Eötvös Loránd Tudományegyetem

Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Konkurens vegetáció detektálás és változáselemzés ALS pontfelhőkön

Témavezető:
Dr. Cserép Máté András
egyetemi adjunktus

Szerző:
Papp Péter
programtervező informatikus MSc

Budapest, 2025

Tartalomjegyzék

1. fejezet

Bevezetés

A növényzet és épületek változásai nagy hatással vannak életünkre, ezért fontos, hogy vizsgáljuk őket és beavatkozzunk, amennyiben szükséges. A környzetre hatással van a városiasodás, az iparosodás, de a klíma változás is. A nyomonkövetés segítségével megkönnyíthető a várostervezés, akár a növényzet telepítésének tervezése. Az épület változások vizsgálatával földrengés esetén, akár életet is menthetünk. Tegyük fel, hogy van egy korábbi időpillanatban egy ország épületeiről három dimenziós pontfelhőnk. Földrengés esetén, ha ismét készül egy felvétel ugyanarról a területről, akkor a korábbi pontfelhő összevetésével és különbségeik meghatározásával könnyebben lehet csoportosítani a mentőalakulatokat, ezáltal akár több embert megmentve. A növényzet vizsgálatával mezőgazdasági területek alakulása is megfigyelhető, például művelik-e a területet, ha nem akkor az elfásodás során mely fa fajták lesznek jelen. Erdő tarvágása után, akár könnyítheti a rekonstrukció folyamatát, de az illegális fakitermelést is segíthet észrevenni.

Az Eötvös Loránd Tudományegyetemen egy olyan keretrendszert dolgoztak ki, amely légi lézeres távolságmérésből (airborne laser scanning = ALS) származó hatalmas adatmennyiségek feldolgozására képes. A projekt amelybe becsatlakoztam mind az épület-, mind a növénydetektálással és azok változásaival foglalkozik. Egy ország vizsgálata esetében rendkívül nagy mennyiségű adatról beszélünk, ezért nagyon fontos az optimalizált algoritmusok tervezése és megvalósítása, hiszen egy esetleges katasztrófa esetén, a mentés esetében az idő az egyik legfontosabb kérdés. A növényzet változás vizsgálatánál is egy meghatározó szempont a gyorsaság. Ezért célul tűztem ki, hogy a vegetáció detektáló és változást vizsgáló algoritmuson minél többet gyorsítsak.

Az algoritmus megismerése után megvizsgáltam, hol és hogyan lehetne a legoptimálisabb. A fejlesztés során két féle módszert valósítottam meg. Az egyikben térbeli indexelést, azon belül is negyedelő fát alkalmaztam. A másikban pedig a bejövő adat, csempékre osztását és azoknak párhuzamos feldolgozását, a már meglévő algoritmus komponensek segítségével.

Diplomamunkám során a 2. fejezetben írok a kapcsolódó szakirodalmi háttérismeretről, többek között a LiDAR technológiáról, a digitális domborzatmodellről. Bemutatom a dolgozatban használt térbeli adatok tárolására szolgáló adatszerkezeteket, indexelésüket, majd a párhuzamosításhoz szükséges csempe alapú feldolgozást és a "map reduce" modellt, amelyet alapul vettem a csempésítés folyamatában.

A 3. fejezetben a módszertan keretein belül bemutatom a projektet, amelyen dolgoztam. Részletezem az optimalizálni kívánt algoritmus lépéseit, bemutatom a szűk keresztmetszeteit, a területet, amelyen a futtatás megvalósult. Szó lesz a csempe alapú párhuzamosított feldolgozásról, az algoritmusban alkalmazott negyedelő fákról.

Az 4. fejezetben írok a használt eszközökről, könyvtárakról. Bemutatom az új folyamatábrát és az általam készített/módosított osztályokat.

Az 5. fejezetben bemutatom a kiindulási algoritmus által tesztelt területeken, hogy a negyedelőfa, és a csempealapú párhuzamosítás külön-külön mennyit gyorsított az algoritmuson és egyben, továbbá összehasonlítom a korábbi eredményekkel. Kielemzem a kapott futási és fa detektálási eredményeket.

2. fejezet

Szakirodalmi áttekintés

Ebben a fejezetben kifejtem a diplomamunkám során érintett témákat, a program optimalizálásához használt módszerek megértéshez szükséges szakirodalmi háttérismereteket. A lézeralapú távérzékelés technológia segítségével állnak elő a bemeneti adatok, ezért írok kialakulásáról, felhasználásáról, működéséről. A térinformatikában elengedhetetlen a digitális domborzatmodell ismerete, amely magába foglalja a digitális felszín és digitális terepmodellt.

A dolgozat egyik központi témája a térbeli adatmodellek és térbeli indexelés. A dolgozat szempontjából ismeretük fontos, hiszen a negyedelő-fa is egy ilyen adatstruktúra, amelyet a programban több helyen tudtam használni. A négyfák több típusát is bemutatom.

Bemutatásra kerül a dolgozat másik fő témája a csempealapú párhuzamos feldolgozás, amelyet a térinformatikában megjelenő hatalmas adatmennyiség miatt több helyen használt MapReduce technológia ötletével vegyítettem.

2.1. LiDAR

A lézeralapú távérzékelés (Light detection and ranging = LiDAR) a fényvisszaverődés elvére épül. Működése során távolságot lehet meghatározni egy fényt kibocsátó eszköz és egy visszaverődési felület között.

2.1.1. Története

Az első LiDAR alapú rendszert 1961-ben Malcolm Stitch mutatta be, egy évvel a lézerfény felfedezése után. Kezdetben koherens lézeralapú távérzékelésnek (Coherant

2.1. ábra. Különböző LiDAR eszközök
Forrás:[1]

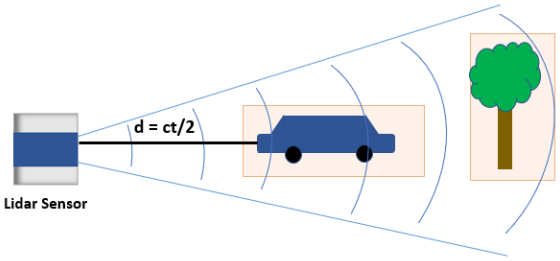
light detection and ranging = CoLiDAR) nevezték és főként katonai célokra használták. A LiDAR kifejezést 1963-tól kezdték használni, a „light"(= fény) és „radar" kifejezések kombinálásával. LiDAR gyors fejlődésének köszönhetően folyamatosan bővült felhasználásának lehetőségei. 1970-es években fő felhasználási irányai voltak a jégtakarók feltérképezése, erdők lombkoronáinak meghatározása, óceánok, atmoszféra tulajdonságainak mérése. Pontos LiDAR képalkotási adatokat az 1980-as évektől lehetett készíteni, a Globális Helymeghatározó Rendszer (Global Positioning System = GPS) és (Inertial Measurement Units = IMU) megjelenéseiket követően.[2]

2.1.2. Működése

Működése során az eszköz lézerfényt bocsájt ki, amely levegőben, vízben, légkörben terjed. A fény objektummal való találkozása esetén a fény egy része szétszóródik, egy része visszaverődik az érzékelő felé. A visszavert fénysugarakat a LiDAR érzékeli. A visszaverődés idejét mérve és a fénysebesség ismeretében könnyen meghatározható a megtett távolság, amelyet a következő egyenlettel tudunk meghatározni

$$D = c\left(\frac{\Delta T}{2}\right),$$

ahol D: távolság az mérőeszköz és az objektum között, c: fénysebesség, T: a fény kibocsátása és visszaérkezése között eltelt idő.[2]



2.2. ábra. LiDAR működése
Forrás: `https://de.mathworks.com/help/lidar/ug/lidar-processing-overview.html`

2.1.3. Típusai és felhasználási területei

A LiDAR-nak alapvetően négy típusát különböztetjük meg.[3] A fénysebesség gyorsaságának köszönhetően a LiDAR alapú rendszerek rendkívül gyorsak és pontosak, ennek következtében rengetek területen használják őket. A mozgásban lévő járművekre szerelt LiDAR mellett szükség van egyéb szenzorokra (GPS, IMU) is az adatok meghatározására.

Légi LiDAR

A légi LiDAR-t repülőgépekre, drónokra szerelik. Segítségével olyan területeket is tudnak vizsgálni, amelyek megközelíthetetlenek. Nagyléptékű térképezéshez, környezeti megfigyelésekhez, változáselemzéshez használják.

Földi LiDAR

A földi LiDAR-t általában állványra szerelik, amely környezetét függőleges, vagy vízszintes síkban pásztázza. Főbb felhasználási területei a infrastruktúra állapotának felmérése, földmérés, építészeti modellezés.

Mobil LiDAR

A mobil LiDAR mozgás közben tudja végezni a méréseket, amelyeket autóra, kamionra, hajóra, vonatra szerelve közlekedési infrastruktúrák minőségének felmérésére, tervezésére használják.

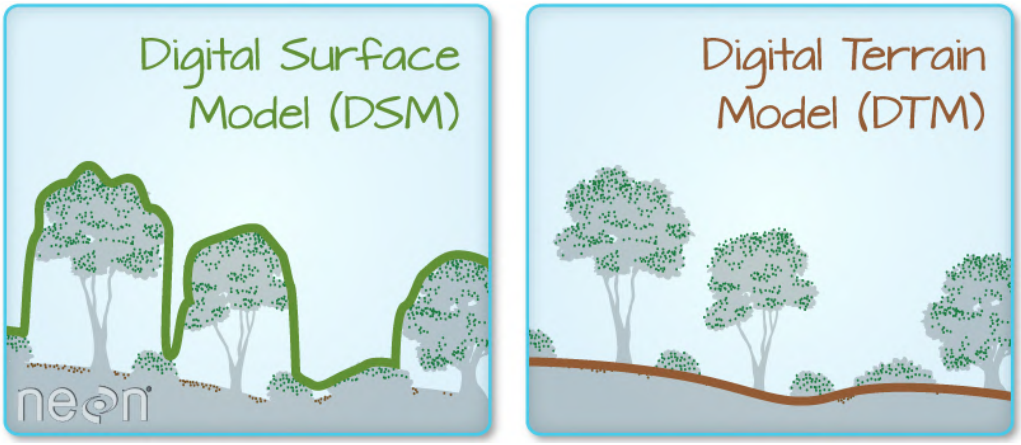
Űrbéli LiDAR

Az űrbeli LiDAR-t műholdakra, vagy űrszondákra telepítik. Segítségükkel a Föld légkörét, domborzatát és növényzetét tudják vizsgálni, következtetéseket levonni.

2.2. Digitális domborzatmodell

Ebben a témakörben nem alakult ki egy egységes szóhasználat, ezért előfodulhat, hogy a következő fogalmaknak több féle értelmezése is lehet, a dolgozatomban a gyakrabban használt definícióit ismertetem. A Digitális Domborzatmodell (Digital Elevation Model = DEM) magában foglalja a Digitális Terepmodellt (Digital Terrain Model = DTM) és a Digitális Felszínmodellt (Digital Surface Model = DSM). A DEM-et lehet készíteni földfelszín felületről, de lehet mélyebben húzódó geológiai rétegről, illetve talajvíz-tükör felületről is. A DEM-ek alapvetően olyan adatbázisok, amely topográfiai felület térbeli koordinátáit tárolják. A DEM-et széles körben használják környezetünk leírására, elemzésére. Segítségével fel lehet mérni erdők faállományait, azok térbeli eloszlását, törzsvastagságukat, méretüket. Használható még talajpusztulás felmérésére, már nem működő vulkán rekonstruálására, hogy milyen volt aktív korában, szintkülönbségek számolására, árvízeknél vésztározók hatékonyságának számolására. Erdők és épületek változáselemzéseire és még nagyon sok mindenre alkalmazhatóak. 2,5-dimenziós modellnek is szokták hívni, hiszen az áthajlások függőlegesnél is meredekebb felületekre általában nem megoldott, de vannak olyan esetek, amikor a tényleges 3D megkövetelt.

Ahogy a 2.3. ábra is mutatja a DTM és DSM közti különbség annyi, hogy míg a DTM a tereptárgyak nélkül a szilárd kőzetburok felszínét modellezi, addig DSM a szilárd kőzetburok mellett a tereptárgyakat is beleveszi a modellbe. Föld felszíni DSM-et leghatékonyabb módon repülőre szerelt LiDAR-al lehet készíteni, míg DTM-et szintén LiDAR-al a felszín bejárásával.

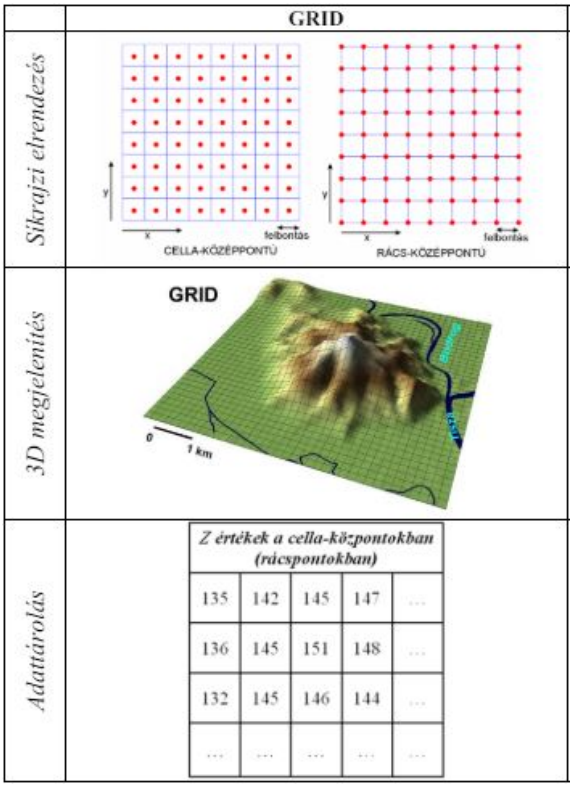


2.3. ábra. Digitális Felszínmodell és Digitális Terepmodell
Forrás: `https://github.com/NEONScience/NEON-Data-Skills`

Sokan a DEM-et a rács (= GRID) adatszerkezetű modellekkel azonosítják, de tágabb értelemben vett definíciója alapján több fajtáját különböztetjük meg aszerint, hogy milyen módon tárolja az egyes pontokat. Az egyik a GRID modell, amely a raszteres modellek közé sorolható, másik a Szabálytalan Háromszögháló (Triangulated Irregular Network = TIN), amely vektoros modell. Az előbb említett modelleken kívül léteznek hibrid modellek, amelyek a raszteres és vektoros modelleket ötvözik. A GRID modellt mutatom be részletesebben.[4]

Rács típusú domborzatimodellek

Ahogy a 2.4. ábra mutatja a területet téglalapokból, négyzetekből álló szabályos rácshálóval fedjük le. A rácsháló könnyen ábrázolható egy mátrixszal, amelyben a szabályos elrendezés miatt elég tudni a sarokkoordinátákat és a rácstávolságokat, amiből könnyen meghatározható bármely másik pont. Ennek előnye a könnyű algoritmizáció. Két szomszédos rácsvonal távolságát felbontásnak nevezzük, minél kisebb ez a távolság annál pontosabb a modellünk, viszont annál több hardver igénye van az adatokon történő műveleteknek. A magasság megadása jelentheti az adott pont magasságát, de lehet a cellányi terület átlagértéke is. Felszín magasság esetében is két féle megadás létezik, az egyik amikor a négyzetek középpontjában (cella középpontú), a másik mikor a rácspontokban (rács-középpontú) adjuk meg.[4]



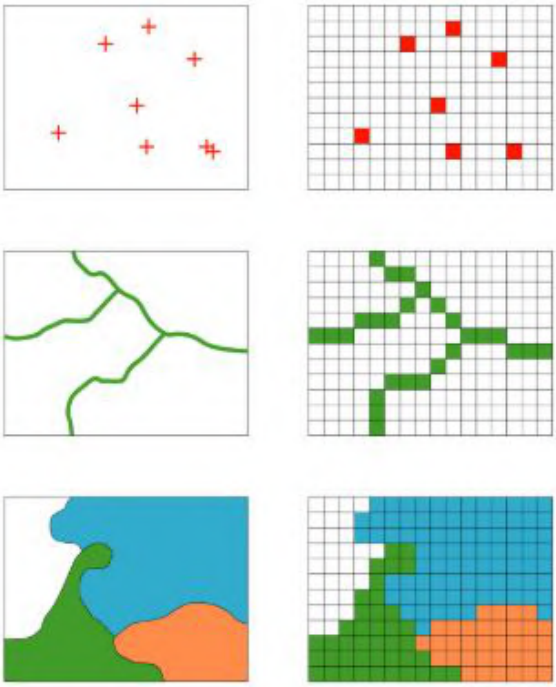
2.4. ábra. GRID modell
Forrás:[4]

2.3. Térbeli adatok tárolása és indexelése

Térbeli adatoknak tekintjük a minket körülvevő mesterségesen, vagy természetesen keletkező objektumokat. A több dimenziós térbeli adatok hatékony reprezentálása, tárolása fontos kérdés az applikációk tervezésénél. Térbeli adatok például a városok, utak, folyók, termőföldek, megyék, hegyvonulatok, stb, amelyeket pontokkal, vonalakkal, régiókkal, felületekkel, térfogattal tudunk reprezentálni. Sokszor szoktak még a térbeli adatok mellé egyéb nem térbeli információkat is csatolni például magasságértékeket, városneveket stb.[5] A térbeli komponens és a leíró komponens közti konzisztencia nehezen tartható fenn. Nagyméretű térinformaitkai rendszerek (Geographical Information System = GIS) esetén ezt a konzisztenciát fent kell tartani, ezért szükség van speciális adatmodellekre. Ezek az adatmodellek térbeli objektumok esetében raszteres, illetve vektoros adatmodellek általában.

2.3.1. A térinformatikai adatmodellek

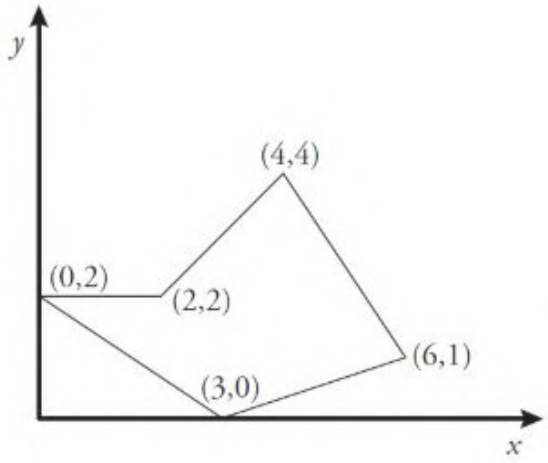
Raszteres adatmodell bemenete egy bármilyen megfigyelt területről, objektumról készített digitális kép, amelyet képalkotó rendszerek egy nxm-es képmátrixra képeznek le és elemeihez pixeleket rendel. A pixelek azonos dimenziójú egész típusú vektorok, amelynek dimenziói a sávok száma, értékei a sávokban mért intenzitás értékek. Raszteres adatmodellek tárolnak továbbá egyéb fontos georeferenciai adatokat, rasztert felépítő pixelek sorainak, oszlopainak számát. A modell előnye, hogy könnyen vizsgálhatóak a geometriai relációk, a mintavételezés gyors, egyenletes. Hátránya, hogy minden pixelt ismerni kell, ezért az adatbázisnak nagyméretűnek kell lennie, nehezen definiálhatók az objektumok, nehezen kiépíthető relációs adatbázis-kapcsolatok.



2.5. ábra. Pontok, vonalak és poligonok raszteres rendszerben
Forrás:[6]

Vektoros adatmodellek esetén a terület, objektum reprezentálása helyvektorokkal történik, oly módon, hogy az objektum jellegzetes pontjait helyvektoruk koordinátáival adjuk meg, illetve egy összekötési szabállyal garantáljuk, hogy az alakzatnak megfelelő pontok legyenek összekötve. Az alakzatok az objektumdefiníciókban vannak leírva. Megkülönböztetünk pontszerű objektumot, vonalas objektumot és területi objektumot. Pontszerű objektumnál elég a koordinátáit eltárolni. A vonalas objektum egy vonallánc, amelynél a töréspontok koordinátáit tároljuk el. A területi

annyiban különbözik a vonalas objektumtól, hogy a vonallánc zárt és itt a csúcspontokat tároljuk. Előnye, hogy komplex objektumokat tudunk építőelemei segítségével felépíteni és az objektumokat intelligens kapcsolatokkal lehet összekötni. Előnye, hogy kevés adat mennyiséggel is képes a felületet pontosan leírni. Lehet vele készíteni objektumorientált megközelítésű rendszereket. Mintavételezése egyenletes, gyors.[6]



2.6. ábra. Poligon vektoros ábrázolása
Forrás:[6]

2.3.2. Térbeli indexelés

Az egydimenziós indexstruktúrák nem használhatóak térbeli alakzatok lekérdezéséhez, amelynek fő okai, hogy a többdimenziós teret nem lehet távolságtartással egydimenziósra leképezni, illetve nehézkes az adatmozgatás, továbbá nagy tárigénye van. Ahogy a térinformatikai adatok tárolására külön módszereket kellett kidolgozni, így a ezek indexeléseire és az adatok hatékony lekérdezésére különböző térbeli indexelési algoritmust hoztak létre. Egy térkép esetében általában egy adott részére vagyunk kíváncsiak, ezért fölösleges az összes elemet beolvasni, ami nagyban lelassítaná a feldolgozást.

Nagyon sok féle térbeli indexelési struktúra létezik, amelyek osztályozása korántsem triviális a sok féle variáció, megnövekedett paraméterek miatt. Többféle képpen is lehet csoportosítani, egyik ilyen egyszerűbb például, hogy az indexelni kívánt objektum pontszerű vagy terület alapú.[7]

Egy másik ilyen osztályozás lehet, hogy az adatszerkezetek területalapú, vagy adatvezérelt módszerek. Ennél a csoportosításnál az számít, hogy az adott struktúra a területet osztja-e fel, vagy a térbeli objektumokból indulnak ki. Területalapú mód-

szerek közé sorolják a Grid indexet, a kd-fát és adaptív kd-fát, BSP-fát, negyedelőfát, Z-rendező fát. Adatvezérelt módszerek például az R-fa, R+-fa, R*-fa.[6]

A fa struktúra hatékonyan szolgálja egy-egy régió fókuszálását, ami megkönnyíti a gyors algoritmusok tervezését. Az implementációban a negyedelő- és nyolcadoló-fa alkalmazása merült fel, ezért a a következőkben ezek működését mutatom be.

2.3.3. Negyedelő-fa

A negyedelő-fák, vagy másnéven négyfák hierarchikus térbeli adatstruktúrák, amelyeket objektumok indexelésére alkalmaznak. Az adatszerkezetet Finkel és Bentley fejlesztették ki 1974-ben[8], egyike volt az első olyan adatstruktúráknak, amely alkalmazható volt magasabb dimenziójú adatokra. Sok féle téradatot lehet vele ábrázolni, például pontokat, vonalszakaszokat, téglalapokat, sokszögeket, felületeket, térfogatokat. A negyedelőfa elnevezést működésének elve alapján kapta, amely a tér rekurzív dekompozíciójára épül. Egy régiót mindig négy részre darabol, északnyugati (=NW), délnyugati (=SW), északkeleti (=NE) és délkeleti (=SE) kvadránsokra, ahol a tér elválasztók mindig párhuzamosak a koordinátatengelyekkel. Az így kapott régiókat a feldarabolt régió gyermekeiként tartja számon. A fa struktúra előnye, hogy könnyű frissítési műveleteket végezni rajta és régiókat könnyen lehet fókuszálni. A négyfa kiterjeszthető több dimenzióra, ekkor a feldarabolt régió mindig 2^d tartományra bontódik szét. A négyfa elnevezést szokták általánosan is használni függetlenül a tér dimenzió számától. A négyfáknak sokféle változata létezik, amelyeket a dekompozíciós folyamatban használt elv különböztet meg.

A négyfa létrehozásakor a dimenzónak megfelelő tartományból indulunk ki, amely körbeveszi a vizsgált térbeli adatokat.A dekompozíciós folyamat fontos kérdése, hogy a dekompozíciót a bemeneti adatok vezérlik-e, vagy a tér egyenlő felosztásán alapul. A fa magassága kiegyensúlyozható, ha kezdetben az összes bemeneti adat rendelkezésre áll és a fa ezek alapján épül fel. Amennyiben a tér egyenlő felosztásának elvét választjuk, a fa magassága a térbeli adatok eloszlásától függ.

Szintén fontos kérdés a régió felosztási folyamat során mi a befejezési feltétel. Előre meghatározott adatszerkezetnél akkor áll le, amikor a dekompozíció egy előre beállított felbontást elér, változó adatszerkezet esetén, ha egy bemeneti adaton alapuló tulajdonság teljesül, de vannak olyan négyfák, ahol hibrid módon használják.

A leggyakrabban használt negyedelő-fa a pont négyfa (Point quadtree), a pont-

tartomány négyfa (Point Region quadtree) és a tartomány négyfa (Region quadtree). Mindhárom esetben egy olyan négyzet alakú régióból indulunk ki, amely az összes pontot tartalmazza.[6, 9]

Pont négyfa

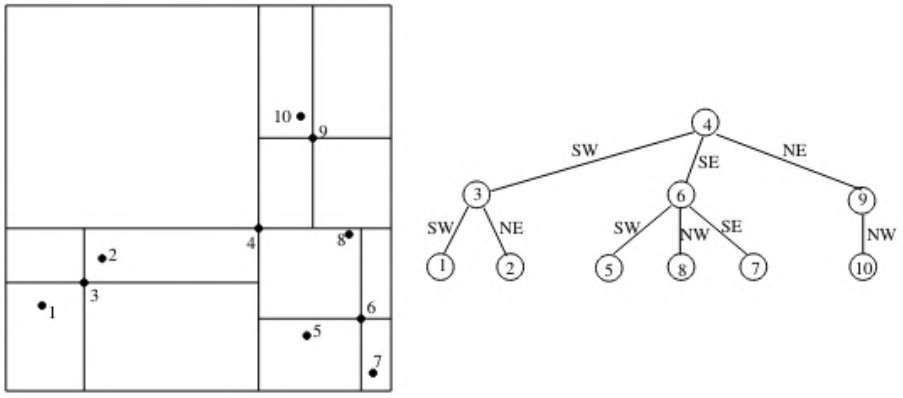
Pontszerű objektumok tárolására alkalmas, ahol mindig a csúcsokban tároljuk a pontokat. Az adatpontokban metszük el a koordinátatengellyel párhuzamos egyenesekkel. Addig ismétlődik a rekurzív térfelosztás, amíg van input pont a térben, amelyet még nem metszettünk el. A négyfa pontok szerinti felépítéshez mindig kell egy kezdőpontot kiválasztani (gyökér csomópont), amihez a feldarabolást követően kapcsolódik mindig négy másik csomópont, amelyeket az NW, NE, SW, SE korábban említett címkékkel jelölik. A címkékre azért van szükség, hogy a szűlőben tárolt ponthoz képest a gyermeke melyik kvadránsban helyezkedik el. Ebben az esetben a feldarabolást követően kapott régiók nagy valószínűséggel nem lesznek egyenlő méretűek, hiszen a pont elhelyezkedése határozza meg az egyes régióméreteket.

Pont keresése esetén a gyökér csomóponttól végig megyünk a már beszúrt csúcsokon. A csúcsokben egy döntési algoritmus az x, illetve y koordináta alapján határozza meg azt, hogy az elem mely kvadráns felé menjen tovább. Új elem beszúrást a kereső algoritmus alkalmazásával tudunk végrehajtani, annyi különbséggel, hogy a végén a megfelelő levél végére beszúródik a pont.

A csúcsbeszúrás és keresés átlagos költsége $O(log_4 n)$, amely kikövetkeztethető a fa felépítésre szánt költségből. A felépítés a fa teljes úthosszának költsége, amely $O(n * log_4 n)$, ahol n a véletlen elhelyezkedő pontok száma.

A fa magassága a pontok beszúrási sorrendjétől függ. Amennyiben a pontok eloszlása egyenletlen a magasság megnyúlik. Amennyiben a pontok sorrendjét ismerjük optimálisan felépíthető a fa, amennyiben nem, dinamikusan a fa építésekor átrendezésekkel tudunk valamelyest optimalizálni. Amennyiben a fa kiegyensúlyozatlan a tartománylekérdezések is megnőhetnek, amelynek legroszabb esetben a költsége $O(k * n^{1-\frac{1}{k}})$, ahol k a dimenzió.

A pontadatokhoz számos adatlekérdezési módszer tartozik. Az egyik ilyen például a tartomány lekérdezés, amikor az értéktartományon belüli összes pontot kérjük le. Egy másik ilyen példa a gömbi régió lekérdezése, ahol egy lekérdezési pont körüli összes pontot szeretnénk megkapni egy adott sugárnyi területen belül.[6, 9]



2.7. ábra. Pont négyfa
Forrás:[9]

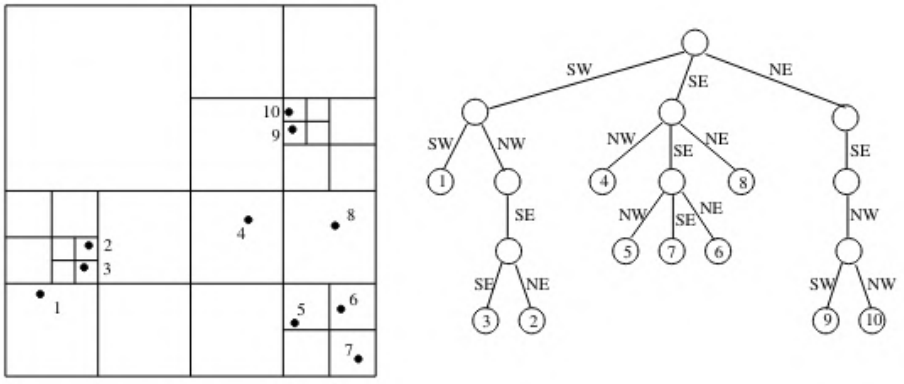
A 2.9. ábra szemlélteti a kétdimenziós pontok negyedelő-fává való alakítását. Jól látható, hogy a 4. csúcs a gyökér elem és a pontok szét szórtan helyezkednek el, így viszonylag kiegyensúlyozott fát lehet felépíteni. Azért csak három gyereke van a gyökér elemnek, mert jól látszik, hogy NW-ben (első negyedben) nincs elem. A többi elem gyerekei hasonló képpen határozható meg, a baloldali kétdimenziós téglalapok alapján.

Ponttartomány négyfa

Ponttartomány négyfa nem csak pontok tárolására alkalmas, segítségével komplex alakzatokat is tudunk indexelni. Régió szerinti feldarabolás esetén mindig négy egyenlő méretű részt kapunk egy régió feladarbolása után, amelyeket a korábbiakkal megegyezően címkézzük.

Pontok indexelése: A pontnégyfával ellentétben a fa csúcsai helyett a levelein fogjuk tárolni a pontokat, ebből az következik, hogy annyi levele lesz a fának, ahány pont van (2.8. ábra). A csúcsokban tároljuk a csúcshoz tartozó terület középpontját és a gyerekekre mutató négy címkét. A levelek legfeljebb egy elemet tárolhatnak.

Pont beszúrás esetén mindig a gyökér csomóponttól indulva, koordináták összehasonlításával haladunk lefelé a fában, addig amíg egy levélhez érünk. Mivel a levélnek legfeljebb egy pontja lehet, ezért megvizsgáljuk, hogy üres-e a levél, vagy sem. Amennyiben üres a levél, úgy beszúrjuk a pontot, amennyiben nem üres, térfelosztást kell végrehajtani. A tér dekompozícióját addig kell folytatni, ameddig a két pont nem egy tartományba esik. Abban az esetben, ha a két pont közel van egymáshoz és sokszor kell megismételni a felosztást, úgy a fa megnyúlik és kiegyensúlyozatlan lesz.



2.8. ábra. Ponttartomány négyfa
Forrás:[9]

Pont keresése esetén is a gyökérből kiindulva kell eljutni a pontot tároló levélig. A pont négyfához hasonlóan itt is minden csúcsban egy döntési algoritmus megvizsgálja, hogy mely régió felé kell tovább haladni.

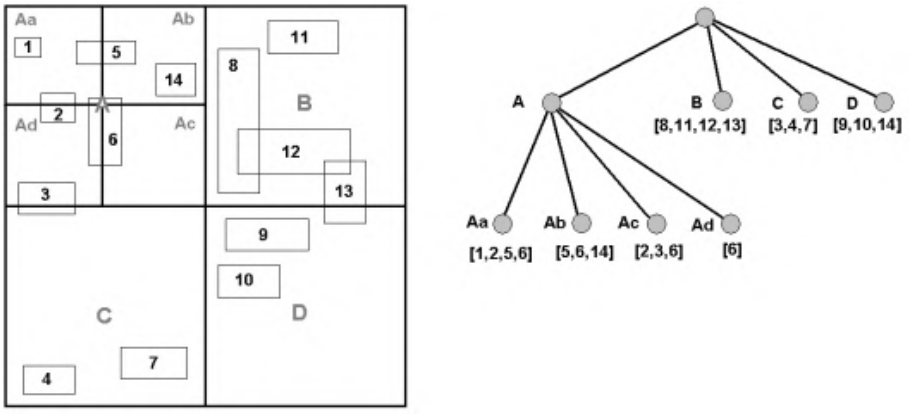
Pont törlésekor, ha egy levele maradna egy csúcsnak, célszerű összevonni a tartományokat, ennek következtében a levelet tartalmazó csúcs lesz az új levél.

Az egyes műveletek futási ideje a fa magasságától függ, tehát $O(h)$, ahol h a fa magassága. A fa magassága annál rosszabb minél közelebb helyezkednek el a pontok egymáshoz. A fa magasságának felső korlátját a pontok egymáshoz való távolsága közül a legkisebbet választva kapjuk meg. A legkisebb térrész hossza - amely tartalmazza ezt a két pontot - az $\frac{s}{\sqrt{d}}$, ahol s a legkisebb távolság két pont között, d pedig a dimenziók száma. Két dimenzió esetén $\frac{s}{\sqrt{2}}$. Addig folytathatjuk a rekurzív felosztást, amíg egy tartomány nagysága kisebb nem lesz, mint az előbb meghatározott hossz. A felezések miatt, a fa mélységének a felsőkorlátját a legkisebb k fogja adni, ami $k = \lceil log\frac{\sqrt{d}D}{d} \rceil$, ahol D a teljes térrész hossza.

A fa felépítése történhet beszúrások egymásutániságával, vagy szintenkénti felépítéssel. Mindkét esetben a legrosszabb futási idő $O(n*log\frac{D}{s})$. A műveletek költsége a fa maximális mélységétől függenek $O(h)$, ahol h a fa magassága. Optimális, kiegyensúlyozott fa esetén a minimális költség $\lceil log_4(n-1) \rceil$.

Egy fa tárolási igénye $O(n * N)$, - ahol n az inputok száma - ha a térben az inputok lefedhetőek egy 2^Nx2^N méretű négyzetráccsal.

Téglalap indexelés: Összetett objektumokat a legkisebb befoglaló téglalapjukkal határozhatjuk meg és ekkor a téglalapokat tudjuk indexelni ponttartomány négyfával az alábbi módon.



2.9. ábra. Összetett objektumok indexelése
Forrás:[6]

Felépítése a pontok indexelése képpen történik. A tér dekompozícióját addig végezzük, amíg minden tartományban maximum egy adott számú objektum nem marad. Itt is a leveleken fogjuk tárolni az objektumokat, annak megfelelően, hogy a befoglaló téglalap mely tartományokban helyezkedik el. Három eset lehetséges:

a, Tartományban teljesen benne van egy téglalap.

b, Tartomány és téglalap metszik egymást.

c, Téglalapban teljesen benne van a tartomány.

A beszúrás és törlés ugyanúgy történik, de mivel egy téglalap több levélben is megjelenhet egyszerre, ezért a pontok beszúrásához képest a költség növekedhet. Túlcsordulás esetén itt is további térfelosztás, törléskor tartomány összevonások szükségesesek.

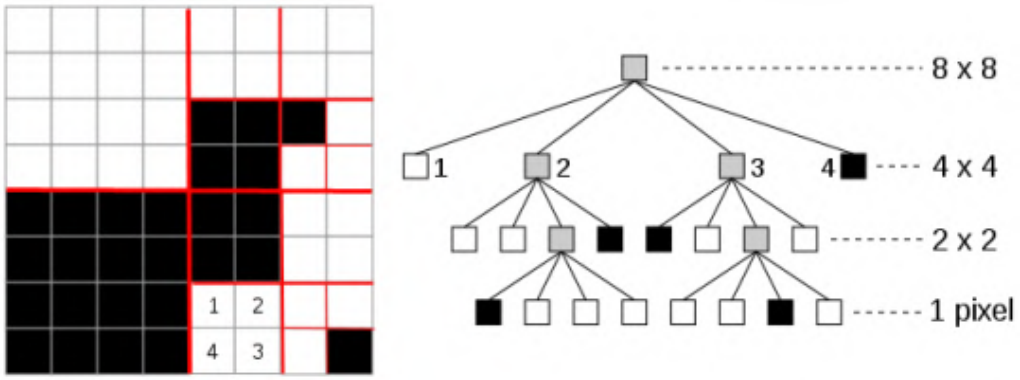
Kereséskor minden levélhez el kell jutni, amely tartalmazza az adott téglalapot. Ezért sokszor nem elég egy utat bejárni.

A fa méretét az objektumok méretei és a térben elhelyezkedésük is befolyásolja. Minnél nagyobb egy objektum annál több tartományban kell tárolni, annál több utat kell bejárni kereséskor. Az objektumok közeli elhelyezkedése pedig a pont indexeléshez hasonlóan több darabolást fog igényelni.[6, 9]

Tartomány négyfa

A tartomány négyfát raszteres képek indexelésére alkalmazzák. Az egyszerűbb fekete-fehér, illetve színes képek indexelése is megvalósítható.

Alapvető ötlete, hogy egy tartomány csak azonos színű pixeleket tároljon. A tér itt is négy egyenlő részre osztódik, addig amíg egy tartományban található más szín is.



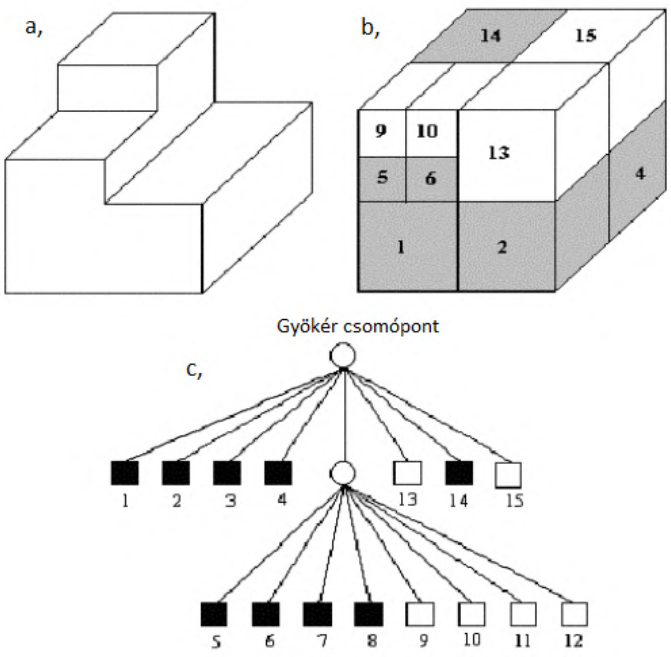
2.10. ábra. Tartomány négyfa
Forrás:[6]

Az összes művelet a ponttartomány négyfa képpen működik. Lekérdezéskor az azonos színű pixelek megtalálásához elengedhetelen az össze olyan levelet megtalálni a fában, amely tartalmazza az adott színt.

Színes képek esetén az indexelés ugyanígy történik, annyi különbséggel, hogy árnyalat intervallumokat határozunk meg egy adott képhez, amely pixelek hasonló intezitásértékkel rendelkeznek, ők szerepelhetnek egy tartományban.[6]

Nyolcadoló-fa

A nyolcadoló-fa a négyfa háromdimenziós kiterjesztése, mindegyik korábban említett négyfának létezik kiterjesztése több dimenzióra. Ebben az esetben 2^3-on azaz 8 elemre bontható szét egy régió. Legtöbbször háromdimenziós képek ábrázolására alkalmazzák. A nyolcfa nem effektív, azokban az esetekben, ha az objektumok a képen nem egyenletesen oszlanak el. [9]



2.11. ábra. Nyolcadoló-fa működése
Forrás: [9]

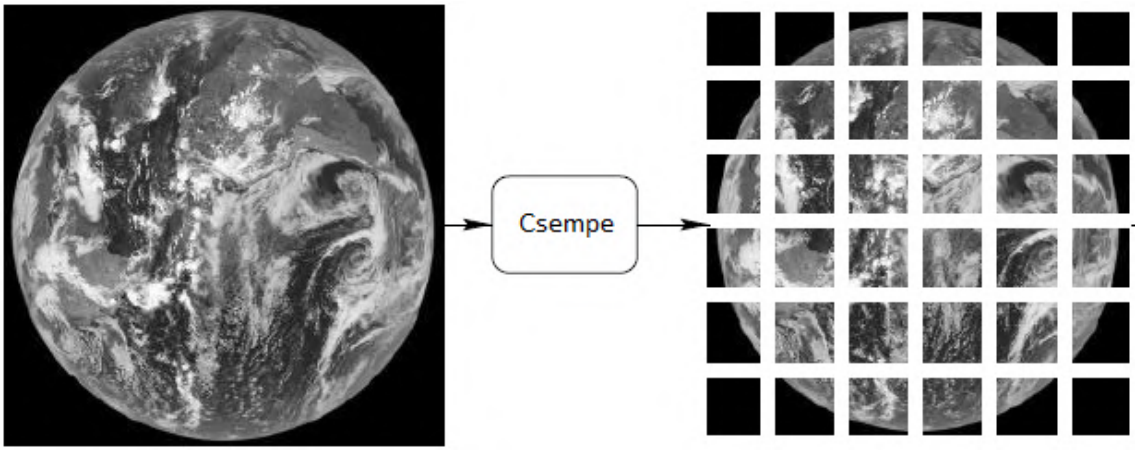
A 2.11. ábrán láthatjuk egy nyolcadoló-fa megvalósítását. Az a, jelű háromdimenziós objektumot szeretnénk tárolni nyolcfa adatszerkezetben. A b, kocka ábrázolja az a, jelű objektumot, ahova kiterjed azt szürke színnel jelöli, ahova nem azt fehérrel. A c, adatstruktúra a b, kocka nyolcadoló-fás megvalósítása.

2.4. Csempe alapú feldolgozás

A térbeli adathalmazok gyorsan növekvő méretei miatt szükség volt ezen adatok hatékony kezelésére, manipulálására. A csempe alapú feldolgozást az idő és hardver véges keretei miatt találták ki. Alapkoncepciója, hogy egy nagy méretű területet felosztunk kisebb méretűekre, ezáltal nem kell annyi adatot a memóriában tárolnunk, továbbá ha műveletet szeretnénk az egész területen végrehajtani párhuzamosítható is. A felosztott területek legtöbbször négyzet, vagy téglalap alakúak.

Tekintsünk egy hatalmas térképet, amit meg szeretnénk nézni, illetve adatmanipulációt szeretnénk végezni rajta. Nem kell az egész térképet betöltenünk, ahhoz hogy egy részét a képernyőn megjelenítsük. Osszuk fel a térképünket kisebb méretű csempékre és töltsük be a használni kívánt adatterületet, azaz csempét. Mivel csak egy részét töltjük be, ezáltal maximalizálja a frissítési sebességet, lemezes I/O műveleteket redukálja.

Párhuzamosan több szálon futó programoknál is hatékonyan lehet használni a csempézést. Az egy egy szálnak beadott kisebb területű csempék egy idejű futása és feldolgozása esetén jelentősen gyorsíthatók az alkalmazások.

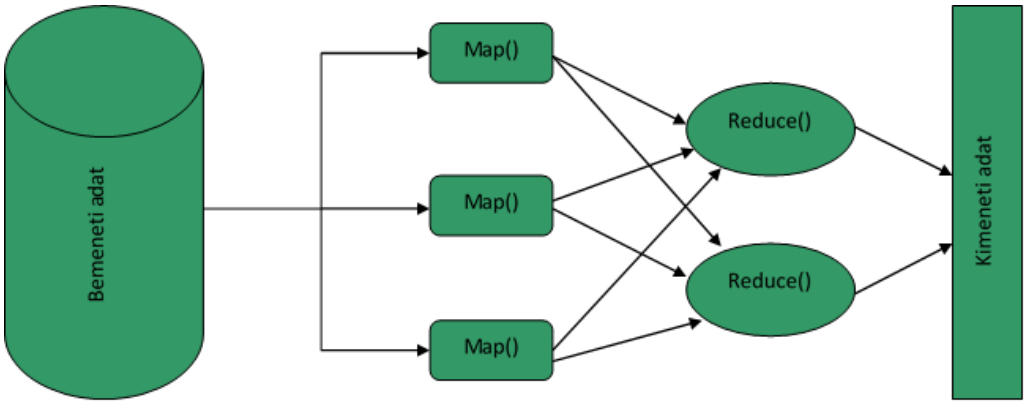


2.12. ábra. Csempékre osztás
Forrás:[10]

2.12. ábra mutatja egy kép kisebb csempékre való darabolását. Többféle csempézési módszer létezik, akár több felbontású csempézés is, ahol a csempézési és skálázási módszereket vegyítik. [10]

2.5. MapReduce

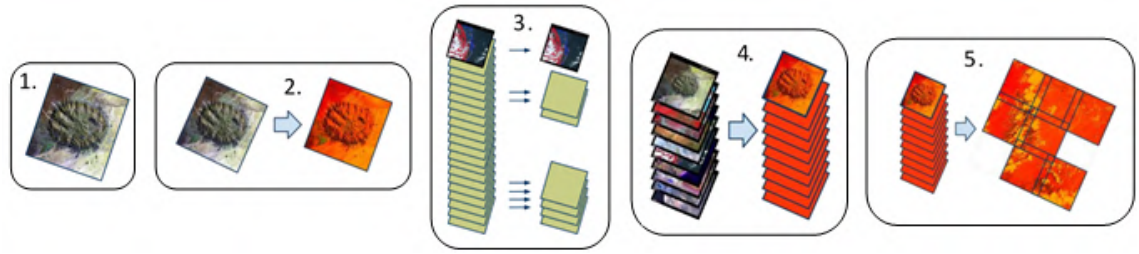
A MapReduce technológiát a Google mutatta be[11], amely segítségével nagymennyiségű adat párhuzamos feldolgozását tudjuk megvalósítani. Ahogy a neve is mutatja két fázisból áll egy "map" és egy "reduce" műveletből. Az előbbi egy lekérdezési függvény, amely szűrést és rendezést végez, az utóbbi egy redukciós függvény, amely az eredményösszefésülésért felel. Tehát a bemeneti adathalmazt kisebb feladatokra osztjuk, azokat párhuzamosan végrehajtjuk és a végén a sok kis részeredményt redukáljuk egy eredménnyé.



2.13. ábra. MapReduce leegyszerűsített működése

Ahogy a 2.5. ábárán is látható a bemeneti nagy mennyiségű adatokat kisebb részekre bontják, majd ezeket átadva a Map függvényeknek a feldolgozó algoritmus párhuzamosan fut a kisebb adathalmazokra. Amint a map algoritmusok végeznek átkerülnek a reduce fázisba, ahol a kiértékelt adatokat, egy adathalmazba rendezik vissza.

Ennek a technológiának több féle implementációs megvalósítása lehet, amit ebben a fejezetben bemutattam a működésének az alapkoncepciója. Térinformatikai alkalmazásokban is a nagy adatmennyiségek feldolgozása miatt használják. Az egyik ilyen például a sokak által ismert Google Earth Engine. A Google Earth Engine egy "big data" platform távérzékeléshez. Gyors számítási és vizualizációs rendszer globális méretű térinformatikai elemzéshez, amelyhez tartozik egy tudományos adatkatalógus.[12]



2.14. ábra. Google Earth Engine
Forrás: Szerkesztett kép [12] alapján

A 2.14. ábra szemlélteti a működését, amelynek 5 lépése a következő.

1. Válasszunk egy képet, ahol meghatározzuk a vetítés, felbontás, sávok, határoló téglalap és vizualizáció nagyságát.

2. Futassunk le egy általunk megírt, vagy könyvtárból vett algoritmust.

3. Szűrjük a képgyűjteményt idő, tér és metaadatok alapján.

4. "Map" lépés. Algoritmus leképezése a képgyűjteményre $N \to N$.

5. "Reduce" lépés. $N \to 1$, vagy $N \to M$.

Egy másik ilyen példa a Marmot, amely nagyteljesítményű, szintén MapReduce technológián alapuló térinformatikai nagy adat feldolgozó rendszer. A Marmot a térindexeket tartalmazó összetett és időigényes lekérdezések kezelésében általában felülmúlja a SpatialHadoop térbeli big data keretrendszert.[13]

3. fejezet

Módszertan

Egy algoritmus futási idejének javításához legelső lépésként azt kell megvizsgálni, hogy egyáltalán az adott algoritmus gyorsítható-e. Ehhez a programban olyan szűk keresztmetszeteket kell keresni, amelyek áthidalásával, optimalizálásával gyorsabb futási időket érhetünk el. Diplomamunkám során kétféle optimalizálást alkalmaztam, amellyel a program jelentősen gyorsult.

Ebben a fejezetben bemutatom a kiindulási algoritmus[14] részegységeit, a programban található szűk keresztmetszeteket és az általam adott megoldásokat ezen keresztmetszetek kiküszöbölésére, optimalizálására. Írok az algoritmus korábban használt futtatási területeiről, amelyeket én is használtam.

Ezután bemutatom a csempealapú párhuzamos feldolgozásra adott ötleteimet, amikor a csempék pontosan illeszkednek és ennek továbbgondolását, amikor fedik egymást. Ezután bemutatom a módszer lépéseit és müködését.

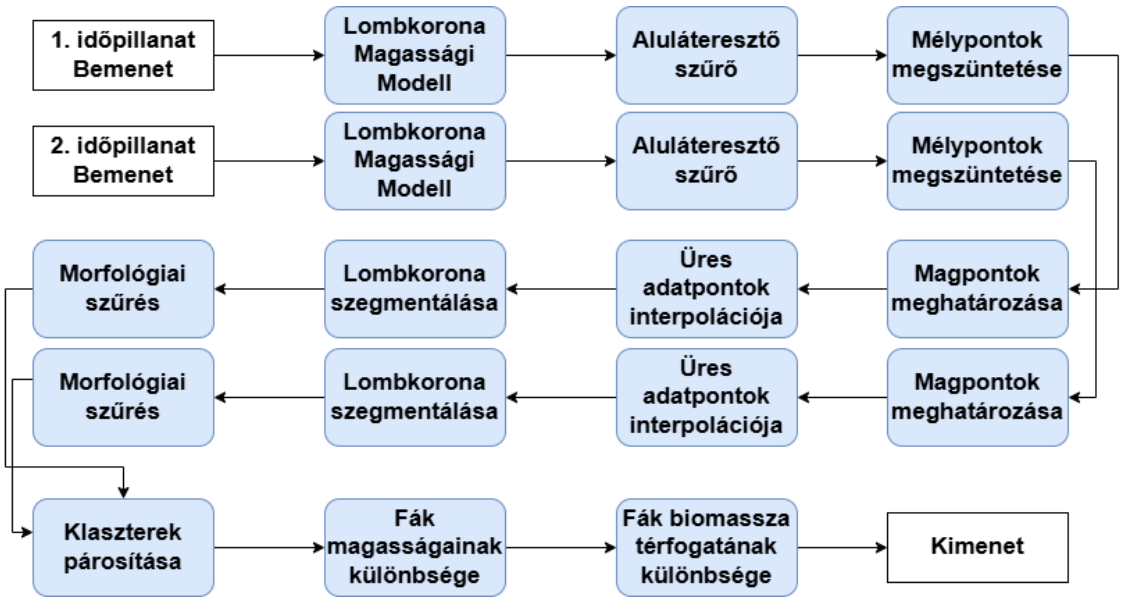
3.1. Kiindulási algoritmus

A projekt keretein belül épület detektálással, vegetáció detektálással és ezek változáselemzéseivel foglalkoztak. Az én munkám a vegetáció detektáláshoz és annak változáselemzéséhez köthető, amelynek keretein belül nem csak az egyes fák detektálása, hanem azok két különöböző időpillanatban bekövetkezett változásának meghatározása is cél volt.

3.1.1. Algoritmus bemutatása

A korábban megvalósított algoritmus két nagy részből áll. Egy előfeldogozó részből, amely a vegetáció detektálást végzi és a bemeneti adatok feldolgozása után az azonosított fák gyűjteményét adja vissza az adott időpillanatban. És egy utófeldolgozó részből, amely a változáselemzésért felel és párosítja a különböző időpillanatokból származó fákat, továbbá kielemzi a különböző időpillanatok közti különbségeket. Az alábbiakban ismertetem a két nagy egység kisebb részegységeit.

Az előfeldolgozó részben történik meg a digitális terepmodell és digitális felszín modell beolvasása, amelyek előfeldolgozott pontfelhőket tartalmaznak.



3.1. ábra. Kiindulási algoritmus folyamatábrája
Forrás: [14]

A 3.1. folyamatábra lépéseinek rövid leírásával mutatom be a kiindulási algoritmust.

Vegetáció detektálás

A program két különböző időpillanatban készült ugyanazon terület DTM, illetve DSM paramétereit várja bemenetként. A felhasználó futtatási kapcsolóként meg tudja adni, hogy a program párhuzamosan, vagy szekvenciálisan fusson a két különböző időpillanatra. Az időpillanatok futtatási területeire az alábbi részegységek futnak le szekvenciálisan.

1. **Lombkorona Magassági Modell:** Lombkorona Magassági Modell (Canopy Height Model = CHM) az egyes fák magasságát írja le. Kiszámolása során a DTM-et kivonjuk a DSM-ből.

2. **Aluláteresztő szűrő:** A fák magpontjaikból ponthalmazokat hoz létre. Ahhoz hogy elkerüljük, hogy egy fához több maximum pont is tartozzon eliminálják a felesleges csúcsokat Gauss-féle elmosódási transzformációval.

3. **Mélypontok megszüntetése:** 1,5 méternél kisebb küszöbérték alatti objektumokat megszünteti, hiszen az biztosan nem fa.

4. **Magpontok meghatározása:** A fák csúcspontjai lesznek a fák magpontjai. 3x3-as kernelmátrixon belül, ha egy pont magasabban helyezkedett el, mint az összes szomszédos pontja, akkor az a fa csúcspontja.

5. **Csomópontok interpolációja:** A potenciális fa közelében lévő csomópontokhoz, - amelyekhez nem tartozik érték - értéket rendel, a szomszédos pontok számtani átlagát véve.

6. **Lombkorona szegmentálása:** Ahhoz, hogy tényleges lombkoronát kapjunk a magpontokat további lombkorona pontokkal kell kiterjeszteni, ez fogjuk klaszternek hívni. Az algoritmus több olyan esetet is megvizsgál, amikor több magpontot össze kell vonni, mert az egy fa.

7. **Morfológiai szűrés:** Morfológiai szűrés során az algoritmus pontokat ad, vagy pontokat vesz el a már elkészült klaszterből, ezáltal módosítja a klasztert.

Változáselemzés

Miután mindkét időpillanatra lefutott a vegetáció detektálás elérkezünk a második nagy blokkhoz, mikoris összehasonlításokat végzünk a két területen.

1. **Klaszterek (fák) párosítása:** A két különböző időpillanatból származó immáron azonosított fákat párba állítja az algoritmus. Két számolási módszert implementáltak, amelyet szintén a felhasználó tud megadni kapcsolókkal a program futtatásánál. Mindkét számolásnak van előnye, hátránya.

   Az egyik ilyen módszer a fák központjait (= Centroid) párosítja és lineárisan aszimptotikus komplexitású $\theta(n_1 * n_2)$, ahol n_1 és n_2 a klaszterek száma a két klasztertérképen.

A másik módszernél a Hausdoff-távolságot[15] alkalmazták, ahol egy maximum függvény alapján kiszámolják az egyik fa legtávolabbi pontját hasonlítják a másik fa legközelebbi pontjával. Ennek futási ideje $\theta(n_1 * n_2 * m_1 * m_2)$, ahol n_1 és n_2 klaszterek száma, m_1 és m_2 az átlagos pontok száma a két időpillanatban.

A fák párosítása szempontjából három féle opció lehetséges:

- Megtalálja ugyanazt a fát mindkét időpillanatban.
- A későbbi időpillanatban nem talál párt az adott fához, ekkor feltételezhető a fa kivágása.
- A korábbi időpillanatban nem talál párt az adott fához, ekkor valószínűleg fa ültetés történt.

2. **Fák magasságainak különbsége:** Miután a párok meghatározásra kerültek, többféle összehasonlítást lehet végezni a fa párokon. Ilyen például a magasság különbségek, amelyet a két időpillanatban lévő ugyanazon klaszter legmagasabb pontjából határoznak meg. Ebből egy terület átlagos magasságkülönbségét is meghatározhatjuk.

3. **Fák biomassza térfogatának különbsége:** A megfelelő évszakokban meghatározható a fák lombkorona térfogata is, amely egy raszterháló segítségével történik.[14]

3.1.2. Algoritmus optimalizálási lehetőségei

Miután megértettük a komplex algoritmus és egyes részegységeinek működését, elgondolkodhatunk azon, hogy hol és miképpen lehetne gyorsítani rajta úgy, hogy közben a kimeneti adatok ne módosuljanak.

Három különböző helyet találtam, amivel jelentősen csökkenthető a futási idő. A kiinduló fázisban az algoritmus futása előtt, amely egy csempealapú párhuzamosítási lehetőség. A vegetáció detektálás lombkoronák szegmentálása lépésénél. A változáselemzés fák párosítása lépésénél. Utóbbi két optimalizációs lépés megoldása ugyanazon negyedelő-fás elvre épülnek.

Kiinduló fázis

A hatalmas vizsgált terület adja magát, hogy osszuk fel kisebb területekre és arra futassuk le az algoritmust olyan kisebb módosításokkal, hogy az kompatibilis legyen a konkurens feldolgozással.

Lombkorona szegmentálási fázis

Mint ahogy a 3.1.1. alfejezet *Vegetáció detektálás* szakaszában írtam, a lombkorona szegmentálás lépésénél az algoritmus folyamatosan kiterjeszti egy klaszter pontjait. Itt történik a klaszterek eggyé olvasztása amennyiben szükséges a számítások alapján. A számításnál egy klasztert a többivel kell összehasonlítani a pontjaik alapján. Ennek futási ideje négyzetes $\theta(n * n)$, ahol n a klaszterek száma. Itt mondhatjuk azt, hogy az aktuális klasztert ne minden klaszterrel hasonlítsunk össze, hanem csak a tőle adott területen belül elhelyezkedőkkel. Ezzel a futási idő lineárisra csökkenthető.

Fák párbaállítási fázis

Hasonlóan a lombkorona szegmentáláshoz, a fák párbaállításánál is minden fát minden fával összehasonlítunk azok egymásnak megfeleltetése kapcsán. Mind a központi, mind a Hausdorff távolságok meghatározásánál szükség van erre a lépésre, ezért mind két esetben a párbaállítás folyamata négyzetes futási idejű $\theta(n * n)$, ahol n a klaszterek száma. Ötletünk éppúgy, mint az előbbi esetben az, hogy ne minden fával hasonlítsunk össze, csak egy tőle adott területen belüliekkel.

3.1.3. Futtatási területek

Az algoritmust korábban holland és észt területeken futtaták. Összehasonlíthatóság kedvéért én is ezeket a területeket használtam az észt területek kivételével.

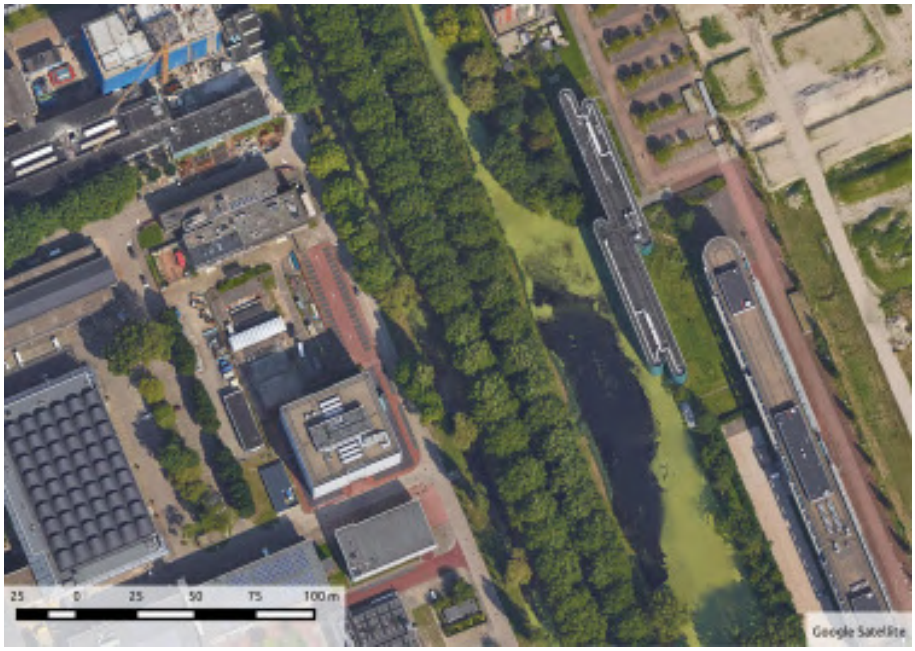
A hollandia egész területét az Actueel Hoogtebestand Nederland (AHN) fedi le, melynek mérete 41526 km^2. Ebből a területből három különböző időpillanatban történt adatgyűjtés létezik, AHN-1, AHN-2, AHN-3. Az adatokat AHN-2 és AHN-3 gyűjteményből választották a nagyobb pontsűrűség miatt (6-10 pont/m^2). Az adatgyűjtések az alábbi időszakokban történtek.

- AHN-2: 2007-2012
- AHN-3: 2014-2019

Az AHN területek 1372 csempéből épülnek fel és egyenként 31,25 km^2. Ezek túl nagy területek a folyamatos tesztelések során, ezért négy kisebb AHN területet használtak, amelyeket a következők.[16]

- Egy utca: 0,103 km^2
- Amszterdam városközpont: 1,937 km^2
- TU Delft Kampusz: 2,143 km^2
- Delft városközpont: 8,640 km^2

Ezekről a területekről előfeldolgozott DEM-ek állnak rendelkezésre, 0,5 m felbontásban, GeoTiff formátumban.[14]



3.2. ábra. Demonstrált terület
Forrás: [14]

Az algoritmus működésének szemléltetésére a korábban említett 0,103 km^2 méretű fás utcát használták, amely a 3.2. ábrán látható Google Satellit kép. Én is ezen fogom demonstrálni a csempézési eredményeimet a 5.3. alfejezetben.

3.2. Csempealapú párhuzamosítás

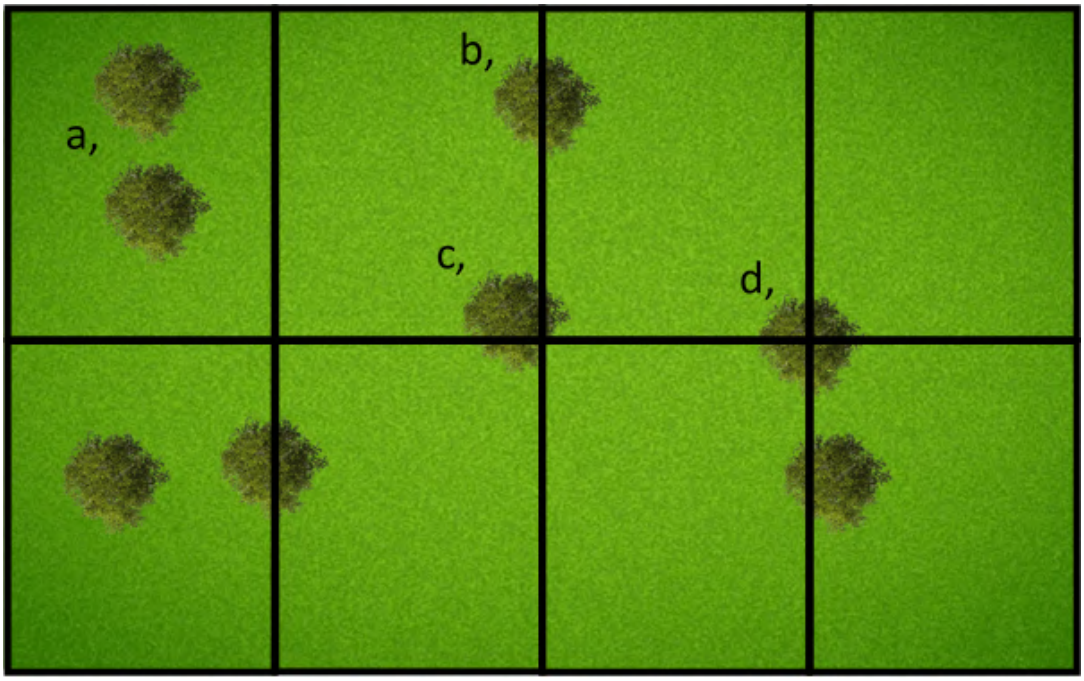
Egy program optimalizálása során talán a legalapvetőbb gondolat, hogy tudnánk-e az algoritmust párhuzamosítani és több szálon futtatni. A több magú processzorok lehetőséget biztosítanak az egyes részegységek konkurens végrehajtására, amely nagy adatmennyiségek esetén jelentős gyorsuláshoz vezethet.

Olyan összetett algoritmusnál, amelyből jelen esetben kiindulunk korántsem evidens a többszálúsítás és annak megvalósítása. Talán az egyik legegyszerűbb megközelítés, hogy a kezdeti nagy adatmennyiséget osszuk fel több kisebb csempére és ezekre futtassuk le az algoritmust konkurens feldolgozásban.

A két féle módszer, - amelyeket a következőkben fogok bemutatni - ugyanaolyan elven működnek, különbség köztük az, hogy a csempék fedik-e egymást, vagy sem.

3.2.1. Pontosan illeszkedő csempék

Az első módszernél a csempék nem fedik egymást, hanem pontosan egymás mellé vannak illesztve. Ennek a módszernek az előnye egyben a hátránya is. Mivel nincs átfedő terület, ezért minimálisan kisebb a feldolgozási idő, de ez a produktum rovására fog menni, hiszen a csempe határokon lévő fákat az algoritmus nem fogja tudni jól detektálni, vagy akár félredetektálja, ami fals pozitív fákat fog eredményezni.



3.3. ábra. Pontosan illeszkedő csempék
Forrás: Szerkesztett kép (háttér: ShutterStock, fa: Depositphotos)

A 3.3. ábra egy feldolgozni kívánt területet mutat be, amelyet nyolc egyenlő méretű csempére osztottam fel. Láthatjuk rajta az összes olyan fa detektálási lehetőséget, amellyel a kép csempékké való felosztását követően, az algoritmusnak szembe kell nézni. Mivel egy fa legfeljebb négy csempében tud egyszerre jelen lenni, ezért négy esetet tudunk megkülönböztetni, amelyeket az ábrán a, b, c, d, -vel jelöltem.

a, A fa a csempén belül helyezkedik el, semmilyen része nem lóg ki az adott csempéből. Ekkor nincs semmilyen probléma az algoritmus eredeti módon tudja detektálni a fát.

b, A fa két csempe határán helyezkedik el, ekkor az egyik része átlóg egy másik csempébe. Az algoritmus mindkét csempében detektálni fogja ugyanazt a fát, tehát kétszer kerül megtalálásra. Ekkor három lehetséges opció történhet a kisebb darabokkal. Az algoritmus működése szerint a csempén belül lévő részt

1. külön fának detektálja.
2. összevonja egy közelben lévő másik fával.
3. több másik fával is összevonja.

c, A fa három csempe határon helyezkedik el, az egyik csempéből átlóg két másikba. Ekkor a három darab rész hasonló képpen kerül feldolgozásra, mint a b, pontban említett fa részek.

d, A fa négy csempe határán helyezkedik el, az egyik csempéből átlóg három másikba. Ekkor a négy darab rész hasonló képpen kerül feldolgozásra, mint a b, pontban említett fa részek.

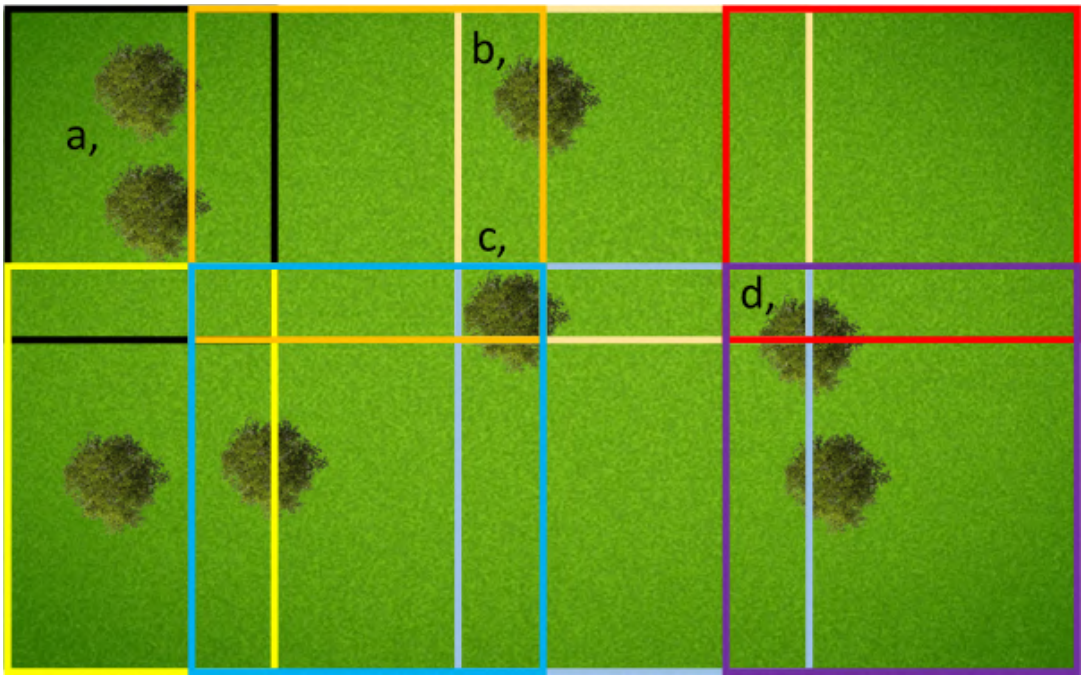
Jól látszik, hogy egy fa több csempébe való átlógása és összevonása több másik fával, sok kombinációt tud eredményezni és képes teljesen átvariálni az eredeti eredményeket. Nyilván minél több fa helyezkedik el a csempe határokon, annál rosszabb eredményeket kapunk. Valamelyest tudunk javítani a csempék számával, hiszen a kevesebb csempe, kevesebb csempehatárt is eredményez, így csökkenthető a hibafaktor. Viszont nagy területek esetén, megfelelő kapacitások mellett, célszerűbb több csempére felosztani egy adott területet.

Mivel egyik alapvető szempont, hogy úgy csökkentsük a futási időt, hogy közben minél kevésbé rontsuk az eredeti algoritmus detektálási és elemzési eredményeit, ezért ez a módszer számunkra nem volt ideális.

3.2.2. Egymást átfedő csempék

A második módszer az előző módszer továbbgondolása. A csempék ne pontosan illeszkedjenek egymáshoz, hanem szélei fedjék egymást, hogy a határmentén helyezkedő fákat is jól detektálja az algoritmus.

Az átfedés oly módon történik, hogy mindig a soron következő csempe egy adott mérettel átnyúlik az azt megelőzőbe és a felette lévőbe. Az első sor felfele nem fed, hiszen nincs fölötte több csempe. Az első oszlop pedig ugyanilyen okból az előző oszlopba nem fed.



3.4. ábra. Egymást átfedő csempék
Forrás: Szerkesztett kép (háttér: ShutterStock, fa: Depositphotos)

A 3.4. ábra ugyanaz mint, a 3.3. ábra annyi kiegészítéssel, hogy a csempék - a korábban leírtak szerint - előre és felfelé fedik egymást. A jobb átláthatóság kedvéért különféle színekkel jelöltem a csempéket. A 3.3. ábra $a, b, c, d,$ jelű problémákra csak a megoldást írom le.

a, Semmi nem változik, ugyanúgy megtalálja, mint az előző módszernél.

b, A világos barna csempe előre fele való kiterjesztésébe beleesik, így az algoritmus megtalálja.

c, Ez az egyik legrosszabb eset. A világos barna és a kék színű csempe nem tudja lefedni teljesen. Tehát itt egyszerre kell érvényesülnie a felfelé és előre felé tartó

megnyújtásnak. A világos kék csempe kiterjesztésével sikeresen le tudtuk fedni. Így egy olyan csempével fedtük le, amibe az előző módszerben a fa semennyire sem lógott bele.

d, Hasonló a c, jelű megoldásához, annyi különbséggel, hogy itt belelógott a kiterjesztett csempébe. A fa megtalálását a lila csempe fogja eredményezni.

A csempe átfedést a legrosszabb eset szerint kell meghatározni, ami jelen esetben a c, jelű fa. Az átfedésnek akkorának kell lennie, hogy egy teljesen kifejlődött fa lombkoronájának átmérője is beleessen az átfedésbe. A tesztelgetések során a 20 méteres csempe átfedésre esett a választásom.

Az előző módszerhez képest futási időben egy kicsit több, hiszen az átfedő területek duplikálva kerülnek lefutattásra. Minél több csempére szeretnénk felosztani egy területet, annál több ez a futási idő.

3.2.3. Működésük

Az átfedő csempék konkurens feldolgozásához a korábban említett MapReduce technológia ötletét vettem alapul. A két időpillanathoz tartozó digitális terep- és digitális felszínmodellek beolvasása után a következő lépések valósulnak meg.

1: Csempésítés

Az adathalmaz beolvasása után, az algoritmus felosztja az elvárt csempemennyiség szerint és megcsinálja az átfedéseket az előzőekben ismertettek alapján. Az átfedés egy fa maximális lombkorona átmérőjének kell legyen. Az így generált, újonnan létrejött kisebb adathalmazokat eltároljuk.

2: Párhuzamos futtatás

Az így kapott kisebb adathalmazok mindegyikére az algoritmus meghívja párhuzamos feldolgozással az eredeti algoritmusban szerinti vegetáció detektálást. Amely mindegyik csempére visszaad egy klasztertérképet, amely az egyes fákat tárolja, különböző információkkal együtt.

3: Összefésülés

Az így kapott klasztertérképeket össze kell fésülni egy darab nagy klasztertérképbe. Az összefésülés úgy történik, hogy az első klaszter térképben lévő fákat teljes egészében belerakom az új klasztertérképbe. Az ezután következő klasztereket először megnézem, hogy létre tudom-e hozni.

Akkor nem fogom tudni létrehozni, ha egy másik - már a közös klasztertérképbe beszúrt - fában már jelen van a magpontja. Ekkor lekérdezem melyik klaszter ez és összehasonlítom a két klaszter méretét a pontjaik alapján. Amennyiben az újonnan érkező klaszter a nagyobb, akkor a már beszúrtat törlöm és a helyére az új kerül.

Amennyiben sikerül létrehozni a klasztert, akkor a pontokat is megpróbálom beszúrni hozzá. Abban az esetben, ha nem sikerül az történik, ami az előző bekezdésben, lekérdezem a klasztert és összehasonlítom őket, törlöm a kisebbet.

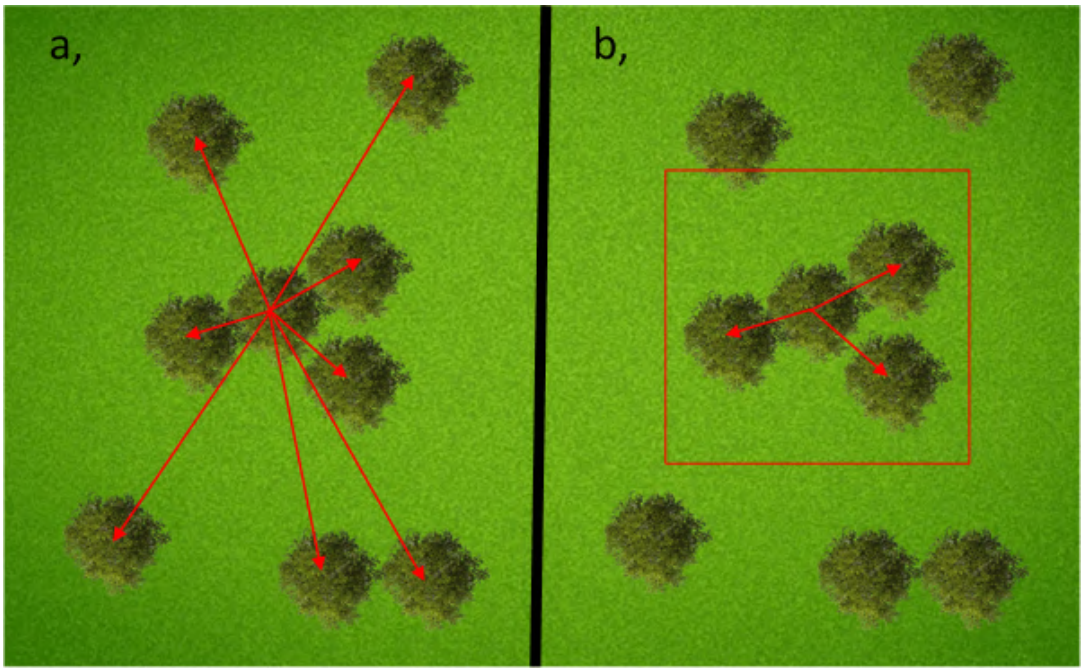
Amennyiben egy fa több fát is elmetsz, akkor addig folytatom az eljárást, amíg, vagy be nem rakja az új fát, vagy elveti és bent hagyja az elmetszettet.

4: Változás elemzés

Az összefésülés után kapott klasztertérképre fut tovább az eredeti változás elemzés szakasza.

3.3. Negyedelő-fa

Negyedelő-fát a lombkorona szegmentálási és fák párbaállításánál tudtam alkalmazni. A 3.1.2. alfejezet lombkorona szegmentálás és fák párbaállítása fázisainál írtam-e két hely problémáiról. Mindkét szakasz esetében az egész tér fái kerültek összehasonlításra egymással, ahelyett, hogy csak a szomszédos fákkal történt volna ugyanez. Tehát mindkét probléma visszavezethető ugyanarra a megoldásra.



3.5. ábra. Eredeti és negyedelő-fa szerinti összehasonlítás
Forrás: Szerkesztett kép (háttér: ShutterStock, fa: Depositphotos)

A 3.5. ábra *a*, jelű képe szemlélteti az eredeti algoritmus működését, ahol minden fát minden fával összehasonlít. Ez nagy adatmennyiségek esetén rengeteg összehasonlítás. A *b*, jelű mutatja a negyedelő-fás megvalósítást, amikor csak az adott fa megadott távolságán belüli fákkal hasonlítjuk össze az adott fát.

A negyedelő-fát nem mindegy milyen pontok szerint építjük fel. Felépíthetőek a fák magpontjai alapján, de a fák két dimenziós középpontjai szerint is. Mivel a klasztereket eleve a magpontjaik alapján hozzuk létre, ezért a magpontok el vannak tárolva az egyes klaszterekhez. A magpontokat egy lekérdezéssel elérjük, míg középpontokat számolás útján. Ezért a kevesebb számolás költség miatt, minden esetben a magpontokból építjük fel a négyfákat.

A fák lekérdezésénél célszerű egy megfelelően nagy határoló téglalapot választani ahhoz, hogy egy fa minden szomszédja beleférjen. Legrosszabb esetben a két szomszédos fa magpontja a két fa ellentétes oldalán helyezkedik el. Tehát ebben az esetben két nagy kiterjedésű fa lombkoronájának mérete kell legyen a téglalap oldala. Több kísérletezés után a fa magpontjától minden irányban 18 - 18 métert választottam, tehát a befoglaló téglalap oldalainak mérete 36 méter.

A sokféle negyedelő-fa típus közül a ponttartomány négyfát használtam. A fák lombkorona pontjaihoz x, y, z, koordináta tartozik, de mivel két dimenzióban kérjük le a fák elhelyezkedését, ezért nincs szükség nyolcadoló-fára.

4. fejezet

Implementáció

Implementáció során a terület csempézését és a negyedelő-fás gyorsítást kellett végrehajtani a már meglévő algoritmusban. Ebben a fejezetben ismertetem a használt programokat és könyvtárakat. A régi folyamatábra kiegészítésével bemutatom az algoritmusba újonnan bejövő lépéseket. Ezt követően bemutatom az újonnan létrehozott osztályokat.

Munkámat az Eötvös Loránd Tudományegyetemen fejlesztett CloudTools nevezetű keretrendszer vegetation moduljában valósítottam meg, amely itt érhető el `https://gitlab.inf.elte.hu/gislab/cloudtools/`.

4.1. Használt programok és könyvtárak

A fejlesztés során a Microsoft Visual Studio 2022-es verziójú fejlesztőkörnyezetet használtam, verzió követéshez git-et és TortoiseGit-et. A kapott eredmények vizuális ellenőrzését QGIS földrajzi információs szoftverrel végeztem, amely lehetővé teszi a tiff kiterjesztésű fájlok vizuális vizsgálatát.

Az eredeti algoritmus futásához szükség volt Boost minimum 1.58-as verziójú könyvtárra, a térinformatikai adatformátumok kezeléséhez a GDAL minimum 2.2 verziót használták.

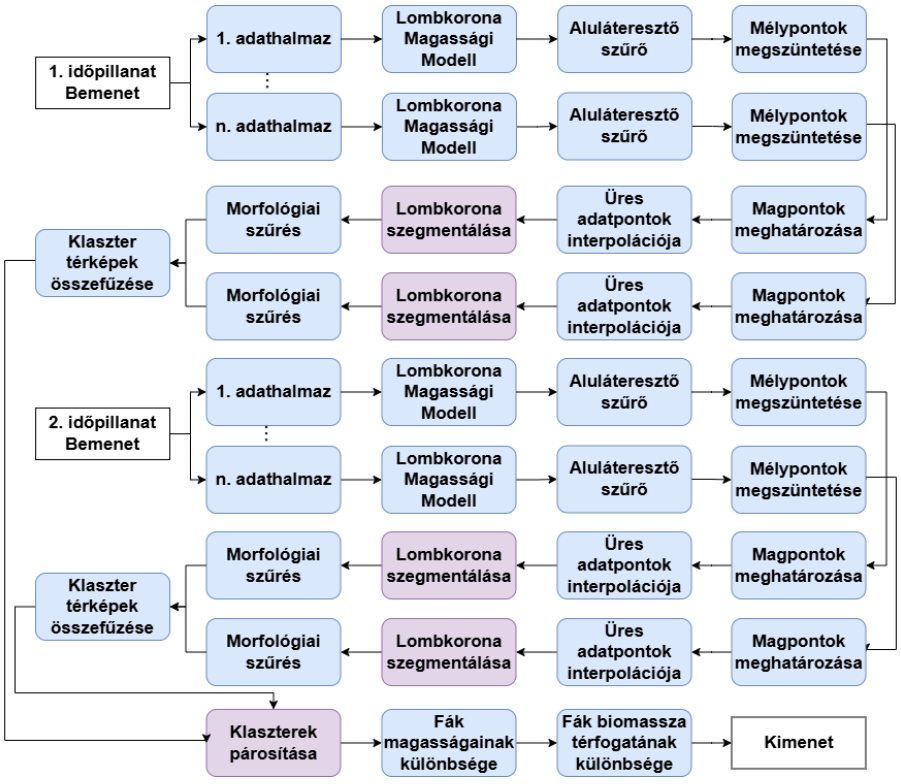
Az eredeti C++11-es verzió helyett a C++17-es standard könyvtárat használtam, mert ebben elérhető a STL (= Standard Template Library) párhuzamosított algoritmusai. Ezen algoritmusok lehetővé tesznek további paraméterezéseket. A paraméterek a következőek lehetnek:

- **execution::seq** Szekvenciálisan egy szálon futattja le az algoritmus. Bemeneti adatok sorrendje nem változik.
- **execution::par** Párhuzamosan futattja le az algoritmus több szálon. Sorrend nem garantált.
- **execution::par_unseq** Párhuzamosan futattja le algoritmus több szálon, vektorizációt használhat. Sorrend nem garantált.
- **execution::unseq** Egy szálon fut, de vektorizált utasításokat használhat. Sorrend nem garantált.

Az *execution* névtér a C++ párhuzamos és vektorizált irányelveit tartalmazza. Én a for_each ciklus $execution :: par$ paraméterrel hívtam meg, ami lehetővé tette a csempék konkurens feldolgozását.

A negyedelő-fa megvalósításához a CPL-t (= Common Portability Library) használtam, amely függvénykönyvtár egy ponttartomány négyfát definiál, annak műveleteivel.

4.2. Az algoritmus folyamatábrája



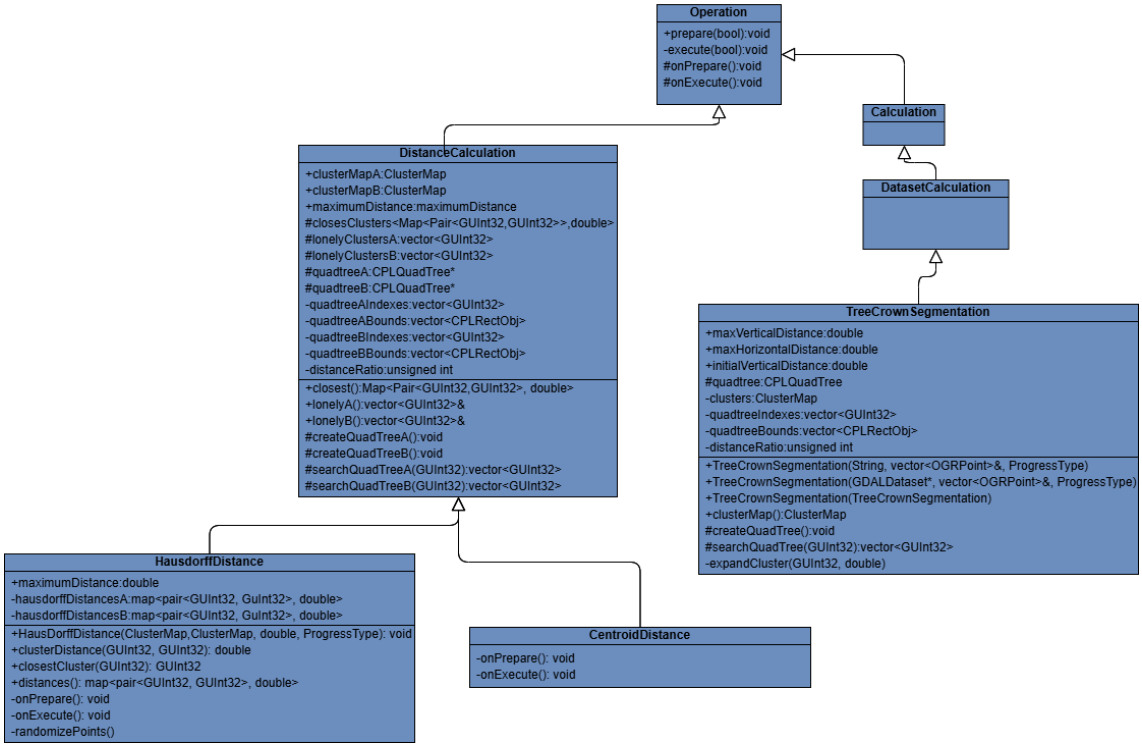
4.1. ábra. Algoritmus folyamatábrája

Az algoritmus folyamatábrájának megváltozását a 4.1. ábra mutatja be. A korábban leírtak szerint mindkét időpillanat DSM és DTM bemeneti paramétereit n darab csempére osztjuk és mindegyikre lefuttatjuk a vegetáció detektálást az eredeti módon, majd az így kapott n darab klasztertérképet összefűzzük egy darabbá és meghívjuk rá a változáselemzés metódusait.

A lilával megjelölt részekben került felhasználásra a ponttartomány négyfa.

4.3. Létrehozott, módosított osztályok

Ebben az alfejezetben bemutatom az általam módosított, vagy készített osztályokat.



4.2. ábra. Osztálydiagram a negyedelő-fa használatáról

A "DistanceCalculation" osztályt beszúrtam az "Operation" és a két számolási módszer osztálya közé. Ez az osztály biztosítja a negyedelő-fa használatához szükséges alapvető műveleti függvényeket. A két számolási osztályban kérdezzük le egy adott magpont környezetében elhelyezkedő többi magpontot. A másik osztály, ahol szintén használni lehetett a "TreeCrownSegmentation", ebben az esetben itt az osztályba tároltam el a létrehozott negyedelő-fát és ide implementáltam a szükséges függvényeket. A "HausdorffDistance" és "CentroidDistance" osztályok-

ban került meghívásra a negyedelő-fa keresése. További módosításokat végeztem a "ClusterMap" osztályban, ahol a klaszter mag és központi pontjainak határoló dobozát kérdezem le.

A csempe alapú feldolgozáshoz két osztályt használtam, amelyek a "main"-ből kerülnek meghívásra. Az egyik a "TileArea", amely a bejövő adatok csempésítését végzi. A másik a "MergeClusters", amely a részegységek klasztertérképbe való összefűzéséért felel. A "main"-ből hívom az előfeldolgozó részt párhuzamosan futtatva.

5. fejezet

Eredmények

Ebben a fejezetben bemutatom az általam elért eredményeket. A kiindulási algoritmus az én számítógépemen a különböző méretű területekre az 5.1. táblázatban bemutatott futási idő eredményeket produkálta. A táblázatban látható a futatott mintaterület, annak mérete, továbbá a két időpillanat eredményeinek párosítási módja, amely vagy Centroid, vagy Hausdorff. Az utolsó oszlop tartalmazza a futási időket.

Mintaterület	**Méret**	**Párosítási mód**	**Futási idő**
Utca	0,103 km^2	Centroid	4,25 mp
		Hausdorff	11,12 mp
Amszterdam városközpont	1,937 km^2	Centroid	16 p 39 mp
		Hausdorff	17 p 14 mp
TU Delft kampusz	2,143 km^2	Centroid	20 p 25 mp
		Hausdorff	25 p 49 mp
Delft városközpont	8,640 km^2	Centroid	8 ó 39 p
		Hausdorff	9 ó 41 p

5.1. táblázat. Kiindulási algoritmus futási eredményei
Forrás:[14]

A 5.2. táblázatban látható a mintaterület, a két időpillanatban talált klaszterek száma, a párosítási mód és a talált párok száma.

Mintaterület	1. időpillanat klaszterek	2. időpillanat klaszterek	Párosítási mód	Fa párok
Utca	168	173	Centroid	112
			Hausdorff	106
Amszterdam városközpont	3865	3585	Centroid	2157
			Hausdorff	2170
TU Delft kampusz	3834	4128	Centroid	2115
			Hausdorff	1922
Delft városközpont	17375	17634	Centroid	9878
			Hausdorff	9137

5.2. táblázat. Kiindulási algoritmus klaszterek és klaszterpárok száma
Forrás:[14]

5.1. Negyedelőfa eredményeinek bemutatása

A negyedelő-fa implementálása után az algoritmus nagyon jó futási eredményeket produkált. A futási eredmények összehasonlítását a 5.3. táblázat mutatja be, ahol az eredeti futási idő a kezdeti algoritmus futási ideje, míg a futási idő a negyedelő-fás módszeré. Minden területhez két futási idő van, mindig a felső a Centroid, az alsó a Hausdorff, ahogy a korábbi táblázatokban szerepelt.

Mintaterület	Méret	Eredeti futási idő	Futási idő
Utca	0,103 km^2	4,25 mp	3,24 mp
		11,12 mp	9,48 mp
Amsterdam városközpont	1,937 km^2	16 p 39 mp	1 p 29 mp
		17 p 14 mp	2 p 12 mp
TU Delft Kampusz	2,143 km^2	20 p 25 mp	2 p 4 mp
		25 p 49 mp	5 p 29 mp
Delft városközpont	8,640 km^2	8 ó 39 p	1 ó 11 p
		9 ó 41 p	1 ó 8 p

5.3. táblázat. Negyedelőfa futási eredményeinek összehasonlítása a kiindulási algoritmus futási eredményeivel

A táblázatból kiolvasható mind a Hausdorff, mind a Centroid párosításnál sokat gyorsult az algoritmus. A kis adatmennyiségű területeken nem számottevő a különbség, de a nagyobb területeken akár tizenhatszoros gyorsaság is megfigyelhető az implementáció előtti állapothoz képest.

Mintaterület	1. időpillanat klaszterek	2. időpillanat klaszterek	Párosítási mód	Fa párok
Utca	168	172	Centroid	110
			Hausdorff	104
Amszterdam városközpont	3877	3581	Centroid	2155
			Hausdorff	2171
TU Delft kampusz	3828	4124	Centroid	2107
			Hausdorff	1915
Delft városközpont	17370	17658	Centroid	9883
			Hausdorff	9145

5.4. táblázat. Negyedelőfa implementáció után a klaszterek és klaszterpárok száma

A két időpillanatban megfigyelhetünk kisebb számú fa detektálási eltérést a kiindulási algoritmus eredményeihez képest, amely kihatással van a talált párokra is. Ezen eltérések lehetséges okaira a 5.3. fejezetben fogok kitérni.

5.2. Két módszer egyszerre való alkalmazása eredményeinek bemutatása

A csempék számának és az átfedő terület változtatásával is próbálkoztam. Több átfedési méret kipróbálása után a 20 métert választottam, a csempék számát 2, 4, 8 darabra teszteltem. A 5.5. táblázat tartalmazza futtatási területek neveit rövidítve, a párosítási módot, a kiindulási algoritmus futási idejét, és mellettük ugyanazon terület, ugyanazon párosítási mód futási idejét a különböző darabszámú csempékre. Ezeket összehasonlítva látható, hogy komoly eltérés van az egyes csempék esetén, amelyeket nagy mértékben torzít a 5.6. táblázatban található detektálási eredmények. Amelynek okait kifejtem a 5.3. alfejezetben.

Terület	Pár. Mód	Eredeti futási idő	Futási idő 2 csempe	Futási idő 4 csempe	Futási idő 8 csempe
Utca	Centroid	4,25 mp	3 mp	2,57 mp	2,47 mp
	Hausdorff	11,12 mp	9,51 mp	9 mp	12,58 mp
Amsz. vk.	Centroid	16 p 39 mp	1 p 6 mp	33 mp	26,73 mp
	Hausdorff	17 p 14 mp	2 p 1 mp	1 p 25 mp	1 p 27 mp
TU Delft k.	Centroid	20 p 25 mp	1 p 49 mp	54 mp	43 mp
	Hausdorff	25 p 49 mp	3 p 58 mp	2 p 57 mp	3 p 3 mp
Delft vk.	Centroid	8 ó 39 p	23 p 15 mp	8 p 15 mp	4 p 7 mp
	Hausdorff	9 ó 41 p	30 p 21 mp	15 p 48 mp	12 p 14 mp

5.5. táblázat. Futási eredmények változtatott csempeszámok mellett

A 5.6. táblázatban láthatjuk az adott mintaterület mellett, az adott számú csempére, milyen detektálási és párosítási eredményeket kaptam.

Terület	Csempe szám	1. időpill.	2. időpill.	Pár. mód	Fa párok
Utca	2	166	172	Centroid	108
				Hausdorff	102
	4	154	167	Centroid	106
				Hausdorff	97
	8	152	168	Centroid	109
				Hausdorff	99
Amsz. vk.	2	3851	3544	Centroid	2209
				Hausdorff	2252
	4	3702	3400	Centroid	2251
				Hausdorff	2323
	8	3545	3254	Centroid	2327
				Hausdorff	2415
TU Delft k.	2	3596	3834	Centroid	2028
				Hausdorff	1839
	4	3255	3422	Centroid	1880
				Hausdorff	1726
	8	2881	2871	Centroid	1761
				Hausdorff	1589
Delft vk.	2	16233	16315	Centroid	9443
				Hausdorff	8771
	4	14298	14519	Centroid	8680
				Hausdorff	8108
	8	12373	11899	Centroid	7578
				Hausdorff	7075

5.6. táblázat. Detektálási és párosítási eredmények változtatott csempeszámok mellett

Megfigyelhető, hogy több csempe szám esetén, az algoritmus jóval kevesebb fát

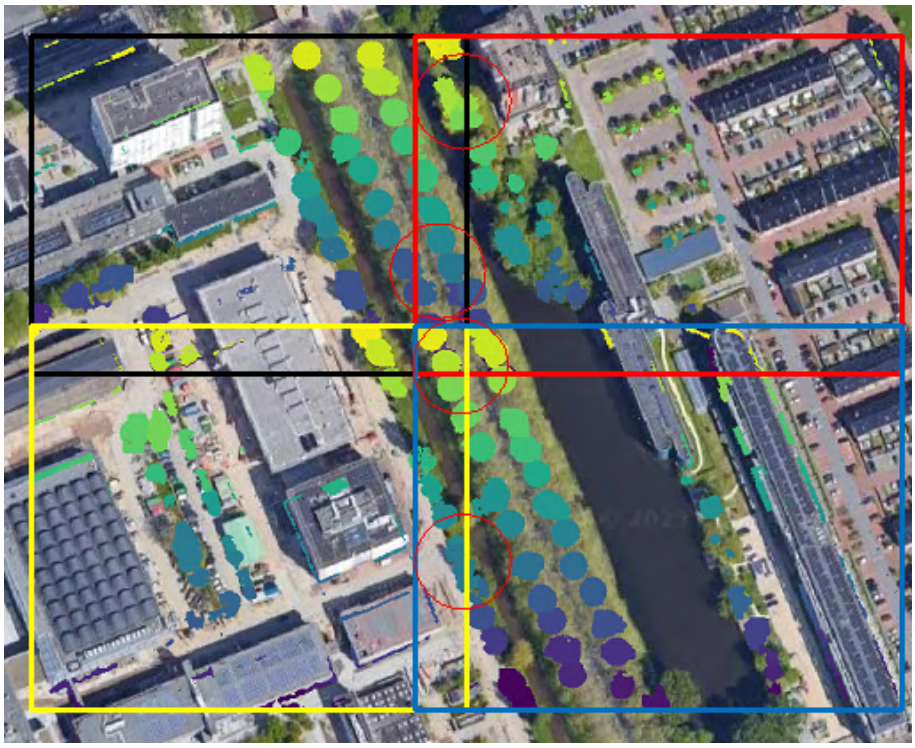
detektál az egyes csempékben, viszont arányaiban nem annyival kevesebb párt talál. Ennek miértjeire és a futási idők a detektálási eredményekkel való összefüggéseire a 5.3. fejezetben fogok kitérni.

5.3. Diszkusszió

Ebben az alfejezetben kielemzem a kapott futási időket a vegetáció detektálás és azok párosítása viszonylatában. Amennyiben egy kapott megoldásban a vegetáció detektálás és párba állítás jóval eltér a kiinduláshoz képest, nem érdemes futási időt nézni hiszen a különbség az adódhat a fák nagyarányú meg nem találása és ezáltal a kevesebb párosítási műveletből és nem a módszer hatékonyságából.

A 5.1. alfejezetben bemutatott negyedelő-fa futási eredményeinél látszott, hogy van pár fa eltérés a vegetáció detektálásnál. Az eredeti algoritmus minden fát minden fával összehasonlított. A negyedelő-fa a fák magpontjai szerint épül fel, ezért valószínűleg a magpontok meghatározásánál lehet valamilyen probléma, hiszen a magpontól való reális lekérdezési távolság határát jóval átlépve adta a kiindulási algoritmus megoldásait a program. Ez a kis mennyiségű detektálási eltérés a futási időre nem gyakorol hatást, tehát a négyfa módszernek köszönhető amilyen eredményeket a 5.3. táblázatban kaptunk.

A 5.2. alfejezetben a két megközelítés egyidejű alkalmazása esetén történő nagyarányú fa eltéréseket a csempe alapú megközelítés torzítja ekkora mértékben, hiszen a negyedelő-fa ugyanúgy működik, mint a korábbiakban, csak kisebb adathalmazokra. Ekkora arányú fa eltérés kihatással van a futtatási időkre, ezért a futási időt nem érdemes elemezni a módszer viszonylatában. Azt érdemes viszont, hogy mi okoz ekkora eltérést. A 5.6. táblázatból kiolvasható, hogy minél több csempére osztjuk fel az adathalmazt, annál rosszabb eredmények jönnek ki. Tehát feltételezhető az, hogy a csempe átfedéseinél lehet a probléma, hiszen minél több a csempeszám, annál több az átfedési terület.



5.1. ábra. Utca csempésített eredménye

A 5.1. ábrán a fák szegmentálásai láthatóak az egyes csempékben. A csempéket a jobb megkülönböztethetőség kedvéért külön színekkel jelöltem. Az ekkora méretű eltérés a csempehatárok miatti fák összevonása miatt történhet meg, amelyek közül párat jelöltem az ábrán, piros körökkel. A 3.4. ábrán ismertett csempehatárokon lévő fák a kiterjesztett csempében megtalálásra kerülnek jól, viszont abban a csempébe amelyikbe csak egy kicsit lóg bele, az a csempe összevonhatja egy másik fával és az eredeti algoritmusban talált fához képest, egy teljesen más milyen fa jön létre, mint a kiindulási algoritmusban.

A fáknak elkerülhetetlenül lesznek közös pontjaik és nem tudjuk eldönteni, hogy a közös pontok melyik fához tartozzanak. Amennyiben törlünk bármely fából pontokat és úgy próbáljuk meg beszúrni a közös klasztertérképbe, deformálatlan fákat fogunk kapni, ami szintén kihatással lesz az eredményekre. Ezért döntöttem úgy, hogy megnézem melyik klaszterban van több pont és azt rakom be a közös klaszter-térképbe.

6. fejezet

Összefoglalás

A cél egy olyan algoritmus futási idejének csökkentése volt, amelyet nagyon sok féle területen, nemes célokra tudnak alkalmazni. Diplomamunkám során két féle optimalizálási módszert valósítottam meg, amellyel a vegetáció detektálást és annak változáselemzését nagy mértékben sikerült gyorsítani úgy, hogy az egyik módszer hatására, az eredmény csak kis mértékben tért el, míg a másiknál jelentős eltérés volt a klaszterek találata és párosítása során.

Az algoritmus szűk keresztmetszeteinek megismerése után, több féle módszert és lehetőséget megfontolva, került sor egy negyedelő-fa alkalmazására - amely két helyen is használható volt - és egy csempealapú párhuzamosítás megvalósítására. Részletesen kitértem az alternatív ötletekre és azok elvetésének miként jére, továbbá bemutattam az alkalmazott módszerek háttereit és működéseit.

Végül összehasonlítottam a kiindulási algoritmus futási idejével és bemutattam az elért eredményeket, ahol akár kilenc-, tízszeres gyorsulás is megfigyelhető a negyedelő-fa esetén. A csempe alapú párhuzamos feldolgozás nagy mértékű adateltérése miatt a futási idő elemzése helyett, az eltérések okait mutattam be.

A jövőbeli munkák során érdemes lenne megállapítani a negyedelő-fák esetében a vélhetően hibás magpontokat, hogy az algoritmus a kiinduló eredményeket tudja produkálni.

További fejlesztési lehetőség a csempealapú párhuzamos feldolgozás klasztertérképek összefűzésésének lépése, ahol valószínűleg egy sokkal bonyolultabb számolási eljárás szükséges, hogy vissza tudjuk kapni a kiindulási eredményeket.

Köszönetnyilvánítás

Szeretném köszönetemet kifejezni Dr. Cserép Máté András témavezetőmnek, aki elvállalta témavezetésemet, számomra konzultációs időpontokat biztosított. Köszönöm türelmét, hogy végig segítette munkámat, ötleteket adott, bármikor fordulhattam hozzá implementációs, vagy diplomamunkámmal kapcsolatos kérdéseimmel.

Irodalomjegyzék

[1] Mingxuan Hu, Yajun Pang és Long Gao. „Advances in silicon-based integrated LiDAR". *Sensors* 23.13 (2023), 5920. old.

[2] Ninad Mehendale és Srushti Neoge. „Review on lidar technology". *Available at SSRN 3604309* (2020).

[3] Dr.Yashwant Arjunrao Waykar. „Lidar Technology: A Comprehensive Review and Future Prospects". *Journal of Emerging Technologies and Innovative Research* (2022).

[4] Telbisz Tamás–Székely Balázs–Timár Gábor. „Digitális terepmodellek". *Budapest, Eötvös Loránd Tudományegyetem Természettudományi Kar Földrajz-és Földtudományi Intézet Természetföldrajzi Tanszék* (2013).

[5] Hanan Samet. *Spatial Data Structures.* 1995.

[6] Orgován Krisztina. „Konvex- és mozgó objektumok indexelése". Dipl. Eötvös Loránd Tudományegyetem, 2011.

[7] Yannis Manolopoulos, Yannis Theodoridis és Vassilis J Tsotras. *Spatial Indexing Techniques.* 2009.

[8] Raphael A Finkel és Jon Louis Bentley. „Quad trees a data structure for retrieval on composite keys". *Acta informatica* 4 (1974), 1–9. old.

[9] Dinesh P Mehta és Sartaj Sahni. *Handbook of data structures and applications.* Chapman és Hall/CRC, 2004.

[10] K Palaniappan és J Fraser. „Multiresolution tiling for interactive viewing of large datasets". *17th Int. Conf. on Interactive Information and Processing Systems (IIPS) for Meteorology, Oceanography and Hydrology.* 2001, 338–342. old.

[11] Jeffrey Dean és Sanjay Ghemawat. „MapReduce: simplified data processing on large clusters”. *Communications of the ACM* 51.1 (2008), 107–113. old.

[12] Noel Gorelick. „Google earth engine”. *EGU general assembly conference abstracts*. 15. köt. American Geophysical Union Vienna, Austria. 2013, 11997. old.

[13] Junghee Jo és Kang-Woo Lee. „High-performance geospatial big data processing system based on MapReduce”. *ISPRS International Journal of Geo-Information* 7.10 (2018), 399. old.

[14] Anett Fekete és Mate Cserep. „Tree segmentation and change detection of large urban areas based on airborne LiDAR”. *Computers & Geosciences* 156 (2021), 104900. old.

[15] R Tyrrell Rockafellar és Roger JB Wets. „Set convergence”. *Variational analysis* (1998), 108–147. old.

[16] Anett Fekete és Máté Cserép. *Tree Segmentation and Change Detection of Large Urban Areas Based on Airborne LiDAR: Datasets and Supplementary Materials*. Mendeley Data. V2. 2021. jún. DOI: 10.17632/9thyzzwd5d.2.

Ábrák jegyzéke

Táblázatok jegyzéke