TAMPEREEN
AMMATTIKORKEAKOULU

PAIKKATIETOPROSESSIEN AUTOMATISOINTI DIGIRIISTAMETSÄ-HANKKEESSA

Thomas Jensen

Opinnäytetyö
Maaliskuu 2020
Metsätalouden koulutus

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Metsätalouden koulutus

JENSEN, THOMAS:
Paikkatiedonprosessien automatisointi Digiriistametsä-hankkeessa

Opinnäytetyö 69 sivua, joista liitteitä 19 sivua
Maaliskuu 2020

Digiriistametsä on Suomen metsäkeskuksen ja Riistakeskuksen yhteinen hanke, jonka tarkoituksena oli löytää metsäkanalinnuille soveltuvia elinympäristöjä. Hankkeen tavoitteena oli selvittää, mitä paikkatiedon prosesseja tarvitaan ja mihin aineistoon, jotta löydetään metsäkanalinnuille soveltuvia elinympäristöjä ja saadaan automatisoitua koko prosessi. Tutkimusmenetelmänä käytettiin paikkatietoaineiston analysointia ArcMap-ohjelmistolla, jonka tulokset jatkojalostettiin ja vietiin Zonation-ohjelmaan.

Tulokseksi saatiin useampi rasteri, jotka vaikuttavat metsäkanalintujen elinympäristöön joko positiivisesti tai negatiivisesti. Näitä aineistoja ovat mm. vesistöt, puuston pituus ja rakennukset Näitä rastereita hyödynnetään kytkeytyneisyysanalyysissä, joka suoritetaan Zonation-ohjelmalla.

Saadut rasterit ovat uskottavia, ja prosessin automatisointi on onnistunut. Automatisoinnissa hyödynnetään useampaa prosessorin ydintä, jolloin käsittelyajat ovat lyhyemmät. Zonation-analyysin tulokset koealueista ovat uskottavia ja analyysin automatisointi on tehty asetuksia vaille valmiiksi.

ABSTRACT

Tampere University of Applied Sciences
Forestry Education

JENSEN, THOMAS:
Automatization of Spatial Processes in Digiriistametsä Project.

Bachelor's thesis 69 pages, appendices 19 pages
March 2020

Digiriistametsä is a joint project between Suomen metsäkeskus (Finnish Forest Centre) and Riistakeskus (The Finnish Wildlife Agency). The goals of the project are to find suitable habitats for grouse, what spatial processes are needed, what sources are needed and to automate the whole process. In this project spatial analyzing software called ArcMap was used to analyze and to handle the source material. After the material has been worked with, the results are sent to Zonation software to further analyze the material.

The results were a multiple rasters that affects the habitats of the grouse either positively or negatively. These results were watery areas like rivers and lakes, human constructions like roads and buildings and forest traits like tree length and tree volume. These rasters are used in Zonation analyze to identify connected reserve networks.

The results are convincing and the process was successfully automatized. The automatized process uses three cores to improve the processing speed. The results from the Zonation analyze are also convincing and starting Zonation analyze is part of the automatized process, excluding the settings that are still missing.

SISÄLLYS

1 JOHDANTO

Opinnäytetyö käsittelee omaa osuuttani Digiriistametsä-hankkeessa, joka alkoi 2017 ja päättyi 2019. Hankkeen tavoitteena oli löytää metsäkanalinnuille sopivia elinympäristöjä käyttämällä FME sovelluksen ArcMapin Python kirjastoa työstämään Suomen Metsäkeskuksen, Maanmittauslaitoksen ja Riistakeskuksen aineistoja.

Kyseessä oli valtakunnallinen hanke, ja aineistokin on valtakunnallinen, minkä vuoksi aineiston sujuva käsittely edellytti toimintojen automatisointia. FME sovelluksella käsiteltiin vektoriaineistoa ja tietokantahaut, jotka välitettiin eteenpäin ArcMapille, jossa käytettiin Python ohjelmointikieltä toimintojen automatisointiin. Näiden prosessien tuloksena saatiin useampi rasteritiedosto, jotka vietiin Zonation sovellukseen, joka suoritti elinympäristöjen kytkeytyneisyysanalyysin. Tästä saatu tieto on tarkoitus viedä Metsään.fi-palveluun, josta metsänomistajat ja toimijat pystyvät paremmin huomioimaan metsäkanalinnut metsänhoitotoimenpiteissä.

Oma osuuteni sijoittui ajanjaksolle, joka kesti vuoden 2018 elokuusta vuoden 2019 helmikuuhun asti. Opinnäytetyön tavoitteena on selvittää, mitä paikkatiedon prosesseja tarvittiin ja mihin aineistoon. Kun tarvittavat prosessit ja aineistot oli selvitetty, ohjelmoitiin koko prosessi Pythonilla. Prosessin tuloksena saatiin rasteriaineisto, jota voitiin hyödyntää Zonation-analyysissä. Tarkoituksena oli, että koko hanke on automatisoitu, jotta riista-aineisto, rasterit ja Zonation analyysi voidaan päivittää vuosittain.

2 PAIKKATIETO

Paikkatieto on tietoa, johon voidaan liittää sijainti, joka metsätaloudessa tarkoittaa useimmiten puusto- ja kuviotietoja. Tietoja voidaan työstää ja analysoida, jolloin tuotetaan uutta tietoa. Tätä prosessia kutsutaan paikkatietoanalyysiksi. Analyysit voidaan jakaa kahteen ryhmään: visuaalisiin analyyseihin ja laskennallisiin analyyseihin. (Paikkaoppi n.d.)

Visuaalinen analyysi on yksinkertainen tapa hyödyntää paikkatietoa. Siinä eri paikkatietoaineistoja asetetaan näkyväksi päällekkäin ja tasojen järjestystä voidaan muuttaa. Muodostuneesta karttakuvasta voidaan tehdä johtopäätöksiä tutkittavasta ilmiöstä. (Paikkaoppi n.d.)

Laskennallisissa analyyseissä hyödynnetään paikkatieto-ohjelman operaatioita. Tavallisimpia analyyseja ovat päällekkäisanalyysit, yhdistävyysanalyysit ja naapuruusanalyysit. Nimensä mukaisesti päällekkäisanalyyseissa aineistoja verrataan päällekkäin ja pyritään löytämään laskennallisesti niiden välisiä yhteyksiä. Yhdistävyysanalyyseissä tarkastellaan verkostoja, kuten teitä. Tällaisia analyyseja voidaan käyttää löytämään lyhin reitti tai kustannuksiltaan halvin reitti. Naapuruusanalyysien avulla tutkitaan kohteen yhteyttä ympärillä oleviin kohteisiin. (Paikkaoppi n.d.)

2.1 Laserkeilausaineisto

Laserkeilausaineisto kerätään laserkeilaimella. Laserkeilain lähettää lasersäteen, joka kohteeseen osuessaan heijastuu takaisin laitteeseen ja tallentuu pisteeksi. Laserkeilauksen tuloksena syntyy kolmiulotteinen pistepilvi, jonka tarkkuus määräytyy kaluston, mittausetäisyyden ja mittausresoluution mukaan. Kohde voidaan tallentaa digitaalisen muotoon, jota kutsutaan pistepilvimalliksi. Tallenne sisältää yhden pisteen koordinaattitiedon ja intensiteettiarvon, joka kuvaa mittauskohteeseen lähetetyn lasersäteen takaisinheijastusarvoa. (Neopoint n.d.)

2.2 Digiriistametsä

Digiriistametsä-hanke alkoi vuonna 2017 ja päättyi 2019. Hankeen tavoitteena oli tunnistaa metsäkanalinnuille soveltuvia elinympäristöjä ja parantaa metsänomistajan mahdollisuuksia riistametsänhoitoon ja lisätä metsäammattilaisten ja metsänomistajien tietoa sekä osaamista riistametsänhoidosta. Tuotettu tieto viedään Metsäkeskuksen Metsään.fi-palveluun. Hanke oli osa valtakunnallista metsänkanalintujen hoitosuunnitelman toimeenpanoa ja kytkeytyi METSO-ohjelmaan ja kansalliseen metsäohjelmaan (Miettinen & Kesälä 2018, 36–37). Hankkeessa hyödynnetään Metsäkeskuksen paikkatietoaineistoja. Tämä tieto auttaa metsänomistajia ja toimijoita paremmin huomioimaan metsäkanalintujen elinympäristöjä hakkuita ja metsänhoitoja suunniteltaessa. Hanke toteutettiin Suomen Metsäkeskuksen ja Suomen Riistakeskuksen yhteistyönä, jossa Riistakeskus on päätoteuttaja. (Digiriistametsä 2020.)

2.3 Metsävaratieto

Suomen yksityismetsistä kerätään eriasteista tietoa. Tätä tietoa kutsutaan metsävaratiedoksi. Metsävaratieto on esimerkiksi tietoa puustosta, hakkuumahdollisuuksista ja metsän arvosta. Metsävaratieto auttaa suunnittelemaan metsän hoitoa kestävästi ja kannattavasti. (UPM metsä 2019.)

Metsäkeskuksen ja Maanmittauslaitoksen aineistojen ajantasaistuksessa käytetään ilmakuvia ja keilausaineistoja (Maastotietokanta n.d.). Tämän lisäksi Metsäkeskus tekee koealamittauksia ja kohdennettuja maastoinventointeja. Metsäkeskuksella menee kymmenen vuotta, jotta Suomen metsät saadaan inventoitua. Inventoitavien alueiden valintaan vaikuttavat

- metsävaratiedon ikä
- inventoinnin tasainen alueellinen eteneminen
- inventointialueiden yhtenäisyys
- resurssit
- yhteistyömahdollisuudet

Kerätystä metsävaratiedosta lasketaan metsätalousmaille puustotiedot hilaruuduille. Hilaruudut ovat 16 m x m16 m kokoisia alueita, joista inventointialue koostuu. Lopullinen kuviointi tehdään kartta- ja kaukokartoitusaineiston perusteella. Kuvioiden puustotiedot

lasketaan hilaruutujen summa- tai keskiarvotietoina ja alueelle haetaan kasvupaikkatiedot. (Metsätiedonkeruu 2019.)

3 TUTKIMUSAINEISTO JA -MENETELMÄT

3.1 Aineisto

Hankkeessa käytettiin Metsäkeskuksen metsävaratietoa, Maanmittauslaitoksen maastotietokantaa ja Suomen Riistakeskuksen riistakolmioaineistoa. Metsäkeskuksen metsävaratietoaineistosta saatiin puustotiedot, joita täydennettiin LUKEn valtakunnallisella metsäaineistolla. Maanmittauslaitoksen aineistosta käytettiin mm. virtauksia, soita, teitä, rakennuksia ja vastaavia aineistoja. Riistakeskuksen riistakolmioaineiston olennaisimmat tiedot olivat lajit ja havaintojen koordinaatit.

3.2 Elinympäristöt

Hankkeessa yhdistettiin metson, teeren ja pyyn aineistot yhdeksi tiedostoksi, joka nimettiin metepy:ksi. Näiden lisäksi käsiteltiin riekkoaineistoa, mutta se käsiteltiin erikseen. Hankkeessa todettiin, että metsäkanalintujen elinympäristöt eivät poikenneet toisistaan riittävästi, jotta ne kannattaisi käsitellä erikseen. Poikkeuksena oli riekko. Riekko päätettiin käsitellä erikseen, koska se on luokiteltu uhanalaiseksi lajiksi.

Metsäkanalinnuille hyväksi elinympäristöksi katsottiin ne alueet, joissa on riittävä latvuspeittävyys, alikasvos, puuston pituus ja järeys, heterogeeninen puusto ja jotka sijaitsevat luonnontilaisella ja ojitetulla suolla. Zonation-vaiheessa jokainen edellä mainittu piirrekerros saa painotuksen, jonka perusteella Zonation laskee kytkeytyneisyyden.

3.3 Työkalut

3.3.1 FME

FME on tiedon käsittely- ja hallintasovellus. Sen avulla tietoa voidaan hakea useista eri lähteistä ja eri formaatteina. Haettua tietoa voidaan käsitellä, tallentaa ja viedä tietoa eteenpäin, ja koko prosessi voidaan automatisoida. (Spatialworld 2018).

3.3.2 Python ja ohjelmointi

Python on ohjelmointikieli, joka soveltuu skriptien kirjoittamiseen. Skripti on ohjelma, jonka ajossa käytetään toista ohjelmaa, tässä tapauksessa Python-kääntäjää. Python-kääntäjä mahdollistaa skriptin nopean kirjoittamisen ja testaamisen. Python on yksinkertainen, selkeä, korostaa luettavuutta ja hyödyntää moduulien sekä kirjastojen käyttöä. Hankkeessa käytettiin Pythonia, koska se mahdollisti paikkatietoanalyysien automatisoinnin käyttämällä ArcMapin Python-kirjastoa ilman, että ArcMap-sovellusta tarvitsee käyttää. Itse skriptin kirjoittamisessa käytettiin Notepad+ -sovellusta. (Python 2020.)

3.3.3 ArcMap

ArcMap on paikkatieto-ohjelma, jolla voidaan esittää maantieteellistä informaatiota lajiteltuna erinäisiin kerroksiin. ArcMapilla voidaan työstää, tulostaa ja julkaista karttoja sekä käsitellä ja analysoida paikkatieto aineistoa. (What is geoprosessing? n.d.)

Hankkeessa käytettiin ArcMap Python kirjaston työkaluja suorittamaan erinäisiä paikkatiedon analyysejä. Näitä analyysejä olisi voinut tehdä FME:llä ja QGIS:llä, mutta FME rasterityökalut eivät olleet yhtä tehokkaita kuin ArcMapin työkalut. Työt aloitettiin ArcMapilla, eikä ollut mitään syytä vaihtaa QGIS:iin kesken projektin.

3.3.4 ArcMap toiminnot

Projektissa tarvittava aineisto tallennettiin geodatabaseen. Geodatabase on keskitetty kokoelma maantieteellisiä tietoja. Geodatabasessa on ArcMapin ensisijainen tietoformaatti tiedon hallintaan ja muokkaukseen. Geodatabase voi pitää sisällään rastereita, attribuutti taulukoita ja feature classeja (What is a geodatabase). Feature class on tietoformaatti, joka

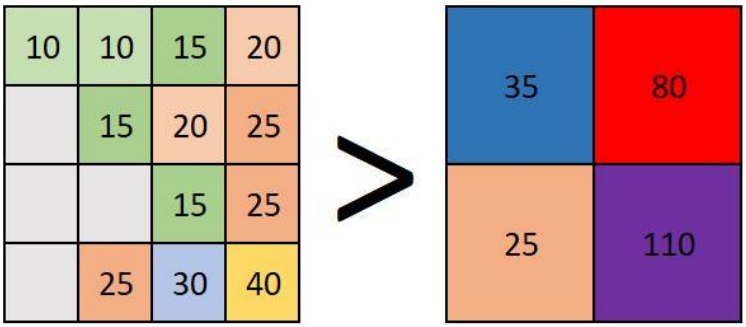
pitää sisällään vektoritietoa, kuten pisteet, viivat ja aluemaiset kohteet, eli polygonit. (Feature class basics n.d.)

Projektissa kaikki aineistot piti muuttaa rasteriksi, jotta Zonation analyysi onnistui. Tämän vuoksi käytettiin linestatistics työkalua, joka muuttaa viiva-aineiston, kuten tiet, rasteriksi. LineStatistics toiminto laskee jokaisen luotavan solun sisään jäävän viiva-aineiston ominaisuustiedot. Näiden tietojen laskennassa käytetään seuraavia metodeja: enemmistö, vähemmistö, suurin arvo, pienin arvo, uniikkien arvojen määrä, arvoalue, keskiarvo, mediaani ja pituus. Jos solu ei sisällä linjoja, saa solu arvoksi NODATAn, paitsi jos käytetään vaihtelevuutta tai pituutta, jolloin solut saavat arvoksi nollan. Enemmistö-, vähemmistö-, keskiarvo- ja mediaanimetodeissa arvo on painotettu linjan pituuden mukaan, eli jos linja1 on kaksi kertaa pidempi kuin linja2, laskennassa linja1 arvot esiintyvät kahdesti. (Line Statistics n.d.)

Aggregate-työkalulla voidaan yhdistää soluja, jolloin soluista tulee suurempia ja solujen määrä vähenee (kuvio 1), jolloin aineiston käsittely nopeutuu ja tiedostojen koot pysyvät kohtuullisina. Työkalua kannattaa käyttää, kun aineistoa on runsaasti ja sen tarkkuudesta voidaan karsia. Aggregate-työkalun etuna on käsittelyajan lyhentyminen, koska työstettäviä soluja on vähemmän. Koska Digiriistametsä-hanke käyttää valtakunnallisia aineistoja, pitää tarkkuudesta karsia, jotta käsittelyajat pysyvät kohtuullisina. Aggregatessa soluja yhdistetään käyttämällä eri menetelmiä, jotka ovat summa, maksimi, minimi, keskiarvo ja mediaani. Alla on lyhyt esimerkki Aggregate-työkalusta, jossa käytetään summa-menetelmää, lyhyesti Aggregate(rasteri1, 2, ’SUM’).

Toiminnon sisältö on seuraava: Aggregate saa ensimmäiseksi argumentiksi rasterin. Toiseksi argumentiksi tulee solukerroin, joka on tässä tapauksessa 2. Tämä tarkoittaa sitä, että solun sivun pituus kaksinkertaistuu. Aikaisemmin solun sivun pituus oli 1 ja nyt se

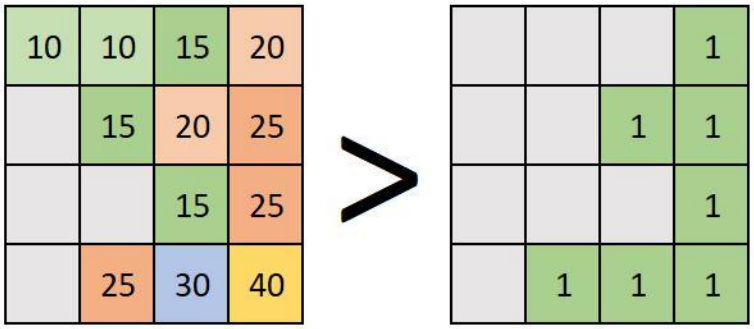
on 2. Viimeiseksi argumentiksi tulee menetelmä, joka on tässä tapauksessa summa, joka laskee yhdistettävien solujen arvot yhteen.



KUVIO 1. Esimerkki rasterista ennen Aggregate-työkalua, SUM asetuksella ja jälkeen

Con-työkalun avulla muutettiin rasterin pikselien syvyyttä, mikä pienensi rastertiedostojen kokoa ja nopeutti käsittelyaikaa. Con-työkalun avulla rasterista voidaan valita solut, jotka täyttävät annetun ehdon, kuten esim. solut, joilla on tietty arvo tai solut, jotka omaavat vähintään/enintään tietyn arvon. Con-työkalulla on helppo hakea ne solut, jotka ovat metsäkanalinnuille tärkeitä, esim. puustoiset metsät. Alapuolella olevassa esimerkissä haetaan solut, joiden arvo on vähintään 20. Annetaan näille soluille arvo 1 ja muille NODATA arvo, lyhyesti: Con(rasteri1, 1, ’NODATA’, ’Value >= 20’) (kuvio 2).

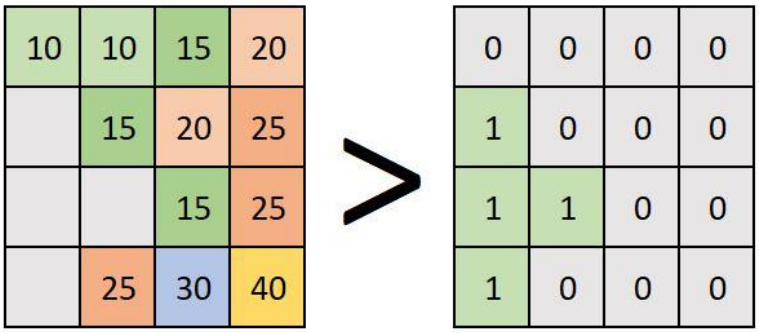
Yllä olevassa esimerkissä Con-työkalu saa ensimmäiseksi argumentiksi rasterin. Seuraava argumentti on arvo, kun ehto täyttyy. Tässä tapauksessa, solut joiden arvo on vähintään 20, saavat arvoksi 1. Seuraava argumentti on arvo, kun ehto ei täyty, eli solut, joiden arvo on alle 20, saavat arvon NODATA. Viimeinen argumentti on ehto, eli solun arvo pitää olla vähintään 20.



KUVIO 2. Esimerkki rasterista ennen Con työkalua ja jälkeen

Projektissa oli tarpeellista yhdistää kahta eri aineistoa, joista muodostuu rasteri. Toinen näistä aineistoista sisältää tuoreempaa tietoa, jonka vuoksi se aineisto priorisoidaan. Tämä

toteutetttiin IsNull-työkalun avulla, joka toimii samalla tavalla kuin Con-työkalu, mutta ehtoa ei voi muuttaa. IsNull-työkalun avulla haettiin ne solut, joista ei ole tuoretta tietoa ja niihin soluihin sovellettiin vanhempaa aineistoa. IsNull-työkalu hakee kaikki solut, joiden arvo on NODATA ja antaa niille arvon 1. Muut solut saavat arvon 0. Kuvio 3 lyhyesti: IsNull(rasteri1).
IsNull saa vain yhden argumentin, rasterin.



KUVIO 3. Esimerkki rasterista ennen IsNull työkalua ja jälkeen

Riistakolmioaineiston havaintopisteet ovat harvassa, minkä vuoksi aineisto piti interpoloida, jotta aineiston saadaan kattamaan koko maan.

Kriging on geostatistinen interpolointimenetelmä, joka perustuu tilastolliseen mallintamiseen ja jota käytetään spatiaalisesti jatkuvien ilmiöiden mittausten analysointiin. Geostatistisessa interpoloinnissa pyritään ymmärtämään pisteiden arvoihin liittyvää säännönmukaisuutta, jota pyritään käyttämään pisteiden ulkopuolella olevien arvojen ennustamiseen. Jotta ennustaminen on mahdollista, tulee alueellinen muuttuja purkaa kolmeen rakennekomponenttiin, jotka ovat yleinen trendi, spatiaalinen autokorrelaatio ja spatiaalisesti riippumaton satunnaisvaihtelu.

Yleisessä trendissä alueellinen systemaattinen vaihtelu on tutkittavaa ilmiötä laajempi. Spatiaalisesti autokorreloituneen aineiston osa on se, jota geostatistisilla menetelmillä voidaan mallintaa. Tämä tarkoittaa yksinkertaistettuna, lähekkäiset kohteet ovat samantapaisia toistensa kanssa, ja kohteiden samantapaisuus laskee etäisyyden kasvaessa. Spatiaalisesti riippumaton satunnaisvaihtelu tarkoittaa mallintamatonta osaa aineistosta, josta on poistettu trendi ja mallinta.

Geostatistisen mallinnuksen tavoitteena on määritellä malli, joka parhaiten kuvaa havaittuja arvoja käyttäen kolmea rakennekomponenttia. Geostastistinen interpolointi edellyttää, että aineisto edustaa spatiaalisesti jatkuvaa ilmiötä ja on autokorreloitunut (Antikainen, Määttä-Juntunen & Ujanen 2015, 75–76.)

3.3.5 ArcMap asetukset

Ensimmäinen asia, mitä pitää tehdä, jotta skripti toimii, on säätää asetukset oikein.

```
with open('{0}\\attribuutit.txt'.format(osoite), 'r') as tied:
    attriLista = tied.read().split(';;')
paaOsoite = attriLista[1]
valiOsoite = '{0}\\vali\\'.format(paaOsoite)
env.scratchWorkspace = env.workspace = valiOsoite
CheckOutExtension('Spatial')
env.overwriteOutput = True
env.snapRaster = env.extent = attriLista[4]
if not os.path.exists(valiOsoite): os.makedirs(valiOsoite)
```

KUVA 1. ArcMapin asetukset

Kuvassa 1 ensimmäisellä kahdella rivillä luetaan asetukset, jotka määritetään FME:ssä. Näitä ovat mm. tallennuskansio, koordinaatiston sijainti ja mitä rasteria käytetään ankkurirasterina. Ankkurirasterin avulla kaikkien uusien rasterien solut kohdistetaan ankkurirasterin kanssa. Tämä tarkoittaa, että rasterien solut alkavat samasta koordinaatista. With open -komennolla määritellään, mitä tiedostoa ollaan avaamassa ja mitä toimintoja sille ollaan tekemässä. Kuvan 1 esimerkissä tiedostoa vain luetaan, minkä vuoksi open komennon viimeisenä argumenttina on ’r’, joka tulee sanasta read. Kun tiedosto on valittu, tehdään siitä muuttuja ja annetaan sille nimi, joka on tässä tapauksessa tied. Seuraava rivi on sisennetty, mikä tarkoittaa, että kyseinen rivi on osa edellistä komentoa. Tällä rivillä määritellään muuttuja attriLista, jolle annetaan tied-muuttujan sisältö, eli asetustiedoston sisältö. Tämä sisältö pilkotaan split()-komennolla, jolloin attriLista-muuttujasta tulee listamuuttuja. Tämä mahdollistaa sen, että asetuksia on helppo kutsua silloin kun niitä tarvitaan.

Lista on kokoelma arvoja, merkkejä ja/tai muuttujia, joita kutsutaan alkioksi. Lista voi koostua muista listoista ja listan tunnistaa (), [] tai {} sulkeista. Listan arvoihin pääsee

käsiksi kutsumalla listaa ja antamalla sille indeksinumeron, esim. lista[0]. Indeksinumero alkaa nollasta. Listan avulla tietoa on helppo tallentaa ja käsitellä ilman että tarvitsee luoda jokaisesta listan alkiosta omaa muuttujaa.

```
ihmisLinjatLista = (('22311', '14111'), 'in_memory\\kaikkiviivat', 'viivarcl.tif')
```

KUVAN 2. Esimerkki Python-listasta

Kuvan 2 listan ensimmäinen muuttuja on toinen lista, toinen ja kolmas muuttujat ovat merkkijonoja. Kuvan 2 listan ensimmäisen muuttujaan pääsee käsiksi komennolla ihmisLinjatLista[0], joka palauttaa arvon ('22311', '14111'). Viimeiseen muuttujaan pääsee käsiksi komennolla ihmisLinjatLista[2] tai vaihtoehtoisesti ihmisLinjatLista[-1], joka palauttaa arvon 'viivarcl.tif'. Ensimmäinen tapa, ihmisLinjatLista[2], edellyttää, että tiedetään, kuinka pitkä lista on kyseessä, kun taas jälkimmäisessä tavassa, ihmisLinjatLista[-1], listan pituutta ei tarvitse tietää. Toiseksi viimeiseen alkioon pääsee käsiksi samalla periaatteella, eli ihmisLinjatLista[-2].

Seuraavaksi Arcmapin asetuksisssa (kuva 1) määritetään tallennuskansio, paaOsoite, joka luetaan attriLista muuttujasta kutsumalla kyseinen muuttuja ja antamalla sille indeksinumero, joka hakee oikean asetuksen. Seuraavalla kahdella rivillä määritetään väliaikaisten tiedostojen sijainti. Määrityksellä env.scratchWorkspace() ilmaistaan ArcMapin oma väliaikaisten tiedostojen sijainti, jossa tiedostot poistetaan, kun niitä ei enää käytetä. EnvWorkspace() on kansio, jonne tiedostot tallennetaan ja jossa ne pysyvät, ellei niille määritetä prosessia niiden poistamiseksi.

Tämän jälkeen haetaan Spatial-lisenssiä CheckOutExtension()-komennolla, jotta jokaista työkalua voidaan käyttää. Seuraavalla rivillä annetaan ArcMapille lupa tallennettaessa korvata olemassa olevia tiedostoja. Seuraavaksi määritetään ankkurirasteri. Lopuksi tarkistetaan, löytyykö tallennuskansiota os.path.exist()-komennolla ja tarvittaessa luodaan se os.makedirs() komennolla.

3.3.6 Zonation

Zoantion on ohjelma, jolla voidaan suunnitella elinympäristöjen turvaamista laajemmassa mittakaavassa. Ohjelma tunnistaa alueita, jotka ovat tärkeitä turvaamaan elinympäristöjä

ja kytkee niitä muihin ympäristötekijöihin, kuten kasvupaikka, muu lajisto tai ihmisen rakennelmat. (Moilanen 2014, 10.)

Zonation hyödyntää rasteriaineistoja, joiden perusteella lasketaan uusi rasteriaineisto. Jokainen solu saa arvon, josta käy ilmi, kuinka hyvin kyseinen alue sopii lajistolle. Laskennassa on huomioitu kaikki piirrekerrokset ja miten ne ovat kytkeytyneet toisiinsa. Piirrekerroksilla tarkoitetaan rastereita, joita käytetään Zonation-analyysissa. Zonationin avulla saadaan kattava kuva, missä lajille sopivat elinympäristöt sijaitsevat. Periaatteena on, ettei yksittäinen riistahila ole niin merkittävä kuin useamman hilan keskittymä tai laajempi hyvien kohteiden verkosto. (Miettinen & Kesälä 2018, 50–51.)

4 PIIRREKERROSTEN PROSESSI

4.1 Piirrekerrokset 1-5 / MVMI ja laserkeilausaineisto

Piirrekerrokset ovat rastereita, joita käytetään Zonation-analyysissä. Piirrekerrokset kuvaavat elinympäristöjä ja niihin vaikuttavia tekijöitä, kuten puustoisuus, vesistöt ja rakennukset (Moilanen 2014, 19).

Pythonissa funktio määritellään def-komennolla, jonka jälkeen tulee funktion nimi ja sulut. Sulkujen sisäpuolelle tulee funktion tarvitsemat argumentit. Funktion komennot on sisennetty sarkaimella tai neljällä välilyönnillä. Kuvassa 3 nähdään, että funktion nimi on VMItoimenpide ja funktiolla on kaksi argumenttia, tiedosto ja niLista. VMI-toimenpidefunktiolla on viisi riviä komentoja, jotka funktio suorittaa rivi riviltä.

VMItoimenpidefunktiota käytetään monilähteisen valtakunnan metsien maastoinventointiaineiston työstämisessä. Investointiaineistosta käytetään maastotietojen lisäksi mm. satelliittikuvia, korkeusmalleja ja numeerisia peruskarttoja. Näiden avulla voidaan käyttää lähimmän naapurin interpolaatiota, jossa yleistetään koealueilta mitatut tiedot koealojen ulkopuolelle (Metla n.d.). Kyseisestä aineistosta hyödynnetään kokonaistilavuutta, latvuspeittävyyttä sekä koivun, kuusen, männyn ja muun lehtipuun tilavuuksia.

```
def VMItoimenpide(tiedosto, niLista):
    VMInimi = tiedosto.split('\\')[-1][:6] + '_con.tif'
    rasTied = Raster(tiedosto)
    aggre = Aggregate(rasTied, 4, 'MEAN')
    Con(aggre, aggre, '', 'Value > 0 AND Value < 1000').save(VMInimi)
    niLista.append(VMInimi)
```

KUVA 3. Funktio monilähteiselle valtakunnan metsien inventointiaineistolle

Funktion argumentit ovat ”tiedosto”, joka on käsiteltävän tiedoston sijainti ja ”niLista”, joka on lista ja johon lisätään funktion lopuksi uuden tiedoston nimi.

Funktion ensimmäisellä rivillä määritetään tulevan tiedoston nimi pilkkomalla tiedoston osoite ”\\” merkkien kohdalta, josta muodostuu lista. Tämän jälkeen valitaan listan viimeinen alkio, josta otetaan kuusi ensimmäistä merkkiä ja lisätään loppuun ”_con.tif”.

Seuraavalla rivillä luetaan tiedosto ja kerrotaan arcpy:lle, että kyseessä on rasteritiedosto Raster()-komennon avulla, jonka jälkeen tiedostolle suoritetaan Aggregate()-toiminto, jossa tulos on 4 solun keskiarvo, jolloin solun koko nelinkertaistuu. Ilman Raster() komentoa saataisiin virheilmoitus, että tiedostolle ei voida suorittaa Aggregate() toimintoa, koska tiedosto ei ole rasteritiedosto. Aggregate()-toiminto saa ensimmäiseksi argumentiksi käsiteltävän tiedoston sijainnin, toiseksi solukertoimen, eli tässä tapauksessa arcon neljä, ja viimeinen argumentti on aggregaatiotyyppi, joka on tässä tapauksessa ”MEAN”, eli keskiarvo. Tämän jälkeen suoritetaan Con()-toiminto Aggregate-tiedostolle, jossa valitaan kaikki ne solut, joiden arvo on suurempi kuin 0 ja pienempi kuin 1000. Con() saa ensimmäiseksi argumentiksi käsiteltävän tiedoston sijainnin, toiseksi arvon/rasterin, kun ehto täyttyy, joka tässä tapauksessa on lähdetiedosto. Kolmas argumentti on arvo/rasteri, kun ehto ei täyty, eli tässä tapauksessa ’’, joka on tyhjä solu ja viimeinen argumentti on ehto. Saatu tiedosto tallennetaan kovalevylle ja lisätään tiedoston nimi listalle myöhempää käyttöä varten. Lopputulokseksi saadaan 6 rasteria.

lk_mvmi-funktiossa käsitellään hilaruudun tietoja, jotka on kerätty laserkeilauksella ja johon yhdistetään VMItoimenpidefunktion tiedostot.

```
def lk_mvmi(lista, mvmi, hilaMaaraNimi = ''):
    nimi = lista[3][:4] + '.tif'
    arcpy.FeatureToRaster_conversion(lista[0], lista[3], nimi, 16)
    aggre = Aggregate(nimi, 4, 'MEAN')
    if hilaMaaraNimi:
        coni = Con(nimi, 1, '', 'Value > 0')
        Aggregate(coni, 4, 'SUM').save(hilaMaaraNimi)
    coni = Con(IsNull(aggre) == 1, mvmi, aggre)
    if arcpy.GetRasterProperties_management(coni, 'VALUETYPE') != lista[2]:
        nimi = 'coni_kopio.tif'
        arcpy.CopyRaster_management(coni, nimi, pixel_type = lista[2])
    else:
        nimi = coni
    Con(nimi, nimi, '', 'Value > 0').save(lista[1])
```

KUVA 4. Muutetaan laserkeilausaineisto rasteriksi ja lisätään MVMI rasteri

Funktio saa ensimmäiseksi argumentiksi ”lista”, joka sisältää hilatiedoston sijainnin, uuden tiedoston nimen, pikselisyvyyden ja käsiteltävän kentän, esim. tilavuus. Tämän lisäksi funktio saa argumentiksi ”mvmi”, joka on VMItoimenpide funktion vastaavan tiedoston sijainti ja ”hilaMaaraNimi”, joka on valinnaisen tiedoston sijainti.

Ensimmäisellä rivillä määritellään uuden väliaikaisen tiedoston nimi ja rasterityyppi, joka on TIFF-rasteri. TIFF-formaattia käytetään, koska Zonation suosii tätä formaattia. Zonationissa TIFF-formaatin vaihtoehtona olisi grid-formaatti, mutta tämän formaatin käyttäminen voi täyttää kovalevyn ja Zonation laskenta hidastuu merkittävästi (Moilanen 2014, 92).

Toisella rivillä muutetaan hilaruutuaineisto 16 x 16 metrin rasteriksi FeatureToRaster_conversion()-toiminnolla. FeatureToRaster_conversion()-toiminto saa ensimmäiseksi argumentiksi käsiteltävän tiedoston sijainnin, toiseksi uuden tiedoston pikselisyvyyden, kolmanneksi uuden tiedoston nimen ja viimeiseksi solunkoon. Seuraavalla rivillä saadun tiedoston solukoko suurennetaan 64 metriin Aggregate()-toiminnolla, jolle annetaan kertoimeksi 4 ja tyypiksi ”MEAN”. Tätä tiedostoa ei tallenneta kovalevylle, mutta se pysyy tietokoneen muistissa funktion aikana. Aggregate-toiminnolla syntyneen tiedoston muuttujan nimi on ”aggre”.

Neljännellä rivillä tarkistetaan, annettiinko hilaMaaraNimi-argumenttia. Jos annettiin, suoritetaan annetulle tiedostolle Con()-toiminto, jolloin kaikki solut, joiden arvo on suurempi kuin 0, saavat arvoksi 1. Tämän jälkeen tiedostoa työstetään lisää Aggregate() toiminnolla, joka saa kertoimeksi 4 ja tyypiksi ”SUM”, eli summan. Tulokseksi saadaan rasteri, jonka solujen arvot ovat 1–16. Tiedosto tallennetaan yhtenä piirrekerroksena. Tämä prosessi suoritetaan vain kerran ja sillä ei ole merkitystä, mille hilatiedostolle se suoritetaan. Tässä skriptissä käytetään tilavuuden hilatiedostoa. hilaMaara-rasteri kertoo, kuinka monta hilaa 64 m solussa on. Suurin arvo on 16, koska 64 m x 64 m soluun mahtuu 16 kappaletta 16 m x 16 m soluja ja pienin arvo 1, koska tyhjiä soluja ei käytetä.

Seuraavalla rivillä yhdistetään aikaisemmin luotu aggre muuttujan tiedosto ja ”mvmi” tiedosto Con() ja isNull() -toiminnoilla. isNull()-toiminto muuttaa tyhjät arvot tai ”null” arvot, ykköseksi (1). Tässä tapauksessa isNull()-toiminnolla muutetaan aggre-tiedoston tyhjät solut arvoksi 1, jonka jälkeen haetaan Con()-toiminnolla kaikki solut, joiden arvo on 1. Tällä tavalla saadaan haettua kaikki aggre-tiedoston tyhjät solut. Tämän jälkeen

solut, jotka saivat arvoksi 1, korvataan ”mvmi”-tiedoston solun arvoilla ja loput solut pitävät aggre-tiedoston solun arvon. Tämä prosessi yhdistää ”mvmi”-tiedoston ja aggre-tiedoston solujen arvot priorisoiden aggre-tiedoston solujen arvot. Toisin sanoen, jos aggre-solulla ei ole arvoa, saa kyseinen solu mvmi-tiedoston solun arvon.

Seuraavaksi tarkastetaan, mikä on rasterin pikselin syvyys getRasterProperties()-toiminnolla. Jos äskettäin luodulla coni-muuttujan pikselin syvyys on eri kuin mitä ”lista”-argumentissa, suoritetaan seuraava komento. Ensiksi määritellään väliaikaisen tiedoston nimi, jonka jälkeen kopiodaan coni-tiedosto uudeksi tiedostoksi arcpy.CopyRaster_management()-toiminnolla. Ensimmäinen argumentti on äskettäin luotu coni-tiedosto ja toinen on äskettäin määritelty tiedoston nimi ja lopuksi määritellään pikselin syvyys, joka saadaan ”lista”-argumentista. Tällä tavalla pidetään tiedostojen koot pienenä, mikä nopeuttaa tiedoston lukua ja kirjoitusta. Tämä pitää huolen siitä, että tiedostot pysyvät muutamissa megatavuissa satojen megatavujen sijaan. Jos pikselisyvyys on sama kuin ”lista”-argumentissa, tallennetaan nimi-muuttujaan coni-muuttujan arvo.

Viimeisellä rivillä suoritetaan Con()-toiminto, joka suoritetaan nimi-muuttujalle, joka on sama kuin coni-muuttujan tiedosto oikealla pikselisyvyydellä. Coni-muuttujan tiedostosta valitaan kaikki solut, joiden arvo on suurempi kuin 0. Lopuksi tiedosto tallennetaan piirrekerrokseksi ja lopputulokseksi saadaan rasteri, jossa ei ole tyhjiä soluja ja jossa on yhdistetty laserkeilauksen puustotieto ja MVMI puustotieto.

Tiheys-funktiossa käsitellään tiheys-tiedoston laserkeilausaineistoa. Tiheysaineisto kertoo, kuinka paljon alueella on alakaikuja, eli kuinka paljon kaikuja pääsi latvuston läpi. Tässä hankkeessa alakaiut olivat 0,5–5 metrin korkeudella. Vertaamalla tiheystietoa latvuspeittävyyteen voidaan päätellä, kuinka kerroksellista puuston on. Kyseinen funktio

tekee saman kuin lk_mvmi-funktio, mutta se ei yhdistä laserkeilausaineistoa ja MVMI-aineistoa yhdeksi rasteriksi, koska MVMI-aineistossa ei ole tiheysrasteria.

```
def tiheys(tasLis):
    arcpy.PolygonToRaster_conversion(
            tasLis[0], tasLis[3], 'tiheys.tif', '', '', 16
            )
    aggre = Aggregate('tiheys.tif', 4, 'MEAN')
    arcpy.CopyRaster_management(
            aggre, 'tih_ras.tif', pixel_type = tasLis[2]
            )
    Con('tih_ras.tif', 'tih_ras.tif', '', 'Value > 0').save(tasLis[1])
```

KUVA 5. Tiheys-tiedoston laserkeilausaineiston rasterointi vaatii oman funktionsa

Kuvassa 5 ensimmäisenä toimintona muutetaan polygoni rasteriksi PolygonToRaster_conversion()-toiminnolla. Toiminto saa ensimmäiseksi argumentiksi polygon tiedoston sijainnin ja seuraavaksi argumentiksi, mitä kenttää käytetään rasterin solujen arvon määrittämiseen. Kolmas argumentti on uuden rasterin nimi. Seuraavat kaksi argumenttia ovat solun arvon määrittäminen, jos soluun mahtuu useampi polygoni, ja seuraava argumentti on, mitä kenttää käytetään, jos solulla on useampi polygoni, jolla on sama arvo. Tässä tapauksessa nämä kaksi argumenttia on jätetty tyhjiksi. Viimeinen argumentti on solun koko, joka on 16 metriä.

Tämän jälkeen saatuun rasteriin suoritetaan Aggregate()-toiminto samalla tavalla kuin lk_mvmi-funktiossa, ja aivan kuten edellisessä funktiossa, tässä pidetään huoli, että rasterilla on järkevä pikselisyvyys arcpy.CopyRaster_management()-toiminnolla. Seuraavaksi suoritetaan rasterille Con()-toiminto, josta valitaan kaikki solut, joiden arvo on suurempi kuin nolla, minkä jälkeen rasteri tallennetaan yhdeksi piirrekerrokseksi.

```
maksi = (
        1 - (CellStatistics([manty, muulp, kuusi, koivu], 'MAXIMUM', 'DATA') / tilavuus)
        ) / (1 - (1/puulajienmaara))
```

KUVA 6. Puulajien suhteellisen osuuden laskeminen

Valitaan männyn, koivun, kuusen ja muiden lehtipuulajien tilavuusrastereista suurin arvo solulle CellStatistics()-työkalun avulla (kuva 6). CellStatistics()-toiminto saa ensimmäiseksi argumentiksi listan, joka pitää sisällään kaikki ne rasterit, joita verrataan toisiinsa. Seuraava argumentti on mitä haetaan, joka tässä tapauksessa on suurin arvo, ja

lopuksi määritetään, huomioidaanko tyhjät solut vai vain solut, joilla on arvo. Tässä tapauksessa huomioidaan vain solut, joilla on arvo. Lopputulokseksi saadaan rasteri, johon on haettu suurin arvo kaikista puulajeista.

4.2 Piirrekerrokset 7 & 8 / Metsäkanalintuaineisto

Funktio GDBTarkistus saa argumenteiksi kansion, jossa geodatabase sijaitsee tai jonne se tallennetaan ja geodatabasen nimen, tosin tämä on valinnainen argumentti. GDBTarkistus-funktiolla tarkistetaan, löytyykö haettu geodatabase. Jos haettu geodatabase löytyy, funktio palauttaa sen sijainnin. Jos ei löydy, kuvan 7 funktio luo geodatabasen annettuun sijaintiin annetulla nimellä, minkä jälkeen funktio palauttaa geodatabasen sijainnin. Tällä pidetään huolta siitä, että feature class -tiedostot tallennetaan aina geodatabaseen.

```
def GDBTarkistus(vaPo, geDB = '\\vali.gdb'):
    paikka = vaPo + geDB
    if not arcpy.Exists(paikka):
        arcpy.CreateFileGDB_management(vaPo[:-1], geDB)
    return paikka + '\\'
```

KUVA 7. Tarkistetaan löytyykö ja tarvittaessa luodaan haluttu geodatabase.

Funktio kanaAinesto muuttaa riistakolmion Excel tiedoston feature classiksi jokaisen neljän kanalinnun kohdalla ja lopuksi yhdistää metson, teeren ja pyyn tiedot yhdeksi tiedostoksi, jolloin saadaan metepy-tiedosto ja riekko-tiedosto. Tämä funktio palauttaa listan, ja funktiota käytetään muuttujan arvon määrittämisessä.

```
def kanaAineisto(valiPo, riistaOsoite, koordit, nimi1, nimi2):
    riista = GDBTarkistus(valiPo) + '\\riistaPisteet'
    valiNimi = '\\vali.gdb\\valiriista'
    kanaLista = (nimi1.split('\\')[-1][:-4], nimi2.split('\\')[-1][:-4])
    arcpy.MakeXYEventLayer_management(
            riistaOsoite, 'kolmio_keskipiste_x', 'kolmio_keskipiste_y',
            valiNimi, koordit
            )
    arcpy.FeatureToPoint_management(valiNimi, riista)
    arcpy.MakeFeatureLayer_management(riista, 'kana_taso')
    kanaNimiLista = Manager().list()
    for k in kanaLista:
        kanaGDB = GDBTarkistus(valiPo, '\\{0}.gdb'.format(k))
        ilmaisu = 'laji_txt = \'{0}\''.format(k)
        if k == kanaLista[0]:
            ilmaisu = 'laji_txt = \'{0}\' OR laji_txt = \'{1}\' OR laji_txt = \'{2}\''
                    .format('metso', 'teeri', 'pyy')
        if arcpy.Exists('{0}{1}_86hakutj'.format(kanaGDB, k)):
            arcpy.Delete_management('{0}{1}_86hakutj'.format(kanaGDB, k))
        uusiNimi = '{0}{1}_86hakutj'.format(kanaGDB, k)
        arcpy.SelectLayerByAttribute_management('kana_taso', 'NEW_SELECTION', ilmaisu)
        arcpy.Dissolve_management(
                    'kana_taso', uusiNimi + 'dissolve', 'knum', [
                        ['n', 'SUM'], ['kolmio_keskipiste_x', 'FIRST'],
                        ['kolmio_keskipiste_y', 'FIRST']
                        ],
                    'SINGLE_PART')
        kanaNimiLista.append(uusiNimi + 'dissolve')
    return kanaNimiLista
```

KUVA 8. Muutetaan metsäkanalintu Excel-tiedosto Feature Classiksi

Funktio kanaAineisto saa argumenteiksi väliaikaisen kansion osoitteen, riistakolmiotiedoston sijainnin, koordinaattitiedoston sijainnin ja kaksi tiedoston nimeä luotaville tiedostoille. Ensimmäiseksi tarkistetaan/luodaan tuleville aineistoille geodatabase-tietokanta. Tämän jälkeen määritellään väliaikaisen tiedoston nimi ja heti perään luodaan lista, joka pitää sisällään molemmat uudet tiedostojen nimet. Seuraavaksi luodaan riistakolmiotiedostosta feature class -tiedosto MakeXYEventLayer_management-työkalun avulla. Työkalu saa argumentiksi riistakolmiotiedoston, mitä attribuuttia käytetään x- ja y-koordinaatin määrittämisessä, luotavan tiedoston nimi ja mitä tasokoordinaattijärjestelmää, joka Suomessa on nimeltään ETRS-TM35FIN. ETRS on koordinaattijärjestelmän nimi. TM on karttaprojektio. 35 kertoo kaistanumero ja FIN kertoo, että kyseessä on suomalainen sovellus UTM standardissa (ETRS89). Tämän jälkeen luodusta tiedostosta tehdään pistetiedosto FeatureToPoint_management-toiminnolla, jonka tuotos muutetaan karttatasoksi MakeFeatureLayer_management-työkalun avulla. Attribuuttihakuja voidaan tehdä

vain karttatasoille, minkä vuoksi tiedosto piti muuttaa karttatasoksi. Tämän jälkeen luodaan rinnakkaisprosessiin soveltuva lista, kanaNimiLista, jotta luettavaa/käsiteltävää tiedostoa ei lukita ja aiheuteta virhetilannetta. Tämän jälkeen iteroidaan alussa luotu kanaLista läpi. Ensimmäiseksi tarkistetaan/luodaan uusi geodatabase käsiteltävälle kanalinnulle. Tämä luodaan, jotta tiedosto ei mene lukkoon ja aiheuta virhetilannetta rinnakkaisprosessissa. Tämän jälkeen luodaan muuttuja, joka sisältää ominaisuushakukomennon, mitä kanalintulajia haetaan. Seuraavaksi on ehtolause, joka tarkistaa, onko kyseessä kanaLista-muuttujan ensimmäinen alkio, jolloin hakulauseke muuttuu. Uusi hakulauseke hakee metson, teeren ja pyyn tiedot. Voi vaikuttaa oudolta, että luodaan hakulauseke ja korvataan se toisella heti perään. On muistettava, että jokainen iteraatio tämän jälkeen ei ole kanaListan ensimmäinen alkio ja näin ollen hakulauseke ei muutu ensimmäisen määrittelyn jälkeen. Tämä on vain yksi tapa monista. Tämän jälkeen tarkistetaan ja poistetaan vanha tiedosto, jos sellainen löytyy. Tämä tehdään virhetilanteiden välttämiseksi. Seuraavaksi määritetään luotavan tiedoston nimi, joka on sama kuin juuri poistettu. Tämän jälkeen suoritetaan ominaisuus haku SelectLayerByAttribute_management, joka saa argumenteiksi aiemmin luodun karttatason, ’NEW_SELECTION’ komennon, eli uusi haku ja lopuksi ilmaisu-muuttujan sisältämän hakukomennon.

Tässä vaiheessa haetaan metsäkanalintujen tiedot. Ensimmäisellä iteraatiolla haetaan metson, teeren ja pyyn tiedot ja toisella iteraatiolle riekon tiedot. Seuraavaksi suoritetaan Dissolve_management-komento. Dissolve-toiminto saa ensimmäiseksi argumentiksi käsiteltävän tiedoston ja toiseksi argumentiksi uuden tiedoston nimen. Seuraava argumentti on kentät, jotka yhdistetään, ja niiden yhdistämisen tapa. Tässä tapauksessa yhdistetään havaintojen lukumäärä summametodilla, havaintoalueen keskipisteen x- ja y-koordinaatti, joista valitaan ensimmäinen vaihtoehto. Viimeiseksi argumentiksi annetaan ’SINGLE_PART’-arvo, jolloin jokainen havaintoalue on yksittäinen alue yhden kokonaisuuden sijaan. Tämän jälkeen luodun tiedoston tiedostopolku lisätään kanaNimiLista

-muuttujaan. Kun iteraatio on suoritettu loppuun, palautetaan kanaNimiLista-muuttujan sisältö funktion kutsujalle myöhempää käyttöä varten.

Funktio krigiFunk suorittaa kriging-toimenpiteen kanaAineisto-funktiosta saatuun aineistoon, muuttaa pikselien syvyyden 8-bittiseksi ja poistaa aineistosta Suomen ulkopuolelle menevän osan (kuva 9).

```
def krigiFunk(tiedosto, kanaNimi, valiPo, talPo,  pohAi, suomi, etaisyys_m = 50000):
    kriNimi = '{0}\\{1}.tif'.format(talPo, kanaNimi)
    conimi= 'con{0}'.format(kanaNimi)
    krigi = Kriging(tiedosto, 'SUM_n', KrigingModelOrdinary(
                    'GAUSSIAN', '', etaisyys_m, 9, 2
                    ),
                    64, RadiusVariable(12, etaisyys_m))
    sleep(3)
    arcpy.CopyRaster_management(krigi, conimi, pixel_type = '8_BIT_UNSIGNED')
    extrac = ExtractByMask(conimi, suomi)
    Con(extrac > 0, extrac).save(kriNimi)
    pohAi.append(kriNimi)
```

KUVA 9. Muutetaan metsäkanalintu Feature Class -tiedosto rasteriksi ja yleistetään Kriging-toiminnolla

KrigiFunk-funktio saa argumenteiksi kana-tiedoston, uuden tiedoston nimen, pääkansion ja väliaikaisen kansion osoitteet, tyhjän listan, Suomen kartan ja etäisyyden. Tämä toiminto suoritetaan kahdesti. Ensimmäiseksi luodaan muuttuja, joka saa arvoksi piirrekerroksen tiedoston nimen ja sijainnin, ja heti perään luodaan toinen muuttuja, joka saa arvoksi väliaikaisen tiedoston nimen. Seuraavaksi käytetään ArcMapin kriging-toimintoa. Krigingissä käytetään SUM_n attribuutti-kenttää. Tämä kertoo, kuinka monta metsäkanalintuhavaintoa kolmiossa on. Seuraavaksi valitaan gaussian-asetus semivariogrammitekniikan malliksi.

Etäisyydeksi annetaan etäisyys-argumentissa oleva arvo, sill saa arvon 9 ja nugget arvon 2. Nugget kuvaa aineiston vaihtelua pienillä etäisyyksillä, sill kuvaa vaihtelua maksimitasolla ja range on pisteparien välinen etäisyys, jolla sill-arvo saavutetaan, ja etäisyys, jossa autokorrelaatio loppuu (GIS-analysointimenetelmät, 75–76). Solun koko on 64. RadiusVariable saa pisteiden määräksi 12 ja etäisyydeksi etäisyys-argumentin arvon. Tämä määrittelee, mitä pisteitä käytetään interpoloinnissa (Kriging). Tämän jälkeen odotetaan 3 sekuntia. Kriging-toiminto tuotti tässä vaiheessa ongelmia, jotka ratkesivat lisäämällä pienen viiveen toiminnon perään. Seuraavaksi kopioidaan kriging-toiminnosta saatu tiedosto ja määritellään sille uusi pikselisyvyys, joka on 8 bittiä. Tämän jälkeen kopiosta

leikataan Suomen rajat ylittävät osiot ja heti perään poistetaan nolla-arvot, jolloin saadaan piirrekerrostiedosto. Lopuksi tallennetaan piirrekerroksen tiedostopolkulistaan. Tämä toiminto ei tee mitään tällä hetkellä, mutta jos kriging-toiminnot suoritettaisiin eri prosessin ytimissä ja samanaikaisesti, tämä lista pitäisi huolen, että tiedostoa ei lukita toisiltaan ja aiheuteta virhetilannetta.

4.3 Piirrekerrokset 6, 9 & 10 / Maastotietokanta & suoaineisto

Haettaessa maastotietokannasta työstettävät tiedostot, kuten esim. vesistöt käytetään hakufunktiota. Tätä tarkoitusta varten luodaan pohjaLista-funktio (Kuva 10).

```
def pohjaLista(lukuOsoite, haLista):
    omaLista = []
    tiedOso = arcpy.da.Walk(lukuOsoite)
    for dirpath, dirnames, filenames in tiedOso:
        for filename in filenames:
            tied = filename.lower()
            for haku in haLista:
                if haku in tied and 'bm' not in tied and 'lehti' not in tied:
                    omaLista.append(os.path.join(dirpath, filename))
    return sorted(omaLista)
```

KUVA 10. Haetaan syötetyt tiedostot kansiosta.

Funktio saa kaksi argumenttia: mistä kansiosta haetaan tiedostot ja mitä tiedostoja haetaan. Ensimmäiseksi luodaan tyhjä lista, johon lisätään löydetyt tiedostot. Tämän jälkeen käytetään ArcMapin walk -komentoa ja luodaan muuttuja, joka pitää sisällään kansion rakenteen. Seuraavaksi käydään jokainen kansiossa sijaitseva tiedosto läpi käyttäen kahta for komentoa. Ensimmäinen for käy kansiot läpi ja jälkimmäinen tiedostot. Jokainen kansiossa oleva tiedoston nimi muutetaan pieniksi kirjaimiksi, minkä jälkeen tiedoston nimeä vertaillaan haettavan listaan, joka annettiin argumentissa. Tämän lisäksi tarkistetaan, ettei nimi sisällä ”bm” tai ”lehti” -merkkijonoa, koska kyseessä on tällöin eri tiedosto, jolla on lähes sama nimi kuin haettavalla tiedostolla. Kun nämä ehdot täyttyvät, lisätään löydetty tiedoston polku alussa luotuun listaan. Lopuksi palautetaan lista funktion kutsujalle.

ArcMapin Merge-toimintoa varten luodaan merge-funktio, jossa yhdistetään maastotietokannan luokka-aineistoa yhdeksi tiedostoksi. Funktio koostuu kolmesta ehtolauseesta, eli haluttaessa funktio ei tee mitään (kuva 11). Tämä tekee funktiosta joustavan, minkä

vuoksi sitä voidaan käyttää, kun työstetään monta tiedostoa tai kun työstetään yhtä tiedostoa.

Prosessissa käsitellään useita maastotietokannan luokkia. Luokkien käsittely veisi aikaa, jos näitä käsiteltäisiin yhtenä kokonaisuutena. Tämän vuoksi käsitellyistä luokista erotelaan neljä suurinta luokkaa, jotka käsitellään erikseen ja jotka vasta lopuksi yhdistetään muihin luokkiin. Tämä nopeuttaa luokkien käsittelyä huomattavasti. Merge-funktio on rakennettu käsittelemään niin tiedostoja kuin tiedostoa. Tällä tavalla ei kirjoiteta koodia kahteen kertaan, vaan hyödynnetään olemassa olevaa koodia.

```
def merge(mergeLista, mergeNimi, rasteriNimi = '', rclNimi = '', maastoTietoKanta = ''):
    if maastoTietoKanta:
        mergeLista = pohjaLista(maastoTietoKanta, mergeLista)
    if type(mergeLista) in (list, tuple, set):
        arcpy.Merge_management(mergeLista, mergeNimi)
        mergeLista = mergeNimi
    if rasteriNimi:
        arcpy.PolygonToRaster_conversion(
                mergeLista, 'LUOKKA', rasteriNimi, 'MAXIMUM_COMBINED_AREA', '', 64)
        Reclassify(
            rasteriNimi, 'Value', RemapRange([[1,99999, 1]]), 'NODATA').save(rclNimi)
```

KUVA 11. Yhdistetään ja rasteroidaan maastotietokannan Feature Classeja

Merge funktio saa argumenteiksi, mitkä tiedostot yhdistetään, saadun tiedoston nimen, kaksi rasterin nimeä ja maastotietokannan sijainnin, josta tiedostot haetaan. Ensimmäiseksi tarkistetaan, onko maastotietokanta-argumenttia annettu. Jos argumentti on annettu, niin haetaan tiedostot käyttäen pohjaLista-funktiota. Tässä päivitetään mergeLista-argumentti ja tehdään siitä funktion sisäinen muuttuja, koska mergeLista-muuttujaa käytetään useammassa vaiheessa, jotka eivät ole riippuvaisia toisistaan. Tällä tavalla funktion toimintoja voidaan käyttää erillään toisistaan, jolloin funktio on joustava. Tämän jälkeen tarkistetaan, onko mergeLista-argumentti lista vai merkkijono. Jos kyseessä on merkkijono, argumentiksi annetaan yksittäinen tiedosto ja siirrytään seuraavaan kohtaan. Jos kyseessä on lista, suoritetaan ArcMapin Merge.management-komento. Komennolle annetaan argumentiksi mergeLista, joka pitää sisällään yhdistettävät tiedostot, ja mergeNimi joka annetaan syntyneelle tiedostolle. Seuraavaksi päivitetään mergeLista-muuttuja ja annetaan sille mergeNimi-arvo. Lopuksi tarkistetaan, onko rasteriNimi-argumentille annettu arvo. Jos arvo löytyy, muutetaan mergeLista rasteriksi käyttäen luokka-attribuuttia. Rasterin solukoko on 64 metriä, ja solun arvo saadaan mergeLista-tiedoston luokka-attribuutista. Tämän jälkeen rasterille suoritetaan ArcMapin reclassify-komento ja annetaan

soluille arvo 1, jolloin rasterin tiedoston koko pienenee ja rasterin käsittely nopeutuu. Uusi rasteri saa nimekseen rclNimi-argumentin arvon.

Funktiolla ihmisLinjat on tarkoitus selvittää ihmisten rakentamien verkostojen sijainnit, kuten tiet, joista valitaan ne alueet, jotka vaikuttavat metsäkanalintujen elinympäristöön, (kuva 12.)

```
def IhmisLinjat(pk10lista, tyoTaso):
    ihmisLista = pohjaLista(lukuOsoite = tyoTaso, haLista = pk10lista[0])
    merge(ihmisLista, pk10lista[1])
    viivaStats = LineStatistics(pk10lista[1],'None', 64, 64, 'LENGTH')
    viivaHaku30 = viivaStats >= 30
    arcpy.CopyRaster_management(viivaHaku30, pk10lista[2], pixel_type = '1_BIT')
```

KUVA 12. Funktio, joka käsittelee ihmisten luomia linjoja, esim. teitä

IhmisLinjat-funktio saa kaksi argumenttia. pk10Lista-argumentti on lista, joka sisältää maastotietokannan luokat, väliaikaisen tiedoston nimen ja lopulliset tiedoston nimen. Toinen argumentti on tyoTaso, joka on kansio, josta haetaan tarvittavat tiedostot, tässä tapauksessa kyseinen kansio on maastotietokanta.

Ensimmäinen komentorivi luo listan, joka koostuu tiedostopoluista. Listan sisältö valitaan pk10lista-argumentissa esiintyvien luokkatunnusten perusteella.

Toisella rivillä kutsutaan merge-funktiota, joka kutsuu ArcMapin merge-toimintoa ja yhdistää vektoriaineistot yhdeksi tiedostoksi. Argumentiksi annetaan edellisellä rivillä luotu lista, joka koostuu tiedostopoluista, ja pk10Listan toisen indeksin arvo, joka kertoo, minne uusi tiedosto luodaan ja millä nimellä.

Kolmannella rivillä käytetään ArcMapin komentoa LineStatistics, joka luo vektoriaineistosta rasterin. LineStatisticsille määritetään tiedosto, kenttä, solukoko, hakuetäisyys ja tilastotyyppi. Tässä tapauksessa tiedosto on kakkosrivillä luotu tiedosto, solukoko ja hakuetäisyys ovat molemmat 64 metriä, ja tilastotyyppi on pituus, jonka vuoksi kentäksi voidaan antaa ’None’-arvo. Näillä argumenteilla luodaan rasteri, jonka solujen arvo on vektoriaineiston pituuden summa.

Neljännellä rivillä luodaan väliaikainen tiedosto, johon valitaan äskettäin luodusta rasterista kaikki solut, joissa on vähintään 30 metriä teitä ja muita linjoja. Viidennellä rivillä

kutsutaan ArcMapin komentoa CopyRaster_management, joka saa argumenteiksi kopioitavan tiedoston, uuden tiedoston nimen ja pikselisyvyyden. Tehdään väliaikaisesta tiedostosta pysyvä tiedosto, joka tallennetaan haluttuun paikkaan ja nimellä, eli kuvasta 12 nähdään, että nimi on viivarcl.tif ja tallennuspaikkana on oletuspaikka. Pikselin syvyys on 1 bittiä, koska kiinnostavaa on, onko solulla vähintään 30 metriä teitä vai eikö ole. Kiinnostavaa ei enää tässä vaiheessa ole, kuinka paljon teitä solulla on. 1-bittinen pikselin syvyys saa arvot 0 tai 1. Pikselisyvyydellä on merkitystä, koska se vaikuttaa tiedoston kokoon ja käsittelyaikoihin, mikä pitää huomioida valtakunnallisessa aineiston käsittelyssä.

Kuvassa 13 esitetty ihmiset-funktio yhdistää kaikki ihmisen tekemät avoimet alueet ja rakennelmat yhdeksi rastertiedostoksi.

```
def ihmiset(MTK, ihmisRaLi, listatut, IhmisLinLis, paaPol, zonaNimi):
    rasLista, rclLista =
                    tuple(map(lambda x: 'ras{0}.tif'.format(x), range(5))),
                    tuple(map(lambda x: 'rcl{0}.tif'.format(x), range(5)))
    pk10Lista = pohjaLista(MTK, ihmisRaLi)
    suuret = set([(p) for p in pk10Lista for l in listatut if l in p])
    uusiLista = [set(pk10Lista) - suuret]
    uusiLista.extend(suuret)
    merge(tuple(uusiLista[0]),  '\\vali.gdb\\valiMerge', rasLista[0], rclLista[0])
    merge(uusiLista[1], uusiLista[1], rasLista[1], rclLista[1])
    merge(uusiLista[2], uusiLista[2], rasLista[2], rclLista[2])
    merge(uusiLista[3], uusiLista[3], rasLista[3], rclLista[3])
    merge(uusiLista[4], uusiLista[4], rasLista[4], rclLista[4])
    IhmisLinjat(IhmisLinLis, MTK)
    arcpy.MosaicToNewRaster_management(
            [rclLista, IhmisLinLis[2]], paaPol, zonaNimi[len(paaPol):],
            '', '1_BIT', 64, 1, 'FIRST')
```

KUVA 13. Funktio, jossa käsitellään ihmisten rakennelmat, kuten tiet ja talot.

Ihmiset-funktio saa argumenteiksi maastotietokannan (MTK), ihmisten rakennukset (ihmisRaLi), eritetyt tiedostojen nimet (listatut), ihmisten rakentamat linjat (ihmisLinat), kansion, jonne rasterit tallennetaan (paaPol) ja rasterin nimen (Zonation). Ensimmäiseksi luodaan kaksi listaa, jotka iteroidaan läpi ja täytetään väliaikaisten rasterien nimillä. Seuraavaksi luodaan uusi listamuuttuja käyttämällä pohjaLista-funktiota. Tämän jälkeen luodaan uusi kokoelmamuuttuja (suuret), joka pitää sisällään suurimmat tiedostot, jotka on manuaalisesti eritelty. Kyseinen kokoelma on set-kokoelma. Funktion käytön kannalta merkittävin ero tavalliseen listaan on, että set-kokoelma on indeksisoimaton eikä sisällä kopioita. Nämä tiedostot haetaan aikaisemmin luodusta pk10Lista-muuttujasta. Tämän jälkeen luodaan uusi listamuuttuja, uusiLista, jolle annetaan ensimmäiseksi muuttujaksi

pk10Lista sisällön, josta on poistettu suuret-muuttujan sisältö. Seuraavaksi lisätään uusiLista-muuttujaan suuret-muuttujan sisältö extend komennolla. Tämä saattaa vaikuttaa oudolta, mutta tämä muuttaa listan rakennetta. Muuttujan uusiLista ensimmäinen alkio on lista ja tämän jälkeiset alkiot ovat merkkijonoja. Jos tätä ei olisi tehty, jokainen uusiLista-alkio olisi merkkijono (kuva 14). Tällä on merkitystä funktion kannalta.

```
uusiLista = ( pk10Lista[0], pk10Lista[1], pk10Lista[2], suuret[0], suuret[1], suuret[2] )
uusiLista = ( ( pk10Lista[0], pk10Lista[1], pk10Lista[2] ), suuret[0], suuret[1], suuret[2] )
```

KUVA 14. Listan rakenne. Ensimmäisenä on lista muuttuja, johon on lisätty pk10Lista sisältö ja josta ei ole poistettu suuret-muuttujan sisältämiä tiedostoja erikseen, ja jälkimmäisenä on lista muuttuja, josta poistetaan suuret- muuttujan tiedostot pk10Lista-muuttujasta ja lisätään myöhemmin uusiLista-muuttujaan.

Tämän jälkeen suoritetaan merge-funktio kokonaisuudessaan uusiLista-muuttujan ensimmäiselle alkiolle, joka muutetaan set-tyypistä lista-tyypiksi, jotta merge-funktio tunnistaa ja kykenee työstämään kyseistä kokoelmaa. Tämän jälkeen suoritetaan merge-funkio jokaiselle jäljelle olevalle alkiolle. Koska kyseessä on merkkijono eikä lista, merge-funktio suorittaa ensimmäisen ja kolmannen ehtolausekkeen ja hyppää toisen ehdon yli, joka tarkistaa, onko kyseessä lista-kokoelma. Tämän jälkeen suoritetaan ihmisLinjat-funktio ja lopuksi yhdistetään kaikki luodut rasterit yhdeksi tiedostoksi käyttämällä ArcMapin MosaicToNewRaster_management-funktiota. Rasteri tallennetaan pääkansioon 64 m solukoolla ja se saa pikselin syvyydeksi yhden bitin.

4.4 Zonation

Zonation-funktiolla kirjoitetaan asetustiedostot Zonationille, jonka pohjalta Zonation luo rasterin. Asetustiedostot koostuvat BAT-tiedostosta, SPP-tiedostosta ja DAT-tiedostosta.

```
def zona(batlista, datlista, spplista, batpolku, matriisi, interaktio, blp):
    if not os.path.exists(batlista[1]):
        os.makedirs(batlista[1])

    # BAT
    with open(batlista[1] + '\\' + batlista[3], 'w') as bat_tiedosto:
        bat_tiedosto.write(
                'call {0} -r {1} {2} {3} {4}  -threads -grid-output- compressed-tif\n'
                    .format(batlista[0], batlista[1] + '\\' + datlista[6], batlista[1] +
                        '\\' + spplista[0], batlista[1] + '\\' + batlista[4], batlista[2]))

    # Matriisi
    if matriisi:
        with open(batlista[1] + '\\' + matriisi[0], 'w') as matriisi_tiedosto:
            for m in matriisi[3:]:
                matriisi_tiedosto.write('{0}\n'.format(m))

    # DAT
    with open(batlista[1] + '\\' + datlista[6], 'w') as dat_tiedosto:
        dat_tiedosto.write(
            '[Settings]\nremoval rule = {0}\nwarp factor = {1}\nedge removal = {2}\n'
                .format(datlista[0], datlista[1], datlista[2]))
        if interaktio:
            dat_tiedosto.write('\nuse interactions = {0}\ninteraction file = {1}\n'
                .format(interaktio[1], batlista[1] + '\\' + interaktio[0]))
        if datlista[5]:
            dat_tiedosto.write('\nmask missing areas = {0}\narea mask file = {1}\n'
                .format(datlista[4], datlista[5]))
        if matriisi:
            dat_tiedosto.write(
            '\n[Settings]\nsimilar matrix = {0}\nconnect file = {1}\nconnectivity = {2}\n'
                .format(matriisi[1], batlista[1] + '\\' + matriisi[0], matriisi[2]))
        if blp:
            dat_tiedosto.write('\nBLP = {0}'.format(blp))

    # SPP
    with open(batlista[1] + '\\' + spplista[0], 'w') as spp_tiedosto:
        for spp in spplista[1:]:
            spp_tiedosto.write('{0} {1}\n'.format(spp[0], spp[1]))

    # Interaction
    if interaktio:
        with open(batlista[1] + '\\' + interaktio[0], 'w') as interaktio_tiedosto:
            for a in interaktio[2:]:
                interaktio_tiedosto.write('{0}\n'.format(a))

    os.system(batpolku)
```

KUVA 15. Luodaan Zonationin asetustiedostot ja käynnistetään Zonation. Kuvan sisältöä muutettu, jotta kuva olisi helpommin luettavissa. Tämän vuoksi kuvan prosessi tuottaa virheellistä tietoa. Funktion oikea muoto löytyy liitteestä 4.

Kuvassa 15 esitellään Zona-funktio, jonka rakennetta on hiukan käsitelty, jotta kuvaa olisi helpompi lukea. Zonation-funktiolle annetaan arvot batlista, datlista, spplista, batpolku, matriisi, interaktio ja blp. Batlista, datlista ja spplista sisältävät tekstitiedoston sisällön. Batpolku kertoo, minne BAT-tiedosto kirjoitetaan ja mistä se jatkossakin löytyy. Matriisi,

interaktio ja blp-muuttujat kertovat, miten piirrekerrokset vaikuttavat toisiinsa. Tässä käytetään matriisitapaa. Interaktio ja blp on tehty valmiiksi, jos niitä joskus tarvittaisin. Niitä ei käsitellä tämän enempää.

Ensimmäiseksi tarkistetaan, löytyykö Zonation-kansio ja jos ei löydy, niin luodaan kansio, jonne DAT, BAT ja SPP tiedostot kirjoitetaan.

Seuraavaksi kirjoitetaan BAT-tiedosto with open -toiminnolla. BAT-tiedosto on Windows käyttöjärjestelmän komentojonotiedosto, jonka avulla käyttöjärjestelmälle voidaan kirjoittaa useita komentoja, kuten tiedostojen kopiointi, siirto ja poisto. BAT-tiedosto saa ensimmäiseksi arvoksi kansion sijainnin, toiseksi tiedoston nimen päätteellä ja kolmanneksi arvoksi ’w’, joka tulee englannin sanasta write. Seuraavaksi suoritetaan itse kirjoituskomento ja kerrotaan, mitä tiedostoon kirjoitetaan. BAT-tiedostoon kirjoitetaan komento call ja sen perään Zonation-tiedoston sijainti. Call-komento käynnistää tässä tapauksessa Zonation-ohjelman. Tämän lisäksi BAT-tiedostoon kirjoitetaan DAT- ja SPP-tiedoston sijainti sekä lokitiedoston sijainti. Lopuksi kirjoitetaan Zonation-analyysin asetuksia, jotka eivät vaikuta itse analyysiin, esim. kuinka monta prosessorin ydintä käytetään analyysissä. (Zonation manual n.d. 84–85.)

BAT-tiedoston jälkeen kirjoitetaan tarvittaessa matriisitiedosto. Matriisikytkeytyneisyydellä ilmaistaan elinympäristöjen samankaltaisuutta toisiinsa nähden. Matriisitiedostoon kirjoitetaan miten tiedostot/elinympäristöt painottuvat toisiinsa. (Zonation manual n.d. 42, 135.)

DAT-tiedosto pitää sisällään itse asetukset. DAT-tiedostossa tarkistetaan, käytetäänkö matriisia, interaktiota vai bpl:ää. Jokaisella on omat asetukset, jotka pitää kirjoittaa.

```
[Settings]
removal rule = 2
warp factor = 10000
edge removal = 1

mask missing areas = 1
area mask file = D:\Aineistot\test_trkj\pilotkun.tif

[Community analysis settings]
load similarity matrix = 1
connectivity similarity matrix file = D:\Aineistot\Zona\drm_zonation2_matrix.txt
apply to connectivity = 1
```

KUVA 16. Zonation-analyysilaskennan asetukset

Kuvassa 16 ensimmäinen asetus on removal rule. Tämä asetus määrittelee, millä tavalla rasterien solut poistetaan. Tässä asetuksena on 2, jolloin Zonation laskee kaikkien piirrekerrosten yhteenlasketun vaikutuksen soluun ja poistaa solun, joka saa pienimmän arvon. Mitä suurempi arvo, sitä tärkeämpi solu on lajille/elinympäristölle (Zonation n.d. 33). Warp factor -asetus määrittelee, kuinka monta solua poistetaan jokaisen iteraation jälkeen, joka on tässä 10000 solua. Zonation laskee jokaiselle solulle arvon removal rule -asetuksen mukaan. Kun jokaiselle solulle on laskettu arvo, siitä poistetaan warp factor -asetuksen määrä soluja, jonka jälkeen soluille lasketaan uudet arvot, koska solujen välinen vaikutus on muuttunut (Zoantion n.d. 103, 29). Edge removal -asetus määrittelee, poistetaanko soluja alueen sisältä (asetus 0) vai reunoista (asetus 1). Tässä käytetään asetusta 1, joka pitää laskenta-ajat lyhyenä ja parantaa alueiden kytkeytyneisyyttä. (Zonation n.d. 103.)

Mask missing areas ja area mask file -asetukset määrittelevät, rajataanko alue ja mitä tiedostoa käytetään rajaamaan aluetta. Tässä tapauksessa Zonationia testattiin pilottikunnissa, minkä vuoksi Zonation-analyysi rajattiin koskemaan vain näitä alueita.

Community analysis settings -asetuksissa määritellään matriisin käyttöä. Load similarity matrix määrittelee, käytetäänkö matriisia vai ei. Tässä tapauksessa käytetään, jolloin se saa arvoksi 1. Connectivity similiarity matrix file kertoo, missä matriisitiedosto sijaitsee. Matriisitiedosto pitää sisällään piirrekerrosten painotukset toisiinsa nähden (Zonation n.d. 119). Apply to connectivity -asetuksella määritetään, käytetäänkö kytkeytyneisyysmatriisia suojeltavien aluiden kytkeytyneisyyden priorisoinnissa. (Zonation n.d. 109)

SPP tiedosto kertoo piirrekerrostiedostojen sijannit ja niiden painotukset.

```
3.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\tilavuus.tif
3.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\tiheys.tif
3.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\latvus.tif
3.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\puulajit.tif
2.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\hilaMaara.tif
2.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_0402_testi\ojittamaton_con.tif
3.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_0402_testi\ojittam_euc.tif
1.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_0402_testi\ojitettu_con.tif
0.5 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\luonto.tif
1.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\metepy.tif
1.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\riekko.tif
-1.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\ihmiset.tif
-1.0 0.002 1 1 0.25 D:\Aineistot\test_trkj\testi\Koeala_2102\vesistot.tif
```

Kuva 17. Zonation-analyysissä käytettävät tiedosto, niiden sijainnit ja painotukset

Ensimmäinen arvo kertoo, miten kyseinen piirrekerros vaikuttaa priorisointiin. Mitä korkeampi arvo, sitä positiivisemmin se vaikuttaa. 0-arvo ei vaikuta priorisointiin lainkaan ja negatiivinen on haitaksi. Seuraavaa arvoa käytetään matriisikytkeytyneisyyden takia, ja se on saanut kokeilujen ajaksi arvon 0,002. Seuraava kahta arvoa ei käytetä, mutta ne pitää kuitenkin määritellä, minkä vuoksi molemmille on annettu arvo 1. Viimeinen arvo määrittelee, kuinka paljon alue menettää suojeluarvoa soluja poistattaessa. Viimeiseksi kerrotaan, missä piirrekerros sijaitsee. (Zonation 94–98.)

Lopuksi kutsutaan os.system-komentoa, joka saa arvoksi BAT-tiedoston sijainnin. Kyseinen komento käynnistää BAT-tiedoston, joka vuorostaan käynnistää Zonationin.

4.5 Oja- ja suoaineisto / piirrekerrokset 11-13

Käytettävissä aineistoissa ei ollut eroteltu ojitettuja ja ojittamattomia soita, minkä vuoksi oli luotava metodi/malli, joka erottelisi suot toisistaan. Tässä metodissa hyödynnetään suoaineistoa ja pienvirtausaineistoa. Pienvirtausaineistoa analysoitiin ja työstettiin, jotta sieltä saatiin eroteltua ojat luonnollisista virtauksista. Erotus tapahtui käyttämällä geometrisia kaavoja vektoriaineistoon. Näiden kaavojen avulla pystyttiin laskemaan vektorin suunnan, pituuden ja kahden vektorin välinen kulma, joita työstämällä voitiin laskea, vaihtuvatko vektorien suunnat ja kuinka monta kertaa sekä kuinka suuria kulman muutokset ovat.

Geometrisien kaavojen lisäksi hyödynnettiin vektorien ID-tunnuksia, joiden avulla voitiin pitää kirjaa, kuinka monesta vektorista virtauslinjat koostuivat. Virtauslinjonen vektorien määrästä voitiin päätellä, että jos linjalla on runsaasti vektoreita, kyseessä on todennäköisesti luonnollinen linja. Tämän lisäksi ID-tallentaminen mahdollisti linjojen pilkkomisen yksittäisiin vektoreihin ja lopuksi vektorien palauttamisen alkuperäiseen linjaan.

Kun suot on eroteltu toisistaan, ajetaan Python-skripti. Python-skripti työstää suoaineistoja maakunta kerrallaan, jotta kuormitus pysyy kohtuullisena ja käsittelyajat eivät turhaan pitkity. Kaikki ojat bufferoidaan 50 m:n säteellä, koska hankkeessa todettiin, että ojan vaikutus metsäakanalinnuille on 50 m. Suoaineistosta leikataan bufferoitu alue pois ja yhdistetään toisiaan koskettavat suot yhdeksi kokonaisuudeksi. Tämä tehdään, koska

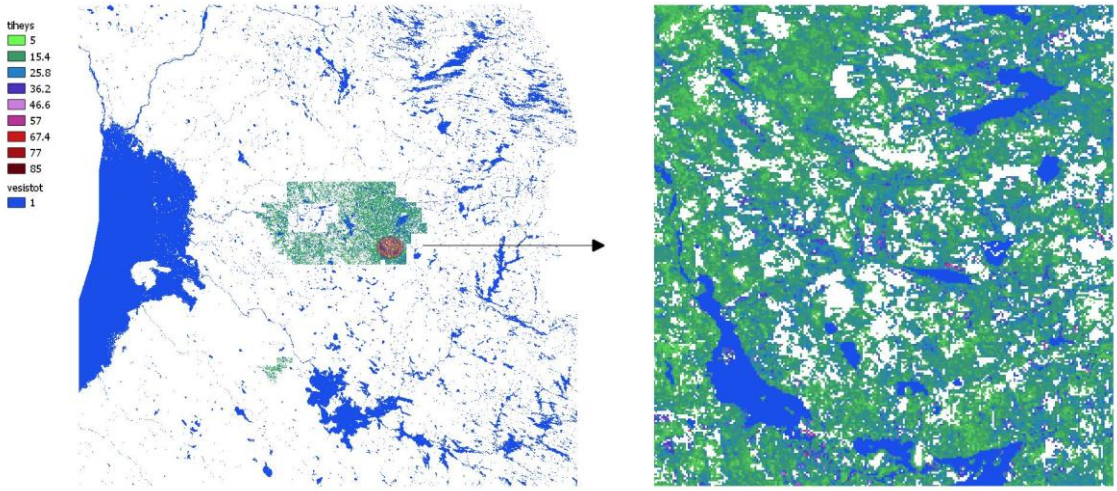
suot on eritelty tiloittain ja palstoittain. Tämän jälkeen valitaan soista kaikki, joiden pinta-ala on vähintään 5 hehtaaria. Näihin soihin suoritetaan euclidean distance. Kyseinen työkalu luo rasterin. Tästä rasterista käy ilmi, kuinka kaukana rasterin solut ovat lähdeaineistosta eli suosta.

Tämän jälkeen pidetään huoli, että aineisto on eheä, repair geometry -työkalun avulla. Saatu aineisto poistetaan suoaineistosta, jolloin jäljelle jää ojitetut suot ja alle 5 hehtaarin luonnolliset suot. Tämän jälkeen ojitetuille soille suoritetaan samat toimenpiteet kuin ojittamattomille soille.

5 TULOKSET

5.1 Tiheys

Tiheys-piirrekerros kertoo, kuinka paljon alueella on aluskasvillisuutta, joka tarjoaa suojaa metsäkanalinnuille. Tätä ja jokaista piirrekerrosta käytetään Zonation-analyysissä, jossa tiheys-piirrekerros saa positiivisen arvon, eli piirrekerroksella on myönteinen vaikutus metsäkanalintujen elinympäristöön.

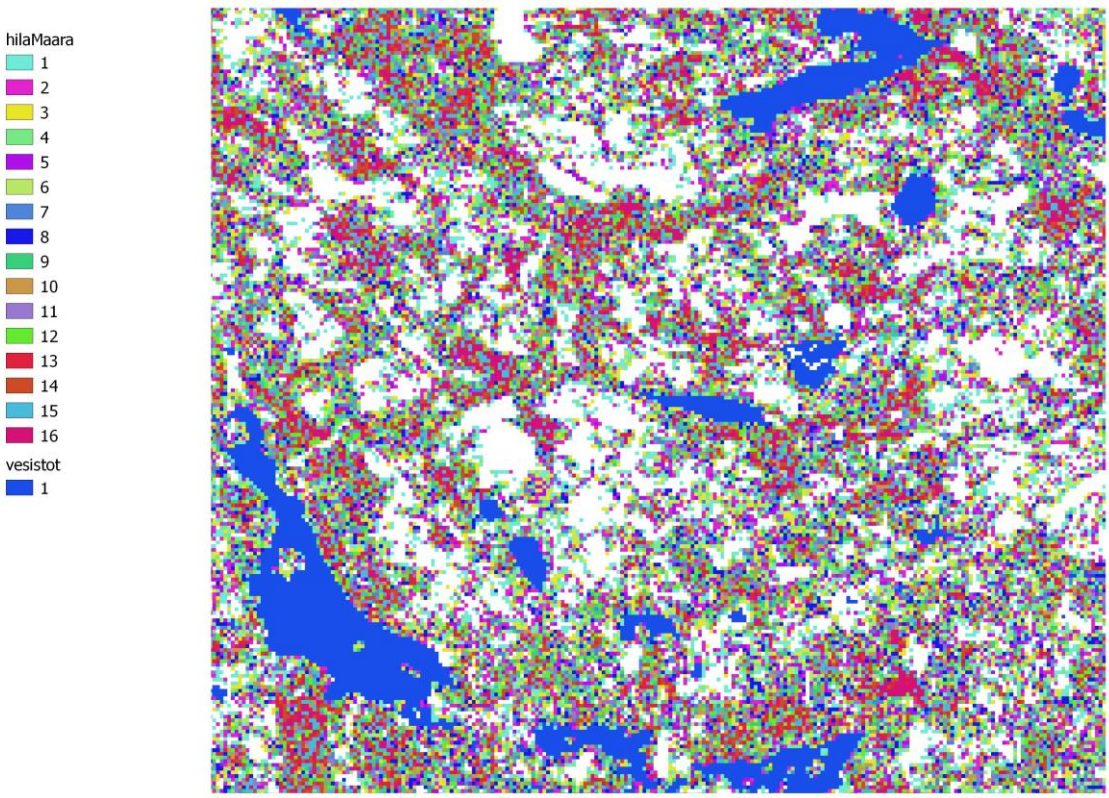


KUVIO 4. Tiheys, eli alakaiut pilottikunnan alueella

Kuviossa 4 nähdään metsän tiheys 64 m:n solun alueella, jossa suurempi arvo edustaa tiheämpää metsää. Kyseinen alue on rajoitettu hankkeen pilottikuntaan, koska hanketta ei vielä tässä vaiheessa ollut suoritettu valtakunnallisesti ja toimenpiteitä vielä testattiin.

5.2 Hilamäärä

Hilamäärä-piirrekerros kertoo, kuinka monesta riistahilasta solun arvo koostuu. Mitä suurempi luku, sitä luotettavampi tieto. Zonation-analyysissä tämä piirrekerros saa positiivisen painotuksen.

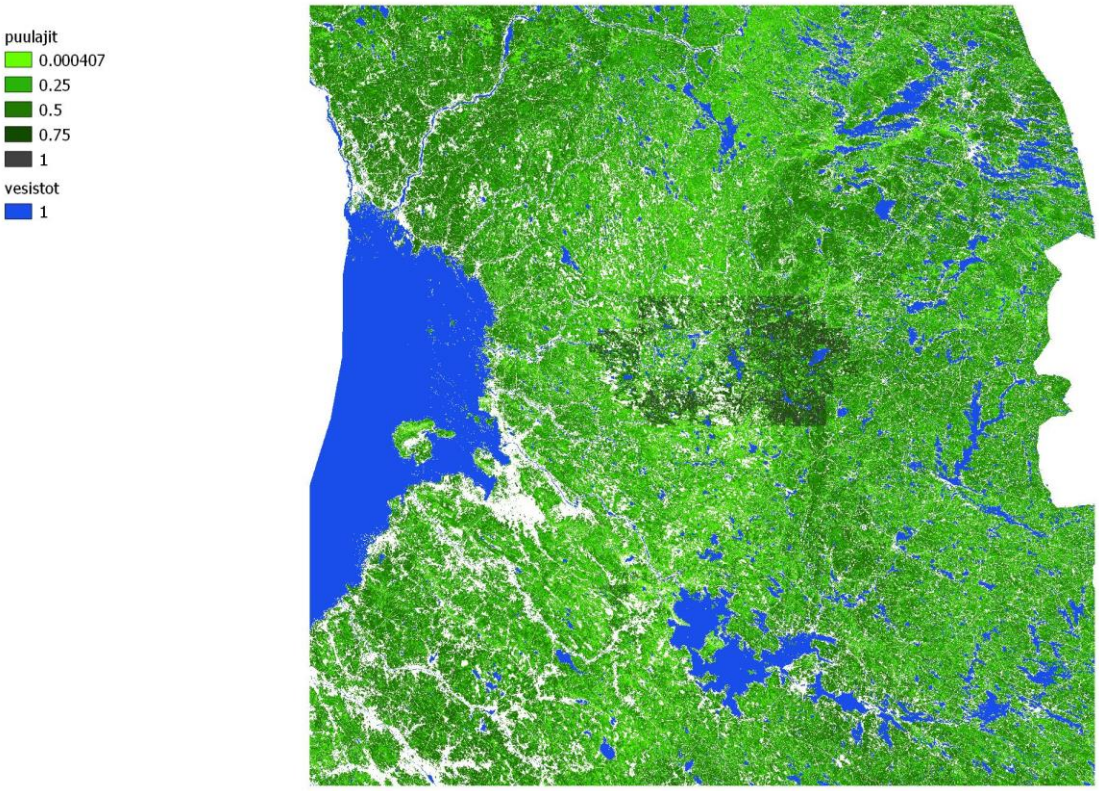


KUVIO 5. Rasteriaineisto, jossa näkyy, kuinka monta riistahilaa käytettiin jokaisessa rasterin solussa.

Hilamäärä kertoo, kuinka monta 16 m x 16 m solua jokaisessa 64 m x 64 m solun sisällä on. Tämä tieto kertoo, kuinka monta solua laskennassa on käytetty, mikä puolestaan kertoo, kuinka paljon solun tietoja yleistettiin. Kuviossa 5 näkyy esimerkki hilamäärä-rasterista pilottikunnan alueella. Esimerkiksi vaaleansiniset alueet ovat 64 m solun alueita, joiden laskennassa käytettiin yhtä riistahilaa. Alueet, joissa käytettiin 16 kappaletta riistahiloja, ovat purppuroita. Valkoisilla alueilla ei ollut yhtään riistahilaa.

5.3 Puulajit

Puulajit-piirrekerros kertoo, kuinka suuri osuus yhdellä puulajilla on alueella. Arvo 1 tarkoittaa, että alueella on kolme eri puulajia, eli metsä ei ole homogeeninen. Zonation-analyysissä tällä piirrekerroksella on positiivinen painotus.

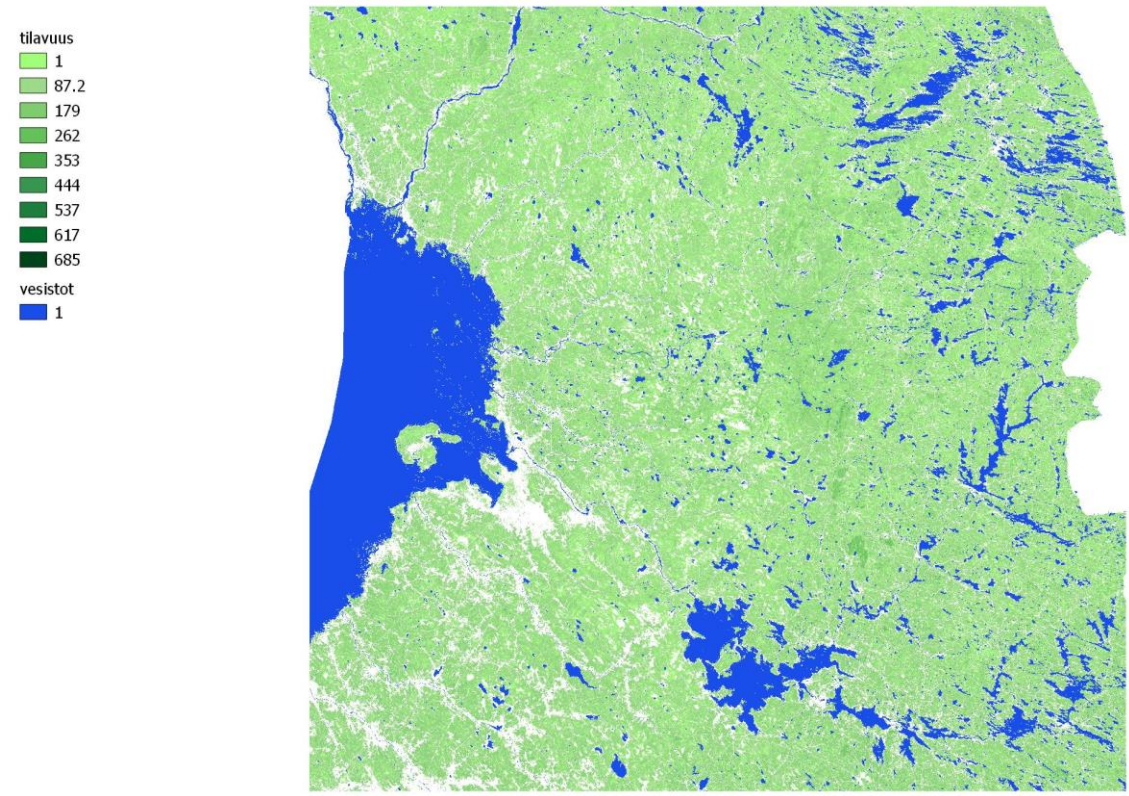


KUVIO 6. Puulajien määrä koko Suomessa. Pilottikunta erottuu tummempana.

Kuviossa 6 nähdään puulaji-rasterit. Mitä vaaleampi alue, sitä suurempi osuus yhdellä puulajilla on tilavuudesta. Kuvassa nähdään, miten puulajien määrä on keskittynyt pilottikunnan alueella. Pilottikunnissa puustotiedot on laskettu riistahiloilla, kun muussa Suomessa on käytetty rasteriaineistoa. Tällä hetkellä riistahila-aineisto kattaa koko maan tai lähes koko maan ja rasteriaineistoa käytetään vain täydentämään puutteelliset alueet.

5.4 Tilavuus

Tilavuus-piirrekerros kertoo, kuinka järeää puuta alueella on. Mitä suurempi arvo, sitä järeämpää puuta. Zonation-analyysissä tämä piirrekerros saa positiivisen painotuksen.

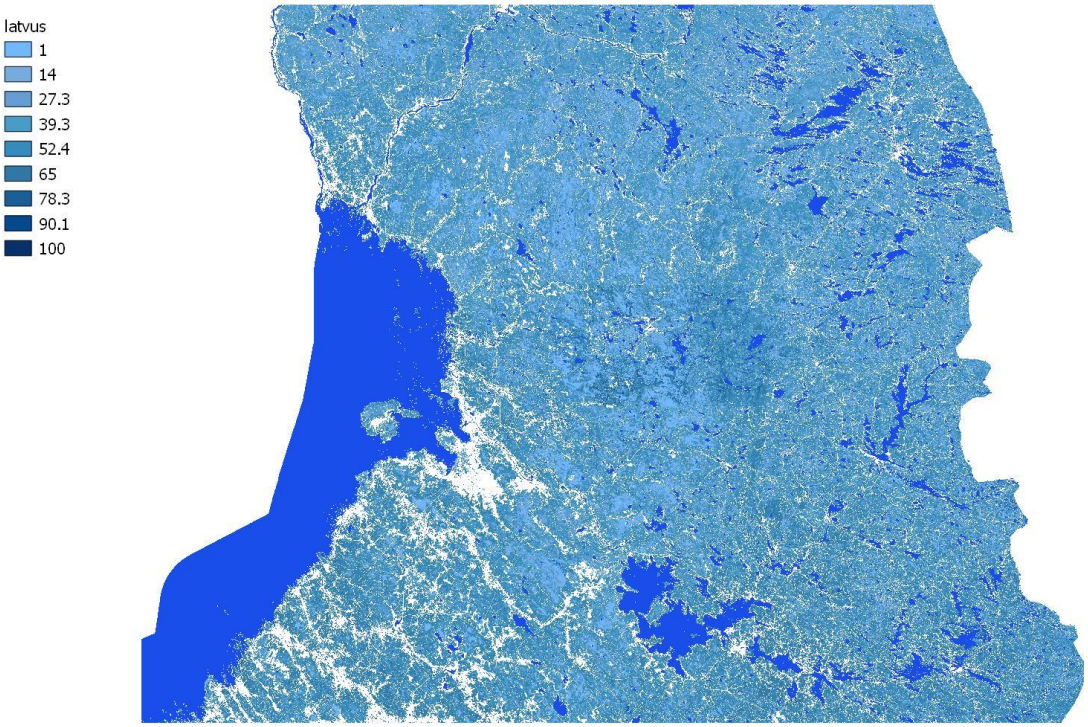


KUVIO 7. Metsien tilavuus kuutiometreinä

Kuviossa 7 näkyy tilavuus-rasteri. Mitä tummempi vihreä, sitä suurempia puita alueella on. Kuvassa näkyy, että pilottikuntien alue ei erotu muusta alueesta, koska riistahila-aineiston tilavuustieto tulee samasta lähteestä kuin muu Suomi.

5.5 Latvuspeittävyys

Latvuspeittävyys-piirrekerros kertoo prosentteina, kuinka suuri osa alueen pinta-alasta on latvuksen peittämänä. 100 tarkoittaa, että latvukset peittävät koko alueen. Zonation-analyysissä tämä piirrekerros saa positiivisen painotuksen.

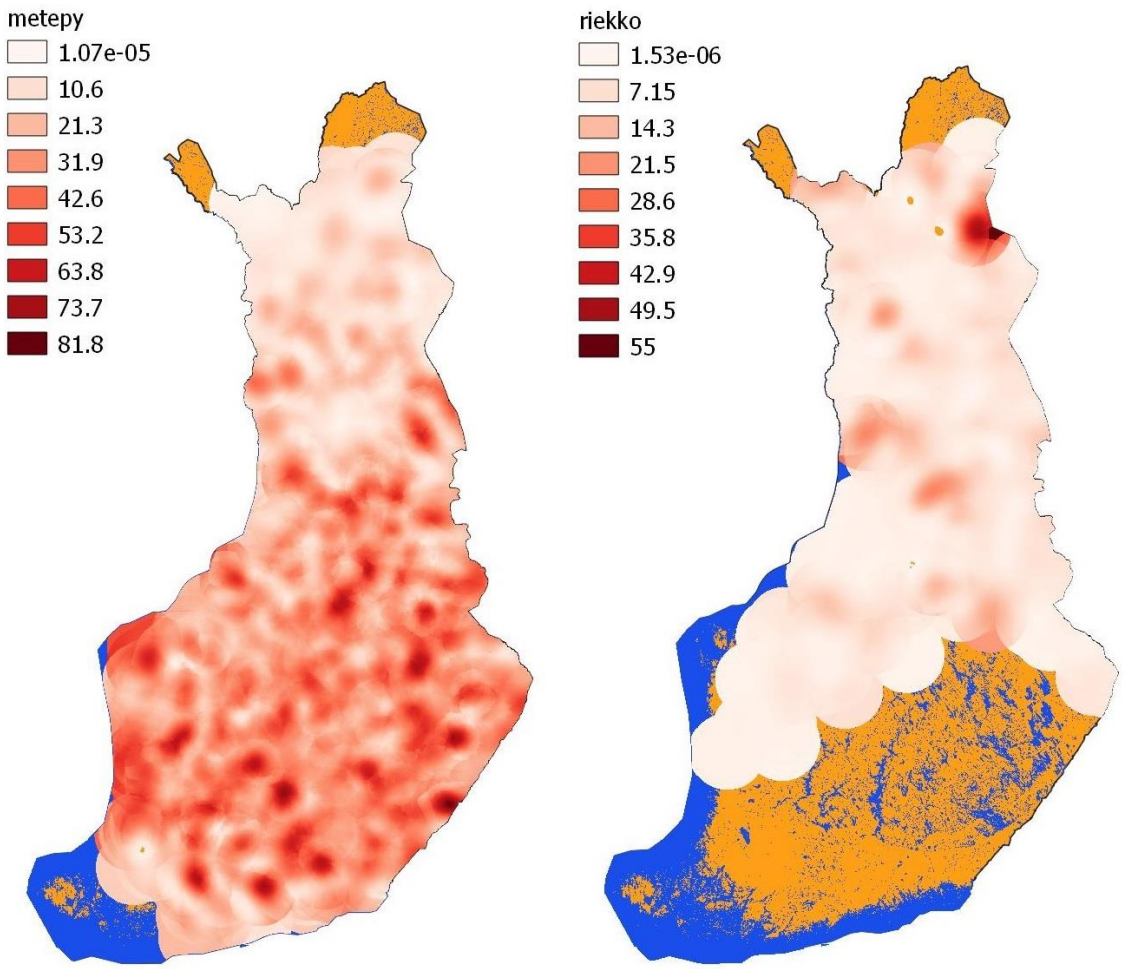


KUVIO 8. Metsien latvuspeittävyys prosentteina.

Kuviossa 8 on latvuspeittävyyden rasteri. Aivan kuten puulaji-rasterissa, pilottikunta erottuu muusta maasta samasta syystä.

5.6 Metsäkanalinnut

Metsäkanalinnut koostuu kahdesta piirrekerroksesta. Metso, teeri ja pyy on yhdistetty yhdeksi piirrekerrokseksi ja riekko omaksi piirrekerrokseksi. Nämä piirrekerrokset ovat krieging interpolpointi-metodin tuloksia. Piirrekerrokset kertovat, miten laji on levittäytynyt ja saavat positiivisen painotuksen Zonation-analyysissä.



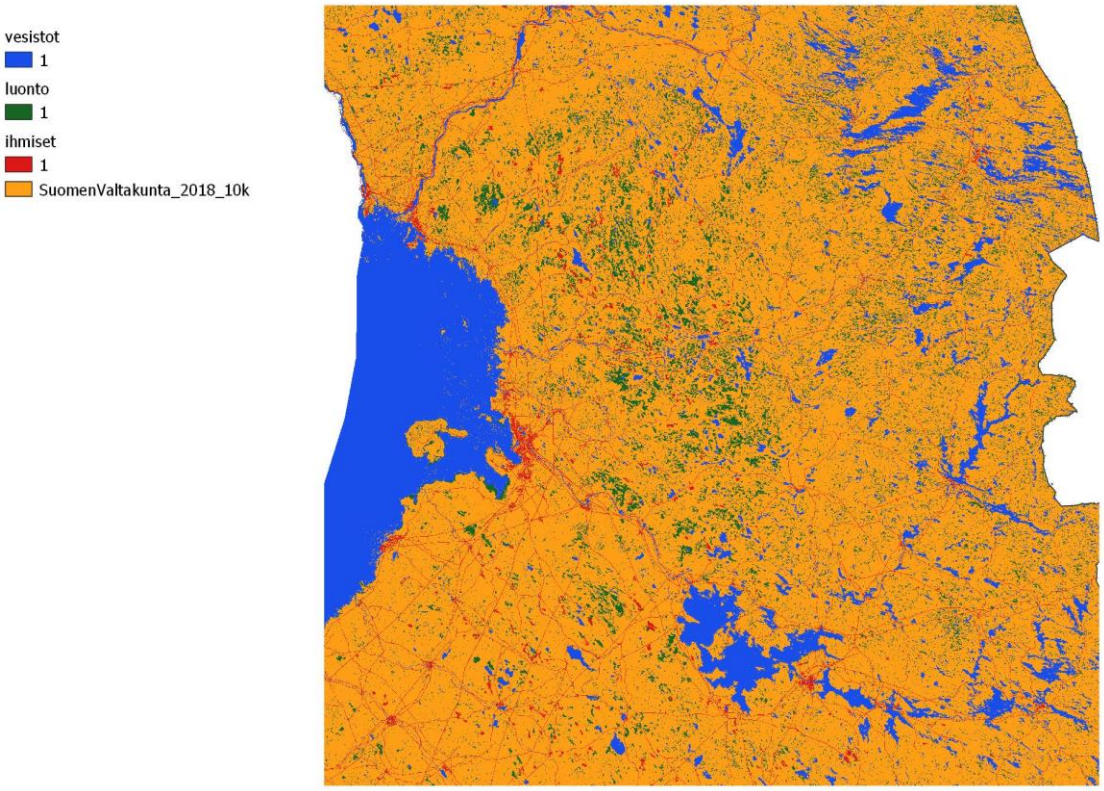
KUVIO 9. Vaemmalla näkyy metson, teeren ja pyyn levittäytyminen ja oikealla vastaavasti riekon levittäytyminen. Tummempi väri kertoo, kuinka paljon alueella lajia esiintyy

Kuviossa 9 näkyy metsäkanalintujen interpoloidut elinalueet. Metepy tarkoittaa metsoa, teertä ja pyytä. Kuvassa nähdään, että nämä lajit ovat levinneet tasaisesti ympäri Suomea, kun taas riekko on keskittynyt pohjoiseen. Metson, teeren ja pyyn elinympäristöt ovat riittävän lähellä toisiaan, minkä vuoksi ne on yhdistetty yhdeksi, kun taas riekon elinympäristö poikkeaa muista merkittävästi, minkä vuoksi se on pidetty erillään. Zonation-analyysissä molemmilla piirrekerroksilla on painotuksena 1.

5.7 Maastotietokanta

Maastotietokanta-piirrekerrokset ovat vesistöt, tiet ja ihmisten rakennelmat sekä luonnon aukot. Nämä piirrekerrokset kertovat, esiintyykö alueella piirrekerroksen sisältöä, esim.

vesistöä. Zonation-analyysissä luonto piirrekerros saa painotuksen, joka on lähellä nollaa. Ihmiset ja vesistöt-piirrekerrokset saavat negatiivisen painotuksen.

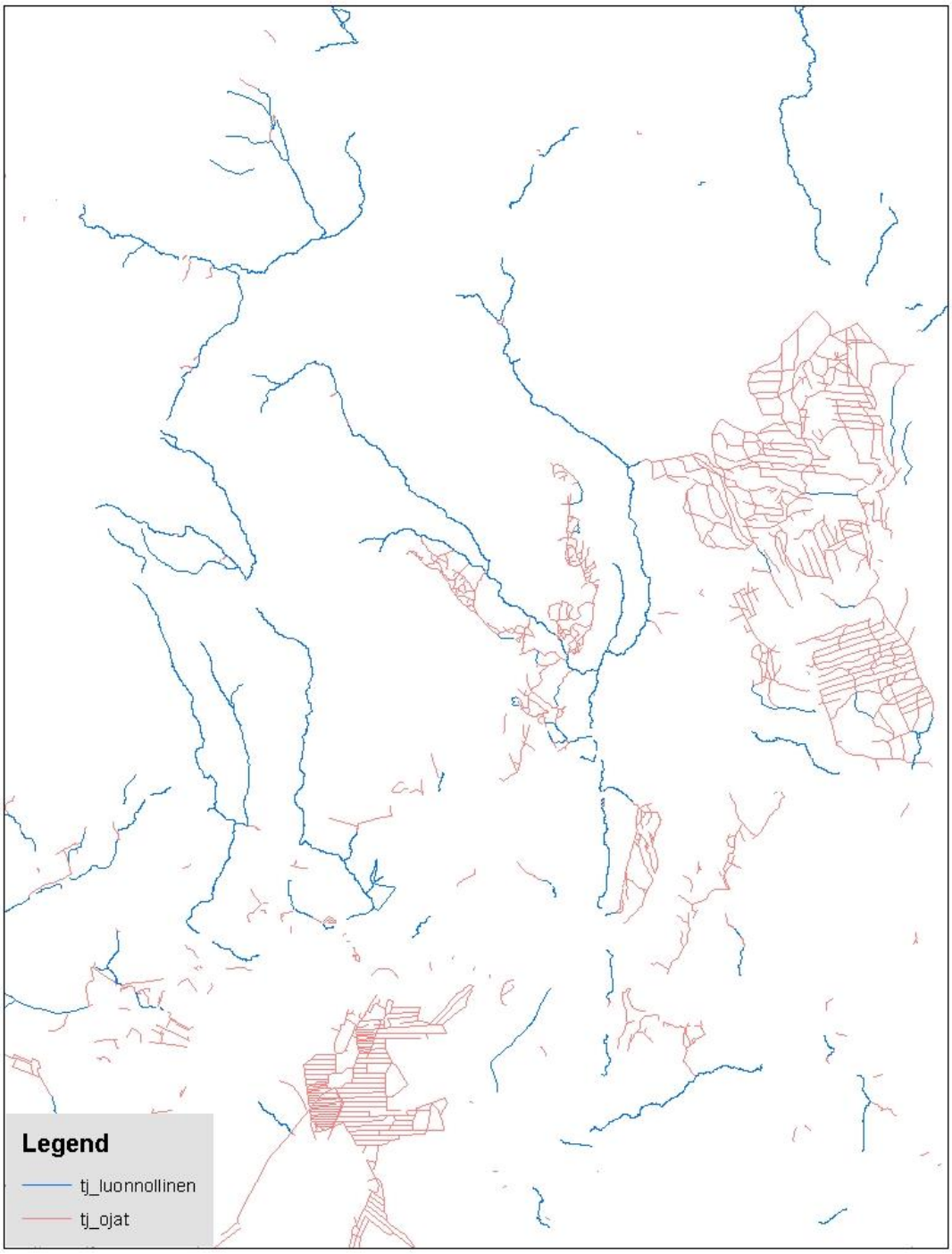


KUVIO 10. Maastotietokanta-aineisto rasteri muodossa.

Kuviossa 10 näkyy ihmisten rakennelmat, vesistöt ja avoimet ympäristöt. Vesistöt ja ihmiset -piirrekerrokset vaikuttavat negatiivisesti metsäkanalintujen elinympäristöön, myös luonto-piirrekerroksella on laskeva vaikutus.

5.8 Suoaineisto

Suoaineisto-piirrekerrokset ovat ojitettu suo, ojittamaton suo ja etäisyys ojittamattomasta suosta. Ojitetun ja ojittamattoman suon erottelussa käytetään virtausaineistoa, josta erotellaan ojat luonnollisista virtauksista matemaattisen kaavan ja erinäisten ehtojen avulla. Suoaineistot saavat positiivisen painotuksen Zonation-analyysissä.

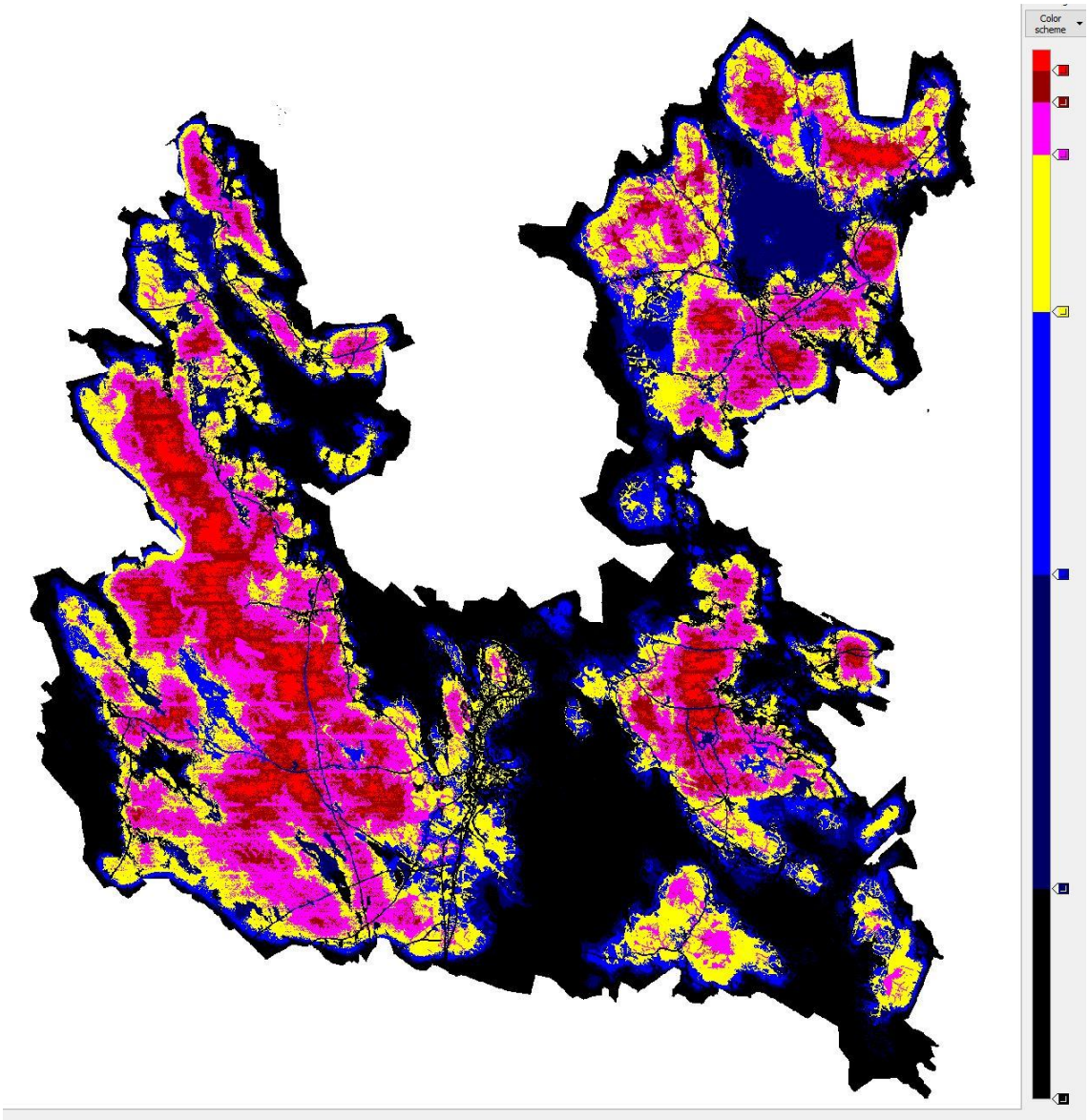


KUVIO 11. FME-suomallin tulos. Sininen osoittaa luonnollisia virtauksia ja punainen ojia.

Kuviossa 11 näkyy ojitettujen ja ojittamattomien soiden erottelu mallin avulla. Kuvasta nähdään, että malli ei ole täysin valmis, mutta selkeimmät tapaukset on löydetty. Metodia/mallia pitäisi vielä työstää, jotta siitä tulisi luotettavampi, mutta alku on lupaava.

5.9 Zonation

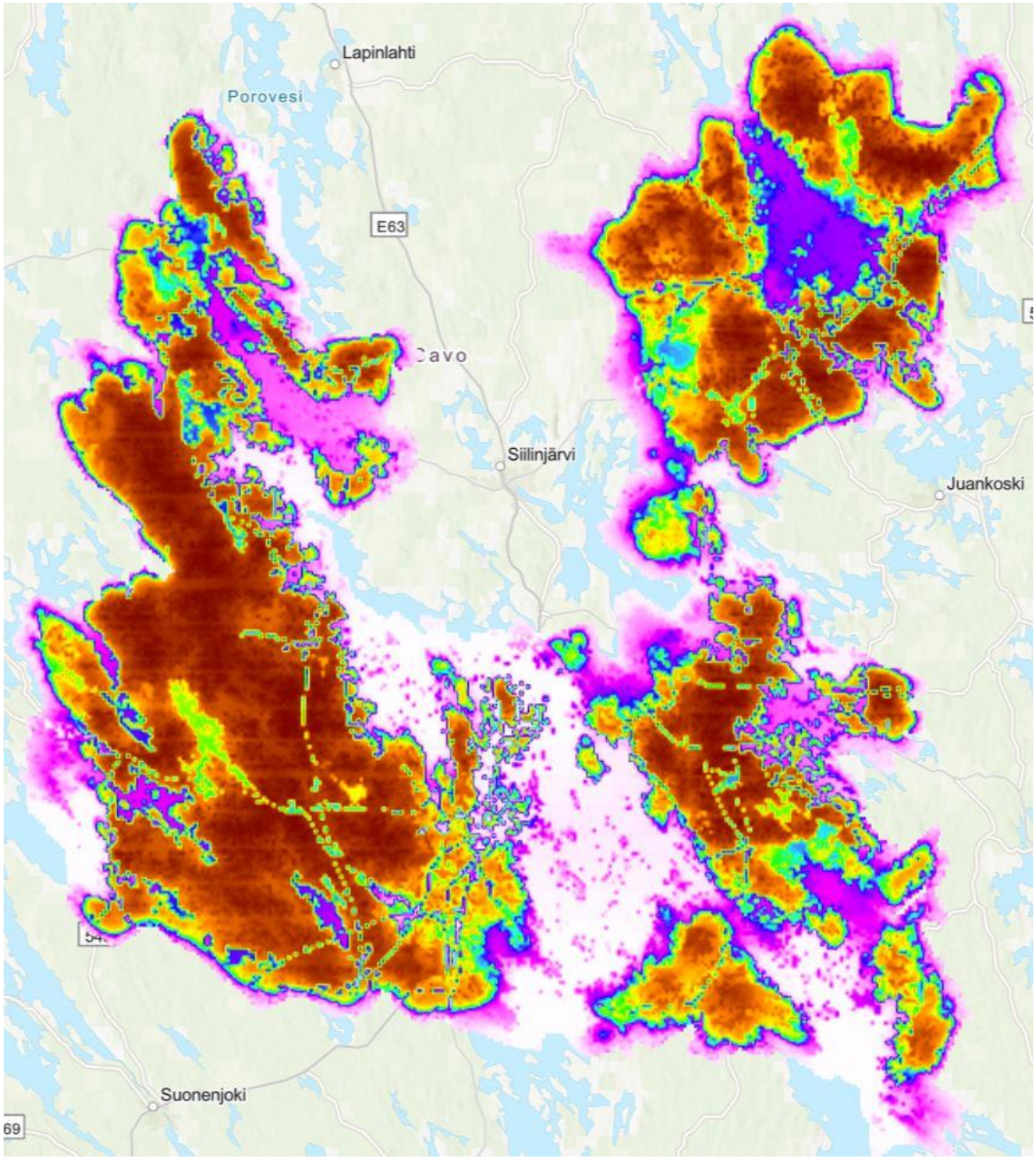
Zonation-analyysissä käytetään 13 piirrekerrosta. Kuvassa 20 nähdään näiden piirrekerrosten painotus Zonation-analyysissä ensimmäisessä sarakkeessa, mutta nämä painotukset eivät vielä tässä vaiheessa ole valmiita. Nämä piirrekerrokset määrittelevät potentiaaliset metsäkanalintujen elinympäristöt.



KUVIO 12. Zonation-analyysin tulos. Tummanpunainen väri on erittäin hyvä elinympäristö ja vastaavasti musta erittäin huono elinympäristö.

Kuviossa 12 näkyy yksi Zonation-analyysin tulos pilottikunnan alueelta. Kyseinen analyysi oli kokeilu eikä edusta lopullista analyysiä. Analyysissä käytettiin piirrekerroksia,

joita oli yhteensä 13 kappaletta. Tässä opinnäytetyössä käytiin pääasiassa läpi, miten 10 näistä piirrekerroksesta syntyi. Kolme muuta piirrekerrosta saatiin FME-mallista, joka tuotti ojitettu suo, luonnontilainen suo ja etäisyys luonnontilaisesta suosta -piirrekerrokset. Analyysi tehtiin aineistojen ja painotusten testaamiseksi, jotta lopullinen analyysi olisi mahdollisimman oikea. Kuvan oikealla reunalla on väriasteikko, jossa korkein väri, joka on punainen, edustaa parasta mahdollista elinympäristöä, ja alhaisin väri taas huonointa mahdollista elinympäristöä. Kuviossa 13 nähdään, että tulokset poikkeavat hieman kuvion 12 tuloksista, koska kuvion 12 painotukset ovat vielä testivaiheessa toisin kuin kuvion 13 painotukset.



Kuvio 13. Zonation analyysin tulos Kuopion alueelta. (Metsäkeskus aineisto)

Työni tarkoitus oli luoda prosessi, joka luo piirrekerrokset ja käynnistää Zonation-analyysin. Prosessien tuottamat piirrekerrokset ja Zonation-analyysin tulos ovat uskottavia eikä niissä esiinny mitään selkeää virhettä. Prosessien automatisointi on rakennettu rinnakkaisprosessia käyttäen, jonka avulla voidaan suorittaa useampia prosesseja samanaikaisesti hyödyntäen prosessorin ytimiä. Rinnakkaisprosessointi on rakennettu käyttämään kolmea ydintä ja kaikki muokkaukset ytimien määrään täytyy tehdä manuaalisesti. Hankkeessa ei ollut aikaa perehtyä mahdolliseen dynaamiseen ytimien määrän säätämiseen, vaan priorisoitiin toimiva ratkaisu.

Saatuja tuloksia ei ole käyty maastossa tarkistamassa, minkä vuoksi tulosten oikeellisuudesta ei voi olla täysin varma. Tietojen oikeellisuuden varmistaminen edellyttäisi koko Suomen kattavan koealaotannan, jota ei suoritettu sinä aikana, kun olin hankkeessa mukana. Ojitettujen soiden tunnistaminen jäi puutteelliseksi ja hieman virheelliseksi, mutta alustavat kokeilut ovat lupaavia ja mallia hyödyntämällä muissa hankkeissa tai mallia kehittämällä voidaan löytää entistä paremmin luonnontilaisia ojia.

6 POHDINTA

Skripti on rakennettu noudattamalla hyvää ohjelmointietikettiä. Ulkoasussa on pyritty selkeyteen ja helppolukuisuuteen. Funktioiden sisällä ei ole muuttujia, elleivät arvot ole vakioita ja omia kokonaisuuksiaan. Funktiot on tallennettu erilliselle tiedostolle ja niitä kutsutaan main/päätiedostosta. Tämä helpottaa skriptin/funktioiden kopioimista vastaaviin tehtäviin, jolloin muiden toimijoiden on helpompi lukea ja ymmärtää selkeästi kirjoitettua skriptiä.

Ojittamattomien ja ojitettujen soiden erottelumalli ja skripti eivät ole täysin valmiita, vaan sitä pitäisi kehittää paremmin tunnistamaan ja erottelemaan ojat luonnollisista virtauksista. Tämän lisäksi erottelu ojitetuiksi ja ojittamattomiksi voitaisiin vaihtaa asteikkoon, joka kertoo, kuinka todennäköisesti virtaus on oja tai luonnontilainen puro. Mallissa on myös pari virhettä. Malli erottelee epävarmat virtaukset joko ojiksi tai luonnollisiksi virtauksiksi. Malli ei myöskään osaa huomioida suoaineistoja, jotka ylittävät maakuntien rajat järkevästi, vaan käsittelee aineiston useampaan otteeseen. Mallin ei pitäisi jättää mitään aineistoa käsittelemättä.

Mitä tulee automatisaatioon, tavoite saavutettiin. Skripti toimii ja tuottaa aineistoa, mutta koska testit on tehty koealueille, mahdollisia muutoksia ei voida poissulkea. Nämä muutokset ovat todennäköisesti suorituskykyyn vaikuttavia, kuten esim. rinnakkaisprosessien määrä. Prosessin tuottama aineisto on uskottavaa ja käyttökelpoista Zonation-analyysille. Zonation-analyysin automatisointi on rakennettu ja tarvitsee vain oikeat painokertoimet ja asetukset, jotta analyysi voidaan suorittaa. Zonation-analyysi tuottaa uskottavaa aineistoa, eli ilmiselvät epäsopivat alueet on rajattu pois, mutta ilman oikeita painoarvoja on vaikea sanoa, kuinka todenmukaista aineistoa se tuottaa.

Zonation-analyysin tuloksen avulla voidaan huomioida metsäkanalintujen elinympäristöjä, kun suunnitellaan hakkuita ja metsänhoitotoimenpiteitä. Metsäkanalintujen huomioiminen ylläpitää niiden elinympäristöjä ja mahdollistaa kannan elpymisen ja vahvistumisen. Zonation-analyysiä ei suoritettu valtakunnallisella tasolla, mutta tulokset ovat olemassa pilottikunnista. Kyseinen aineisto löytyy Metsäkeskuksen aineistoista.

Metsäkeskuksen Zonation-analyysin mukaan vesistöt ja tiet eivät ole sopivia elinympäristöjä metsäkanalinnuille. Kyseisillä alueilla on negatiivinen vaikutus metsäkanalintujen elinympäristölle, joten oli odotettavissa, että nämä alueet saavat Zonation-analyysissä heikot arvot. Tosin ainoastaan suurimmat ja selvimmät alueet ovat saaneet negatiiviset arvot. Rasterien suuri solukoko voi selittää analyysissä esiintyvät epätarkkuudet. Analyysistä näkyy, että suot sopivat metsäkanalinnuille. Metsäkanalinnut viihtyvät soisilla alueilla, joten tämä ei tullut yllätyksenä. Analyysi on huomioinut elinympäristöjä määrittäessään ojitettujen ja luonnontilaisten soiden eron, tosin näiden soiden arvot ovat lähellä toisiaan. Tämä viittaa siihen, että suopiirrekerrosten painotukset ovat samankaltaiset ja kun katsotaan kuvan 17 painotuksia, niin niiden alustavat painotukset poikkeavat selvästi toisistaan, minkä vuoksi on vaikea sanoa, miten piirrekerrosten painotukset ovat muuttuneet. Tämän lisäksi ei voida poissulkea mahdollisuutta, että analyysissä olisi voitu käyttää toista suoaineistoa tai muuta aineistoa.

LÄHTEET

Antikainen H., Määttä-Juntunen H., Ujanen J. 2015. GIS-analyysimenetelmät ArcGIS 10.2.1 -ohjelmistolla. Oulu. Oulun yliopisto.

Digiriistametsä. Metsäkeskus. Luettu 01.27.2020 https://www.metsakeskus.fi/digiriista-metsa

ETRS89. Luettu 9.3.2020 https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/old/ETRS89koordinaattijarjestelma_kayttoon.pdf

Feature class. ArcGis for desktop. Luettu 30.9.2019 https://desktop.arcgis.com/en/arcmap/latest/analyze/main/what-is-geoprocessing.htm

Geodatabase. ArcGis for desktop. Luettu 30.9.2019 http://desktop.arcgis.com/en/arcmap/10.3/manage-data/geodatabases/what-is-a-geodatabase.htm

Kesälä, M. Miettinen, J. Rantala, M. Svensberg, M. 2018. Digiriistametsä-hanke. Metsästäjä 02/2018. 36-37.

Kriging. ArcGis for desktop. Luettu 2.11.2019 http://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/kriging.htm

Line Statistics. ArcGis for desktop. Luettu 30.09.2019 http://desktop.arcgis.com/en/arcmap/10.3/tools/spatial-analyst-toolbox/line-statistics.htm

Maastotietokanta. Luettu 30.9.2019 https://www.maanmittauslaitos.fi/kartat-ja-paikkatieto/asiantuntevalle-kayttajalle/tuotekuvaukset/maastotietokanta-0

Metla. Luke. Luettu 24.9.2019 http://www.metla.fi/ohjelma/vmi/vmi-moni.htm

Metsäkeskuksen aineisto. Luettu 12.2.2020 http://www.arcgis.com/home/webmap/viewer.html?url=http%3A%2F%2Faineistot.metsakeskus.fi%2Fmetsakeskus%2Frest%2Fservices%2FRiistanhoito%2FRiistapiirre_zonation%2FImageServer&source=sd

Metsätiedonkeruu. Metsäkeskus. Luettu 30.9.2019 https://www.metsakeskus.fi/metsatiedon-keruu

Miettinen, J, Kesälä, M. 2018. Digiriistametsä-hanke. Metsästäjä 05/2018, 50-51.

Moilanen A. 2014. Zonation user manual. Helsinki. Helsingin Yliopisto.

Neopoint OY, Laserkeilaus. Luettu 27.09.2019 https://www.neopoint.fi/fi/laserkeilaus

Paikkaoppi. Luettu 30.09.2019 http://www.paikkaoppi.fi/fi/paikkatiedon-hyodyntaminen-ja-paikkatietoanalyysit-2/

Python. Luettu 25.9.2019 https://www.python.org/doc/essays/blurb/

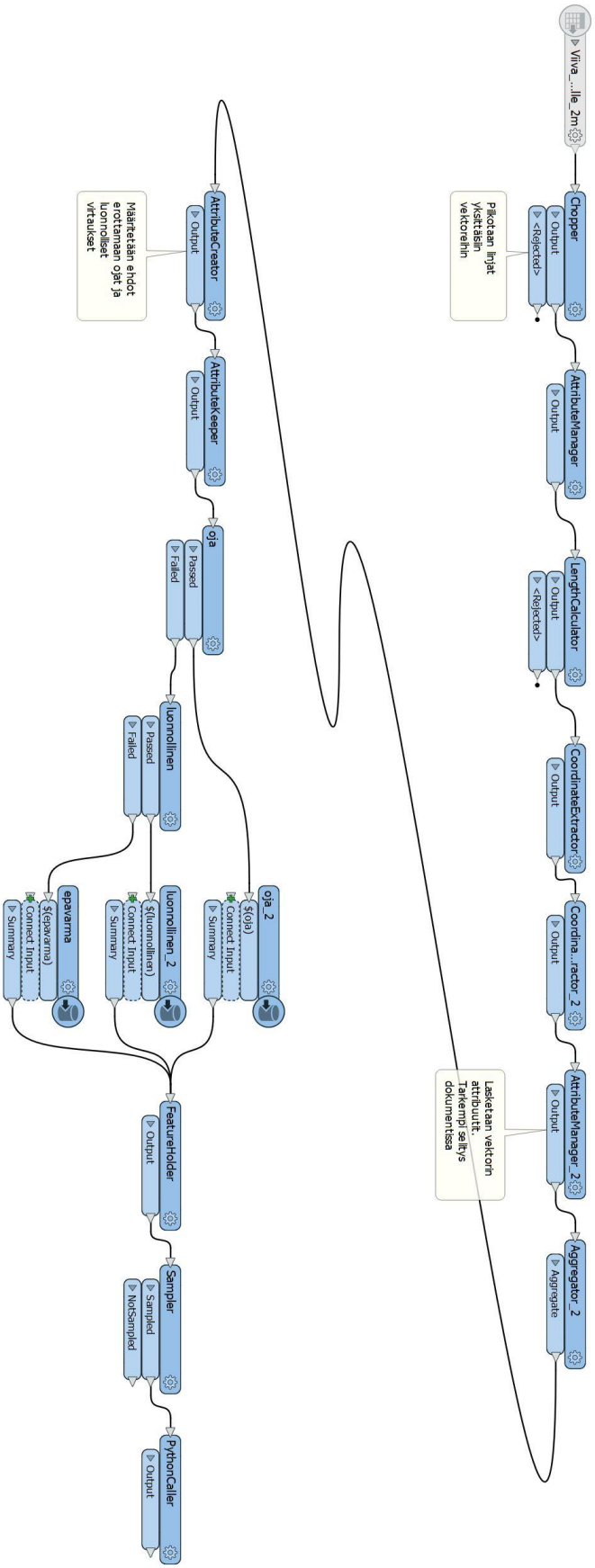
Spatialworld. FME. Luettu 24.9.2019 http://www.spatialworld.fi/fi/fme/

UPM. Luettu 1.11.2019 https://www.upmmetsa.fi/tietoa-ja-tapahtumia/artikkelit/mita-on-metsavaratieto/

What is geoprosessing?. ArcGis for desktop.Luettu 01.27.2020 https://pro.arcgis.com/en/pro-app/help/analysis/geoprocessing/basics/what-is-geoprocessing-.htm

LIITTEET

Liite 1. FME malli.



Liite 1 selitys:

1. Luetaan tietokannasta virtausaineisto
2. Pilkotaan vektoriaineisto yksittäisiin vektoreihin
3. Määritetään uusi tunniste ominaisuus, jolle määritetään id perustuen mihin vektorikokonaisuuteen kuuluu. Tämä tehdään, jotta vektorit voidaan palauttaa alkuperäiseen muotoon.
4. Lasketaan jokaiselle vektorille pituus
5. Määritetään vektorin ensimmäiselle ja toiselle pisteelle x- ja y-koordinaatit
6. Määritetään ja lasketaan uudet ominaisuudet
    a. pituusC: Lasketaan kahden vektorin hypotenuusa.
    b. delta. Vektorin x-koordinaattien erotus.
    c. atan. Lasketaan vektorin kulman x-akselin suhteen. Nähdään, onko kyseessä nouseva vai laskeva vektori, jota käytetään suunta ominaisuustiedossa.
    d. kulma. Lasketaan kahden vektorin välinen kulma.
    e. suunta. Annetaan arvot 1,0 tai -1. 1, jos kyseessä on nouseva vektori ja -1, jos kyseessä on laskeva vektori. 0 annetaan, jos vektori on vaakasuora.
    f. sVaihto. Käytetään suunta ominaisuustietoa ja lasketaan, kuinka monta kertaa vektorin suunta vaihtuu vektorilinjan sisällä.
    g. kVaihto. Samanlainen kuin sVaihto, mutta lasketaan kulman muutoksia. Huomioidaan vain ne vektorit, joiden kulma on alle 135°
    h. summaKulma. Lasketaan kaikki vektorilinjan kulma ominaisuustiedot yhteen.
    i. ykkoset. Jokainen vektori saa arvon 1, joka summataan, kun rakennetaan vektorilinja uudestaan. Tämän avulla saadaan selville, kuinka monesta vektorista vektorilinja koostuu.
7. Yhdistetään vektorit alkuperäiseen muotoon ja yhdistetään ominaisuudet joko summa tai keskiarvo asetuksella. Kohdassa 6:d ominaisuus keskiarvoistetaan ja kohdat 6:f – 6:i ominaisuudet summataan
8. Määritetään ehdot ojitetuille ja luonnollisille virtauksille, joiden perusteella vektorit jaetaan.
    a. Oja saa luonnollisen ojan arvon, kun yksi ehdoista täyttyy:
        i. sVaihto > 3 JA ykkoset > 10
            1. Aineistoa tutkimalla totesin, että ojan vektorit ovat yleensä samansuuntaisia, jonka vuoksi ojien sVaihto ominaisuustieto on yleensä 0-2.
        ii. ykkoset > 20
            1. Aineistoa tutkimalla totesin, että luonnolliset virtaukset koostuvat usein monista vektoreista, toisin kuin ojat, jotka yleensä koostuvat muutamista vektoreista.
    b. Oja saa ojitetun ojan arvon, kun yks ehdoista täyttyy:
        i. sVaihto < 3 JA kulma >= 170 JA kVaihto / summaKulma < 0,2
            1. Aikaisemmin totesin, että ojien sVaihto arvo on pieni, koska ne ovat samansuuntaisia. Samansuuntaisuus tarkoittaa myös, että vektorien välinen kulma on lähellä 180° ja luonnollisilla virtauksilla vektorien välinen kulma vaihtelee suuresti. Viimeinen ehto pitää huolen, että kVaihto ominaisuuden osuus koko vektorilinjasta on alle 20%.
        ii. kulma > 175
        iii. kulma > 170 JA sVaihto < 3
        iv. kulma <= 180 – (10 x ykkoset)

1. Aineistosta kävi ilmi, että ojalinjat saattavat kohdistua kahdesta vektorista, joiden välinen kulma on 90°. Tätä varten piti luoda oma ehto, joka toimii pienissä ojalinjoissa, mutta ei suurissa. Tämän ehdon vaatimukset kiristyvät mitä pidempi ojalinja on kyseessä.

v. (summaKulma - kVaihto) / (ykkoset – (1 + (kVaihto / 135))) >= 165
1. Lasketaan vain suorien ja yhdensuuntaisten vektorien keskiarvo ja tarkastetaan, onko arvo suurempi kyin 165°

c. Jos mikään ehdoista ei täyty, annetaan ojalle epävarma arvo.

9. Poistetaan kaikki ominaisuustiedot, paitsi ojitettu/ojittamaton tieto. Tämä nopeuttaa käsittelyä
10. Kirjoitetaan kolme tiedostoa, ojitetut, ojittamaton ja epävarma.
11. Odotetaan, että kaikki tiedostot ovat kirjoitettu.
12. Ajetaan kirjoitettu aineisto python skriptin läpi.

Liite 2. FME:n mallissa oleva python skripti.

```
import arcpy
from arcpy.sa import Con, EucDistance
import fme
import fmeobjects
arcpy.scratchWorkspace = arcpy.env.workspace = FME_MacroValues['polku']
arcpy.env.overwriteOutput = True
arcpy.CheckOutExtension('Spatial')
# ojitetuissa soissa virhe. mukana ojittamattomia
def FeatureProcessor(feature):
   kissa = """""""
   arcpy.MakeFeatureLayer_management(FME_MacroValues['luonnollinen'], 'luonto_lyr')
   arcpy.MakeFeatureLayer_management(FME_MacroValues['oja'], 'oja_lyr')
   arcpy.MakeFeatureLayer_management(FME_MacroValues['epavarma'], 'epavarma_lyr')
   arcpy.SelectLayerByLocation_management('epavarma_lyr', 'INTERSECT', 'luonto_lyr')
   arcpy.SelectLayerByLocation_management('epavarma_lyr', 'INTERSECT', 'oja_lyr', '', 'REMOVE_FROM_SELECTION')
   arcpy.CopyFeatures_management('epavarma_lyr', 'mergattu')
   arcpy.SelectLayerByLocation_management(in_layer = 'epavarma_lyr', selection_type = 'SWITCH_SELECTION')
   arcpy.CopyFeatures_management('epavarma_lyr', 'uusi_epavarma')
   arcpy.Merge_management([FME_MacroValues['luonnollinen'], 'mergattu'], 'yhdistetty')
   arcpy.Buffer_analysis(FME_MacroValues['oja'], 'oja_bufferi', 100)
#    arcpyDelete_management(FME_MacroValues['epavarma'])
#    arcpyDelete_management(FME_MacroValues['luonnollinen'])
#    arcpyDelete_management(FME_MacroValues['oja'])

   suot = 'metsaiset_suot'
#    arcpy.Merge_management([FME_MacroValues['Yhdistettava_suo1'], FME_MacroValues['Yhdistettava_suo2']], suot)
#    arcpy.RepairGeometry_management(suot)

   arcpy.MakeFeatureLayer_management(suot, 'suo_lyr')
   arcpy.MakeFeatureLayer_management('oja_bufferi', 'bufferi_lyr')
   arcpy.Clip_analysis('suo_lyr', 'bufferi_lyr', 'ojitettu_suo')
   arcpy.Erase_analysis(suot, 'oja_bufferi', 'ojittam_suot') # Ei tallenna, toploginen virhe?
#    arcpyDelete_management(suot)

   # digiriistaosio
   arcpy.env.snapRaster = arcpy.env.extent = FME_MacroValues['extent']
   arcpy.Dissolve_management('ojittam_suot', 'ojittamaton_dissolve')
   arcpy.MultipartToSinglepart_management('ojittamaton_dissolve', 'ojittamaton_explode')
   arcpy.MakeFeatureLayer_management('ojittamaton_explode', 'explode_lyr')
```

```
   arcpy.SelectLayerByAttribute_management(in_layer_or_view = 'explode_lyr',
where_clause = 'SHAPE_AREA > 5000')
   arcpy.CopyFeatures_management('explode_lyr', 'ojittamaton_5ha')
   valiRaster = FME_MacroValues['ojittamaton_suo']#[:-4] + '_c.tif'
   arcpy.PolygonToRaster_conversion(in_features = 'ojittamaton_5ha', value_field
='OBJECTID', out_rasterdataset = valiRaster, cellsize = 64)
   arcpy.AddMessage('1')
   coni = Con(valiRaster, 1, '', 'Value > 0')
   arcpy.AddMessage('2')
   coni.save(FME_MacroValues['ojittamaton_suo'])

   try:
      EucDistance('ojittamaton_5ha', 100, 64).save(FME_MacroValues['euc_ojitta-
maton'])
   except:
      arcpy.AddMessage('\"ojittamaton_5ha\" aineistossa vikaa, ei voitu suorittaa Eucle-
dian Distancea.')
   try:
      EucDistance('ojitettu_suo', 100, 64).save(FME_MacroValues['euc_ojitettu'])
   except:
      arcpy.AddMessage('\"ojitettu_suo\" aineistossa vikaa, ei voitu suorittaa Eucledian
Distancea.')
   arcpy.CheckInExtension('Spatial')
```

Liite 2 selitys:

1. Valitaan suot maakunnittain ja tehdään niistä väliaikainen tiedosto.
2. Valitaan ojat maakunnittain ja tehdään niistä väliaikainen tiedosto.
    a. Lapin ojat jaetaan kahteen osaan lukumäärän perusteella, jotka yhdistetään yhdeksi bufferoinnin jälkeen.
3. Bufferoidaan ojat.
4. Käytetään erase toimintoa suo aineistoon ja leikataan oja bufferi pois, jolloin saadaan ojittamattomat suot.
5. Yhdistetään suot yhdeksi kokonaisuudeksi dissolve toiminnolla.
6. Eritellään suot toisistaan multiPartToSingePart / explode toiminnolla, jolloin suot, jotka koskettavat toisiaan muodostavat yhden kokonaisuuden.
7. Valitaan luodusta suo aineistosta kaikki yli 5 ha suot ja luodaan niistä väliaikainen tiedosto.
8. Tehdään käänteinen valinta ja luodaan niistä oma väliaikainen tiedosto.
9. Suoritetaan euclidean distance yli 5 ha suo aineistolle.
10. Muutetaan euclidean distance rasterin nollat NoDataksi.
11. Muutetaan yli 5 ha suo aineisto rasteriksi.
12. Poistetaan väliaikainen suo aineisto ja varmistetaan aineiston eheys Repair gemoteryn toiminnolla tarvittaessa.
13. Suoritetaan clip toiminto suo aineistolle käyttäen oja aineistoa, jolloin saadaan ojitettu suo aineisto.
14. Valitaan alle 5 ha suo aineistosta kaikki, jotka koskettavat ojitettu suo aineistoa ja yhdistetään molemmat aineistot yhdeksi aineistoksi.
15. Suoritetaan euclidean distance ojitetut suo aineistolle ja muutetaan nolla arvot NoDataksi.

Liite 3. Skripti koostuu kahdesta erillisestä tiedostosta, paa.py ja funktiot.py. Liite 3 on pää Python skripti, joka kutsuu funktioita ja ohjaa prosesseja. Skriptit näyttävät sekavalta, mutta kopioituna muistioon/notepadiin näyttävät niin kuin niiden pitäisi.

```
import os
import sys
from time import sleep
from multiprocessing import Process, Manager
from arcpy import env, CheckOutExtension, CheckInExtension, CopyFeatures_manage-
ment, AddMessage
from arcpy.sa import CellStatistics, Float, Aggregate, Con
from collections import namedtuple
osoite = os.path.dirname(os.path.realpath(__file__))
sys.path.append(osoite)
import funktiot

with open('{0}\\attribuutit.txt'.format(osoite), 'r') as tied:
        attriLista = tied.read().split(';;')
paaOsoite = attriLista[1]
valiOsoite = '{0}\\vali\\'.format(paaOsoite)
env.scratchWorkspace = env.workspace = valiOsoite
CheckOutExtension('Spatial')
env.overwriteOutput = True
env.snapRaster = env.extent = attriLista[4]
if not os.path.exists(valiOsoite): os.makedirs(valiOsoite)

# Jaetaan tehtavat ytimille
def ydin1(kanaNimiLis, valiPo, paaPol, krigLista, suomiMaski, kana1, kana2, tyhjaLis,
aineistoLis, MTK, vesistoLis, luontoLis):
        funktiot.VMItoimenpide(tyhjaLis[4], aineistoLis)
        funktiot.krigiFunk(kanaNimiLis[0], kana1.split('\\')[-1][:-4], valiPo, paa-
Pol, krigLista, suomiMaski)
        funktiot.krigiFunk(kanaNimiLis[1], kana2.split('\\')[-1][:-4], valiPo, paa-
Pol, krigLista, suomiMaski)
```

```
		funktiot.merge(vesistoLis[0], vesistoLis[1], vesistoLis[2], vesistoLis[3], MTK)
		funktiot.merge(luontoLis[0], luontoLis[1], luontoLis[2], luontoLis[3], MTK)

def ydin2(tyhjaLis, aineistoLis, tasLis1, tasLis2, valiPo, paaPol, MTK, ihmisRaLi, IhmisLinLis, zonaNimi, listatut):
		funktiot.VMItoimenpide(tyhjaLis[1], aineistoLis)
		funktiot.VMItoimenpide(tyhjaLis[0], aineistoLis)
		funktiot.VMItoimenpide(tyhjaLis[2], aineistoLis)
		latvus = funktiot.oikea(aineistoLis, 'latv')
		funktiot.lk_mvmi(tasLis1, latvus)
		funktiot.tiheys(tasLis2)
		funktiot.ihmiset(MTK, ihmisRaLi, listatut, IhmisLinLis, paaPol, zonaNimi)

def ydin3(tyhjaLis, aineistoLis, tasLis1, tasLis2, zonaHilaNimi, puulajienmaara = 3):
		funktiot.VMItoimenpide(tyhjaLis[3], aineistoLis)
		funktiot.VMItoimenpide(tyhjaLis[5], aineistoLis)
		tilavuus = funktiot.oikea(aineistoLis, 'tila')
		funktiot.lk_mvmi(tasLis1, tilavuus, zonaHilaNimi)
		# Varmistetaan, etta kaikki tiedostot on kasitelty. Parempaa ratkaisua odotellessa.
		while len(aineistoLis) < 6:
			sleep(10)
		manty, muulp, kuusi, koivu, tilavuus = Float(funktiot.oikea(aineistoLis, 'manty')), funktiot.oikea(aineistoLis, 'muulp'), funktiot.oikea(aineistoLis, 'kuusi'), funktiot.oikea(aineistoLis, 'koivu'), funktiot.oikea(aineistoLis, 'tila')
		maksi = (1 - (CellStatistics([manty, muulp, kuusi, koivu], 'MAXIMUM', 'DATA') / tilavuus)) / (1 - (1/puulajienmaara))
		funktiot.lk_mvmi(tasLis2, maksi)

if __name__ == '__main__':
		# Zonationiin menevat tiedostojen nimet
		suoLista = (attriLista[10], attriLista[11], attriLista[12])
```

```
zonaRasterLista = tuple(map(lambda x: '\\'.join([paaOsoite, x]), ('tila-
vuus.tif', 'tiheys.tif', 'latvus.tif', 'puulajit.tif', 'hilaMaara.tif', 'luonto.tif', 'metepy.tif',
'riekko.tif', 'ihmiset.tif', 'vesistot.tif')))

pohjaAineisto, krigiLista, = Manager().list(), Manager().list()
VMINimi = ('tilavuus_', 'latvuspeitto_','manty_','kuusi_','muulp_','koivu_')
kanaKokoelma = funktiot.kanaAineisto(valiPo = valiOsoite, riistaOsoite =
attriLista[7],  koordit = attriLista[8], nimi1 = zonaRasterLista[6], nimi2 = zonaRaster-
Lista[7])

suomi = funktiot.GDBTarkistus(valiOsoite) + 'Suomimaski'
CopyFeatures_management(attriLista[3], suomi)
vesistoLista = (('36200', 'e_44300', '38700', '38600', '36211', '36313',
'38300'), 'in_memory\\kaikkivesistot', 'vesistoraster.tif', zonaRasterLista[9])
luontoLista = (('39110', '34300', '32800', '35411', '35421', '39130'), 'in_me-
mory\\kaikkiLuonto', 'luontoraster.tif', zonaRasterLista[5])
ihmisRakLista = (
                                                '32611', '42241', '42242', '42250',
'42251', '42251', '42260', '42261', '42262', '42270'
                                                , '42210', '42211', '42212', '42220',
'42221', '42222', '42230', '42231', '42232', '42240'
                                                , '38900', '40200', '32500', '32421',
'32418', '32417', '32416', '32415', '32414', '32413'
                                                , '32412', '32411', '32300', '33000',
'32113', '32112', '32111')

suuretIhmisTiedosto = ('32611', '42211', '42231', '42261') # Listattu 4 suu-
rinta tiedostoa "ihmisRakLista" muuttujasta
ihmisLinjatLista = (('22311', '14111', '12122', '12112', '12121', '12111'),
'in_memory\\kaikkiviivat', 'viivarcl.tif')
MTKAineisto = attriLista[0]
tyhjaLista = funktiot.pohjaLista(lukuOsoite = attriLista[5] + '\\', haLista =
VMINimi)

hilaKentta = ('puulajienlkm', 'v', 'latvuspeittavyys5', 'tiheys5')
```

```
riistaHila = attriLista[9]

tas1Lista = (riistaHila, zonaRasterLista[3], '32_BIT_FLOAT', hila-
Kentta[0])
tas2Lista = (riistaHila, zonaRasterLista[0], '16_BIT_UNSIGNED', hila-
Kentta[1])
tas3Lista = (riistaHila, zonaRasterLista[2], '8_BIT_UNSIGNED', hila-
Kentta[2])
tas4Lista = (riistaHila, zonaRasterLista[1], '8_BIT_UNSIGNED', hila-
Kentta[3])

# Rinnakkais prosessit
pros_ydin1 = Process(target = ydin1, args = (kanaKokoelma, valiOsoite,
paaOsoite, krigiLista, suomi, zonaRasterLista[6], zonaRasterLista[7], tyhjaLista, poh-
jaAineisto, MTKAineisto, vesistoLista, luontoLista))
pros_ydin2 = Process(target = ydin2, args = (tyhjaLista, pohjaAineisto,
tas3Lista, tas4Lista, valiOsoite, paaOsoite, MTKAineisto, ihmisRakLista, ihmisLinjat-
Lista, zonaRasterLista[8], suuretIhmisTiedosto))
pros_ydin3 = Process(target = ydin3, args = (tyhjaLista, pohjaAineisto,
tas2Lista, tas1Lista, zonaRasterLista[4]))

pros_ydin1.start()
pros_ydin2.start()
pros_ydin3.start()

pros_ydin1.join()
pros_ydin2.join()
pros_ydin3.join()

CheckInExtension('Spatial')
del pohjaAineisto, suomi, VMINimi, valiOsoite, MTKAineisto,
pros_ydin1, pros_ydin2, pros_ydin3
del vesistoLista, luontoLista, ihmisRakLista, ihmisLinjatLista, kanaKoko-
elma, tyhjaLista, krigiLista
```

```
del tas1Lista, tas2Lista, tas3Lista, tas4Lista, suuretIhmisTiedosto, riista-
Hila, hilaKentta


znTiedosto = namedtuple('Rasteri', 'painotukset tiedosto')
kernel_10km = round(2.0 / ((1000 / 64) * 64), 5) # etaisyys / solukoko =
hilojen maara. Yksikkona metri
kernel_xkm = round(2.0 / ((x / 64) * 64), 5)
#variantti 1
pk1 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[0])
pk2 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[1])
pk3 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[2])
pk4 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[3])
pk5 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[4])
#variantti 2
pk6 = znTiedosto(painotukset = '2.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = suoLista[0])
pk7 = znTiedosto(painotukset = '3.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = suoLista[1])
pk8 = znTiedosto(painotukset = '1.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = suoLista[2])
#variantti 3
pk9 = znTiedosto(painotukset = '0.5 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[5])
pk10 = znTiedosto(painotukset = '1.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[6])
pk11 = znTiedosto(painotukset = '1.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[7])
pk12 = znTiedosto(painotukset = '-1.0 {0} 1 1 0.25'.format(kernel_10km),
tiedosto = zonaRasterLista[8])
```

```
pk13 = znTiedosto(painotukset = '-1.0 {0} 1 1 0.25'.format(kernel_10km), tiedosto = zonaRasterLista[9])

#zonation
znMask = attriLista[2]
matriisi = ''
interaktio = ''
BLP = ''
if znMask.lower() == 'none':
znMask = None
variaLista = (pk1, pk2, pk3, pk4, pk5, pk6, pk7, pk8, pk9, pk10, pk11, pk12, pk13)
variaTiedMaara = (5, 8, 13)
for x in range(len(variaTiedMaara)):
znBatLista = (attriLista[6], paaOsoite + '\\Zona\\drm_zonation{0}'.format(x), '0.0 1 1.0 1', 'drm_zonation{0}.bat'.format(x), 'drm_zonation{0}_out\\drm_zonation{0}.txt'.format(x))
znDatLista = ('2', '10000', '1', '0', '1', znMask, 'drm_zonation{0}.dat'.format(x))
znSPPLista = ['drm_zonation{0}.spp'.format(x)]
znSPPLista.extend(variaLista[:variaTiedMaara[x]])
znBatPolku = paaOsoite + '\\Zona\\drm_zonation{0}\\drm_zonation{0}.bat'.format(x)
if x >= len(variaTiedMaara)-1:
matriisi = ('drm_zonation{0}_matrix.txt'.format(x), '1', '1'
, ''
, '0.5'
, '0.9 0.3'
, '0.2 0.5 0.3'
, '0.6 0.8 0.7 0.7'
```

```
                                                                    ,  '0.1
0.1 0.1 0.1 0.1'
                                                                    ,  '0.3
0.3 0.5 0.6 0.1 0.8'
                                                                    ,  '0.3
0.3 .05 0.6 0.1 0.6 0.95'
                                                                    ,  '0.1
0.1 0.1 0.1 0.1 0.1 0.1 0.1'
                                                                    ,  '0.2
0.2 0.2 0.2 0.3 0.2 0.2 0.2 0.1'
                                                                    ,  '0.1
0.1 0.1 0.1 0.3 0.9 0.8 0.6 0.1 0.4'
                                                                    ,  '0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.5 0.01 0.01'
                                                                    ,  '0.01
0.01 0.01 0.01 0.01 0.01 0.01 0.01 0.5 0.01 0.01 0.6')
                                        interaktio2  =  """('drm_zonation{0}_interact-
ions.txt'.format(x), '1'
                                                                   ,'1    1
{0} 1 1'.format(kernel_xkm)
                                                                   )"""
                                        #BLP = '0.1'
                                funktiot.zona(znBatLista,  znDatLista,znSPPLista,  znBat-
Polku, matriisi, interaktio, BLP)
```

Liite 4. Funktio skripti, josta pää skripti kutsuu funktioita.

```
import arcpy
import os
from time import sleep
from datetime import datetime
from arcpy.sa import Con, Raster, ExtractByMask, Aggregate, Kriging, Kriging-
ModelOrdinary, RadiusVariable, Reclassify, RemapRange, LineStatistics, IsNull
from multiprocessing import Manager
arcpy.env.overwriteOutput = True

# Pidetaan huoli, etta oikea tiedosto menee muuttujaan
def oikea(kokoelma, haettava):
        for k in kokoelma:
                if haettava in k:
                        return Raster(k)

# Loytyyko geodatabase ja luodaan tarvittaessa
def GDBTarkistus(vaPo, geDB = '\\vali.gdb'):
        paikka = vaPo + geDB
        if not arcpy.Exists(paikka):
                arcpy.CreateFileGDB_management(vaPo[:-1], geDB)
        return paikka + '\\'

# Kaydaan kansio lapi ja palautetaan lista, jossa on haetut tiedostopolut
def pohjaLista(lukuOsoite, haLista):
        omaLista = []
        tiedOso = arcpy.da.Walk(lukuOsoite)
        for dirpath, dirnames, filenames in tiedOso:
                for filename in filenames:
                        tied = filename.lower()
                        for haku in haLista:
                                if haku in tied and 'bm' not in tied
and 'lehti' not in tied:
```

```
                                                    omaLista.ap-
pend(os.path.join(dirpath, filename))
        return sorted(omaLista)

# Kirjoitetaan kaytetty aika lokiin, joka sijaitsee skriptin kanssa samassa kansiossa. Lii-
tetaan halutun funktion loppuun ja alkuun laitetaan muuttuja ja sille arvo datetime.now().
def ajastus(aika, tulosNimi, tyhjennetaan = False):
        aika_tiedosto = '{0}\\aika_tiedosto.txt'.format(os.path.dir-
name(os.path.realpath(__file__)))
        if not os.path.exists(aika_tiedosto) or tyhjennetaan:
                with open(aika_tiedosto, 'w') as tiedosto:
                        tiedosto.write(str(datetime.now()))
        erotus = datetime.now() - aika
        with open(aika_tiedosto, 'a') as tiedosto:
                tiedosto.write('\n{0}:  \t{1}'.format(funkNimi,  str(ero-
tus).split('.')[0]))

# MVMI aineistojen kasittely
def VMItoimenpide(tiedosto, niLista):
        VMInimi = tiedosto.split('\\')[-1][:6] + '_con.tif'
        rasTied = Raster(tiedosto)
        aggre = Aggregate(rasTied, 4, 'MEAN')
        Con(aggre, aggre, '', 'Value > 0 AND Value < 1000').save(VMInimi)
        niLista.append(VMInimi)

# Kasitellaan piirrekerrokset 1-4
def lk_mvmi(lista, mvmi, hilaMaaraNimi = ''):
        nimi = lista[3][:4] + '.tif'
        arcpy.FeatureToRaster_conversion(lista[0], lista[3], nimi, 16)
        aggre = Aggregate(nimi, 4, 'MEAN')
        if hilaMaaraNimi:
                coni = Con(nimi, 1, '', 'Value > 0')
                Aggregate(coni, 4, 'SUM').save(hilaMaaraNimi)
        coni = Con(IsNull(aggre) == 1, mvmi, aggre)
```

```
        if arcpy.GetRasterProperties_management(coni, 'VALUETYPE') != lista[2]:
                nimi = 'coni_kopio.tif'
                arcpy.CopyRaster_management(coni, nimi, pixel_type = lista[2])
        else:
                nimi = coni
        Con(nimi, nimi, '', 'Value > 0').save(lista[1])

# Kasitellaaj tiheys tiedostoa
def tiheys(tasLis):
        arcpy.PolygonToRaster_conversion(tasLis[0], tasLis[3], 'tiheys.tif', '', '', 16)
        aggre = Aggregate('tiheys.tif', 4, 'MEAN')
        arcpy.CopyRaster_management(aggre, 'tiheysraster.tif', pixel_type = tasLis[2])
        Con('tiheysraster.tif', 'tiheysraster.tif', '', 'Value > 0').save(tasLis[1])

# Ihmisen luoma linja, esim tiet
def IhmisLinjat(pk10lista, tyoTaso):
        ihmisLista = pohjaLista(lukuOsoite = tyoTaso, haLista = pk10lista[0])
        merge(ihmisLista, pk10lista[1])
        viivaStats = LineStatistics(pk10lista[1],'None', 64, 64, 'LENGTH')
        viivaHaku30 = viivaStats >= 30
        arcpy.CopyRaster_management(viivaHaku30, pk10lista[2], pixel_type = '1_BIT')

# Kasitellaan ihmisten aukot, joista erotellaan suurimmat tiedostot ja kasitellaan ne yksi-tellen
def ihmiset(MTK, ihmisRaLi, listatut, IhmisLinLis, paaPol, zonaNimi):
        rasLista, rclLista = tuple(map(lambda x: 'ras{0}.tif'.format(x), range(5))), tuple(map(lambda x: 'rcl{0}.tif'.format(x), range(5)))
        pk10Lista = pohjaLista(MTK, ihmisRaLi)
        suuret = set([(p) for p in pk10Lista for l in listatut if l in p])
        uusiLista = [set(pk10Lista) - suuret]
        uusiLista.extend(suuret)
```

```
		merge(tuple(uusiLista[0]),  '\\vali.gdb\\valiMerge', rasLista[0], rclLista[0])
		merge(uusiLista[1], uusiLista[1], rasLista[1], rclLista[1])
		merge(uusiLista[2], uusiLista[2], rasLista[2], rclLista[2])
		merge(uusiLista[3], uusiLista[3], rasLista[3], rclLista[3])
		merge(uusiLista[4], uusiLista[4], rasLista[4], rclLista[4])
		IhmisLinjat(IhmisLinLis, MTK)
		arcpy.MosaicToNewRaster_management([rclLista,  IhmisLinLis[2]],  paa-
Pol, zonaNimi[len(paaPol):], '', '1_BIT', 64, 1, 'FIRST')

# Muutetaan tiedosto rasteriksi ja tapauskohtaisesti luodaan ja yhdistetaan lista
def merge(mergeLista, mergeNimi, rasteriNimi = '', rclNimi = '', maastoTietoKanta = ''):
		if maastoTietoKanta:
				mergeLista = pohjaLista(maastoTietoKanta, mergeLista)
		if type(mergeLista) in (list, tuple, set):
				arcpy.Merge_management(mergeLista, mergeNimi)
				mergeLista = mergeNimi
		if rasteriNimi:
				arcpy.PolygonToRaster_conversion(mergeLista, 'LUOKKA',
rasteriNimi, 'MAXIMUM_COMBINED_AREA', '', 64)
				Reclassify(rasteriNimi, 'Value', RemapRange([[1,99999, 1]]),
'NODATA').save(rclNimi)

# Kriging toimenpide
def krigiFunk(tiedosto, kanaNimi, valiPo, talPo,  pohAi, suomi, etaisyys_m = 50000):
		kriNimi = '{0}\\{1}.tif'.format(talPo, kanaNimi)
		conimi= 'con{0}'.format(kanaNimi)
		krigi = Kriging(tiedosto, 'SUM_n', KrigingModelOrdinary('GAUSSIAN', '',
etaisyys_m, 9, 2), 64, RadiusVariable(12, etaisyys_m))
		sleep(3)
		arcpy.CopyRaster_management(krigi,  conimi,  pixel_type  =  '8_BIT_UN-
SIGNED')
		extrac = ExtractByMask(conimi, suomi)
		Con(extrac > 0, extrac).save(kriNimi)
		pohAi.append(kriNimi)
```

```
# Valmistellaan ja eritellaan kanalinnut krigingia varten
def kanaAineisto(valiPo, riistaOsoite, koordit, nimi1, nimi2):
		riista = GDBTarkistus(valiPo) + '\\riistaPisteet'
		valiNimi = '\\vali.gdb\\valiriista'
		kanaLista = (nimi1.split('\\')[-1][:-4], nimi2.split('\\')[-1][:-4])
		arcpy.MakeXYEventLayer_management(riistaOsoite, 'kolmio_keski-
piste_x', 'kolmio_keskipiste_y', valiNimi, koordit)
		arcpy.FeatureToPoint_management(valiNimi, riista)
		arcpy.MakeFeatureLayer_management(riista, 'kana_taso')
		kanaNimiLista = Manager().list()
		for k in kanaLista:
			kanaGDB = GDBTarkistus(valiPo, '\\{0}.gdb'.format(k))
			ilmaisu = 'laji_txt = \'{0}\''.format(k)
			if k == kanaLista[0]:
				ilmaisu = 'laji_txt = \'{0}\' OR laji_txt = \'{1}\'
OR laji_txt = \'{2}\''.format('metso', 'teeri', 'pyy')
			if arcpy.Exists('{0}{1}_86hakutj'.format(kanaGDB, k)):
				arcpy.Delete_management('{0}{1}_86ha-
kutj'.format(kanaGDB, k))
			uusiNimi = '{0}{1}_86hakutj'.format(kanaGDB, k)
			arcpy.SelectLayerByAttribute_management('kana_taso',
'NEW_SELECTION', ilmaisu)
			arcpy.Dissolve_management('kana_taso', uusiNimi + 'dis-
solve', 'knum', [['n', 'SUM'], ['kolmio_keskipiste_x', 'FIRST'], ['kolmio_keskipiste_y',
'FIRST']], 'SINGLE_PART')
			kanaNimiLista.append(uusiNimi + 'dissolve')
		return kanaNimiLista

# Kirjoitetaan zonation tiedostot
def zona(batlista, datlista, spplista, batpolku, matriisi, interaktio, blp):
		if not os.path.exists(batlista[1]):
			os.makedirs(batlista[1])

		# BAT
		with open(batlista[1] + '\\' + batlista[3], 'w') as bat_tiedosto:
```

```
                bat_tiedosto.write('call {0} -r {1} {2} {3} {4}  --use-threads
--grid-output-formats compressed-tif\n'.format(batlista[0], batlista[1] + '\\' + datlista[6],
batlista[1] + '\\' + spplista[0], batlista[1] + '\\' + batlista[4], batlista[2]))

        # Matriisi
        if matriisi:
                with open(batlista[1] + '\\' + matriisi[0], 'w') as matriisi_tie-
dosto:
                        for m in matriisi[3:]:
                                matriisi_tie-
dosto.write('{0}\n'.format(m))

        # DAT
        with open(batlista[1] + '\\' + datlista[6], 'w') as dat_tiedosto:
                dat_tiedosto.write('[Settings]\nremoval rule = {0}\nwarp fac-
tor = {1}\nedge removal = {2}\n'.format(datlista[0], datlista[1], datlista[2]))
                if interaktio:
                        dat_tiedosto.write('\nuse interactions = {0}\nin-
teraction file = {1}\n'.format(interaktio[1], batlista[1] + '\\' + interaktio[0]))
                if datlista[5]:
                        dat_tiedosto.write('\nmask missing areas =
{0}\narea mask file = {1}\n'.format(datlista[4], datlista[5]))
                if matriisi:
                        dat_tiedosto.write('\n[Community analysis sett-
tings]\nload similarity matrix = {0}\nconnectivity similarity matrix file = {1}\napply to
connectivity = {2}\n'.format(matriisi[1], batlista[1] + '\\' + matriisi[0], matriisi[2]))
                if blp:
                        dat_tiedosto.write('\nBLP = {0}'.format(blp))

        # SPP
        with open(batlista[1] + '\\' + spplista[0], 'w') as spp_tiedosto:
                for spp in spplista[1:]:
                        spp_tiedosto.write('{0}     {1}\n'.format(spp[0],
spp[1]))
```

```
        # Interaction
        if interaktio:
                with open(batlista[1] + '\\' + interaktio[0], 'w') as inter-
aktio_tiedosto:
                        for a in interaktio[2:]:
                                inter-
aktio_tiedosto.write('{0}\n'.format(a))

        os.system(batpolku)
```