jatketaan kunnes uusi alkupiste on löydetty. Rivin päätepisteen kanssa toimitaan vastaavalla tavalla.

3.6.3 Viivat

Viivakohteita luettaessa lasketaan kullekin viivan kulmapisteelle, minkä polygonin sisällä se sijaitsee ja tämä tieto tallennetaan siten, että voidaan hakea vakioajassa lista polygonin sisällä olevista viivapisteistä. Lisäksi viiva jaetaan segmentteihin, jotka ovat enintään yhden polygonin alueella ja enintään käyttäjän määräämään maksimipituuden mittaisia. Jotta viiva saataisiin jaettua segmentteihin, jotka ovat vain yhden polygonin alueella, täytyy viivaa lukiessa selvittää sen ja polygonien leikkauskohdat. Tämä tehdään yksinkertaisesti käymällä kaikki polygonin sivut läpi ja tarkistamalla mitkä niistä leikkaavat viivasegmentin.

Jotta jokaisen viivan pisteen kohdalla ei tarvitsisi käydä lävitse kaikkia polygoneja

leikkauskohtien selvittämiseksi, käydään viivat läpi aloittaen alkupisteestä. Alkupisteen kohdalla joudutaan käymään polygoneja läpi yksitellen, kunnes löytyy polygoni, jonka sisällä alkupiste on. Tämä vaihe nopeutuu huomattavasti, kun jokaiselle polygonille on tallennettu sen bounding box.



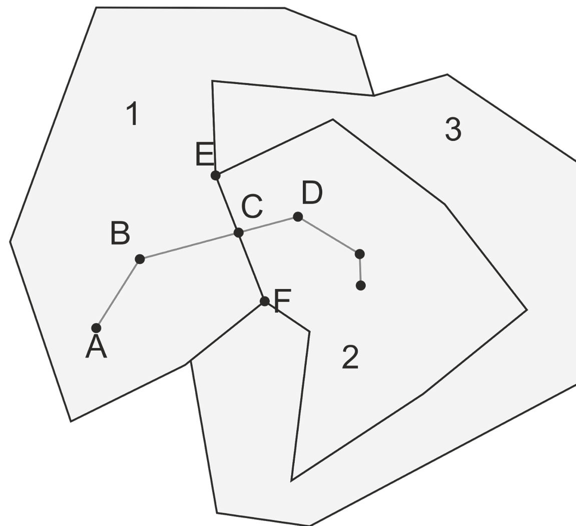
Kuva 17. Viivan pilkkominen yhden polygonin alueella oleviin osiin. Viivan solmujen B ja D välinen segmentti leikkaa polygonien 1 ja 2 reunaa solmujen E ja F välillä. Viivasegmentti (B, C) jaetaan polygonien rajalta kahteen osaan ja viivan käsittelyä jatketaan polygonin 2 sisällä.

Tämän jälkeen katsotaan leikkaako ensimmäisen ja toisen viivasolmun välinen jana polygonia. Jos ei leikkaa, merkataan seuraavakin viivan solmu kuulumaan samaan polygoniin ja lisätään viivasolmut toistensa suoriksi naapureiksi. Näin jatketaan kunnes löydetään segmentti joka leikkaa polygonin (kuva 17).

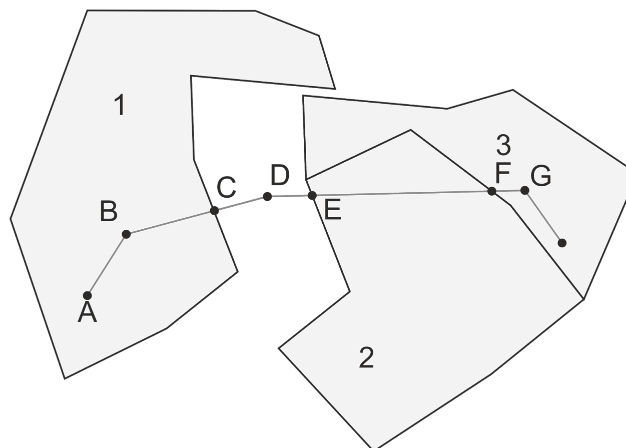
Tällaisessa tapauksessa tarkistetaan mihin kahteen samaan polygoniin leikkauskohdan molemmin puolin olevat solmut kuuluvat. Näistä kahdesta polygonista toinen on nykyinen polygoni ja toinen polygoni, johon viiva seuraavaksi etenee. Polygonien rajalle luodaan uusi solmu, joka lisätään polygonin rajan ylittävän segmentin päätepisteiden naapuriksi ja merkitään kuuluvaksi kumpaankin polygoniin.

On myös mahdollista, että polygonin rajalla olevat solmut kuuluvat useampaan kuin kahteen samaan polygoniin (kkuva 18). Tällöin joudutaan selvittämään missä näistä polygoneista

solmut muodostavat polygonin reunasegmentin. Mikäli viiva kulkee rasterin sisällä, lasketaan kullekin viivan pisteelle minkä rasterin solun alueella se sijaitsee ja lisätään tämän solun keskipisteessä sijaitseva solmu viivan solmun suoraksi naapuriksi. Tällöin kustannus muodostuu rasterin solun kustannuksen mukaan.



Kuva 18. Tilanne jossa solmut E ja F kuuluvat useampaan kuin kahteen samaan polygoniin $(1, 2, 3)$. Kuitenkin E ja F muodostavat polygonin reunasegmentin vain enintään kahdessa polygonissa $(1, 2)$.



Kuva 19. Viivasegmentti (B, C) leikkaa polygonin 1 sellaisen reunan, jonka toisella puolella ei ole mitään polygonia. Tällaisessa tilanteessa joudutaan käymään polygoneja yksitellen läpi, jotta löydetään polygoni, johon viiva seuraavaksi tulee. Viivasegmentti $(D,\ G)$ leikkaa polygonien 2, ja 3 rajat kahdessa eri pisteessä $(E\ \&\ F)$. Näistä valitaan solmua D lähempi (E), jonka jälkeen viivan käsittelyä jatketaan polygonin 2 sisällä.

Mikäli jokin viivan segmentti leikkaa polygonin sellaisen reunan, jonka toisella puolella ei ole uutta polygonia (kuva 19), joudutaan tarkistamaan leikkaako segmentti mitään muuta polygonia, missä tapauksessa voidaan joutua käymään kaikki polygonit läpi. Polygonin ja viivasegmentin leikkausten selvittämistä nopeuttaa huomattavasti kun tehdään esitarkistus, jossa selvitetään leikkaavatko viiva ja polygonin bounding box toisiaan tai sisältääkö bounding box molemmat viivan päätepisteet. Mikäli näin ei ole voidaan siirtyä seuraavaan polygoniin. Mikäli segmentti ei leikkaa mitään muita polygoneja, jatketaan leikkauksen etsimistä seuraavasta segmentistä. Mikäli leikkauskohtia on useampia, siirrytään polygoniin, jota viiva leikkaa lähimpänä alkupään solmua.

3.6.4 Pisteet

Lähtö- ja maalipisteitä luettaessa selvitetään minkä polygonin alueella piste sijaitsee ja merkitään se kuuluvaksi tähän polygoniin. Maalipisteiden polygonin määrittämisessä käytetään hyväksi ensin polygonin bounding boxia ja sen jälkeen raycasting-menetelmää (Shimrat 1962), kuten muidenkin pisteiden kohdalla tehtiin yllä olevissa luvuissa. Jos maalipisteitä on paljon, kuten silloin kun pyritään muodostamaan yhtenäinen kustannusetäisyysrasteri, olisi jokin tehokkaampi menetelmä oleellinen syötteen lukemisen nopeuttamiseksi.

3.7 Käsiteltävän solmun valinta

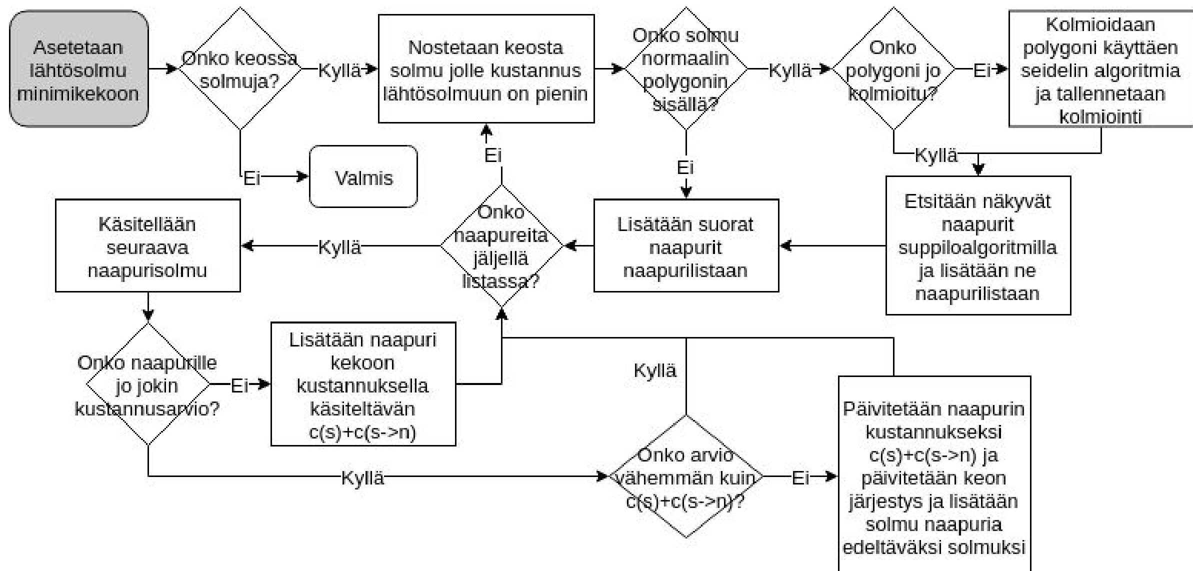
Kun kustannuspinta on luettu, voidaan varsinainen reitinetsintä käynnistää. Toteutetussa sovelluksessa voi valita käytettäväksi joko Dijkstran tai A* algoritmit (Dijkstra 1959, Hart, Nilsson & Raphael 1972) riippuen maalipisteiden määrästä ja sijoittelusta. Koko reitinetsintävaiheen toteutus on kuvattu vuokaaviona (kuva 20).

Dijkstran algoritmissa kustannusarvio solmulle on alhaisin tähän asti solmuun löydetty reitti. Kun solmu lopulta otetaan käsittelyyn, on edullisin solmuun vievä reitti jo löytynyt, eikä edullisempaa reittiä voi enää löytyä (Dijkstra 1959). A* algoritmissa edullisimpaan tähän asti löydettyyn kustannukseen lisätään heuristiikkafunktion tuottama arvio kustannuksesta solmusta lähimpään maalisolmuun (Hart, Nilsson & Raphael 1972).

Tuloksen oikeellisuuden kannalta ainoa vaatimus heuristiikkafunktiolle on, että se ei saa antaa pienempää arviota maalisolmuun, kuin mitä on todellisuudessa mahdollista saavuttaa (Hart, Nilsson & Raphael 1972). Tämän vuoksi sovelluksessa käytetään funktiota $h(s) = min(eucDist(s, m)) * cmin$, eli euklidinen etäisyys lähimpään maalisolmuun kerrottuna alhaisimmalla aineistossa esiintyvällä kustannuksella. Tämä takaa sen, että heuristiikkafunktio ei anna liian pieniä arvioita, mutta toisaalta funktio on varsin hidas laskea. Mikäli maalipisteitä on useampia, onkin parempi käyttää Dijkstran algoritmia.

Molemmat algoritmit vaativat tehokkaasti toimiakseen sen, että seuraavaksi käsittelyyn otettava solmu saadaan tietää nopeasti. Käsittelyyn otetaan aina solmu, jonka kustannusarvio on alhaisin. Koska vielä käsittelemättömien solmujen saavuttamisen kustannus muuttuu algoritmin etenemisen aikana, on järkevää tallentaa käsittelemättömät solmut minimikekoon.

Minimikeko on tietorakenne, josta voidaan tehokkaasti selvittää pienin kekoon tallennettu alkio, lisätä kekoon alkioita, poistaa pienin alkio ja päivittää kekoon tallennetun alkion arvoa pienemmäksi. Minimikeko voidaan toteuttaa usealla eri tavalla, ja eri operaatioiden aikavaativuudet riippuvat toteutustavasta. Laaditussa toteutuksessa käytetään pairing-kekoa (Fredman ym. 1986), koska se tarjoaa vakioaikaiset operaatiot pienimmän alkion tarkistamiselle ja uuden alkion lisäämiselle (Iacono 2000). Pienimmän alkion poistaminen ja alkion arvon pienentäminen ovat aikavaativuudeltaan logaritmisia suhteessa keon kokoon. Jotkin muut keon toteutukset kuten Fibonacci-keko (Fredman & Tarjan 1987) tarjoaisi myös vakioaikaisen operaation alkion arvon pienentämiselle, mutta johtuen pairing-keon yksinkertaisemmasta toteutuksesta se on havaittu käytännön toteutuksissa nopeammaksi (Fredman 1999).



Kuva 20. Solmun käsittely reitinetsinnän aikana. Funktio $c(s)$ kuvaa kustannusta solmuun s ja $c(s \rightarrow n)$ kustannusta solmun s ja n välillä.

3.8 Solmun naapurien löytäminen

Käsiteltävän solmun naapurit koostuvat kahdesta joukosta: suorista naapureista ja näkyvistä naapureista. Suorat naapurit on määritelty jo kustannuspinnan lukuvaiheessa naapureiksi, joihin käsiteltävästä solmusta voidaan siirtyä. Näkyviä naapureita ovat saman polygonin sisällä olevat pisteet, joihin voidaan siirtyä suoraa viivaa pitkin, ilman että leikataan polygonin reunaa. Suoria naapureita ei tarvitse enää etsiä, sillä ne on tallennettu listaan jokaiselle solmulle kustannuspinnan lukuvaiheessa, mutta näkyvyyteen perustuvien naapureiden etsinnässä on tehtävä lisää laskentaa. Käytetty menetelmä pohjautuu suppiloalgoritmiin (Lee & Preparata 1984), mutta sen toimintaa on yksinkertaistettu, koska ei ole tarpeen selvittää kaikkia lyhimpiä polkuja polygonin sisällä, vaan riittää selvittää mitkä solmut on nähtävissä käsiteltävästä solmusta.

Suppiloalgoritmi tarvitsee toimiakseen kolmioidun polygonin (Lee & Preparata 1984). Polygonin kolmiointi saadaan tehtyä tehokkaasti siten, että polygoni jaetaan ensin monotonisiin osiin, minkä jälkeen monotoniset osat pilkotaan kolmioihin (de Berg ym. 2008). Monotonisiin osiin pilkkominen saadaan tehtyä $O(n \log^* n)$ ajassa (Seidel 1991). Tämän jälkeen monotonisten osien kolmiointi saadaan tehtyä lineaarisessa ajassa (de Berg ym. 2008). Kolmioinitialgoritmista saadaan myös suoraan tieto siitä, mitkä kolmiot ovat toistensa

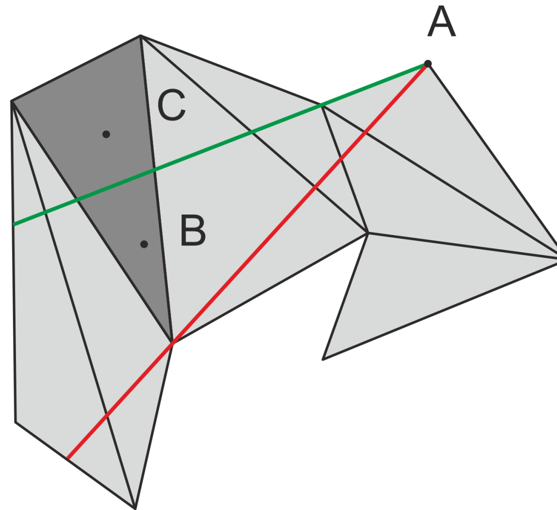
naapureita polygonin sisällä, mikä tallennetaan muistiin suppiloalgoritmin toimintaa varten.

Ei kuitenkaan ole täysin selvää, mitkä pisteet tulisi sisällyttää kolmioinnissa käytettyyn polygoniin. Vaihtoehtoja ovat seuraavat:

1. Vain polygonin kulmapisteet
2. Polygonin kulmapisteet ja suorille osuuksille laskettavat lisäpisteet
3. Edellä mainitut ja polygonin sisällä sijaitsevat pisteet (esim. maalipisteet).

Näistä vaihtoehdoista hieman epäintuitiivisesti yksinkertaisin on kolmas vaihtoehto. Tämä vaihtoehto johtaa kuitenkin tarpeettoman suureen kolmioiden määrään, mikä hidastaa näkyvien naapureiden määrittämistä koko polygonin alueella eikä ainoastaan reunojen tai sisällä olevien lisäpisteiden läheisyydessä, sillä suppiloalgoritmin aikavaativuus on lineaarinen suhteessa kolmioiden määrään (Lee & Preparata 1984). Tästä syystä päädyinkin tässä työssä ensimmäiseen vaihtoehtoon. Näkyvyys pitää kuitenkin määrittää myös lisä- ja sisäpisteisiin, ja tämä tehdään seuraavasti.

Polygonia kolmioidessa tarkistetaan jääkö muodostettavan kolmion sisään pisteitä ja tallennetaan nämä pisteet listaan kolmion sisäisistä pisteistä. Kun suppiloalgoritmi sitten saapuu kolmioon, jolla on sisäpisteitä, tarkastetaan yksitellen jäävätkö sisäpisteet näkyvän sektorin sisään (kuva 21). Tämä tarkistus tehdään laskemalla kummalle puolelle sektorin oikeaa ja vasenta laitaa tarkasteltava solmu jää.

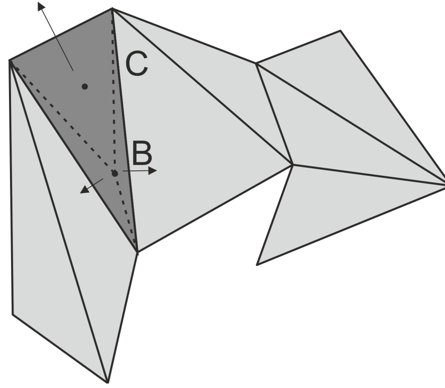


Kuva 21. Solmusta A nähtävissä olevia solmuja selvitetään suppiloalgoritmilla. Kun kolmio, jonka sisällä solmut C ja B sijaitsevat tarkistetaan sijaitsevatko ne A:sta näkyvän sektorin sisällä.

Solmun sijainti suhteessa sektorin laitaan saadaan selville ristitulon avulla: $d = (Py - Jx)(Ly - Jy) - (Py - Jy)(Lx - Jy)$, missä P on tarkasteltava solmu, J juurisolmu ja L sektorin laita. Jos $d > 0$, on tarkasteltava solmu sektorin laidan oikealla puolella, $d < 0$ vasemmalla ja jos $d = 0$ on solmu tasan sektorin rajalla.

Sisäpisteestä pitää kuitenkin myös päästä jatkamaan matkaa, ellei kyseessä ole maalipiste, minkä vuoksi täytyy kyetä myös tarkistamaan sisäpisteestä näkyvät pisteet. Tämä tehdään tuttuun tapaan suppiloalgoritmilla, minkä vuoksi sisäpisteenkin täytyykin kuulua johonkin kolmioon.

Kun sisäpiste otetaan käsittelyyn Dijkstran algoritmin toimesta ja aletaan selvittää siitä näkyviä naapureita, kolmio, jonka sisällä sisäpiste on, jaetaan kolmeen uuteen kolmioon (kuva 22). Nämä uudet kolmiot saavat jokainen yhden naapurin ne sisältävän kolmion naapurista. Tämä naapuruussuhde on kuitenkin yksisuuntainen, eli vanhan kolmion naapurit tai itse vanha kolmio ei saa uusia kolmiota naapurikseen. Näin näkyvyyden tarkastelu voi lähteä sisäpisteestä, mutta luodut sisäkolmiot eivät vaikuta näkyvyyden tarkastamiseen muille pisteille hidastaen suppiloalgoritmin toimintaa.



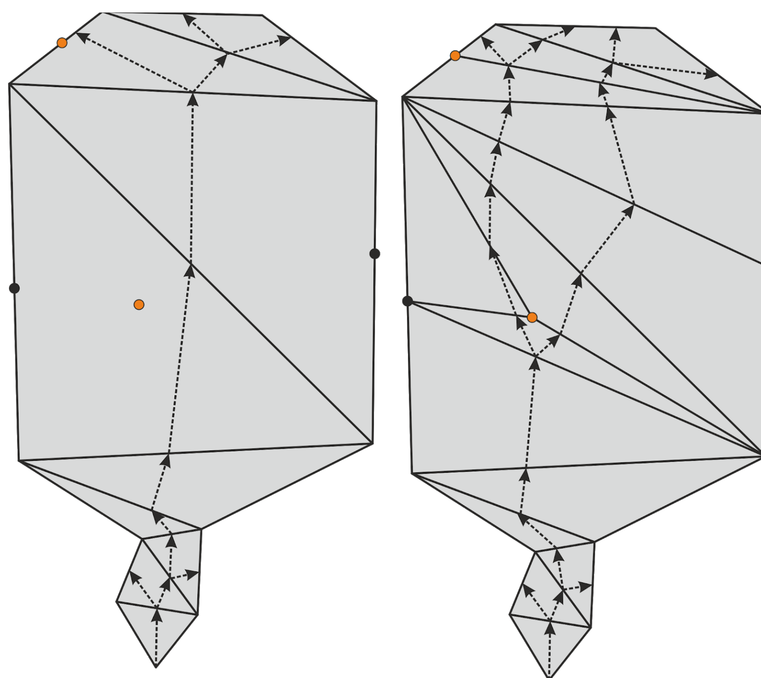
Kuva 22. Reitinetsinnän jatkuessa solmusta B, tulee suppiloalgoritmin kyetä muodostamaan ensimmäiset solmusta B lähtevät suppilot. Tätä varten kolmio, jonka sisällä B sijaitsee, jaetaan kolmeen uuteen kolmioon. Piste C merkitään lisäksi näkyväksi B:stä sen sijaan, että sitä lisättäisiin kuulumaan uuteen kolmioon.

Mikäli jaettavan kolmion sisällä on useita sisäpisteitä, ei ole tarpeen lisätä uusiin kolmioihin niiden sisään jääviä lisäpisteitä. Kaikki alkuperäisen kolmion sisällä olevat pisteet voidaan lisätä käsiteltävän sisäpisteen naapureiksi, sillä kaikki alkuperäisen kolmion sisällä olevat pisteet on nähtävissä toisistaan.

Koska luoduilla lisäkolmioilla on merkitystä vain kun lähdetään eteenpäin sisäsolmusta, on lisäkolmioilla merkitystä vasta kun sisäsolmu otetaan käsittelyyn Dijkstran algoritmin toimesta. Koska Dijkstran algoritmi myös takaa, että jokainen solmu käsitellään vain kerran (Dijkstra 1959), voidaan lisäkolmiot myös unohtaa saman tien, kun sisäsolmun naapurit on löydetty. Näin laskettavat lisäkolmiot eivät vaikuta tallennettavien kolmioiden määrään ja kolmioiden tallentamisesta johtuva muistin tarve ei kasva itse polygonin kolmioiden $n-2$ kolmiosta.

Reunapisteiden kanssa toimitaan samaan tapaan. Koska reunapiste sijaitseen aina kolmion reunalla täytyy sisempiä kolmioita muodostaa vain kaksi. Kumpikin näistä kahdesta kolmiosta saa yhden naapurin vanhan kolmion naapureista. Uusien kolmioiden loput sivut rajautuvat aina toiseen uuteen kolmioon tai polygonin reunaan. Reunapisteistä myös tiedetään niiden olevan käsiteltävän kolmion jollakin kyljellä, joten niiden näkyvyys on tarpeen tarkistaa vain jos kylki, jolle lisäpisteet on laskettu, on ylipäätänsä nähtävissä suppilon juuresta.

Jos yhden kolmion sisällä on suuri määrä sisäpisteitä, jotka eivät ole osa kolmiointia, hidastaa sisäpisteiden yksi kerrallaan läpi käyminen suppiloalgoritmia siinä vaiheessa kun tarkistetaan, mitkä sisäsolmut näkyvät suppilon juurisolmusta. Toisaalta näkyvät sisäsolmut aiheuttavat niiden ollessa osa kolmiointia suppiloalgoritmin haarautumisen useaan suppiloon, mikä tietysti myös hidastaa laskentaa (kuva 23). Maalipisteitä tulee helposti suuri määrä erityisesti jos halutaan luoda yhtenäinen koko tutkimusalueen kattava kustannusetäisyysrasteri, kuten perinteisesti paikkatietosovelluksista löytyvillä rasteripohjaisilla työkaluilla.



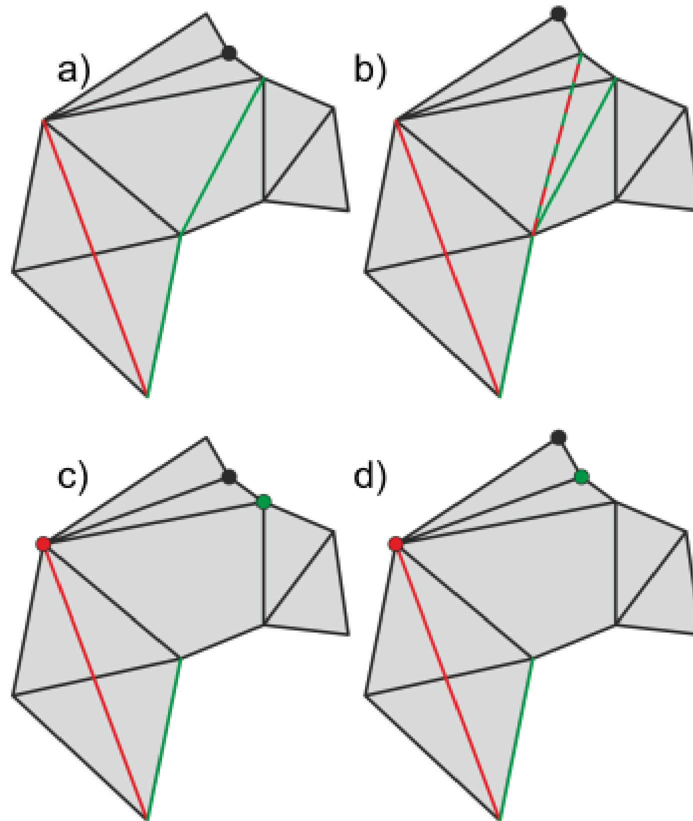
Kuva 23. Vasemmalla suppiloalgoritmin kulku, kun reunoille laskettavat lisäpisteet tai sisäpisteet eivät vaikuta kolmiointiin. Oikealla algoritmin kulku, kun kaikki pisteet otetaan mukaan kolmiointiin. Nuolet kuvaavat suppiloalgoritmin yhtä askelta, eivät löydettyjä polkuja. Nuolen kärki osoittaa kulloinkin käsiteltävänä olevan suppilon suuta. Huomionarvoista on, että vasemmassa kuvassa vain oranssiksi värjätyissä sisä- ja reunapisteissä on edes tarpeen tarkistaa näkyvätkö pisteet suppilon juuresta.

Reunasolmujen osalta ongelma on huomattavasti pienempi, sillä niiden näkyvyyttä ei tarvitse muutoinkaan tarkistaa yhtä usein kuin varsinaisten sisäsolmujen näkyvyyttä yllä kuvatuista syistä. Kun jokin osa polygonin reunasta on nähtävissä, saadaan ensimmäinen ja viimeinen näkyvä reunasolmu määritettyä tehokkaasti binäärihaulla (Knuth 1998), jossa selvitetään oikealta ja vasemmalta suppilon reunalta reunimmainen juurisolmusta näkyvä solmu. Tämän jälkeen voidaan kaikki näiden väliin jäävät reunasolmut merkitä näkyviksi.

Myös itse suppiloalgoritmin toimintaan on mahdollista tehdä yksinkertaistus johtuen siitä, että halutaan löytää näkyvät solmut. Algoritmin alkuperäinen käyttötarkoitus on etsiä lyhin reitti kaikkiin solmuihin polygonin sisällä (Lee & Preparata 1984). Alkuperäisessä suppiloalgoritmissa muodostetaan solmuketjuja ja ketjuun voidaan lisätä uusia solmuja, jolloin suppilo "levenee".

Koko solmuketjun pitäminen tallessa ei kuitenkaan olen näkyvyyden laskemisen tapauksessa tarpeellista vaan riittää pitää kirjaa viidestä solmusta. Nämä viisi solmua ovat suppilon juuri, näkyvän sektorin oikean ja vasemman laidan määrittävät solmut sekä suppilon suun oikea ja vasen laita (kuva 24). Käytännössä suppilon suun molemmista solmuista kirjan pitäminen ei ole tarpeen vaan riittää tietää mikä kolmio muodostaa kulloinkin suppilon suun.

Sektorin laidat muodostuvat ensin sen kolmion, johon käsiteltävä solmu kuuluu, kyljistä. Tämän jälkeen sektoria ainoastaan kavennetaan jakamalla se kahteen osaan, kuten suppiloalgoritmissakin tehdään silloin kun vastakkainen solmu sijoittuu sektorin sisään (Lee & Preparata 1984).



Kuva 24. Alkuperäisessä suppiloalgoritmissa (a & b) vastakkaisen solmun (musta solmu) ollessa suppilon ensimmäisten segmenttien rajaaman sektorin ulkopuolella (a) selvitetään solmuketjussa ensimmäinen solmu, josta vastakkainen solmu näkyy. Tämän jälkeen käsiteltävää suppiloa kavennetaan ja luodaan suppilon jäljelle jäävästä osasta uusi suppilo (b). Kuitenkaan näkyvyyttä selvitettäessä (c & b) ei ole tarpeen luoda uutta suppiloa, eikä selvittää mistä solmusta vastakkainen solmu on nähtävissä. Tämän vuoksi ei myöskään ole tarpeen pitää kirjaa suppilosta sen ensimmäisten ensimmäisten segmenttien ja suun välillä. Riittää tietää, että suuta vastakkainen solmu ei ole nähtävissä suppilon juuresta (c), jolloin suppilon suun toinen reuna (vihreä solmu) siirretään vastakkaiseen solmuun ja otetaan seuraava vastakkainen solmu käsittelyyn (d).

Mikäli tulemme tilanteeseen, jossa suppilon suuta vastakkainen solmu on näkyvän sektorin ulkopuolella, etsitään alkuperäisessä suppiloalgoritmissa solmuketjusta viimeinen segmentti, jonka rajaaman suppilon sisään solmu jää (kuva 24). Tämän jälkeen käsiteltävän suppilon ketju korjataan johtamaan löydettyyn solmuun, ja ketjun jäljelle jäävästä osasta luodaan uusi suppilo (Lee & Preparata 1984).

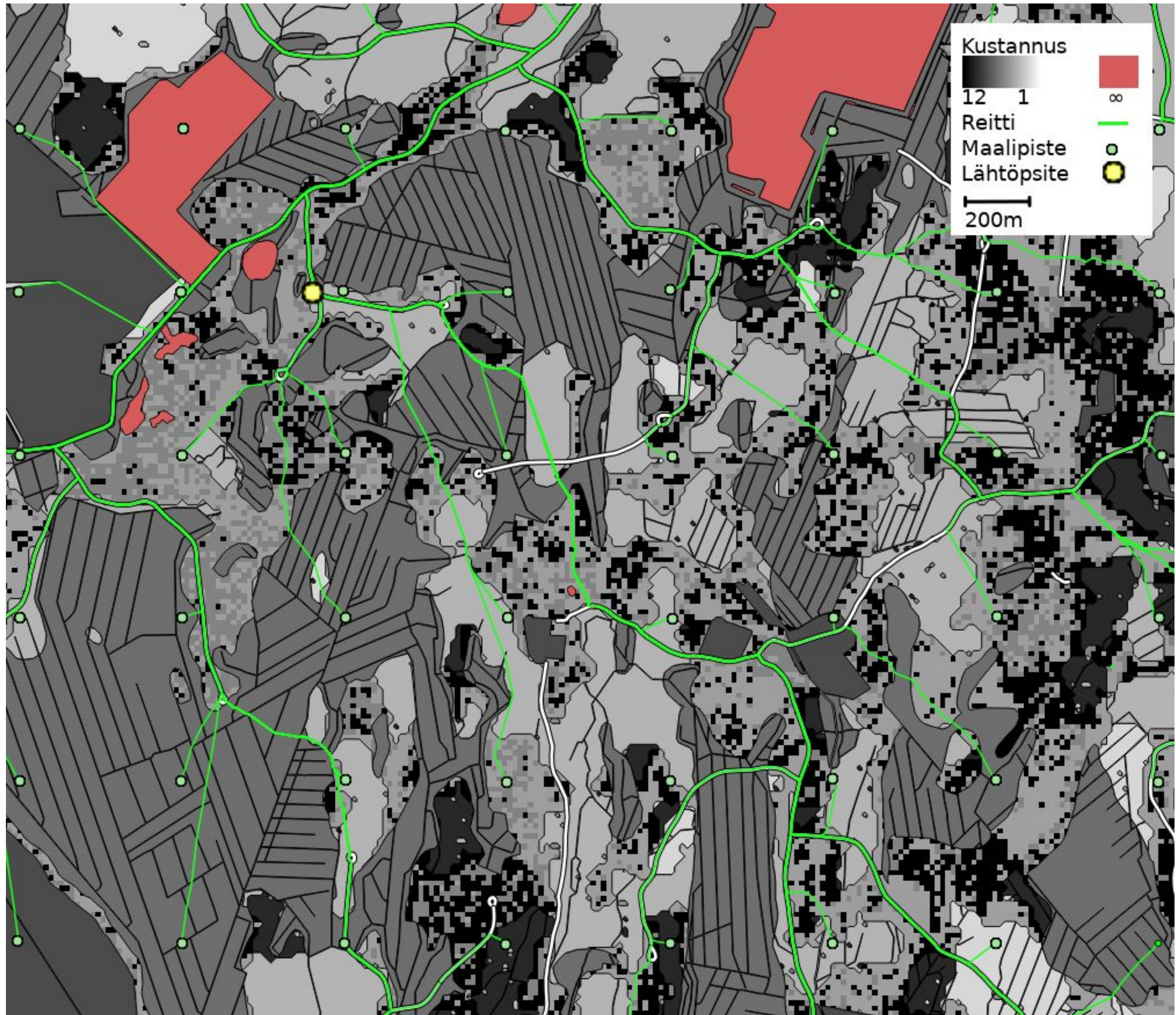
Näkyvyyden tapauksessa emme kuitenkaan ole kiinnostuneita tästä uudesta suppilosta, sillä sen juuri ei ole solmu, josta näkyviä solmuja haluamme selvittää. Toisaalta käsiteltävän suppilon solmuketjun korjaaminen ei myöskään ole oleellista, sillä näkyvyyttä selvitettäessä

ainoastaan ketjun ensimmäisellä ja viimeisellä solmulla on merkitystä. Ensimmäinen solmu rajaa näkyvyyden määrittelevän sektorin laidan, ja viimeinen solmu määrittelee suppilon suun, josta algoritmi jatkaa sitten seuraavaan suuta vastakkaiseen solmuun.

Ensimmäinen solmu, joka määrittää näkyvän sektorin ei voi kuitenkaan vaihtua, sillä jotta näin tapahtuisi, täytyy vastakkaisen solmun olla nähtävissä juurisolmusta eli näkyvän sektorin sisällä, jolloin suppilo jaettiin kahtia. Toisaalta myös suppilon suu on jo tiedossa, sillä sen toinen pää ei muutu ja toinen pää liikkuu aina edellisenä käsiteltyyn suppilon suuta vastakkaiseen solmuun. Näin ollen solmuketjun korjaaminen, uusien suppiloiden, joiden juuri ei ole alkuperäinen juurisolmu, muodostaminen, tai ylipäätänsä solmuketjusta kirjan pitäminen ensimmäisen ja viimeisen solmun välillä, on turhaa. Riittää pitää kirjaa käsiteltävästä kolmiosta, joka määrittää suppilon suun ja sitä vastakkaisen solmun, sekä solmuketjun ensimmäisistä solmuista, jotka määrävät näkyvän sektorin. Uusia suppiloita riittää luoda ainoastaan silloin, kun käsiteltävänä oleva solmu näkyy juurisolmusta.

3.9 Tapaustutkimus maastonavigoinnista

Tässä osassa esitellään tapaustutkimus liittyen maastonavigointiin metsäympäristössä. Tapaustutkimus tehtiin käyttäen nyt kehitettyä reitinetsintäsovellusta ja tutkimuksen tarkoituksena oli arvioida käytännön käyttötapauksen kautta sovelluksen tuottamien reittien (kuva 25) laatua. Lisäksi tapaustutkimuksella pyrittiin esittelemään komposiittikustannuspinnan muodostamiseen käytettyjä työvaiheita sekä arvioimaan työkalun käyttöön liittyviä haasteita.



Kuva 25. Esimerkki kustannuspinnasta ja algoritmin tuottamista poluista. Kuvattu alue on pieni osa käytetystä kustannuspinnasta.

Tapaustutkimuksessa muodostettiin kustannuspinta käyttäen viiva-, rasteri- ja polygoniaineistoja, minkä jälkeen kustannuspinnasta laskettiin reitit tutkimusalueelle tasaisesti sijoitettuihin maalipisteisiin käyttäen Dijkstran algoritmia (Dijkstra 1959). Reittien laskennan jälkeen saatujen reittien laatua arvioitiin silmämääräisesti. Lisäksi ilmoitettujen kustannusten oikeellisuutta tarkasteltiin laskemalla reittien kustannukset uudestaan suoraan kustannuspinnasta ja vertaamalla saatua kustannusta algoritmin ilmoittamiin kustannuksiin.

3.10 Aineiston esittely

Kustannuspinta muodostettiin metsävaratiedon ja maastotietokannan kohteiden perusteella (taulukko 1). Eri maastotyyppien kustannukset asetettiin Antikaista mukaillen (Antikainen

2009). Antikainen on puolestaan määrittänyt eri kustannusarvot perustuen asiantuntijahaastatteluihin.

Kustannukseen vaikutti maaston peitteen lisäksi joidenkin kohteiden osalta miten niitä pitkin tai niiden yli kuljetaan. Esimerkiksi vilkkaiden teiden ja ojien osalta näiden kohteiden ylittämiselle oli määritelty eri kustannus kuin niitä pitkin kulkemiselle. Kuitenkin on hyvä muistaa, ettei käytettyjen kustannusten tarkalla oikeellisuudella ole juurikaan merkitystä tässä tutkimuksessa, jonka tarkoituksena oli vain testata algoritmin toimintaa ja arvioida sen tuottamia polkuja

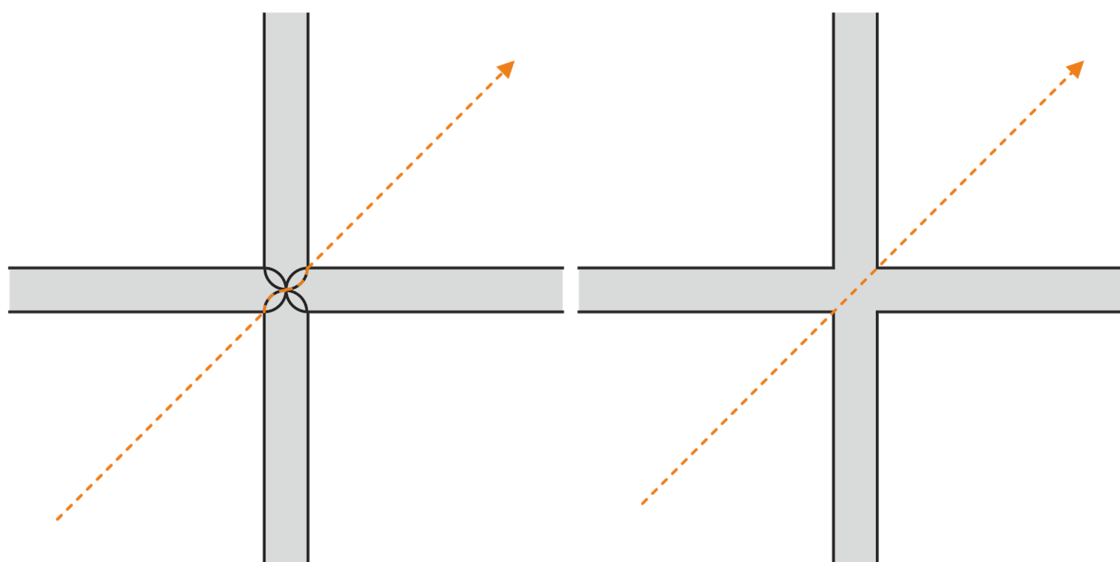
Taulukko 1. Kustannuspinnan muodostamisessa käytetyt aineistot

Kustannuspinnan kohde	Aineisto	Kustannus	Aineiston tyyppi	Tuottaja
Tietä pitkin kulkeminen	Maastotietokanta	1	Viiva	MML
Polkua pitkin kulkeminen	Maastotietokanta	2.25	Viiva	MML
Vilkkaan tien ylittäminen	Maastotietokanta	4.98	Viiva	MML
Avoin alue tai nuori taimikko	Hilamuotoinen metsävaratieto	3.04	Rasteri, 16m	SMK
Varttunut taimikko tai nuori metsä (matala runkoluku)	Hilamuotoinen metsävaratieto	3.95	Rasteri, 16m	SMK
Varttunut taimikko tai nuori metsä (korkea runkoluku)	Hilamuotoinen metsävaratieto	8.12	Rasteri, 16m	SMK
Varttunut metsä	Hilamuotoinen metsävaratieto	3.10	Rasteri, 16m	SMK
Suo	Maastotietokanta	4.98	Polygoni	MML
Maatalousmaa	Maastotietokanta	4.95	Polygoni	MML
Ojan ylitys (alle 2m)	Maastotietokanta	7.64	Viiva	MML
Jyrkänne	Maastotietokanta	11.90	Viiva	MML
Vesistö	Maastotietokanta	∞	Polygoni	MML

3.11 Kustannuspinnan muodostaminen

Viivakohteista polut ja pienemmät tiet säilytettiin kustannuspintaa muodostettaessa viivamaisessa muodossa, sillä näiden kohteiden ylittämiseen ei ajateltu liittyvän mitään niiden ympäristön kustannuksesta poikkeavaa kustannusta (Antikainen 2009). Ojat olivat selkeästi viivamaisia kohteita, joiden ylittämiseen liittyi kustannus. Tämä mallinnettiin jakamalla ojat kahteen luokkaan: alle kaksi metriä leveisiin ja kahdesta viiteen metriä leveisiin. Ojat bufferoitiin käyttäen kahden ja viiden metrin buffereita, minkä jälkeen alle kaksimetrisille ojille asetettiin niiden alueella kulkemiselle korkea kustannus, ja kahdesta viiteen metriä leveät ojat mallinnettiin vesistöinä, joiden alueelle reitinetsintä ei etene lainkaan. Myös jyrkänteet bufferoitiin käyttäen viiden metrin bufferia, ja jyrkänteen alueella kulkemiselle asetettiin korkea kustannus.

Vilkkaasti liikennöityihin teihin liittyi edullinen kustannus kulkea niiden piennarta pitkin, mutta korkea kustannus niiden ylittämiselle. Tämä mallinnettiin kustannuspinnassa bufferoimalla tieviiva alueeksi, minkä jälkeen alueen reunaa pitkin liikkumiselle asetettiin edullinen kustannus ja alueen läpi kulkemiselle korkea kustannus (Antikainen 2009). Algoritmin loogisen toiminnan kannalta oli tärkeää yhdistää bufferoidut alueet, jotta teiden ylittäminen risteysten kohdalta buffereiden päätyjä pitkin ei mallintunut edullisena tien reunaa pitkin kulkemisena (kuva 26).



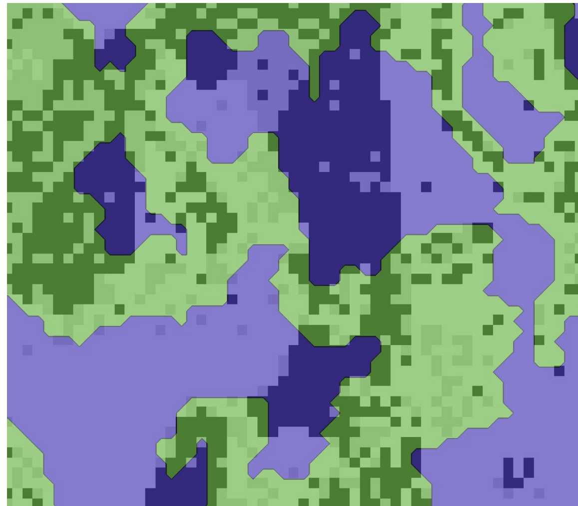
Kuva 26. Vasemalla yhdistämättömät bufferit johtavat edulliseen ylityskustannukseen risteävien korkean kustannuksen alueiden edullisia reunoja pitkin. Oikealla yhdistetyt bufferit tuottavat halutun kaltaisen suoran polun (oranssi katkoviiva)

Kustannuspinnassa käytettävä metsävaratieto oli koko tutkimusalueen kattavassa 16x16 m hilamuodossa. Tämä hila sisälsi sekä homogeenisiä alueita että alueita, joilla kulkukustannus vaihteli naapurisolujen välillä paljonkin. Jotta erityyppiset alueet saatiin analysoitua niille sopivalla menetelmällä, täytyi hilasta eristää homogeeniset kustannusalueet polygoneiksi ja toisaalta määritellä, mitkä alueet olivat niin heterogeenisiä, että oli järkevää käsitellä alue rasterimenetelmin.

Tämän tavoitteen saavuttamiseksi hila muutettiin ensin rasterimuotoon, minkä jälkeen se luokiteltiin heterogeenisiin ja homogeenisiin alueisiin. Luokittelu tehtiin käyttäen edge density -funktiota 5x5 solun liukuvalla ikkunalla. Solujen arvoiksi tuli siis yhteenlasketun kustannusalueiden reunan pituus solua ympäröivällä 5x5 solun alueella (Pallecchi & Metz 2015). Saatu edge density-kartta kynnystettiin binäärimuotoon homogeenisten ja heterogeenisten alueiden erottamiseksi.

Heterogeenisten alueiden osalta riitti tunnistaa itse alueet, minkä jälkeen niiden varsinainen kustannus määräytyi metsävaratietorasterin perusteella. Homogeenisten alueiden osalta oli

tarpeen määritellä myös itse kustannusalueet sillä yksi homogeenisuuden alue saattoi sisältää useita vierekkäisiä homogeenisiä eri kustannuksen alueita (kuva 27).



Kuva 27. Kustannuspinta jakautuu homogeenisiin (violetti) ja heterogeenisiin (vihreä) alueisiin. Varsinainen kustannus vaihtelee kummankin ryhmän sisällä (tummempi sävy kuvaa korkeampaa kustannusta). Heterogeenisten alueiden osalta käytetään rasterialgoritmeja, joten riittää käyttää heterogeenisyyden rajoja rastereiden määrittelypolygoneina. Homogeenisten alueiden osalta pitää vielä muuttaa alueet varsinaisiksi kustannuspolygoneiksi siten, että yhden polygonin alueelle jää vain yhtä kustannusta.

Tätä varten rasterimuotoisesta hilasta laadittiin kustannuspolygonit, minkä jälkeen syntyneiden kustannuspolygonien ja homogeenisuuden rajoja kuvaavien polygonien muodostamat alueet yhdistettiin siten, että jos alue oli heterogeeninen, käytettiin suoraan heterogeenisyyden rajoja kuvaavaa polygonia ja homogeeniset alueet pilkottiin varsinaisiin kustannuspolygoneihin intersect -operaatiolla. Tuloksena prosessista syntyi kustannuspinta, jossa heterogeenisille alueille oli määritelty määrittelypolygonit sekä rasteri, josta varsinaiset kustannusarvot löytyivät. Homogeenisemmille alueille taas oli määritelty yksi kustannus vektoripohjaista reitinetsintää varten.

Ongelmana tässä tavassa muodostaa kustannuspinta oli, että se ei ottanut kovin hyvin huomioon kustannusalueiden rajojen merkityksellisyyttä. Jos kustannus jonkin heterogeenisen alueen välillä vaihteli tiheästi, mutta vain hyvin vähän, sillä ei olisi ollut reitinetsinnän kannalta juurikaan merkitystä, ja olisi ollut järkevää luoda alueesta yksittäinen kustannusalue.

Kun eri metsätyypeistä oli saatu laadittua polygonitaso yhdistettiin siihen vielä maastotietokannan aluemaiset kohteet ja viivamaisista kohteista bufferoidut alueet. Tämä tehtiin käyttäen hierarkkista overlay-operaatiota, jossa alueen, jolla oli useita päällekkäisiä kustannusluokkia, kustannus määräytyi sen mukaan minkä alueella esiintyvän kustannusluokan prioriteetti oli korkein (Antikainen 2009).

Kustannusten prioriteetit (taulukko 2) määräytyivät sen oletuksen pohjalta, että päällekkäisten kohteiden tapauksessa korkeamman prioriteetin kohde oli todennäköisemmin todellisuudessa kulkukelpoisuuden kannalta oleellisin (Antikainen 2009). Jos vesistö ja tie esimerkiksi olivat päällekkäin, oli luultavasti kyseessä silta, ja näin vesistön kulkukelpoisuudella ei ollut merkitystä.

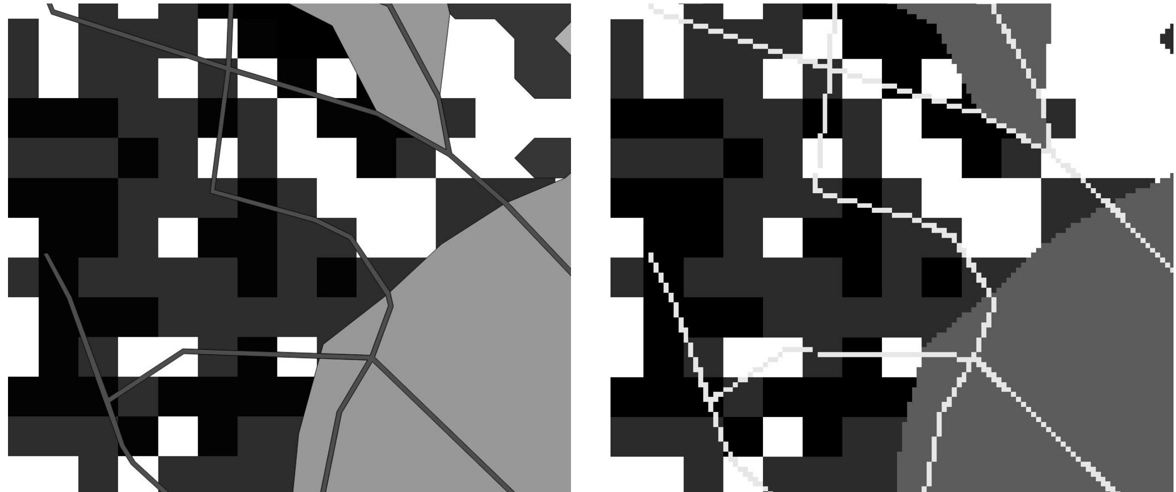
Taulukko 2. Eri kustannusluokkien prioriteetit.

Kustannusluokka	prioriteetti
Tie	7
Vesistö	6
Oja (alle 2m)	5
Jyrkänne	4
Maatalousmaa	3
Suo	2
Metsätyyppi (tai heterogeeninen metsä)	1

Toteutetussa tapaustutkimuksessa käytettiin 625x750 kokoista rasteria, 12 300 polygonista ja 245 656 solmusta koostunutta vektorimuotoista kustannuspintaa sekä 903 viivasta ja 16 058 solmusta koostunutta viivaverkostoa. Maalipisteitä oli 526 kpl. Pienimmät kustannuspinnan kohteet olivat kooltaan kahden metrin luokkaa ja tutkimusalue oli kooltaan 10 km x 12 km.

Tapaustutkimuksessa ei systemaattisesti tarkasteltu eri tekijöiden vaikutusta algoritmin suoritusaikaan, mutta työkalun käyttökelpoisuuden kannalta on kuitenkin tärkeää, että laskentaan käytetty aika ei kasva kohtuuttoman suureksi muihin käytettävissä oleviin

työkaluihin nähden. Tämän vertailun toteuttamiseksi muodostettiin kustannuspinnasta myös yhdestä rasteritasosta koostuva kustannuspinta jolle reitinetsintä toistettiin. Vertailun tarkoituksena ei ollut selvittää kumpi menetelmä olisi yksiselitteisesti nopeampi tai hitaampi, vaan ainoastaan muodostaa jonkinlainen käsitys nyt kehitetyn työkalun käytännön laskenta-ajan tarpeista.



Kuva 28. Vasemmalla komposiittikustannuspinnan osa ja oikealla siitä 1 m solukoolla rasteroitu kustannuspinta.

Jotta laskenta-ajoille saatiin jokin vertailukohta, kustannuspinta muutettiin 10 000 x 12 000 solun rasteriksi. Valittu rasterin koko tarkoitti yhden metrin solukokoa, mikä antoi vastaavaa tarkkuusluokkaa olevan kustannuspinnan. Kustannuspinta sisälsi paljon kahden metrin levyisiä korkean kustannuksen kohteita (kuva 28), joten suurempi solukoko olisi johtanut näiden kohteiden katkeamiseen.

3.12 Tuotettujen reittien arviointi

Saatujen polkujen laatua arvioitaessa oli tarkasteltiin asiaa: Kuinka lähellä saadut polut olivat optimaalista kustannuspinnasta laskettua polkua ja toisaalta kuinka lähellä algoritmin polulle ilmoittama kustannus oli polun todellista kustannusta kustannuspinnassa. Näistä ensimmäisen arviointi oli vaikeaa, sillä optimaalisen polun määrittäminen ei ollut tämän tutkimuksen puitteissa mahdollista. Kuitenkin silmämääräisen tarkastelun perusteella havaittiin poluissa kohtia, jotka selvästikään eivät olleet optimaalisia.

Reittien ilmoitetun kustannuksen ja todellisen kustannuksen eron arvioiminen voitiin tehdä pelkkää visuaalista tarkastelua systemaattisemmin. Jokaiselle algoritmin tuottamalle reitille laskettiin kustannus uudestaan siten, että jokaiselle polun segmentille annettiin sitä vastaavan kustannuspinnan osan kustannus. Tällöin todellinen kustannus saatiin kertomalla polun segmenttien pituudet niiden kustannuksilla ja summaamalla saadut kustannukset yhteen. Koska kustannuspinta muodostui useasta eri tasosta, oli tämä polkujen kustannusten tarkistamiseen käytetyn tason muodostaminen useampivaiheinen prosessi.

Ensin rasterimuotoinen kustannuspinta polygonoitiin, minkä jälkeen polygonoidusta rasterista leikattiin irti clip työkalulla polygoinmuotoisen kustannuspinnan kattamat alueet. Jäljelle jäävät alueet ja polygonimuotoinen kustannuspinta yhdistettiin union operaatiolla. Koska lasketut reitit etenivät myös viivamaisia kohteita pitkin, piti myös viivat ottaa laskennassa huomioon.

Ensin polygonikohteiden, joiden reunoja pitkin liikkumiselle oli annettu erillinen kustannus, reunat muutettiin viivamuotoon, minkä jälkeen niistä ja varsinaista viivakohteista muodostettiin yhden millimetrin bufferilla aluemaiset kohteet. Nämä alueet yhdistettiin reitin arvioinnissa käytettyyn kustannuspintaan union operaatiolla.

Kun kaikki kustannuspinnan kohteet oli yhdistetty yhdeksi aluemaiseksi, kustannuspinnaksi laskettiin jokaiselle alueelle kustannus käyttäen samaa prioriteettijärjestystä kuin aiemmin. Tarkistuksessa ei siis pyritty korjaamaan esimerkiksi ison ja pienen tien risteämisestä aiheutuvaa epäloogisuutta, vaan ainoastaan tarkistamaan onko algoritmin ilmoittama kustannus oikein sillä kustannuspinnalla, jonka algoritmi sai syötteeksi.

Tämän jälkeen jokaisen reitin viivat jaettiin sellaisiin paloihin, että jokainen pala on vain yhden kustannusalueen sisällä, minkä jälkeen reitin segmenteille laskettiin niiden kustannukset. Lopuksi viivasegmenttien kustannukset laskettiin ryhmittäin yhteen maalisolmun mukaan ja saatua tulosta verrattiin reitinetsintätyökalun ilmoittamaan kustannukseen kyseisessä maalisolmussa.

4 Tulokset

4.1 Reitinetsintään käytetty aika

Reitinetsintään kului 2 min 4 s, josta 1 min 22 s meni kustannuspinnan lukemiseen ja suorien naapurien määrittämiseen ja 42 s varsinaiseen reitinetsintään. Syötteen lukemiseen ja reitinetsintään käytetyn ajan osuudet riippuvat voimakkaasti käytetyn syötteen tyypistä. Rasterit, viivat ja maalipisteet lisäävät selvästi syötteen lukuun käytettävää aikaa erityisesti, jos rasteri- ja vektoritasot ovat päällekkäin. Jos aineisto koostuu pääasiassa polygoneista, menee suurin osa ajasta reitinetsintävaiheeseen.

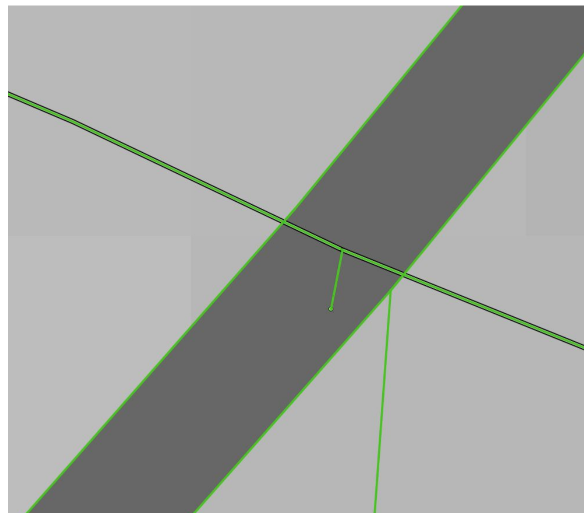
Alhaisimman kustannuksen reitin etsintä vertailukohdaksi muodostetusta rasterimuotoisesta kustannuspinnasta kesti GRASS GIS 7 ohjelmalla 3 min 32 s. Laskenta-aikojen vertailu voisi antaa hyvin erilaisen tuloksen, jos kustannuspinta olisi koostunut erilaisista kohteista. Nyt kustannuspinnassa oli paljon pieniä ja kapeita kohteita, paljon laajoja homogeenisiä alueita ja toisaalta myös heterogeenisiä alueita. Komposiittimenetelmä tietysti soveltui hyvin tällaisen kustannuspinnan käsittelyyn, kun taas rasteripohjaiset menetelmät eivät niinkään.

4.2 Tuotettujen reittien laadun arviointi

Viivojen ja polygonien välillä siirryttäessä ei havaittu mitään lisävirhettä liittyen aineistojen välillä siirtymiseen. Toki sama polygonin reunoille ja viivasegmenteille laskettujen pisteiden harvuus vaikutti myös polygonista viivoihin siirryttäessä samaan tapaan kuin eri polygonien välillä liikuttaessa.

Kustannuspinnassa oli mallinnettu vilkkaat tiet bufferoituina alueina, joille asetettiin korkea kustannus alueen sisällä liikkumiselle ja matala kustannus reunoja pitkin liikkumiselle. Tämän tarkoituksena oli saada tien reunaa pitkin kulkeminen halvaksi, mutta luoda korkeampi kustannus tien ylittämiselle. Koska pienemmät tiet oli mallinnettu yksinkertaisesti viivamaisina kohteina, aiheutui tästä hieman ei-toivottuja tuloksia.

Syntyneistä alhaisimman kustannuksen reiteistä voitiin havaita tilanteita, joissa maalipiste oli keskellä bufferoitua vilkasta tietä (kuva 29). Laskettu polku teki kuitenkin mutkan keskellä tietä ennen maalipisteeseen saapumista, mikä ei selvästikään ollut toivottavaa. Mutka johtui siitä, että tien kanssa ristesi pienempi tie, joka oli mallinnettu viivakohteena. Polku etenikin pienempää tietä pitkin edullisella kustannuksella vilkkaan tien puoliväliin, mistä se suuntasi suoraan maalipisteelle. Vastaavasti ison tien ylittäminen pientä tietä pitkin mallintui edullisena. Ongelman ydin olikin siinä, että pienemmän tien kustannuksen ei olisi pitänyt olla korkeammalla prioriteetilla kuin vilkkaan tien.



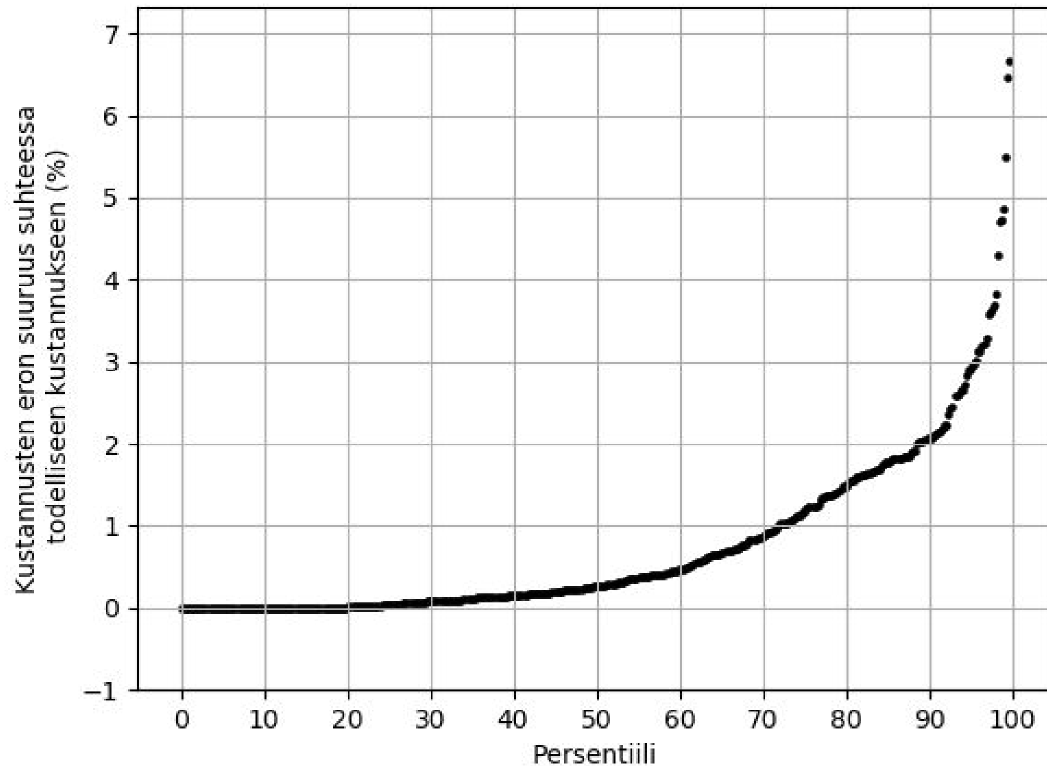
Kuva 29. Pienempi tie ylittää suuremman tien, jonka ylittämiseen liittyy kustannus. Koska pienempi viivamainen tie on lisätty kustannuspintaan korkeammalla prioriteetilla, voidaan suurempi tie virheellisesti ylittää pienempää pitkin ilman tien ylittämisestä aiheutuvaa kustannusta.

Kuitenkin tyypillisesti viivamaisten kohteiden kustannus on haluttu todellinen kustannus, alueiden, joita viivat halkovat, kustannuksen sijaan. Jotta tällainen ongelma saataisiin korjattua, tulisi pieniä teitä kuvaavat viivat katkaista vilkkaan tien reunoille tai vaihtoehtoisesti bufferoida myös pienemmät tiet. Havaitun kaltainen ongelma on siis vältettävissä huolellisella kustannuspinnan suunnittelulla, mutta se kuvastaa hyvin komposiittikustannuspinnan muodostamiseen liittyviä haasteita. Toisaalta syntyneen kaltainen tilanne voisi olla täysin toivottu esimerkiksi alikulkutunnelia mallinnettaessa. Esimerkki osoittaakin hyvin komposiittikustannuspinnan muodostamiseen liittyvien haasteiden lisäksi sen kykyä mallintaa erilaisia kustannuspinnan kohteita ja niiden välillä liikkumisen muotoja.

Koska polkujen arviointiin käytetty kustannuspinta ei ollut täysin sama kuin alkuperäinen kustannuspinta, josta polkujen kustannukset oli laskettu, oli saaduissa reittien pituuksissa odotettavissa pieniä eroja johtuen viivakohteiden pienistä buffereista sekä liukulukulaskennassa väistämättä syntyvistä pienistä virheistä.

Tarkasteltaessa polkujen tarkistettuja kustannuksia, voitiin havaita kymmeneen maalipisteeseen menevän polun kulkevan todellisuudessa alueen lävitse, jonka läpi reitti ei olisi saanut lainkaan kulkea. Koska tällaisen alueen kustannus on käytännössä ääretön, jätettiin nämä polut pois tuloksista. Myös osa maalipisteistä sattui sijaitsemaan alueilla, joille ei ollut lainkaan pääsyä lähtöpisteestä, joten lopulta tarkastettavia polkuja jäi jäljelle 432 kappaletta.

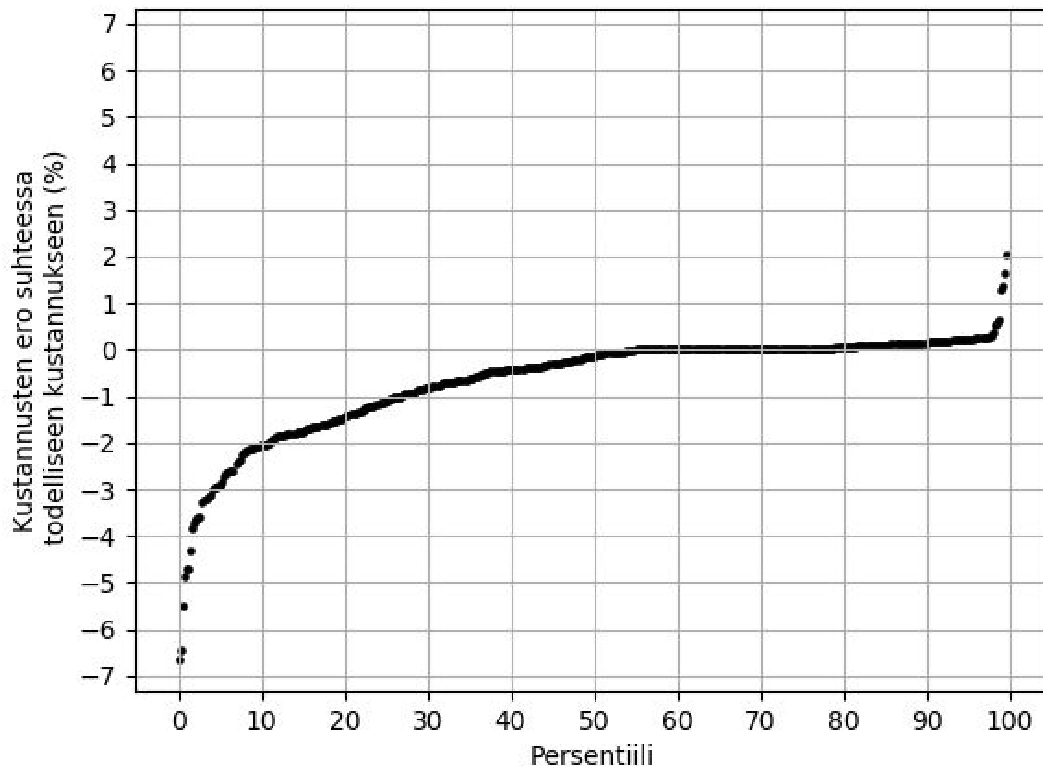
Arvioitaessa polkujen oikeellisuutta käytettiin mittaria $\frac{k-i}{k} \cdot 100$, missä k kuvaa todellista kustannusta ja i algoritmin ilmoittamaa kustannusta. Tämä mittari ilmoitti siis polkujen kustannuksessa olevan virheen suuruuden suhteessa polun pituuteen. Jos olisi käytetty vain absoluuttista polun virhettä oikeellisuuden mittarina, olisi ollut vaikea vertailla keskenään eri kustannuksisten polkujen virheitä. Toisaalta voidaan ajatella, että suuri virhe korkeamman kustannuksen polussa olisi ollut pahempi asia kuin pieni virhe matalamman kustannuksen polussa, vaikka ne olisivatkin olleet prosentuaalisesti saman suuruiset.



Kuva 30. Laskettujen polkujen kustannuksen ja algoritmin ilmoittaman kustannuksen ero prosentteina. Kuvasta voidaan havaita polkujen pituuksissa esiintyvän varsin suuriakin virheitä (6,7 %), mutta toisaalta suuremmat virheet ovat huomattavan harvinaisia.

Tuloksista voitiin havaita noin 20 % poluista olleen sellaisia, joille laskettu tulos oli lähes täysin oikein (kuva 30). Nämä polut olivat sellaisia, joissa polku kulki vain vektorialueiden ja viivakohteiden kautta, jolloin ilmoitettu kustannus olisi pitänyt olla täysin oikein, mutta johtuen tulosten varmennuksen numeerisesta epätarkkuudesta, saaduissa ja varmennetuissa tuloksissa esiintyi alle 0.001 % eroja. Muiden polkujen osalta voitiin havaita suurempien virheiden olleen exponentiaalisesti harvinaisempia kuin pienempien virheiden. Kuitenkin menetelmä tuotti hieman myös varsin suuria virheitä, enimmillään 6,7 %.

Yksi selitys virheiden jakaumalle voisi olla, että virheen suunta (+/-) on sattumanvarainen ja näin suuren virheen saamiseksi pitää tapahtua monta pientä samansuuntaista virhettä peräkkäin. Tämä selittäisi miksi suuret virheet olivat niin harvinaisia, sillä vaikka virheitä oli polun varrella paljon, saattoivat ne kumota toisensa vaikutuksen polun kustannukseen.



Kuva 31. Kun virheen suuruuden lisäksi havainnoidaan sen suuntaa, voidaan havaita tilanteiden, joissa algoritmi on antanut liian matalan kustannuksen, olevan yleisempiä, ja myös virheiden olevan suurempia, kuin tilanteissa, joissa algoritmi olisi antanut liian suuren kustannuksen.

Kun tarkasteltiin virheen suuruuden lisäksi sen suuntaa, voitiin havaita suurimman osan virheellisistä tuloksista olleen sellaisia, joissa algoritmin ilmoittama tulos oli edullisempi kuin varmennettu tulos (kuva 31). Virheet, joissa algoritmi oli antanut liian suuren tuloksen, olivat paitsi harvinaisempia, myös pienempiä. Tämä johtunee siitä, että rasteri- ja vektorialueiden rajoilla oli mahdollista, että polku “hyppää” ohuen vektorimuotoisen kustannusalueen yli, ilman, että tämän alueen kustannus otetaan huomioon polun kustannuksessa. Mekanismin tarkempi toiminta selitettiin kappaleessa 3.4.2. Kustannuspinnassa oli paljon kapeita korkean kustannuksen polygonialueita (ojat), mutta kapeita matalan kustannuksen alueita ei juuri ollut. Tämä selittäisi, miksi virhe edulliseen suuntaan oli tavallisempi, kuin virhe suuntaan, jossa algoritmi ilmoitti liian kalliin tuloksen.

Myöskään kustannuksia, kun siirryttiin rasterin solusta vektorikohteisiin, ei laskettu täysin oikeilla arvoilla, vaan oletettiin kustannuksen olevan sama kuin sen rasterin solun keskipisteeseen, jossa vektorikohteen solmu sijaitsi (kuva 9). Virhe kuitenkin vaikutti vain kaaren kulkukustannukseen, ei sen laskettuun pituuteen. Tämän virheen vaikutus pitäisi olla sattumanvaraisesti positiivinen tai negatiivinen riippuen ympäröivien solujen kustannuksista ja siitä, miten vektorikohteen solmu sijoittui suhteessa rasterin solun keskipisteeseen.

Näitä virheitä pitäisi myös olla paljon enemmän kuin yllä kuvattuja virheitä, joten voisikin olettaa suuren osan pienistä virheistä johtuneen tästä virhelähteestä ja suurempien virheiden selittyvän yllä mainitun ylihyppäysmekanismin avulla. On kuitenkin huomionarvoista, että mikäli käytetty rastertason solukoko oli suuri suhteessa polun pituuteen tai virheellisesti laskettujen rasterin solun alueella oli erityisen suuri kustannus, saattoi myös tästä virheestä johtuva virhe olla huomattavan suuri.

5. Johtopäätökset ja keskustelu

5.1 Keskustelu tapaustutkimuksen tuloksista

Rasteriosuuksilla virhe muodostui rasterimenetelmille tyypillisestä liikkumissuuntien rajoittuneisuudesta, mikä on havaittu myös lukuisissa aiemmissa tutkimuksissa (Goodchild 1977, Bemmelen ym. 1993, Antikainen 2009, Annila 2016). Osuuksilla, joissa polku selvitettiin näkyvyyteen perustuen, muodostui virhe reunapisteiden harvuudesta johtuvista epäoptimaalisista kustannusalueiden reunojen ylityskohdista. Myös tämä virhe on havaittu aiemmissa vektorimenetelmiä koskevissa tutkimuksissa (Antikainen 2009, Annila 2016).

Nämä virheet ovat kuitenkin läsnä vektori- ja rasterimenetelmissä myös silloin, kun niitä käytetään yksinään. Mielenkiintoisimmat virhelähteet komposiittimenetelmään liittyen olivatkin ne, jotka esiintyivät nimenomaan kahden eri aineistotyypin rajoilla. Toteutetussa työkalussa vektori- ja rasterikohteiden välinen siirtymä oli määritelty siten, että rasterin määrittelypolygonin solmusta voitiin siirtyä suoraan määrittelypolygonin solmun sisältävän solun naapurisoluihin epäluonnollisten mutkien välttämiseksi.

Kuitenkin myös tässä menetelmässä ilmeni omat ongelmansa, joita käsiteltiin teoriatasolla kappaleessa 3.6.2. Erityisesti havaittiin ongelmia kaarissa, jotka ylittivät rasterin määrittelypolygonin rajoja. Koska tutkimuksessa käytettiin kuuttatoista naapuria rasterisoluille ja kohtalaisen suurta 16m rasterin solukokoa, kasvoi naapuruston fyysinen koko säteeltään 32 metriin. Kuitenkin kustannuspinnassa oli paljon tätä pienempiä vektorikohteita (kuva 28), jotka jäivät helposti huomioimatta laskettaessa rasteri ja vektorikohteiden välillä siirtymisen kustannusta.

Tämän tapaustutkimuksen valossa vaikuttikin siltä, että paras tapa päästä näistä virheistä eroon olisi ollut käyttää rasterialueilla laajan naapuruston sijaan extended raster menetelmää. Tällöin rasterin solun sisällä sijaitsevasta viiva- tai polygonikohteen pisteestä olisi voitu siirtyä mihin tahansa rasterin solun reunoille luoduista solmuista. Menetelmän etuna olisi ollut naapuruston pienempi fyysinen koko, sekä kaarten sijainti vain yhden rasterin solun alueella. Nämä ominaisuudet olisivat helpottaneet sen tarkistamista, leikkaako kaari

määrittelypolygonia (ks. 5.3.2). Tämän tarkastuksen avulla olisi puolestaan voitu jättää virheellisesti muodostetut kaaret muodostamatta. Myös kun kaaret olisivat sijainneet vain yhden rasterin solun, eli yhden kustannusalueen, alueella olisi kaikille kaarille laskettu niiden kustannus oikein.

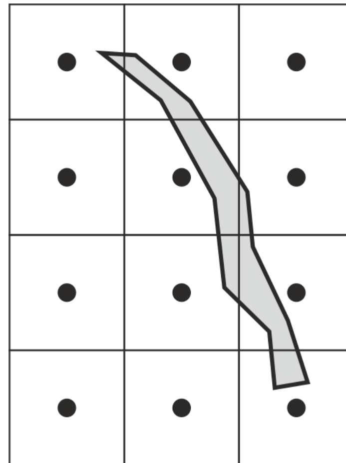
5.2 Kustannuspinnan muodostamisen haasteet

Aiemmissa tutkimuksissa oli todettu erityisesti vektoripohjaisten menetelmien osalta kustannuspinnan muodostamisen olevan käytännössä hankalaa johtuen eri lähteistä haettujen aineistojen huonosta yhteensopivuudesta (Mata & Mitchell 1997, Antikainen 2009). Tässäkin tapaustutkimuksessa havaittiin ongelmia alueiden topologioiden yhdistämisen osalta. Eri lähteistä haettujen polygonien väliin jäi alueita, joita mikään polygoni ei peittänyt, ja polygonien, joiden tulisi selvästi rajoittua toisiinsa , rajat eivät muodostuneet samoista kulmapisteistä. Tällaisia tilanteita esiintyi esimerkiksi järviin ja teihin rajoittuvien kustannusalueiden rajoilla.

Koska reitinetsinnän eteneminen eri alueiden välillä perustuu vierekkäisten kustannuspolygonien yhteisiin solmuihin, oli vektoriaineistojen keskinäinen yhteensopimattomuus ongelmallista. Aiemmissa tutkimuksissa tätä ongelmaa oli korjattu laajentamalla kustannusalueita erilaisilla buffereilla, sekä korjaamalla niiden topologioita yhteensopiviksi tätä varten kehitetyillä algoritmeilla (Antikainen 2009).

Koska kehitetyllä työkalulla oli kuitenkin mahdollista hyödyntää myös rasterimenetelmiä, voitiin koko kustannuspinnan alueelle laatia taustarasteri hilamuotoisesta metsävaratiedosta, joka kattoi koko tutkimusalueen. Näin kustannuspintaa saatiin täydennettyä alueilla, joilla kustannus ei ollut määritelty polygonimuotoisilla kustannuspinnan kohteilla.

Vaikka näin olikin mahdollista paikata puutteellista vektoritietoa rasterimuotoisella aineistolla, ei ratkaisu kuitenkaan ollut ideaali. Erityisesti polygonien väliin jäävät pienet tai kapeat rasterialueet johtivat huonolaatuisiin polkuihin. Kapean rasterin alueella tapahtui usein niin, ettei juuri minkään rasterin solun keskipiste ollut sen määrittelypolygonin sisällä, mikä johti siihen, ettei verkonmuodostuksessa syntynyt kaaria, joita pitkin ohut rasterikaistale olisi voitu ylittää (kuva 32).



Kuva 32. Rasterin määrittelypolygonin ollessa hyvin kapea, on mahdollista, ettei rasterin alueelle synny lainkaan kaaria rasterin solujen välille, mikä käytännössä tekee määrittelypolygonista kokonaan reitinetsinnältä kielletyn alueen.

Metsävaratiedon hyödyntäminen kustannuspinnan täydentämiseen alueiden rajoilla ei myöskään ollut maaston parhaan kuvailun kannalta järkevää. Esimerkiksi, jos tien kummallakin puolella oli peltoa ja voitiin suurella todennäköisyydellä olettaa peltojen rajoittuvan tiehen, löytyi metsävaratiedosta todennäköisesti alueen luokitukseksi avoin alue. Metsävaratiedon avoimet alueet ja pelloiksi luokitellut alueet eivät kuitenkaan vastanneet kustannuksiltaan toisiaan. Vaikka mahdollisuus yhdistellä eri aineistotyyppejä toikin joustavuutta kustannuspinnan muodostamiseen, ei se poistanut eri lähteistä kerättyjen vektoriaineistojen yhdistämiseen liittyviä ongelmia.

5.3 Mahdollisia jatkokehityksen kohteita

5.3.1 Kustannusalueiden spatiaali-indeksointi

Kuten luvussa 3.6 todettiin on yksi nyt kehitetyn algoritmin tehokkuuteen vaikuttava tekijä kustannuspinnan lukeminen ja erityisesti sen määrittäminen, minkä polygonin ja kolmion alueelle maali- ja viivakohteiden pisteet jäävät. Yksi tapa nopeuttaa näytä kyselyjä olisi luoda polygonimuotoisesta kustannuspinnasta spatiaali-indeksointi.

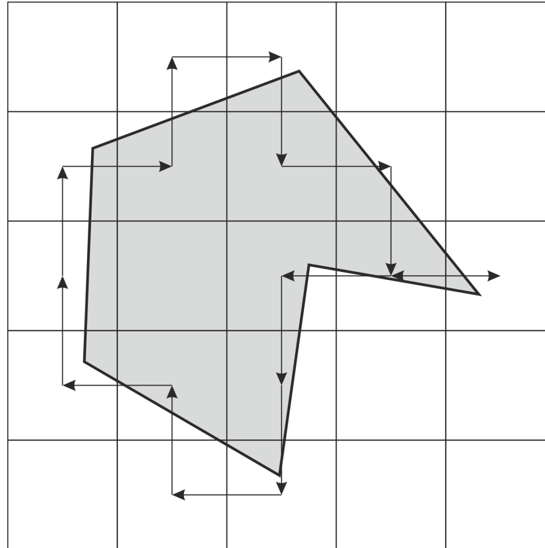
Indeksointi voitaisiin toteuttaa esimerkiksi ryhmittelemällä polygonit hierarkkiseen R-tree

hakupuuhun (Guttman 1984), jossa alemman tason kohteet ryhmitellään isommiksi kokonaisuuksiksi, joille luodaan yhteinen bounding box. Näin voidaan puun ylemmillä tasoilla tarkistaa nopeasti voiko piste kuulua mihinkään haaran polygoneista, minkä jälkeen siirrytään alemmille tasoille niissä haaroissa, joissa on mahdollista löytää polygoni, jonka sisällä piste on.

5.3.2 Rasteritason ja määrittelypolygonin solmujen välisten kaarten määrittäminen

Kuten kappaleessa 3.2 todettiin, ei nyt käytetty menetelmä rasterin solujen ja määrittelypolygonin reunojen yhdistämiseksi tuota täysin oikeita tuloksia, sillä on mahdollista, että kaari leikkaa määrittelypolygonin reunaa, vaikka solun keskipiste olisi määrittelypolygonin sisällä. Tämän virheen poistamiseksi tulisi tarkistaa jokaista kaarta lisätessä leikkaako se polygonin rajaa. Yksi tapa tähän olisi selvittää näkyvyys kaaren päätepisteiden välillä käyttäen suppiloalgoritmia, mutta tehokkaampi tapa saattaisi olla käyttää rasterin sisällä extended raster -menetelmää, jolloin kaaret kulkisivat vain yhden rasterin solun alueella.

Extended raster -menetelmän käytön lisäksi, kun määrittelypolygonin reunoja käytäisiin lävitse, tallennettaisiin jokaiseen rasterin soluun tieto, mitkä polygonin kaaret kulkevat kyseisen solun alueella. Tämä saadaan tehtyä lineaarisessa ajassa suhteessa soluihin, jotka leikkaavat määrittelypolygonin reunoja, olettaen että kovin moni polygonin segmentti ei sijaitse saman solun alueella. Solujen läpikäynti tehdään siten, että aloitetaan jostain määrittelypolygonin pisteestä ja kuljetaan rasterin solusta, jossa määrittelypolygonin solmu on, aina siihen rasterin soluun, johon määrittelypolygonin reunasegmentti seuraavaksi saapuu (kuva 33).



Kuva 33. Rasterin solujen läpikäyntijärjestys, kun selvitetään, minkä rasteritason solun alueella määrittelypolygonin reunasegmentit ovat ovat. Läpikäynnin aloituskohdalla ei ole merkitystä, mutta solut tulee käydä lävitse polygonin reunoja seuraten.

Kun tämä on tehty, voitaisiin soluille, joiden alueella on polygonin reunasegmenttejä, tehdä tarkastus leikkaako lisättävä kaari jotain segmenttiä yksinkertaisesti käymällä läpi kaikki kyseisen solun alueella olevat segmentit. Näitä reunasegmenttejä on oletettavasti enintään muutamia, jolloin koko tarkastus saataisiin tehtyä lineaarisessa ajassa suhteessa rasterin solujen, joita määrittelypolygonin reunat leikkaavat, määrään.

5.3.3 Jatkuvan kustannusrasterin muodostaminen

Mikäli analyysin lopputuloksena halutaan jatkuva kustannusetäisyysrasteri samaan tapaan kuin rasteripohjaisista menetelmistä tulee jokaiseen tuloksena saatavan rasterin solun keskipisteeseen luoda maalisomlu. Tämä on algoritmin nykyisellä toteutuksella ongelmallista sillä maalipsiteiden määrän lisääminen lisää algoritmin tarvitsevaa laskenta-aikaa merkittävästi.

Maalipisteiden määrä lisää algoritmin aikavaativuutta kolmessa kohtaa. Kun syötettä luetaan, tulee jokaisesta maalipisteestä saada selville minkä polygonin sisällä se sijaitsee, polygonia kolmioitaessa tulee maalipisteistä selvittää minkä kolmion sisälle ne jäävät ja suppiloalgoritmin edetessä tulee kolmiota käsiteltäessä niiden sisällä olevien solmujen

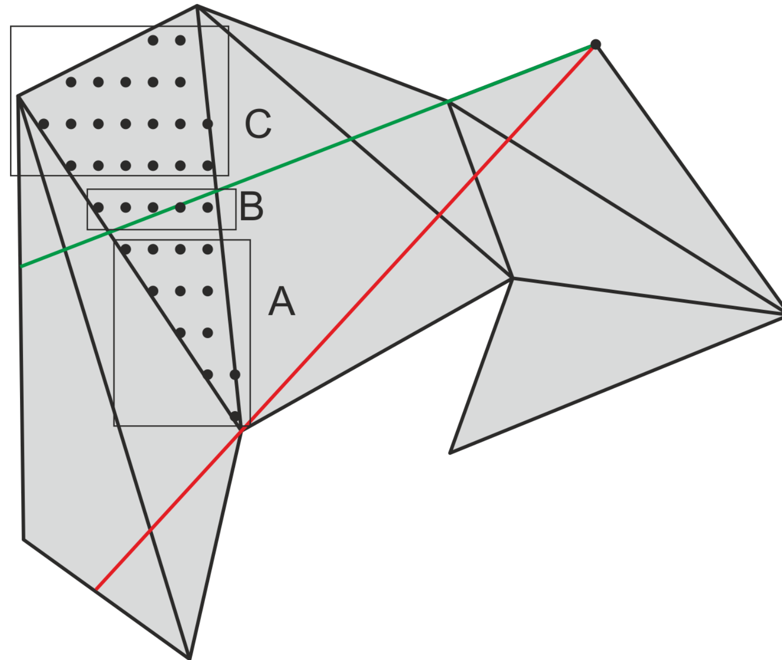
näkyvyys selvittää.

Kuten aiemmin on todettu, ei sen määrittäminen tehokkaasti, mihin polygoniin piste kuuluu, ole täysin yksinkertainen tehtävä. Kuitenkin tietoa siitä, minkä polygonin sisään jokin piste jää, tarvitaan ainoastaan siihen, että piste osataan ottaa huomioon polygonia kolmioitaessa. Koko kustannuspinnan lukeminen voisikin olla järkevämpää tehdä toisella tavalla siten, että jokainen polygoni kolmioitaisiin saman tien sen lukemisen jälkeen ja vain kolmiot ylipäätänsä tallennettaisiin. Kun kaikki polygonit olisi kolmioitu, laadittaisiin kolmioista R-tree spatiaali-indeksi, jonka jälkeen maalipisteitä lukiessa saataisiin nopeasti määritettyä suoraan minkä kolmion alueelle luettavat pisteet kuuluvat.

Kun maalipsteistä on saatu selvitettyä minkä kolmion sisään ne jäävät, lisäävät ne laskenta-aikaa vielä siinä vaiheessa kun selvitetään maalisolmujen näkyvyyttä osana suppiloalgoritmia. Kuitenkin jos solmut on ryhmitelty esimerkiksi hilamuotoon, kuten tehtäisiin jos haluttaisiin laatia lopputuloksena rasterimuotoinen kustannuspinta, voidaan jokaisen maalisolmun erikseen käsittelyltä välttyä käyttämällä binäärihakua seuraavasti:

Jokaisesta rivistä tiedetään, että jos rivin molemmat reunimmaiset solmut näkyvät, täytyy koko rivin näkyä ja jos kumpikaan reunimmaisista pisteistä ei näy, ei mikään muukaan piste voi näkyä (kuva 34). Toisaalta tiedetään, että jos kaksi riviä näkyvät kokonaan, on myös kaikki rivit näiden välissä nähtävissä kokonaan. Tämä on helppo ymmärtää siten, että rivien väliin jäävän alueen ajattelee nelikulmiona. Jos sektorin sisään piirtää nelikulmion, jonka kaikki kulmapisteet ovat sektorin sisällä, on selvää, että tällöin koko nelikulmion pinta-ala on sektorin sisällä. Vastaavasti voidaan tarkistaa onko jokin osa hilasta kokonaan sektorin vasemmalla tai oikealla puolella.

Näitä tietoja yhdistelemällä voidaan ensin tarkastelemalla vain rivien päätepisteitä selvittää binäärihakua käyttäen mitkä rivit näkyvät kokonaan tai ei ollenkaan. Kokonaan näkyvien rivien solmut merkitään näkyviksi ja rivejä, jotka eivät näy lainkaan, ei tarvitse käsitellä enempää. Lopuista riveistä voidaan selvittää binäärihaulla reunimmaiset näkyvät pisteet ja merkitä näiden pisteiden väliset pisteet näkyviksi.



Kuva 34. Mikäli maalipisteet on ryhmitelty hilamuotoon, voidaan ne jakaa kokonaan näkyviin A, osittain näkyviin B ja ei lainkaan näkyviin riveihin C. Nämä jaotukset saadaan selville binäärihaulla, samoin kuin osittain näkyvien rivien kohta, jossa näkyvyys alkaa ja loppuu, mikä tekee hilamuotoon asetettujen pisteiden näkyvyyden tarkastamisesta huomattavasti nopeampaa kuin satunnaisesti sijoitettujen pisteiden tarkastamisesta.

Lähestymistavassa on ongelmana se, että sektorin ollessa pystysuuntainen voi käydä usein niin, että kaikki rivit tullaan luokittelemaan osittain näkyviksi, jolloin jokaisen rivin solmujen näkyvyys pitää selvittää erikseen. Jos rivit ovat vielä kovin lyhyitä, eli kolmio on hyvin korkea ja kapea, voidaan tulla tilanteeseen, jossa olisi tehokkaampaa vain käydä solmut yksitellen lävitse. Joka tapauksessa binäärihakuun perustuvan menetelmän pitäisi olla useimmissa tapauksissa niin paljon nopeampi, että kokonaislaskenta-aika on selvästi pienempi.

5.3.4 Tuotettujen reittien korjaaminen jälkikäteen

Myös menetelmiä reittien korjaamiseen reitinetsinnän jälkeen on tutkittu (Bemmelen ym. 1993, Saab & VanPutte 1999, Antikainen 2009). Rasteripohjaisissa menetelmissä ideana on suoristaa homogeenisten alueiden sisään jäävät polut. Näin saadaankin poistettua paikallisesti rasterimenetelmien tuottamia virheitä, mutta jos polku on mennyt rasterimenetelmien seurauksena kokonaan eri reittiä, kuin globaalisti optimaalinen polku, ei jälkikäteen

korjaaminen poista ongelmaa. Tämä jälkikorjausmenetelmä voitaisiin hyvin liittää osaksi rastereiden alueella tapahtuvaa reitinetsintää komposiittialgoritmin yhteydessä, joskin siihen varmasti liittyisi omat ongelmansa. Erityisen ongelmallista saattaa olla varmistua siitä, että korjattu reitti pysyisi rasterin määrittelypolygonin sisällä. Lisäksi korjausmenetelmä on hyödyllinen vain, jos rasterin alueella on homogeenisia kustannusalueita, jotka olisi joka tapauksessa luontevaa mallintaa polygonimuodossa komposiittimenetelmää käytettäessä.

Myös vektorimuotoisia polkuja on mahdollista korjata jälkikäteen. Antikainen (Antikainen 2009) esittää kaksi korjausmenetelmää. Toisessa saman kustannuksisten alueiden reunan ylittävä polku pyritään suoristamaan ja toisessa eri kustannusalueiden rajan ylittävä polku pyritään saamaan noudattamaan Snellin heijastumislakia. Näistä ensimmäinen on näkyvyyteen perustuvien menetelmien osalta turha, sillä kustannuspinnassa ei pitäisi ikinä olla samaa kustannusta sisältäviä vierekkäisiä polygoneja, vaan nämä tulisi yhdistää yhdeksi polygoniksi. Näkyvyyteen perustuva menetelmä takaa, ettei yhden kustannusalueen sisällä esiinny mutkia, jotka voitaisiin suoristaa (de Berg ym. 2008).

Snellin lakiin perustuva paikallinen rajanylityskohtien optimointi olisi hyödyllinen tällekin menetelmälle, mutta koska kustannusalueet eivät ole kuperia, on optimointi huomattavasti hankalampi toteuttaa. Jos optimoinnin tuloksena polygonien rajanylityskohta siirtyy, täytyy myös tarkistaa, että uusi rajanylityskohta on nähtävissä edellisestä solmusta. Mikäli tämän tarkistuksen laskennalliset kustannukset ovat suuremmat kuin yksinkertaisella rajanylityspaikkojen tihentämisellä varsinaisessa reitinetsintävaiheessa, ei reitin paikallisessa jälkikäteen optimoinnissa ole järkeä.

5.4 Toteutetun algoritmin hyödyllisyys

Esitellyn menetelmän suurin etu on, että se mahdollistaa erilaisten kustannukseen vaikuttavien kohteiden mallintamiseen kustannuspinnassa niitä parhaiten kuvaavilla aineistotyypeillä. Eri aineistotyyppejä yhdistelemällä on myös mahdollista mallintaa kohteita, joiden mallintaminen käyttäen pelkkiä viiva-, polygoni- tai rasterikohteita on vaikeaa tai mahdotonta.

Tämä toisaalta tarkoittaa että algoritmi vaatii syötteenään useampia karttatasoja kuin esimerkiksi pelkkää rasterimenetelmää käyttävä analyysi. Jos aineistot ovat keskenään hyvin yhteensopivia, tämä ei juurikaan aiheuta ongelmia, vaan päinvastoin vältytään aineistojen yhdistämisestä ja muunnoksista johtuvalta työltä. Kuitenkin aineistojen ollessa peräisin eri lähteistä, ovat ne usein keskenään huonosti yhteensopivia, mikä tekee vektorimuotoisen kustannuspinnan muodostamisesta työlästä, kuten aiemmissakin tutkimuksissa on todettu (Mata & Mitchell 1997, Antikainen 2009).

Eri menetelmät alhaisimman kustannuksen reitin etsinnäksi sisältävät omat hyvät ja huonot puolensa. Useissa aiemmissa tutkimuksissa vektoripohjaisia menetelmiä moititaan niiden tehottomuudesta (Bemmelen ym. 1993, Antikainen 2009, Etula & Antikainen 2014). Näissä tutkimuksissa on nähdäkseni kuitenkin joko käytetty tehottomia vektoripohjaisia menetelmiä, tai pyritty ratkaisemaan täysin optimaalinen polku approksimaation sijaan. Verrattaessa vektori- ja rasterimenetelmiä, ei ole mielekästä asettaa vektoripohjaisten menetelmien tavoitteeksi täysin optimaalista polkua, kun myöskään rasteripohjaiset menetelmät eivät tähän pääse. Tehokasta approksimaatiota käyttäen, on mahdollista tuottaa homogeenisistä alueista koostuvasta kustannuspinnasta vektorimenetelmin vastaavassa laskenta-ajassa parempia tuloksia kuin mihin perinteisillä rasterimenetelmillä päästään (Annila 2016).

Toisaalta myös rasteripohjaisilla menetelmillä saadaan hyviä tuloksia, mikäli kustannuspinta on sellainen, että se on luontevaa kuvata rasteriaineistona. Tässä työssä reitin rasteriosuuksien selvittämiseen käytettiin kuusitoistasuuntaista menetelmää, joka on todettu (Bemmelen ym. 1993) parhaaksi kompromissiksi tehokkuuden ja tarkkuuden välillä. Kuitenkin myös esimerkiksi extended raster-menetelmässä on omat hyvät puolensa. Toteutetun tapaustutkimuksen valossa näyttääkin siltä, että nimenomaan extended raster-menetelmällä olisi eniten potentiaalia toimia hyvin yhteen polygonikohteista muodostettavan kustannuspinnan kanssa. Menetelmän hyvä yhteensopivuus johtuu käytettävän naapuruston pienestä fyysisestä koosta ja verkon kaarien sijainnista vain yhden rasterin solun alueella.

Toteutettu tapaustutkimus mielestäni osoitti, että eri menetelmiä yhdistämällä voidaan saada selkeitä etuja yhteen tai kahteen aineistotyyppiin rajoittuneisiin menetelmiin nähden tilanteissa, joissa kustannuspinta muodostuu useista erityyppisistä kohteista. Toisaalta se

myös selvästi havainnollisti eri lähteistä kerättyjen erityyppisten aineistojen yhteenliittämiseen liittyviä haasteita niin kustannuspinnan luonnin, kuin reitinetsintäalgoritmin toteutuksen osalta.

Tapaustutkimuksessa havaittiin nyt kehitetyn työkalun voivan tuottaa huomattavan suuriakin virhearvioita polun kustannuksesta. Virhe on suuri suhteessa hyvin toteutettuihin rasteri- ja vektorimenetelmiin (Annila 2016), ja sen aiheuttavat virhelähteet olisi saatava korjattua, jotta työkalu olisi soveltuva käytännön optimointiongelmien ratkaisemiseen. Kuitenkin mahdollisia virhelähteitä onnistuttiin paikantamaan, ja vaikuttaisi, että näiden virhelähteiden osalta algoritmin korjaaminen olisi toteutettavissa ilman, että laskentaresurssien tarve kasvaisi kohtuuttomasti.

5.5 Vaihtoehtoisia lähestymistapoja kustannuspinnan mallintamiseen

Kuten jo aiemmin todettiin, vektoripohjaisten menetelmien suurin ongelma lienee eri lähteistä haetun vektorimuotoisen datan yhteen sovittamiseen liittyvät vaikeudet (Mata & Mitchell 1997, Antikainen 2009). Vaikka huonosti yhteensopivasta topologiasta johtuvia ongelmia on mahdollista korjata kustannuspintaa muodostettaessa (Antikainen 2009), tai täydentää vektoriaineistoja vaihtoehtoisilla rasteriaineistoilla, on toimivan kustannuspinnan aikaansaaminen joka tapauksessa työlästä.

Koska vektoriaineistoista erityisesti polygoniaineistot muodostavat ongelmallisia tilanteita aineistoja yhdistettäessä, ja erityisesti polygonien yhteensovittaminen automatisoidusti on vaikeampaa kuin esimerkiksi viiva- ja pistekohteiden, voisi olla järkevää kehittää enemmän rasteripohjainen komposiittimenetelmä. Menetelmän tulisi kuitenkin kyetä mallintamaan monimutkaisia kustannuspinnan kohteita, kuten esimerkiksi tässä työssä toteutettu vilkkaiden teiden ylitys. Tämä menetelmä voisi hyödyntää yhdestä rasterista, sekä viivoista ja pisteistä koostuvaa kustannuspintaa. Vastaavia rasteri- ja viivakohteita yhdistäviä menetelmiä on esitetty viivamaisten esteiden (Dean, Vaishnavi & Neeraj 2016) sekä kuljettavan viivaverkoston mallintamiseen (Choi, Um & Park 2014). Viiva- ja rasterimenetelmiä yhdistäviä menetelmiä olisi kuitenkin mahdollista viedä huomattavasti pidemmälle. Menetelmän tulisi kyetä käsittelemään viivakohteita sekä esteinä että kulkuväylinä ja toisaalta nämä olisi kyettävä yhdistämään hyviä tuloksia antaviin tehokkaisiin rasterimenetelmiin.

Heterogeenisillä rasterin alueilla voitaisiin esimerkiksi hyödyntää hyväksi todettua rasterimenetelmää, kuten extended raster menetelmää, ja homogeenisemmilla rasterin alueilla voitaisiin edetä käyttäen esimerkiksi quadtree-menetelmää, joka soveltuu paremmin homogeenisempien alueiden käsittelyyn (Samet 1984, Bemmelen ym. 1993, Vörös 2001).

Tällaisen eri rasteripohjaisia algoritmeja viivakohteisiin yhdistävän menetelmän etuna olisi kustannuspinnan yksinkertaisempi muodostaminen, kyky käsitellä sekä hetero- että homogeenisiä kustannuspintoja kohtalaisen hyvin, sekä kyky käsitellä viivakohteita, mikä mahdollistaisi myös monimutkaisempien kohteiden mallintamisen.

5.6 Loppusanat

Koska rasteri- ja vektoriaineistojen välillä siirtyminen on määritelty rastereiden määrittelypolygonien reunasolmujen avulla, voitaisiin rasterin sisällä käytettävää reitinetsintämenetelmää vaihtaa, jopa rasterista toiseen, ilman että se vaikuttaisi muuhun reitinetsintään. Vastaavasti polygonien sisällä tapahtuva reitinetsintä voidaan toteuttaa eri tavoin ilman että se vaikuttaa rasteri- ja viivaosuuksilla tehtävään reitinetsintään. Näin ollen merkittävin kysymys eri aineistotyyppejä yhdistävän algoritmin osalta onkin se, kuinka eri aineistotyyppien rajoille määriteltävät rajapinnat saadaan toimimaan mahdollisimman saumattomasti, mutta kuitenkin siten, ettei kustannuspinnan lukeminen vie liikaa aikaa.

Nyt kehitetty menetelmä eri aineistojen yhdistämiseen on nähdäkseni melko hyvä lähtökohta aineistotyyppien rajapintojen määrittelylle. Se mahdollistaa yleisesti käytettyjen paikkatietoaineistojen yhdistelyn aiempaa monimutkaisempien ilmiöiden kuvaamiseksi kustannuspinnassa. Kuitenkin myös selviä parannuskohteita havaittiin liittyen algoritmin tehokkuuteen, tarkkuuteen ja yksinkertaisuuteen.

KIRJALLISUUS

Aleksandrov, L., Lanthier, M., Maheshwari, A. & Sack, J. 1998, An ε — Approximation algorithm for weighted shortest paths on polyhedral surfaces, *Algorithm Theory — SWAT'98,* , toim. S. Arnborg & L. Ivansson, Springer Berlin Heidelberg, Berlin, Heidelberg, s. 11-22.

Annila, E. 2016, *Näkyvyyteen perustuva pienimmän kustannuksen polun etsintä, ja sen vertailu rasterimenetelmiin*, Julkaisematon Kandidaatin tutkielma Helsingin yliopisto blogs.helsinki.fi/geoinformatiikka/files/2016/09/Kandidaatintutkielma_Elias_Annila.pdf.

Antikainen, H. 2013, *Using the Hierarchical Pathfinding A* Algorithm in GIS to Find Paths through Rasters with Nonuniform Traversal Cost,* 996-1014 s.

Antikainen, H. 2009, Terrain path optimization using the connectivity graph approach applied to GIS data structures, *Nordia Geographical Publications,* 38:3.

Bagli, S., Geneletti, D. & Orsi, F. 2011, Routing of power lines through least-cost path analysis and multicriteria evaluation to minimise environmental impacts, *Environmental Impact Assessment Review,* 31:3, s. 234-239. DOI //doi.org/10.1016/j.eiar.2010.10.003.

Bemmelen, J., Quak, W., van Hekken, M. & Oosterom, P. 1993, Vector vs . Raster-based Algorithms for Cross Country Movement Planning, *Auto-Carto 11,* American Society for Photogrammetry and Remote Sensing, Bethesda, 30.10 - 1.11.1993, s. 304-317.

Bondy, J.A. & Murty, U.S.R. 2008, *Graph theory,* Springer, New York, NY.

Botea, A., Müller, M. & Schaeffer, J. 2004, Near optimal hierarchical path-finding (HPA*), *Journal of game development,* 1:1, s. 7–28.

Choi, Y., Um, J. & Park, M. 2014, Finding least-cost paths across a continuous raster surface with discrete vector networks, *Cartography and Geographic Information Science,* 41:1, s. 75-85. DOI 10.1080/15230406.2013.850837.

de Berg, M., Cheong, O., van Kreveld, M. & Overmars, M. 2008, *Computational Geometry: Algorithms and applications,* 3. p. 367 s. Springer Berlin Heidelberg, Berlin, Heidelberg.

Dean, D.J., Vaishnavi, T. & Neeraj, S. 2016, Optimal Routefinding Across Landscapes Featuring High-cost Linear Obstacles, *Transactions in GIS,* 20:4, s. 613-625. DOI 10.1111/tgis.12170.

Desrochers, A., Bélisle, M., Morand-Ferron, J. & Bourque, J. 2011, Integrating GIS and homing experiments to study avian movement costs, *Landscape Ecology,* 26:1, s. 47-58. DOI 10.1007/s10980-010-9532-8.

Dijkstra, E.W. 1959, A note on two problems in connexion with graphs, *Numerische Mathematik,* 1:1, s. 269-271. DOI 10.1007/BF01386390.

Dorigo, M. & Stützle, T. 2003, The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances, teoksessa *Handbook of Metaheuristics*, toim. F. Glover & G.A. Kochenberger, Springer US, Boston, s. 250-285.

Douglas, D.,H. 1994, Least-cost Path in GIS Using an Accumulated Cost Surface and Slopelines, *Cartographica: The International Journal for Geographic Information and Geovisualization,* 31:3, s. 37-51. DOI 10.3138/D327-0323-2JUT-016M.

Etula, H. & Antikainen, H. 2014, Reitinoptimoinnin hyödyllisyys metsävaratiedon keruun maastotyössä, *Metsätieteen aikakauskirja,* 2014:2, s. 81-99. DOI 10.14214/ma.6895.

Feldman, S.C., Pelletier, R.E., Walser, E., Smoot, J.C. & Ahl, D. 1995, A prototype for pipeline routing using remotely sensed data and geographic information system analysis, *Remote Sensing of Environment,* 53:2, s. 123-131. DOI 10.1016/0034-4257(95)00047-5.

Fredman, M.L. 1999, On the Efficiency of Pairing Heaps and Related Data Structures, *Journal of the ACM,* 46:4, s. 473-501. DOI 10.1145/320211.320214.
Fredman, M.L., Sedgewick, R., Sleator, D.D. & Tarjan, R.E. 1986, The pairing heap: A new form of self-adjusting heap, *Algorithmica,* 1:1, s. 111-129. DOI 10.1007/BF01840439.
Fredman, M.L. & Tarjan, R.E. 1987, Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, *Journal of the ACM,* 34:3, s. 596-615. DOI 10.1145/28869.28874.
Giesbrecht, J. 2004, *Global Path Planning for Unmanned Ground Vehicles*, Defence Research and Development Canada, Medicine Hat.
Goldberg, A. & Harrelson, C. 2005, Computing the shortest path: A* search meets graph theory, Society for Industrial and Applied Mathematics, Philadelphia, US, 23 - 25.1.2005, s. 156–165.
Goldberg, A., Kaplan, H. & F Werneck, R. 2009, Reach for A*: Shortest Path Algorithms with Preprocessing, AMS, s. 93–140.
Goodchild, M.F. 1977, An Evaluation of Lattice Solutions to the Problem of Corridor Location, *Environ Plan A,* 9:7, s. 727-738. DOI 10.1068/a090727.
Gutman, R. 2004, Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks, *ALENEX,* SIAM, New Orleans, s. 100-111.
Hart, P.,E., Nilsson, N.,J. & Raphael, B. 1972, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths,* 28-29 s.
Hershberger, J. 1989, An optimal visibility graph algorithm for triangulated simple polygons, *Algorithmica,* 4:1, s. 141-155. DOI 10.1007/BF01553883.
Iacono, J. 2000, Improved Upper Bounds for Pairing Heaps, *Algorithm Theory - SWAT 2000,* Springer Berlin Heidelberg, Berlin, Heidelberg, s. 32-45.
Iqbal, M., Sattar, F. & Nawaz, M. 2006, Planning a Least Cost Gas Pipeline Route A GIS & SDSS Integration Approach, IEEE, Islamabad, 2 - 3.9.2006, s. 126-130.
James, D. 2018, *Boost Library Documentation*, Saatavilla: https://www.boost.org/doc/libs/1_67_0/doc/html/hash.html (Haettu 18.7.2017).
Knuth, D. 1998, *The Art of Computer Programming, Volume 3: Sorting and Searching,* 2. p. 780 s. Addison-Wesley Professional.
Latombe, J. 1991, *Robot motion planning,* 7. p. 651 s. Springer US, Boston, MA.
Lee, D.T. & Preparata, F.P. 1984, Euclidean shortest paths in the presence of rectilinear barriers, *Networks,* 14:3, s. 393-410. DOI 10.1002/net.3230140304.
Lösch, A. 1954, *The economics of location,* 2. p. 520 s. Yale Univ. Press [u.a.], New Haven.
Mata, C.S. & Mitchell, J.S.B. 1997, A New Algorithm for Computing Shortest Paths in Weighted Planar Subdivisions, *Proceedings of the Thirteenth Annual Symposium on Computational Geometry,* ACM, New York, NY, USA, s. 264-273.
Mitchell, J. & Papadimitriou, C. 1991, The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision, *Journal of the ACM,* 38:1, s. 18-73. DOI 10.1145/102782.102784.
Pallecchi, S. & Metz, M. 2015, *GRASS GIS Manual*, Saatavilla: https://grass.osgeo.org/grass74/manuals (Haettu 19.7.2018).
Perez Ruy-Díaz, J. & Safar Sideq, M. 2004, Approximating optimal paths in terrains with weight defined by a piecewise-linear function, CCCG, Montreal, s. 72-75.
Pocchiola, M. & Vegter, G. 1996, Topologically sweeping visibility complexes via pseudotriangulations, *Discrete & Computational Geometry,* 16:4, s. 419-453. DOI 10.1007/BF02712876.
Rothley, K. 2005, Finding and Filling the 'Cracks' In Resistance Surfaces for Least-Cost

Modeling, *Ecology and Society,* 10:1, s. 4. DOI 10.5751/ES-01267-100104.
Saab, Y. & VanPutte, M. 1999, Shortest path planning on topographical maps, *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans,* 29:1, s. 139-150. DOI 10.1109/3468.736370.
Sadeghi-Niaraki, A., Varshosaz, M., Kim, K. & Jung, J.J. 2011, Real world representation of a road network for route planning in GIS, *Expert Systems With Applications,* 38:10, s. 11999-12008. DOI 10.1016/j.eswa.2010.12.123.
Samet, H. 1984, The Quadtree and Related Hierarchical Data Structures, *ACM Computing Surveys (CSUR),* 16:2, s. 187-260. DOI 10.1145/356924.356930.
Shimrat, M. 1962, Algorithm 112: Position of Point Relative to Polygon, *Communications of the ACM,* 5:8, s. 434. DOI 10.1145/368637.368653.
Speičys, L. & Jensen, C.S. 2008, Road Network Data Model, teoksessa *Encyclopedia of GIS*, toim. S. Shekhar & H. Xiong, Springer US, Boston, MA, s. 972-978.
Theoharis, T., Platis, N., Patrikalakis, N.M. & Papaioannou, G. 2008, *Graphics and Visualization,* A K Peters/CRC Press, Natick.
Vörös, J. 2001, *Low-cost implementation of distance maps for path planning using matrix quadtrees and octrees,* 447-459 s.
Xu, J. & Lathrop, R.G. 1995, Improving simulation accuracy of spread phenomena in a raster-based Geographic Information System, *International Journal of Geographical Information Systems,* 9:2, s. 153-168. DOI 10.1080/02693799508902031.

LIITTEET:

Liite 1. Toteutettu reitinetsintä sovellus: https://github.com/ealiasannila/lcpc