Beltéri pozicionálási módszerek hatékonyságának javítása adatbányászati módszerek támogatásával

Dr. Sallai Gyula, Nagy István

Budapesti Műszaki és Gazdaságtudományi Egyetem,
Távközlési és Médiainformatikai Tanszék
e-mail:{sallai,nagy.istvan}@tmit.bme.hu

2011. szeptember 26.

Kivonat

A beltéri pozicionálásban használt módszerek közül a példány alapú osztályozó megoldások adják az egyik legegyszerűbb megoldást a jelerősség alapú pozicionálás problémájának megoldására. A példány alapú megoldások azonban egyszerűségük mellett számos tulajdonságukban kevésbé hatékonyak, mint egyéb, adatbányászati módszereken alapuló társaik. A kutatási anyagban az algoritmus hatékonyságát vizsgáljuk meg és javasolunk módszert a számítási hatékonyság növelésére. A megoldások hatását a hatékonyságra valós adatokon végzett tesztekkel mutatjuk be.
Kulcsszavak: beltéri pozicionálás, adatbányászat, klaszterezés, KD-fa

1. Bevezetés

2. Pozicionálás megvalósítása példány alapú osztályozó algoritmussal

A beltéri pozicionáló rendszerek egy része vezeték nélküli hálózatok rádiójeleinek jelerőssége alapján működik, melyet más néven RSSI értéknek neveznek. Az RSSI a rádió-vételi technológiák általános mérőszáma, ez a szám általában láthatatlan a felhasználó számára. Negatív érték, -100-0 intervallumban mozoghat. Érdemes megfigyelni, hogy az AP-tól való távolság nem határozza meg egyértelműen az RSSI értéket a rádiójelek beltéri jelterjedése miatt. A helymeghatározás a felfogható egyfajta osztályozási, vagy regressziós problémaként. Ismeretlen, előre nem meghatározható attribútumokat, az XYZ koordinátákat kell meghatározni más ismert, meghatározható attribútumok alapján, tehát az RSSI vektorok, vagy más néven a helyre jellemző lenyomatok, a fingerprintek alapján.

WiFi alapú megoldás esetén két járható út között kell választani. Csupán a távolság alapján megjósolhatjuk, kiszámíthatjuk az egyes, adott pontból látható adók távolságát a vevőhöz képest, majd ha minimum három darab ilyen távolságot meghatároztunk, akkor az egyszerű háromszög-tétel segítségével meghatározhatók a vevő XYZ koordinátái. Ezzel a megoldással az a probléma, hogy pontatlan, mivel a jelterjedés tulajdonságai miatt, nem lehet egyértelműen megmondani, hogy milyen távol vagyunk az egyes adóktól, hozzáférési pontoktól. Egy másik lehetséges út, ha megelőző méréseket végzünk, majd a pontokból látható jelerősségeket eltároljuk a valós koordinátákkal együtt és ezeket a tárolt értékeket használjuk a helymeghatározáshoz. Mind a két módszernek vannak előnyei, illetve hátrányai. Amennyiben nem végzünk előzetes méréseket, tehát nem kalibráljuk előzetesen a rendszert, a helymeghatározás pontatlanabb lehet, azonban nincs szükség a helyszín pontos ismeretéhez. Amennyiben előzetes konfigurációt végzünk a tárolt adatok helyigényesek és külön időt kell szánni a mérések elvégzésére. Fontos, hogy megtaláljuk azt az optimális mérési mennyiséget, ami már elegendő információt tartalmaz a pontos helymeghatározáshoz, de még nem túl nagy mennyiségű ahhoz, hogy a tárolt adatokban való keresés hátrányosan befolyásolná a pozícionálást. Ennek a megoldásnak az előnye, hogy optimális adatmennyiség és alkalmazott módszerek esetén növelheti a pozícionálás pontosságát. A kutatási anyagban ismeretett pozicionáló rendszert a megfelelő működéséhez először kalibrálni kell. Ez azt jelenti, hogy a helyszínt bejárva sétákat kell tenni és megmondani a rendszernek, hogy adott RSSI értékek mely valós fizikai pontból érhetők el. Ezt a fázist nevezzük kalibrációs fázisnak. A rendszer ebben a fázisban eltárolja az adott pontból látható összes jelerősség értéket hozzáférési pontonként, illetve, hogy az melyik pontból látható. A következő fázis a tanítás. Ebben a részben létre kell hozni egy olyan modellt, amelyet alkalmazva a következő fázisban - a

helymeghatározás fázisában - meghatározhatók azok az értékek amiket nem ismerünk, de ismerni szeretnénk. Helymeghatározás során az előző fázisban létrehozott modellt használva, az adott pontból vett jelerősségek alapján szeretnénk meghatározni a pont koordinátáit. Ebben az esetben a modell bemenete a mért jelerősségekből képzett vektor, a kimenet pedig az térbeli koordináták.

A k legközelebbi szomszéd egy lusta osztályozó algoritmus, amely nem épít külön modellt. A módszer szerint a hasonló attribútumú objektumok hasonló tulajdonságokkal bírnak. A hasonlóságot a pontok Euklideszi távolsága alapján mérjük. A tanuló adatbázist eltároljuk és egy új, ismeretlen pont érkezése esetén megkeressük a hozzá legközelebb eső k darab pontot és az új pontot abba a kategóriába soroljuk, amely a leggyakoribb a k szomszéd között (többségi szavazás). A módszer regresszióra is használható. A k legközelebbi szomszéd segítségével a helymeghatározó rendszer megkeresi az adathalmazban tárolt pontok közül az RSSI vektor szerint vett k darab legközelebbi elemet, majd e pontok RSSI vektor szerint vett Euklideszi távolsága alapján súlyozza valós koordinátáikat. Könnyedén belátható, hogy nagy adathalmaz esetén a k darab legközelebbi pont meghatározása nagyon költséges feladat, mivel minden egyes pontra meg kell határozni két nagy dimenziójú vektor Euklideszi távolságát, közben tárolni kell a k darab legközelebbit.

3. Megoldások a számítási hatékonyság növelésére

A k legközelebbi elem módszerével történő helymeghatározás hátránya, hogy nagy adathalmaz esetén a lineáris keresés miatt nagyon lassú, mivel futási ideje egyenesen arányos az adathalmaz méretével. Olyan módszer létrehozása a cél, amellyel a módszer számítási hatékonysága növelhető. A helymeghatározó rendszer gyorsasága tulajdonképpen a k darab leghasonlóbb múltbéli pont megtalálásának gyorsaságával becsülhető, ezért az alapvető cél, hogy ezt a keresést gyorsítsuk valamilyen módon. A számítási hatékonyság növelésének egyik lehetséges módja, a pozicionáláshoz használt mérések tárolására szolgáló adatstruktúra módosítása. Amennyiben egy új adatstruktúrát használunk, úgy számolni kell az adatstruktúra építésének idejével, de mivel ezt indulásonként csak egyszer kell elvégezni, ezért ez a rendszer valós működési idejét csak nagyon kis mértékben befolyásolja.

A következőkben bemutatjuk azokat az elméleti megoldásokat, amelyek segítségével a megoldás számítási hatékonysága növelhető. Elemezzük azt is, hogy a módszertől miért várható javulás, illetve milyen előnyei, illetve hátrányai vannak az egyes megoldásoknak.

Olyan adatstruktúra keresése, létrehozása a cél, amely a pozícionáló rendszer helymeghatározásának legnagyobb költségét, tehát a keresési időt csökkenti. Ennek érdekében olyan adatstruktúrát kell létrehozni, amelyben a keresés több szinten történik. A legkézenfekvőbb ilyen adatszerkezetek a hierarchikus adatszerkezetek. A [5] könyv alapján az adatszerkezet az adatelemek egy olyan véges halmaza, amelyben az adatelemek között szerkezeti összefüggések vannak. Az összekapcsolás módja határozza meg az elemek egymáshoz való viszonyát, illetve azokat a műveleteket, amelyeket rajta végezhetünk. Formálisan a hierarchikus adatszerkezet olyan $< A, R >$ rendezett párt jelent, amelynél van egy kitüntetett elem, melyet gyökérelemnek hívunk. A gyökérelemre igaznak kell lennie a következő állításoknak:

- Gyökér nem lehet végpont.
- Bármely gyökértől különböző elem egyszer és csakis egyszer lehet végpont.
- Bármely, a gyökértől különböző elem a gyökérből elérhető.

A hierarchikus adatszerkezeteknél az adatelemek között egy-sok kapcsolat áll fenn. Minden adatelem csak egy helyről érhető el, de adott elemekből több adatelem is látható. A következő alfejezetekben bemutatott módszerek is ilyen struktúrára épülnek. Klaszterezés és kétszintű hierarchia esetén is van egy kitüntetett elem, amelyhez több pontot rendelünk hozzá. Ez a kitüntetett elem egyfajta középpontnak tekinthető, amelyhez a hozzá valamilyen értelemben vett leghasonlóbb elemeket soroljuk.

3.1. Kétszintű hierarchia alkalmazása

A kétszintű hierarchia a gyorsítási ötletek közül a legegyszerűbb. Nem használ semmilyen adatbányászati, klaszterezési illetve indexelési algoritmust. Az algoritmus egy paraméterrel rendelkezik (n), amely a kétszintű hierarchia első szintjén lévő, kitüntetett elemek számát adja meg. Az algoritmus a következőképpen működik:

1. Első lépésben kiválasztjuk a kitüntetett elemeket (továbbiaban centroidok). Ezek az elemek az adathalmaz feldolgozása során minden n-ikként beolvasott elemnek felelnek meg.
2. A többi elemet besoroljuk a centroidokhoz az alapján, hogy azok melyik elemhez vannak a legközelebb. Az elemeket csökkenő távolsággal soroljuk a kitüntetett elemhez.

3. Amikor egy adott elemhez keressük a k legközelebbi szomszédot, akkor elsőként redezzük a centroidokat, hogy azok melyik milyen messze vannak az adott ponttól és a legközelebbivel kezdünk, de az összes centroidot megvizsgáljuk.
4. Egy centroid elemein addig megyünk végig, amíg a centroidtól vett távolság és az eddigi legnagyobb távolság (a k legközelebbi közül) még kisebb, mint a cemtroid adott pontjának távolsága az aktuális ponttól (a sorba rendezés miatt, ha ez egyszer teljesül, akkor a többi elemnél is teljesül a listában).

A megoldás előnyez, hogy az adatstruktúrában való keresés esetén nem kell az egész adathalmazt végigolvasni, hogy megtaláljuk a kívánt elemeket, elegendő a legközelebbi csoportokban azokat az elemeket vizsgálni, amelyek távolsága kisebb, vagy egyenlő, mint a keresett pont és az adott csoport centroidja közti távolság, mivel azt feltételezzük, hogy nem teljesen véletlen a kalibrációs adatokban az egymáshoz közeli pontok távolsága. Ezen okból kifolyólag nagymértékű gyorsítás érhető el vele. Az algoritmus hátránya, hogy a csoportok kialakítása során nagymértékű csomósodások jöhetnek létre a térben, mivel kis területen belül, amennyiben nagyon sok pontot vizsgálunk, sok kitüntetett elemet veszünk fel, így sok csoportot kialakítva feleslegesen.

3.2. Klaszterezés alkalmazása

A klaszterezés elemek csoportosítását jelenti, mely során a hasonló elemeket azonos, különböző elemeket külön csoportba szeretnénk sorolni. A csoportok meghatározása nem egyértelmű feladat. Ugyanazon adathalmazt akár többféleképpen is felbonthatunk csoportokra. A klaszterezéshez elsőként meg kell adnunk, hogy mit értünk az elemek hasonlóságán, valamint, hogy mi alapján csoportosítjuk azokat.

A klaszterezés során nehézséget jelent, hogy egy n elemet tartalmazó adathalmaznak $O(e^{n*log(n)})$ (Bell-számnyi) felbontása lehetséges, vagyis az adatpontok számában exponenciálisnál nagyobb méretű keresési térben kívánunk egy optimális klaszterezést megtalálni.

Adott n elem. Tetszőleges két elem között (x, y) értelmezzük a hasonlóságot, azaz ebben az esetben nem az elemek hasonlóságával, hanem azok különbözőségével dolgozunk: $d(x, y)$. A $d(x, y)$-ra igazak az alábbiak:

1. reflexivitás: $d(x, x) = 0$;
2. szimmetria: $d(x, y) = d(y, x)$;
3. háromszög tétel: $d(x, z) < d(x, y) + d(y, z)$.

A klaszterezéssel olyan csoportok léterejöttét akarjuk elérni, amelyekben valamilyen speciális kritérium (sűrűség, távolság) alapján vannak besorolva az egymáshoz hasonló (közel lévő) pontok. Mivel egy adathalmazt a különböző klaszterező algoritmusok más és más klaszterekre bontanak fel, ezért nehéz a megfelelő klaszterező algoritmus kiválasztása, mivel nem ismert a legoptimálisabb klaszterezés, illetve az a tulajdonság, amely szerint a legoptimálisabb klaszterezést kiértékelhetjük. Egyaránt klaszterezhetünk az a mért jelerősségek (RSSI) alkotta térben és valós koordináták szerint is. Az RSSI térben történő klaszterezés esetén olyan csoportokat hozhatunk létre, melyek kialakítása során az RSSI koordináták hasonlóságát veszük figyelembe. Ennek előnye, hogy a keresés során, amikor a bemenetünk egy RSSI értékeket tartalmazó vektor, a klasztereink RSSI értékek szerint vannak csoportosítva. Valós koordináták alapján történő klaszterezés esetén - főles sűrűségalapú megoldásoknál - kiszűrhetőek azok a csomósodások, amelyek azért keletkeztek, mert adott útvonalakat relatíve többször tettünk meg, mint a többit. Az RSSI térben történő klaszterezés azonban nem teljesen független a valós koordinátákon történő klaszterezéstől, mivel az RSSI vektorok függnek a valós koordináták szerinti pozíciótól.

3.2.1. Sűrűség alapú klaszterezés

A legtöbb klaszterező algoritmus csak elliptikus alakú klasztereket képes kialakítani. A sűrűség alapú klaszterező eljárások ennek a hibának a kiküszöbölésére születtek. A klasztereket addig növesztik, amíg egy tartományon a sűrűség meghalad egy bizonyos korlátot, tehát egy adott sugarú körön belül mindig megtalálható bizonyos számú elem. A sűrűség alapú módszerek szerint egy klaszteren belül jóval nagyobb az elemek sűrűsége, mint a klaszterek között. A továbbiakban bemutatunk két klaszterező algoritmust, amely sűrűség alapján határozza meg a csoportokat.

DBSCAN. A DBSCAN[4] (*Density Based Spatial Clustering of Applications with Noise*) algoritmus a legelső sűrűség alapú klaszterező eljárás. A sűrűség meghatározásához két paramétert használ, melyek közül az egyik egy sugárjellegű (*eps*) paraméter, a másik egy elemszám küszöb (*minpts*).

A p elem szomszédjai azok az elemek, melyek p-től legfeljebb *epsz* távolságra vannak. A q elem sűrűség alapon elérhető p-ből ha: $q \in N_{eps}(p)s|N_{eps}(p)| \geq minpts$. A klaszterek határán levő elemek

eps távolságán belül nem mindig van *minpts* számú elem, így nem érhetőek el közvetlenül sűrűség alapon az elemek egymásból.

A q elem sűrűség alapon elérhető p-ből, ha léteznek $p_1 = p, \ldots, p_n = q$ pontok úgy, hogy p_{i+1} közvetlenül sűrűség alapon elérhető p_i-ből.

A p és q elem sűrűség alapon összekötöttek, ha létezik olyan o elem, amelyből p és q is sűrűség alapon elérhető. Tehát akkor tekintjük az elemek egy C részhalmazát klaszternek, ha $p \in C$ és, ha q sűrűség alapon elérhető p-ből, akkor $q \in C$ is teljesül. Ezt a feltételt maximalitásnak nevezzük. Valamint ha $p, q \in C$, akkor p és q sűrűség alapon összekötöttek.

Az algoritmus működése során először választ egy tetszőleges elemet (p) és meghatározza hozzá a sűrűség alapján elérhető elemeket. Amennyiben teljesül, hogy $|N_{eps}(p)| \geq minpts$, akkor meghatároztunk egy klasztert. A feltétel nem teljesülése esetén se mondhatjuk azt, hogy p zaj, mivel lehetséges, hogy p egy másik klaszter határán helyezkedik el, tehát egy másik klaszter eleme, ezért a következő lépésben választunk egy másik (p) elemet. Ha már nem tudunk új elemet választani, akkor az algoritmus véget ér. Egy elem, ha nem tartozik egyik klaszterbe sem, akkor olyan kívülálló pontnak tekintjük, ami zaj a többi adathoz képest. Az algoritmus előnye, hogy tetszőleges alakú klasztert képes felfedni és ehhez csak az elemek távolságát használja. Előfordulhat olyan eset is, hogy egy klaszter teljes egészében körbevesz egy másikat, mégis külön klaszternek tekintendő, mivel sűrűség alapon nem elérhetőek egymásból. Hátránya, hogy rendkívül érzékeny a két paraméterre, sőt amennyiben a klaszterekben található elemek sűrűsége eltérő, nem biztos, hogy található olyan paraméterezés, mellyel a DBSCAN jó eredményt ad. Egy másik hátránya lehet, hogy magas dimenzió esetén a klaszterezésnek nagyon nagy az időigénye, mivel egy pontot akár többször is meg kell vizsgálni. Megfelelő indexeléssel azonban ez leszorítható (n pontú adathalmaz esetén) $O(nlog(n))$-re.

OPTICS. Az OPTICS[1] (*Ordering Points To Identify the Clustering Structure*) egy a DBSCAN hiányosságait orvosolni kívánó, sűrűség alapú klaszterező algoritmus. Az alapkoncepciója megegyezik a DBSCAN algoritmuséval, de annak gyengeségét küszöböli ki, ami a klaszteren belüli különböző sűrűségekből adódik. Működése és paraméterei megegyeznek a DBSCAN algoritmusnál bemutatottakkal, azzal a különbséggel, hogy az *eps* távolságot, amit paraméterül vár, egy maximális távolságnak tekinti. Az algoritmus nem hoz létre konkrét klasztereket a DBSCAN algoritmusnál bemutatottakhoz képest, hanem minden ponthoz két új értéket rendel hozzá, melyek a *core-distance* és a *reachability-distance*. Az első érték a középponttól való távolság, amely kétféle értéket vehet fel: (1) abban az esetben, ha a DBSCAN-nél ismertetett feltétel nem teljesül UNDEFINED (nem definiált) értéket, (2) abban az esetben, ha teljesül a középponttól való távolságot kapja eredményül. A reachability-distance két pont (p és q) között azt jelenti, hogy p-ből q sűrűség alapon elérhető. Abban az esetben, ha p és q távolsága nagyobb mint a core-distance, a valós távolságot adja eredményül, ha kisebb, akkor a core-distance-t, egyébként szintén UNDEFINED.

Ezeket az új értékeket felhasználva megvizsgálhatjuk a tér sűrűségét, illetve a sűrűségváltozásokat, amire DBSCAN esetén nem volt lehetőségünk. Az algoritmus komplexitása megegyezik a DBSCAN algoritmus komplexitásával, így futási ideje is körülbelül azonos. Előnye, hogy kiküszöböli a fent említett problémát, hátránya, hogy nem határoz meg konkrét klasztereket, hanem csak a DBSCAN algoritmushoz használható az algoritmushoz szükséges paraméterek beállításához.

3.2.2. Particionáló klaszterezés

A partícionáló módszerek a pontokat k diszjunkt csoportra osztják úgy, hogy minden csoportba legalább egy elem kerül. Ezek a csoportok reprezentálják a klasztereket. Egy kezdeti partícionálás után újraparticionálás kezdődik, mely során az elemek egyik klaszterből a másikba kerülhetnek. A partícionáló algoritmusok esetén a klaszterek száma előre ismert.

A particionáló algoritmusok legismertebbike a k közép algoritmus[6]. A klaszterezés menete ebben az esetben a következő: a vektortérben megadott elemek közül kezdetben választunk k darab véletlen elemet. Ezután minden elemet besorolunk ahhoz a klaszterhez, amely hozzá a leközelebb helyezkedik el a vektortérben. A besorolás után új középpontot választunk, amely megegyezik a létrehozott klaszter mértani középpontjával. A besorolás és az új középpontválasztás addig ismétlődik, amíg a klaszterközéppontokban nem történik változás. A k közép algoritmus paramétere az a k érték, mely a klaszterek számát adja, valamint, hogy hányszor futtassuk az algoritmust. Ha ismert a k, tehát a klaszterek száma, valamint a d, tehát az adathalmaz dimenziója, akkor a k közép algoritmus költsége: $O(n^{d*k+1} * log(n))$, ha n az adathalmaz rekordjainak a száma. A k közép algoritmus esetén is szükség van a k paraméter optimális megválasztására. Kevés klaszter esetén az átlagos elemszám klaszterenként magas lehet, így a kívánt klaszter megtalálása rövid időn belül történik, viszont a klaszteren belüli keresés sokáig tarthat. Másfelől sok kis klaszter esetén, ha túl nagy a klaszterek száma és azok túl kevés elemet tartalmaznak átlagosan, akkor a megfelelő klaszter kiválasztása tart túl sokáig, a klaszteren belüli elemeket viszont nagyon gyorsan megtaláljuk. A k közép algoritmusnak számos hátránya van: lehet, hogy az algoritmus lokális optimumban áll meg, tehát az iterációk során nem változik semmi, pedig létezhet olyan csoportosítás, amely optimálisabb megoldást ad. Egy

másik probléma, hogy csak olyan elemek csoportosítására használható, amelyek vektortérben vannak megadva, mivel meg kell adni a klaszterek középpontját. A négyzetes hiba csökkentése érdekében célszerű az algoritmust egymás után többször is futtatni, mivel az véletlenszerűen választja ki a kezdeti klaszterközéppontokat, így különböző kezdeti pontokkal is megvizsgálhatjuk, hogy a kapott klaszterezés optimális-e.

3.2.3. Hierarchikus klaszterezés

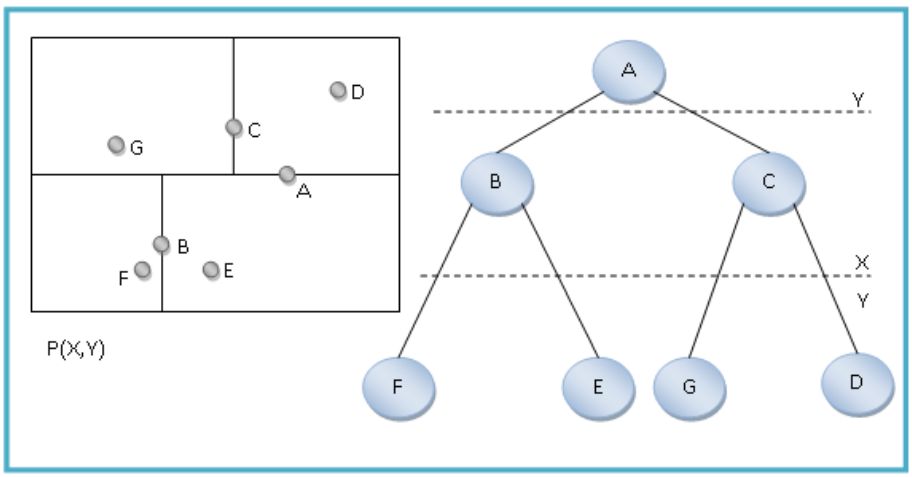
A hierarchikus klaszterező eljárások onnan kapták a nevüket, hogy az adatok egy hierarchikus felbontását hozzák létre, tehát az adatokat egy hierarchikus adatszerkezetbe sorolják be. A hierarchikus klaszterezőknek két fajtája létezik aszerint, hogy hogyan történik a csoportok egymásba ágyazásának sorrendje. Léteznek lentről építkezők, más néven egyesítők és fentről építkezők, avagy osztók. A lentről építkező eljárások esetén először minden elem külön klaszter, majd a nagyon közeli klasztereket egyesíti, amennyiben azok teljesítenek egy bizonyos feltételt. A fentről építkezők fordítva működnek, először minden elemet egy klaszterbe sorolunk be, majd ezt kisebb klaszterekre bontjuk mindaddig, amíg minden elem külön klaszterbe nem kerül, vagy teljesül a leállási feltétel.

Az egyik hierarchikus klaszterző algoritmus a Top-Down algoritmus, melynek különlegessége, hogy más, nem hierarchikus klaszterező algoritmusokat ágyazhatunk bele, mint belső operátor. A külső operátor, tehát a Top-Down algoritmus nem befolyásolhatja a klaszterek számát, tehát nem tudja se beállítani, se lekérdezni, hogy mennyi klaszter jött létre, ez a belső operátor, tehát a beágyazott klaszterező algoritmus feladata. Az algoritmus előnye, hogy sűrűségalapú és partícionáló algoritmusokkal is képes együttműködni, így rugalmas a partíciók létrehozása terén. Két paraméterrel rendelkezik. Az egyikkel megadhatjuk, hogy maximum milyen mély legyen a hierarchikus adatszerkezet, a másikkal beállíthatjuk az elágazási tényezőt, tehát, hogy egy pontnak, hány gyereke lehet.

A Farthest First algoritmus[3] egy nagyon egyszerű hierarchikus algoritmus, amely felosztó megközelítéssel működik. Paraméterül két értéket vár (N, S). Az első paraméterrel megadhatjuk, hogy az algoritmus hány klaszterre bontsa szét az adathalmazt, a másik paraméterrel pedig, hogy kezdetben hány véletlenszerű klaszterközéppontot hozzon létre. Az algoritmus úgy működik, hogy először kiválasztja azt az S db véletlenszerű pontot, amelyek a kezdeti középpontok lesznek. Ezután egy ciklus segítségével kiválasztja a maradék középpontokat, úgy, hogy mindig a már meglévő középpontoktól a legtávolabbi pont lesz középpont. Miután megvan az N darab középpont, a megmaradt elemeket besorolhatjuk a legközelebbi középpontokhoz.

3.3. k-d fa használata

Számos kutatási területen a k-d fa[2] vált a legelfogadottabb, leghatékonyabb adatstruktúrának magas dimenziójú adatok esetén. Működésileg megegyezik a bináris fákkal. Minden csomópontnak pontosan egy szülője és maximum kettő gyereke lehet. Minden csomópont rendelkezik egy vágási tulajdonsággal, ami azt adja meg, hogy az adott pont melyik koordinátája szerint vágja el a teret. Ez azt jelenti, hogy minden pont a szülőcsomópontja által meghatározott hipertestet a vágási koordinátája alapján két részre bontja.



1. ábra. k-d fa felépítése

A fában az egy szinten levő pontok azonos vágási tulajdonságokkal rendelkeznek ([?]). Azonos adathalmazból előállított k-d fa változatosan reprezentálhatja az adathalmazt, attól függően, hogy adott szinten melyik koordinátája szerint történt a vágás, illetve milyen sorrendben adjuk hozzá

a pontokat a fához. A k-d fák hátránya, hogy nem használhatók rajta a bináris fákra jellemző forgatások, így nehéz kiegyensúlyozottá tenni őket.

A k-d fák k paramétere nem azonos a k legközelebbi szomszédban megismert paraméterrel. Jelen esetben a k paraméter az adathalmaz dimenzióját jelenti, innen is kapta a k-d, tehát k dimenziójú fa nevet. Értelemszerűen a k paraméter nem lehet nagyobb az adathalmaz dimenziójánál. Az algoritmus a teret hipertéglatestekre bontja. A hipertéglatestek oldalai párhuzamosak egymással és a tengelyekkel, és egy téglatest kettéosztása mindig egy az oldalfallal párhuzamos sík mentén való kettéosztást jelent. Minden csomóponthoz hozzá vannak rendelve azok a tanítópontok, melyek az adott hipertéglatestbe esnek. K-d fát számos módon lehet építeni, tökéletesen kiegyensúlyozott fa-építési módszer azonban nem létezik. A k-d fába az elemeket szintenként változó koordináták szerint szúrjuk be. Amennyiben az adott eset koordinátája nagyobb, mint az aktuális csomópont, abban az esetben a fában jobbra, egyébként balra haladunk mindaddig, amíg lehetséges, majd beszúrjuk az elemet. Lehetőség szerint ügyelni kell arra, hogy a fa kiegyensúlyozott legyen, mivel ha ki szeretnénk zárni pontokat, akkor sok pontot szeretnénk egyszerre kizárni. A fa építésének időigénye az adathalmaz méretétől függ. Minden egyes pont beszúrásakor futtatunk egy keresést, melynek időigénye megegyezik a bináris fában való kereséssel, amely $O(log(n))$ n darab rekord esetén, valamint ezt n-szer szeretnénk végrehajtani, tehát: $O(n * log(n))$. A k-d fában való keresés megegyezik a bináris fában való kereséssel, azzal a különbséggel, hogy figyelembe kell venni a továbbhaladás során a vágási koordinátát. Időigénye n méretű adathalmaz esetén legjobb esetben $O(log(n))$, azonban rossz esetben közelíthet a lineáris kereséshez, melynek időigénye $O(n)$. A legközelebbi szomszéd keresés a következőképpen történik:

- Mélységi keresés a legközelebbi szomszéd megtalálása érdekében.
- Az így megtalált pont nem feltétlenül a legközelebbi, de egy jó közelítést ad rá.
- Meg kell vizsgálni, hogy az aktuális legközelebbi pont és a keresett pont által meghatározott hipertest, esetleg metszik-e más pontok által meghatározott hipertestet.
- Ha egy másik pont a legközelebbi szomszéd, akkor annak közelibbnek kell lennie a fa gyökeréhez, mint a levél, amit megtaláltunk.
- Fellépünk a szülőjéhez és megvizsgáljuk a szomszédját.
- Ha a szomszéd által reprezentált hipertestet nem metszi a hipergömb, akkor felfele haladunk a fában, egészen addig, amíg el nem érjük a gyökeret. Természetesen eközben folyamatosan megvizsgálunk minden pont testvérét.
- Ha a két hipertest metszi egymást, akkor meg kell vizsgálni az elmetszett hipertestet. Ez azt jelenti, hogy a hipertest gyökere által reprezentált részfán újra le kell futtatni a keresést.

Ahhoz, hogy megvizsgáljuk, hogy adott pont által meghatározott hipergömb metszi-e a szomszédos hipertéglatestet a következő információkra van szükség: a hipergömb középpontja, a keresett pont (t). A hipergömb sugara, a keresett pont és az aktuális legközelebbi pont távolsága (r). A hipertéglatest (H) által meghatározott terület maximuma és minimuma.

A hipergömböt a következőképpen jelöljük: $Y(t, r)$. A hipertest által meghatározott terület: $H = ((H_1^{min}, \ldots, H_n^{min}), (H_1^{max}, \ldots, H_n^{max}))$. Ahhoz, hogy megvizsgáljuk, hogy Y metszi-e H-t, ki kell számítanunk p pontot:

$$p = (p_1, \ldots, p_n)p_i = \begin{cases} H_i^{min}, \text{ ha } t_i \leq H_i^{min} \\ t_i, \text{ ha } H_i^{min} < t_i < H_i^{max} \\ H_i^{max}, \text{ h } t_i \geq H_i^{max} \end{cases} \quad (1)$$

Akkor metszi egymást a két hipertest, ha a p és t pontok Euklideszi távolsága kisebb, mint r.

4. Teszteredmények

Ebben a fejezetben először bemutatjuk a konkrét mérési környezetet valamint az adathalmazt, amin a vizsgálatainkat végeztük. Definiáljuk, hogy milyen szempontok szerint végeztük a vizsgálatot, valamint milyen feltételeket kellett teljesítenie az egyes módszereknek. Az általunk elvégzett méréseket a rendszer eredeti megoldásával fogjuk összehasonlítani.

4.1. Mérési környezet bemutatása

A megoldások teszteléséhez használt adathalmaz egy nagyáruházban került rögzítésre. Az áruházban soronkén két darab vezeték nélküli adó került elhelyezésre (összesen 60 darab), mely képes volt a bevásárlókocsikra szerelt wifiképes esközök jelét venni. A kalibráció során a sorok között útvonalakat tettek meg a felhasználók, melyekben az egyes AP-k mért jelerősségeit a rendszer rögzítette egy központi adatbázisban.

4.2. Adathalmaz bemutatása

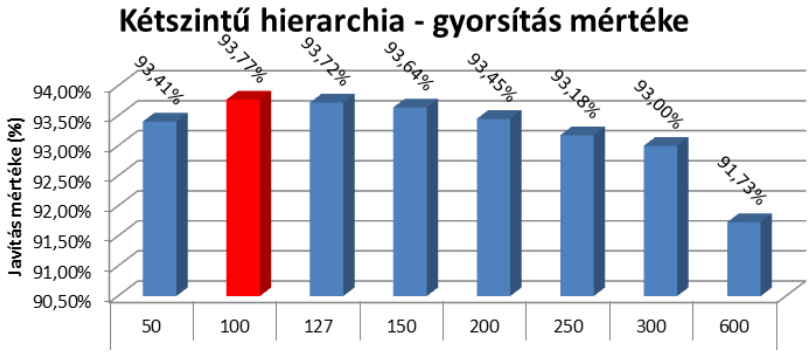
Mint már a fentiekben bemutattuk a rendszer az adatokat egy központi adatbázisban tárolja. Az adatbázis rekordjai úgy épülnek fel, hogy tárolják a időbélyeget, tehát, hogy mikor történt a mintavételezés. Ezen kívül a rendszer tárolja a mérés azonosítóját, tehát, hogy az adott rekord melyik méréshez tartozik. Erre azért van szükség, mert egy mérés során több AP jelerősségét is látjuk, és ez az adatbázis elsődleges kulcsának egyik elsődleges attribútuma, mellyel a méréseket azonosítani lehet. A másik elsődleges attribútum, melyet minden mérés alkalmával tárol a rendszer, a vezetéknélküli adó azonosítója. Ebből az utóbbi két attribútumból, melyek minden mintavételezés során páronként egyediek, állítja össze a rendszer az ujjlenyomatokat a tárolt RSSI értékekből. Az egyes rekordokban ezeken kívül még tárolni kell a mérés helyszínét, tehát a valós koordinátákat. A rendszer az adatbázisból lekéri a rekordokat, majd az elsődleges kulcs, a WiFi tag azonosítója és a mérés azonosítója alapján létrehozza a fingerprinteket. A kalibráció hozzávetőlegesen kétszer fél napig tartott, így 78.775 mérés történt. Ennyi fingerprintet tartalmazó halmazból kell kiválasztani a számunkra megfelelőket, a k legközelebbit.

4.3. A számítás hatékonyságának növelésére kidolgozott módszerek értékelése

Az egyes módszerek esetén vizsgáltuk a tesztadathalmaz pontjai megtalálásának idejét valamint a keresés kimenetét. A keresési időből sok következtetés levonható, melyek közül a legfontosabb az átlagos keresési idő, mivel minél alacsonyabb ez az érték, annál gyorsabb a rendszer működése, tehát a pozíció meghatározása. Érdemes még megfigyelni az egyes módszerek keresési idejének szórását. A szórás megadja, hogy az egyes elemek milyen mértékben térnek el az átlagtól. A feladat szempontjából ez azért fontos, mert folyamatos képet szeretnénk kapni a mozgásróln. Amennyiben a szórás nagyon nagy, tehát előfordulhat, hogy az egyik pontot nagyon hamar, majd a következőt nagyon lassan találjuk meg, a mozgásról alkotott képünk szaggatottá válhat. Fontos még a pontosság mérése is, ugyanis csak abban az esetben beszélhetünk az egyes algoritmusok által elért gyorsítás mértékéről, amennyiben a pontosságuk, tehát a meghatározott pozíciók megegyeznek az eredeti k legközelebbi szomszéd által, tehát a lineárisan megkeresett k darab legközelebbi szomszéddal, ami könnyen belátható, hogy minden esetben valóban a k darab legközelebbi elemet találja meg.

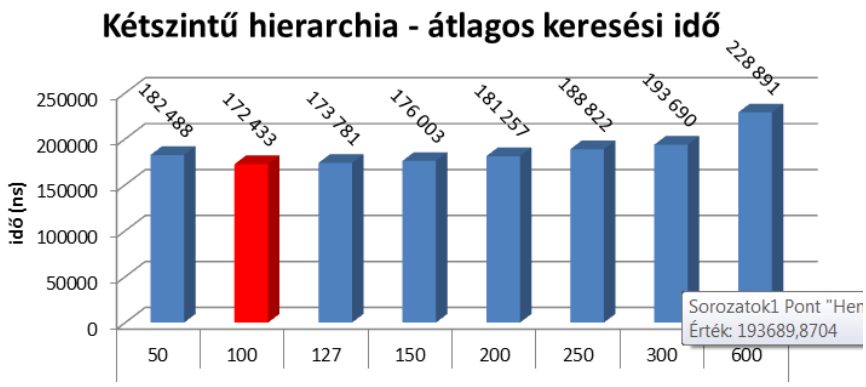
4.3.1. Kétszintű hierarchia

A kétszintű hierarchia adatszerkezetének létrehozása megnövelte a helymeghatározó rendszer inicializációs idejét. A paramétert növelve az inicializációs idő csökkent, megfigyeléseink alapján lineárisan. Ennek oka, hogy a paraméter növelésével egyre kevesebb középpontot választ ki a rendszer és a maradék adatpontok besorolása során egy kisebb halmazból kell kiválasztani a legközelebbi elemet, tehát kevesebb összehasonlítást igényel a megfelelő középpont kiválasztása.



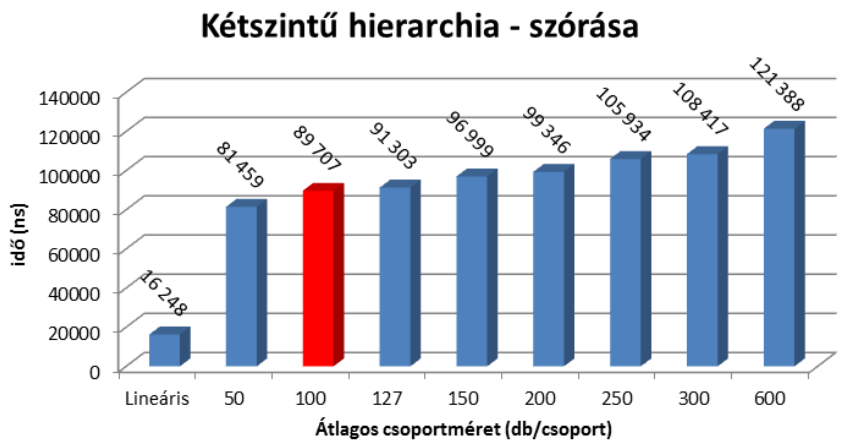
2. ábra. Kétszintű hierarchiával elért gyorsítás mértéke

Ezzel ellentétben viszont a a 2. ábrán látható, hogy a keresési időt sikeresen csökkentette a módszer. Az ábra vízszintes tengelyén látható a centroidok száma, a függőleges tengelye pedig azt mutatja, hogy az így elért keresési idő hány százalékkal kisebb, mint az eredeti megoldás esetén. A paramétert 50-es léptékkel vizsgáltuk, melynek során arra a következtetésre jutottunk, hogy, ha minden századik elemt választjuk ki középpontnak, akkor érjük el a legnagyobb gyorsítást. Ennek oka valószínűleg az, hogy ebben a legoptimálisabb az egyes csoportok mérete. A 3. ábrán látható, hogy az átlagos keresési idők konkrétan hogyan alakultak. A 100-as értéknél mért keresési idő minimális, így ez a legoptimálisabb választás, amennyiben ezt a módszert használná a rendszer.



3. ábra. Kétszintű hierarchiával elért átlagos keresési idők

A 4. ábrán látható, hogy a paraméter növelésével nőt a keresési idő szórása, tehát az előbbiekben említett „jitterszerű" hatás egyre jobban érezhető a rendszeren, egyre szaggatottabbá válhat a megfigyelés. Minél több, tehát átlagosan minél kisebb csoportot hozunk létre, annál nagyobb lesz a keresési idő szórása. Ennek oka feltehetően az, hogy a csoportok mérete, tehát a középponttól a legtávolabbi elemig vett távolság nagyon nagy, így ha a keresett elem legközelebbi középpontja egy nagyon nagy csoport közepe és a keresett elem relatív messze helyezkedik el a középponttól, tehát a csoport szélén, a határon elhelyezkedő pontok a legközelebbi szomszédjai, akkor nagyon sok elemet meg kell vizsgálni, hogy megtaláljuk a megfelelő elemet. Ezzel ellentétben, ha a keresett pont a középponthoz nagyon közel helyezkedik el, akkor nagyon gyors ez a módszer.

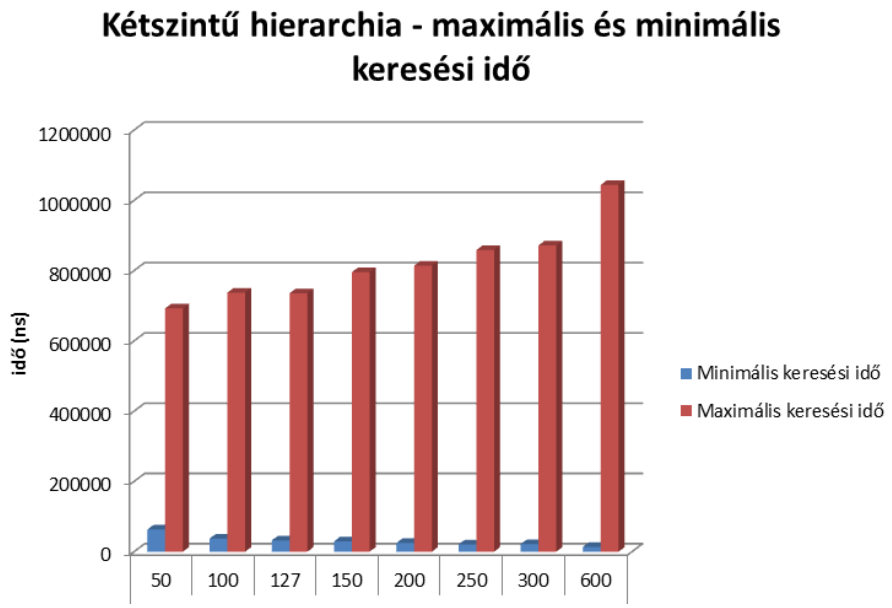


4. ábra. Kétszintű hierarchiával elért keresési idők szórása

Az 5. ábrán látható, hogy a minimális és maximális keresési idők hogyan alakultak a módszer alkalmazása során. Megfigyelhető, hogy nagyon nagy eltérések lehetnek a rendszerben a keresett pont megtalálásának idejében. Ennek oka ugyan azzal magyarázható, mint amit a szórásnál bemutattunk. Legrosszabb esetben előfordulhat, hogy az egymásután következő pontok olyan sorrendben jönnek, hogy mindegyiket a maximális időnek megfelelő idő megtalálni. Ebben az esetben ugyan kvázi folyamatos képet kapunk, de sokkal nagyobb léptékekkel követhető a mozgás. Megfigyelhető azonban, hogy ebben az esetben is gyorsabb ez a megoldás, mint az eredeti.

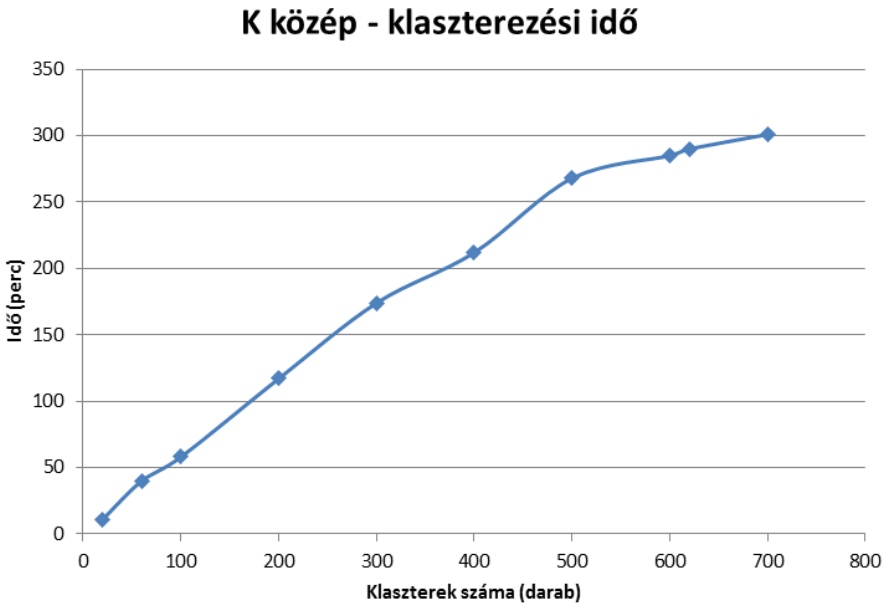
4.3.2. Klaszterezés

A különböző klaszterező eljárásokat egy ingyenes elérhető adatbányászati keretrendszerben teszteltük, majd a klaszterezések kimenetét - a klaszterező algoritmus kimenete alapján - használtuk fel a megfelelő hierarchikus adatstruktúra létrehozása során. Megfigyeltük, hogy az egyes klaszterező eljárások mennyivel hosszabbítják meg az adatstruktúra létrehozásának idejét. Ez természetesen függ a klaszterező eljárás típusától és paraméterezésétől. Ez az idő nem befolyásolja jelentősen a helymeghatározó rendszer gyorsaságát, mivel ez az inicializációs fázisának idejét növeli csak meg, amit csak egyszer kell lefuttatni minden indításnál.



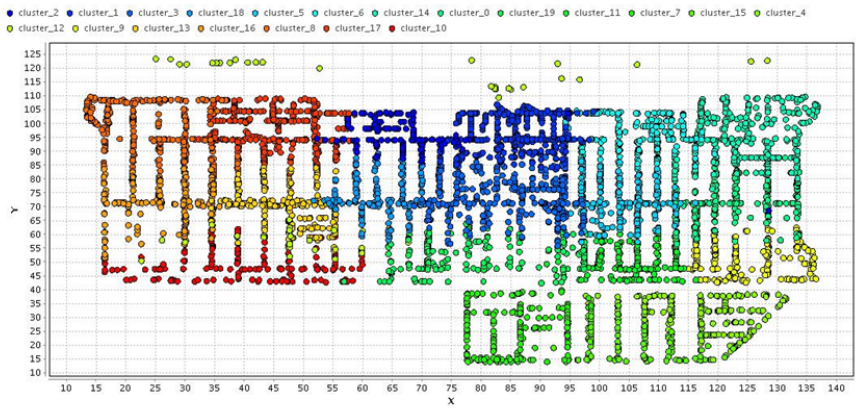
5. ábra. Kétszintű hierarchiával elért maximális és minimális keresési idő

Particionáló klaszterezések értékelése. A particionáló klaszterező eljárások közül a k közép algoritmust választottuk. A klaszterező eljárás kimenete ebben az esetben a klaszterek középpontja volt az RSSI térben, amelyhez meg kellett keresni a legközelebbi elemet, majd besorolni a hozzá legközelebb eső pontokat. A klaszterek számát úgy választottuk meg, hogy folyamatosan növeltük azt egészen addig, amíg folyamatos javulást tapasztaltunk, tehát az átlagos keresési idő csökkent, majd kisebb léptékekkel megkerestük az optimális klaszterszámot. A klaszterező algoritmus futási idejét mutatja a 6 ábra. Megfigyelhető, hogy a futási idő körülbelül lineárisan változott és egyenesen arányos a klaszterek számával.



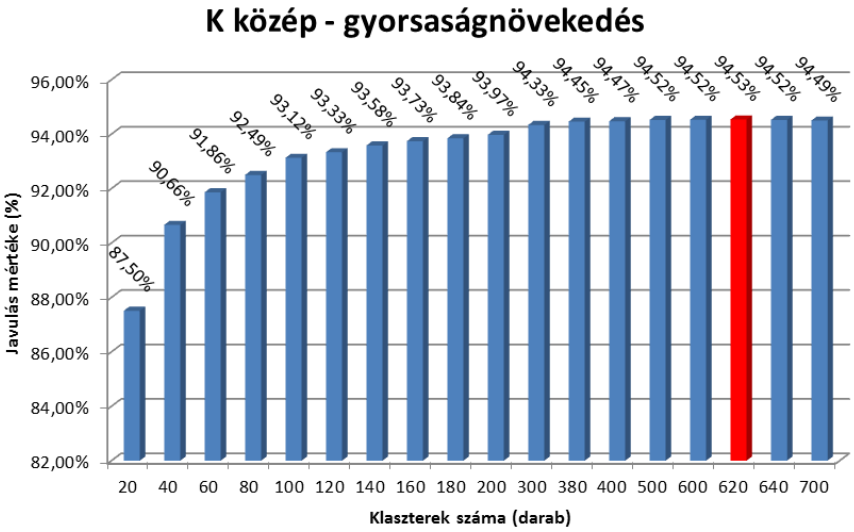
6. ábra. A k közép algoritmus klaszterezési ideje

A klaszterek térbeli elhelyezkedését a 7. ábra reprezentálja. Megfigyelhető, hogy a klaszterek jól körülhatárolhatók és nincsenek szuperklaszterek, melyek lefedik az egész teret, mivel a k közép algoritmus optimalizálta a klaszterek elhelyezkedését, a klaszterközéppontok pozícióját. A k közép algoritmussal létrehozott klaszterek sikeresen gyorsították a rendszer keresési idején (8. ábra). Ennek oka, hogy sikerült olyan csoportokat létrehozni, amelyek optimálisan oszlanak el az RSSI térben. Az



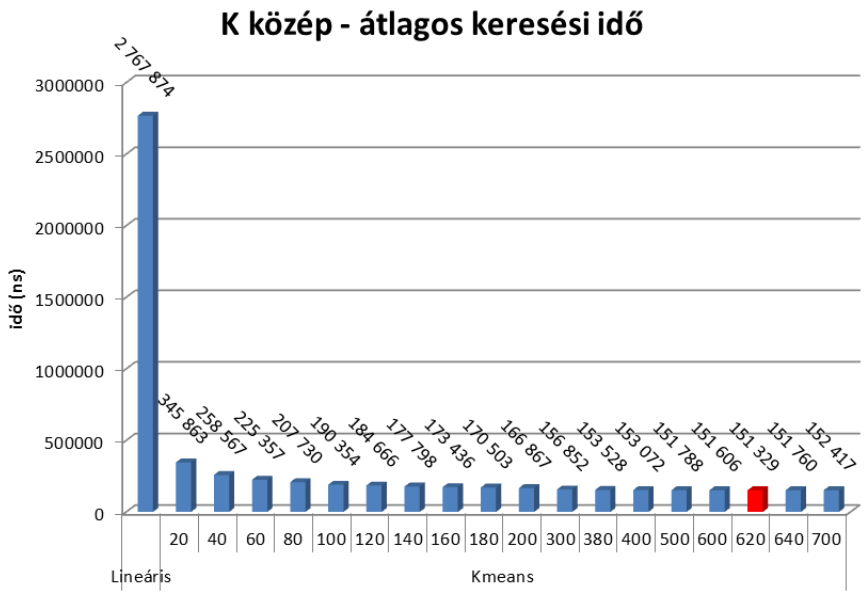
7. ábra. A k közép algoritmus által létrehozott klaszterek

eredeti megoldáshoz képest akár 94,53%-os javulást is sikerült elérnünk, ami kevesebb, mint tizedannyi keresési időt jelent, mint az eredeti megoldás esetében, tehát a k közép algoritmus használata sikeresen gyorsította a rendszer működését, tehát adott időn belül több pozíciót lehet meghatározni, több adat gyűjthető a mozgásról. A 9. ábra mutatja, hogy az eredeti lineáris kereséshez képest, hogy alakultak az átlagos keresési idők. Megfigyelhető, hogy egy bizonyos klaszterszámig (pirossal jelölt) a keresési idő csökkent, majd e fölött a klaszterszám fölött elkezdett növekedni. Ennek oka, hogy ez alatt a klaszterszám alatt a klaszterek mérete túl nagy, túl nagy halmazban kell megtalálni az elemeket. E fölött a klaszterszám fölött viszont túl nagy a klaszterközéppontokat tartalmazó halmaz mérete, vagyis túl nagy halmazból kell kiválasztani, hogy melyik klaszterben, klaszterekben keressük a legközelebbi k darab elemet.

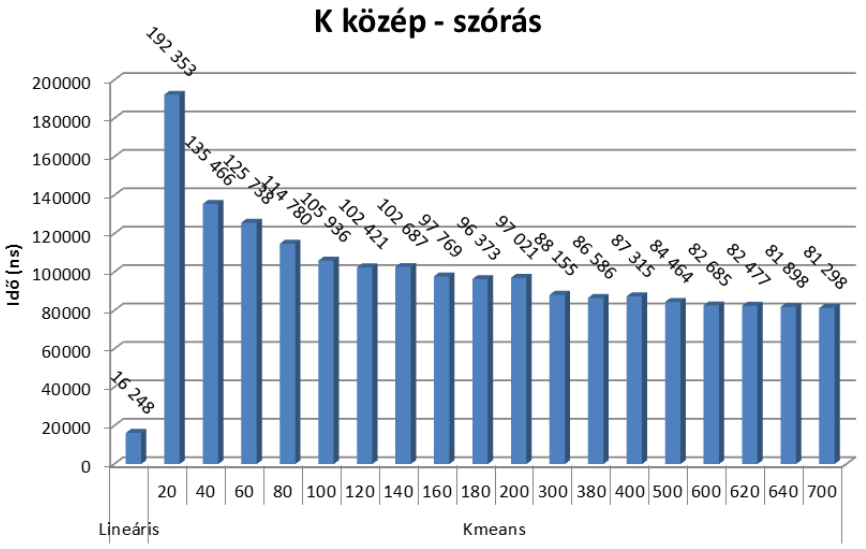


8. ábra. A k közép klaszterezéssel elért gyorsítás mértéke

A 10. ábra mutatja a keresési idők szórását. Látható, hogy az eredeti megoldás esetén ez az érték minimális volt, tehát az egyes pozíciókat szinte azonos idő alatt találta meg a rendszer. Sajnos a hierarchikus elrendezések hátránya a szórásn övekedése. Ahogy a diagramon is látható a szórás egyes esetekben akár tízszeres is lehet, annak ellenére, hogy minimális klaszterszám mellett is gyorsítást értünk el. Előfordulhat, hogy ilyen esetben a mozgásról alkotott kép szaggatottá válhat, nem tűnik olyan folyamatosnak, mint az eredeti esetben. Ez esetenként zavaró lehet, mivel olyan információkat kaphatunk, hogy bizonyos helyeken a megfigyelt objektum áll, közben csak a pozicionáló rendszer keresi az adathalmazban a helyzetét. A 11. ábra a maximális és minimális keresési időkről készült, melyeket a k közép algoritmus használatával értünk el. Megfigyelhető, hogy esetenként hatalmas különbséget tapasztaltunk a keresési időkben, de a klaszterek számának növelésével ez a különbség egyre csökkent. Ennek oka, hogy nagy klaszterben, ha a keresett pont a klaszter határvonalának közelében helyezkedik el, nagyon nagy adathalmazt kell átvizsgálni, hogy megtalálja a legközelebbi



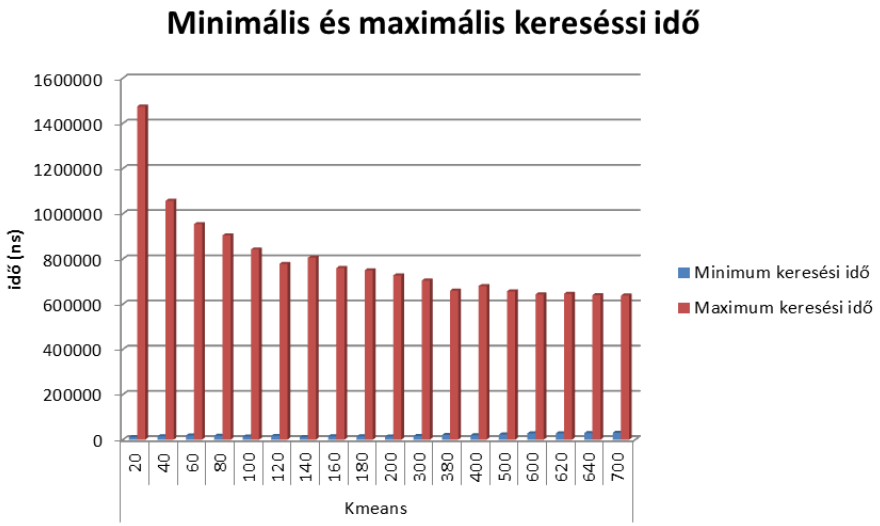
9. ábra. A k közép klaszterezéssel elért átlagos keresési idő



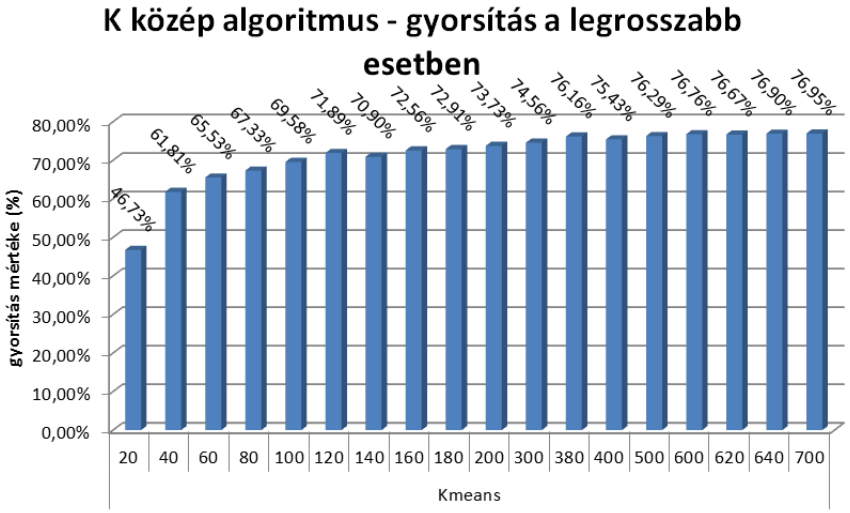
10. ábra. A k közép klaszterezéssel elért keresési idők szórása

elemeket, azonban amennyiben a középponthoz közel helyezkedik el, az adatstruktúra felépítése miatt csak nagyon kevés pontot kell megvizsgálnia, mivel az adatstruktúrában a pontok középponttól mért távolságuk alapján rendezve vannak, így nem vizsgálja meg az egész klasztert. Abban az esetben, ha olyan pontokat kell megtalálnunk, amelyek mindegyike határvonalon helyezkedik el a keresési idő jelentősen megnő az átlaghoz képest, de ahogy a 12. ábra is mutatja, ez még mindig jelentősen kisebb, mint az eredeti esetben.

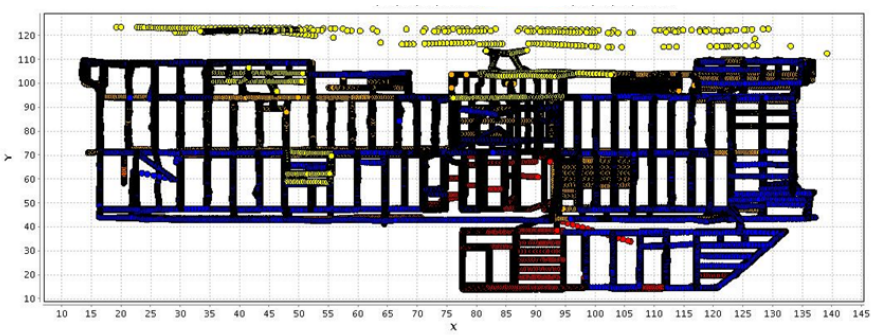
A hierarchikus klaszterezések értékelése. A hierarchikus klaszterező eljárások használatával vegyes eredmények születtek. A Top-Down algoritmusba két féle típusú klaszterező algoritmust ágyaztunk. Először sűrűségalapú klaszterező algoritmussal vizsgáltam, melynél a klaszterek térbeni elhelyezkedése a 13. ábrán látható. Megfigyelhető, hogy a klasztereket nem lehet jól elhatárolni, nincsenek meg azok a határvonalak, melyek mentén a tér feloszthatóklaszterekre. Kék színnel jelölve látható egy nagyon nagyméretű klaszter, mely szinte az egész teret magába foglalja. Ennek oka valószínűleg, hogy a tér sűrűség alapon nem feloszthatóklaszterekre, mivel körülbelül azonos a sűrűségeloszlás.



11. ábra. A k közép algoritmussal elért maximális és minimális keresési ideje

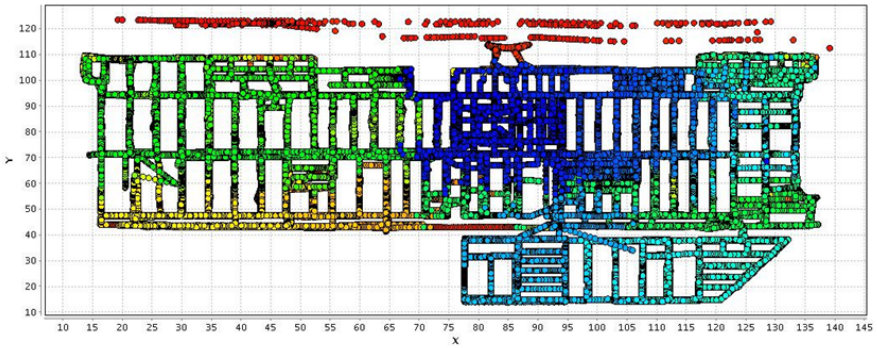


12. ábra. A k közép algoritmussal elért legrosszabb gyorsítás



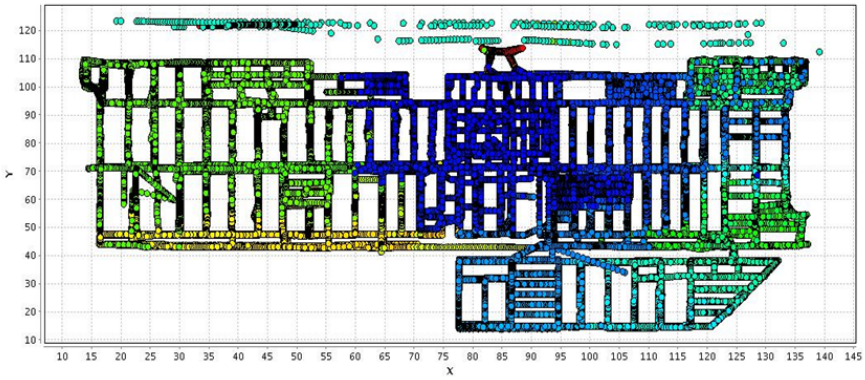
13. ábra. Hierarchikus klaszterező eljárásba ágyazott sűrűség alapú klaszterező eljárás által létrehozott klaszterek elhelyezkedése

A Top-Down algoritmusba ezen kívül még partícionáló algoritmust ágyaztunk. A beágyazott k közép algoritmus a teret sokkal jobban felosztotta, láthatók azok a határvonalak, melyek mentén megállapítható az egyes klaszterek találkozása (14. ábra).



14. ábra. Hierarchikus klaszterező algoritmusba ágyazott partícionáló klaszterező algoritmus által létrehozott klaszterek térbeni elhelyezkedése

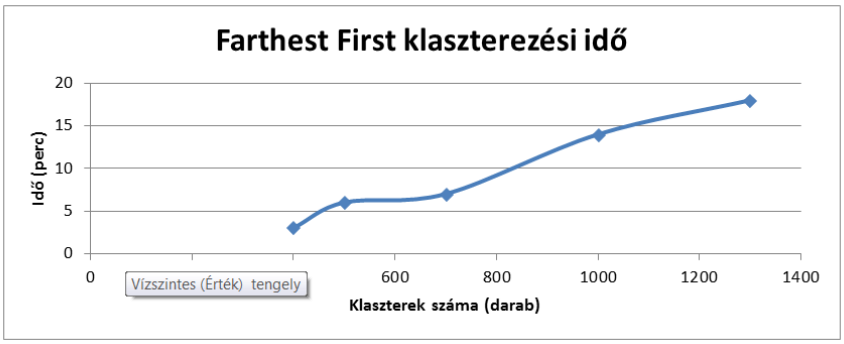
A másik hierarchikus klaszterező eljárás, mellyel vizsgáltam a rendszert, a Farthest First algoritmus, amely a beágyazott k közép algoritmushoz hasonló elhelyezkedésű klasztereket hozott létre (15. ábra).



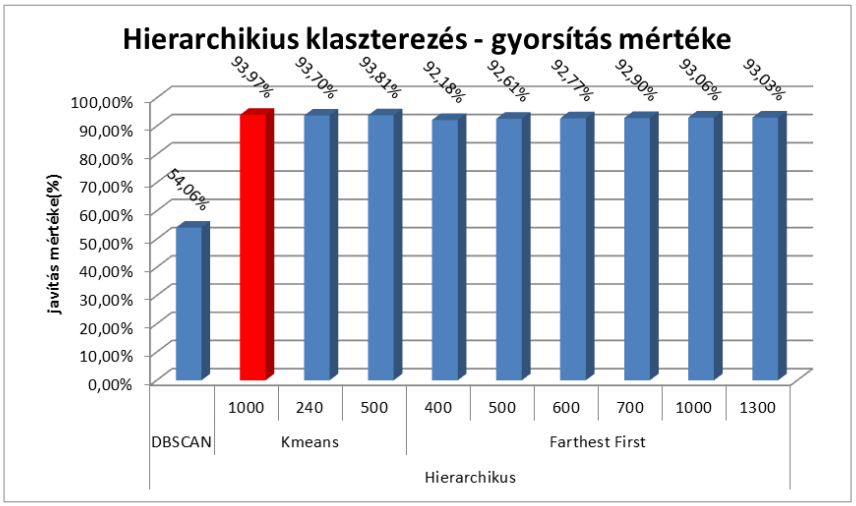
15. ábra. A Farthest First algoritmus által létrehozott klaszterek elhelyezkedése

A klaszterek méretéből és elhelyezkedéséből már következtetni lehetett arra, hogy a beágyazott sűrűségalapú eljárás által létrehozott klaszterek nem fogják olyan mértékben gyorsítani a keresést, mint a k közép és Farthest First algoritmus által létrehozott klaszterek, mivel tartalmaznak olyan klasztert, melynek mérete akár százszorosa is lehet a többi klaszter méretének, ami jelentősen befolyásolja a keresés sebességét. A sejtés a mérés folyamán bebizonyosodott, amelyet a 16. ábra mutat. Megfigyelhető, hogy a sűrűség alapú DBSCAN klaszterező algoritmus közel feleakkora gyorsítást ért el, mint a többi megoldás. Ennek oka, hogy tartalmaz szuperklasztert, amelyben a keresés nagyon lassú, amennyiben a keresett pont a klaszter határvonalán helyezkedik el. A másik két megoldás nagyon jó eredményeket hozott, több mint tízszeres gyorsaságot sikerült elérnünk használatukkal. A leggyorsabb a Top-Down algoritmusba ágyazott k közép algoritmus volt. Megfigyeléseim alapján a Top-Down algoritmus futási ideje közel azonos volt, mint a beágyazott klaszterező algoritmusok futási ideje. A Farthest First algoritmus nagyon gyorsan meghatározta a klasztereket. A klaszterek számával egyenesen arányosan, közel linárisan (17. ábra).

Az átlagos keresési idők a 18. ábrán láthatók. Megfigyelhető, hogy a beágyazott DBSCAN algoritmus által létrehozott klaszterekben való keresés ideje jelentősen nagyobb, mint a másik két megoldás esetén. Ennek oka, hogy amennyiben a pont a szuperklaszter határán helyezkedik el, nagyszámú adatpontot kell megvizsgálnunk. A megoldás nem osztotta fel megfelelően a teret, míg a k közép algoritmus és Farthest First algoritmus sokkal több, kisebb méretű klasztert hozott létre, melyekben a keresés a kisebb elemszám és jobban körülrajzolható határvonal miatt gyorsabb, kevesebb adatpont vizsgálatát igényli. Hierarchikus klaszterező algoritmus használatával a keresési idők szórását a 19. ábra mutatja. Megfigyelhető, hogy beágyazott sűrűség alapú klaszterező eljárás esetén a szórás nagyon nagy, ami jelentős sebességingadozást jelenthet a megfigyelés során. Ennek oka, hogy az egész teret lefedő klaszterben a keresés nagyon lassú, míg egy kisebb klaszterben, amely akár század annyi pontot tartalmaz, nagyon gyors a kevés vizsgálat miatt. Beágyazott k közép algoritmus használata esetén sikerült elérnem a legnagyobb gyorsítást és legkisebb szórást. Más típusú - kétszintű



16. ábra. A Farthest First algoritmus klaszterezési ideje

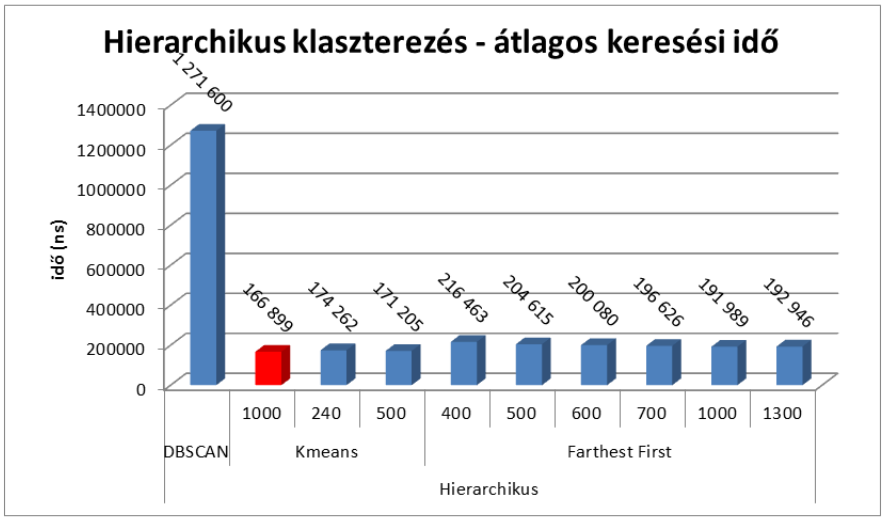


17. ábra. Hierarchikus klaszterezésekkel elért gyorsítás mértéke

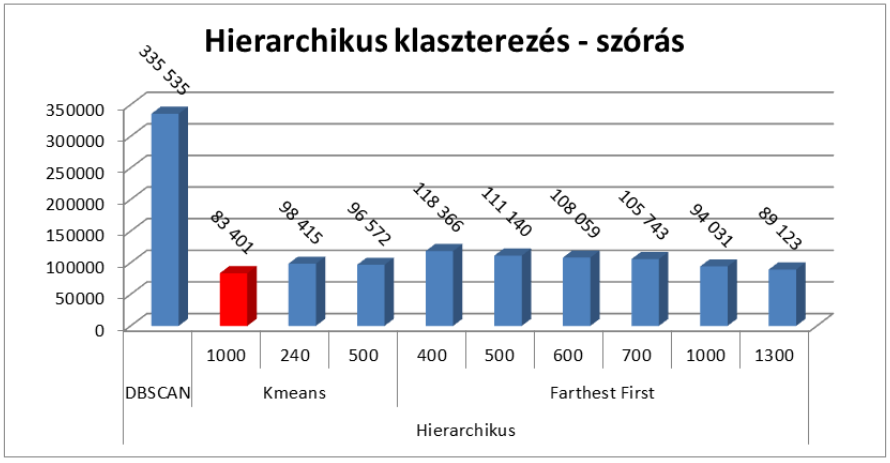
hierarchia, particiós klaszterező algoritmus - eljárások során megfigyeltük, hogy a klaszterek számának növekedésével nem nő minden határon túl a keresési idő, mivel egyre több halmaz közül kell kiválasztani a megfelelőt, így a megfelelő csoport kiválasztása egy bizonyos klaszterszám fölött már sebességcsökkentéshez vezet kisebb klaszterszámokhoz képest, azonban a Top-Down algoritmus nagy memóriaigénye nem tette lehetővé a legoptimálisabb paraméterek megtalálását. Azonban az elmondható, hogy mindenképpen jelentős sebességnövekedéshez vezet ennek a megoldásnak a használata.

A 20. ábrán látható, hogy bizonyos esetekben nagyon nagy különbségek lehetnek a keresési időkben. Ez azért hátrányos a rendszer szempontjából, mert esetenként lehetséges, hogy nagyon sokat kell várni egy új pont meghatározására, mellyel hasznos információt veszítünk. Az eredeti rendszerhez képest, ha minden pont abból a halmazból kerül ki, melyek meghatározása a maximális keresési időhöz közelít, a keresés gyorsítása még mindig jelentős javulást mutat (21), ezért arra a következtetésre jutothatunk, hogy a hierarchikus klaszterező eljárások használata még a legrosszabb esetben is javít a rendszer működésén.

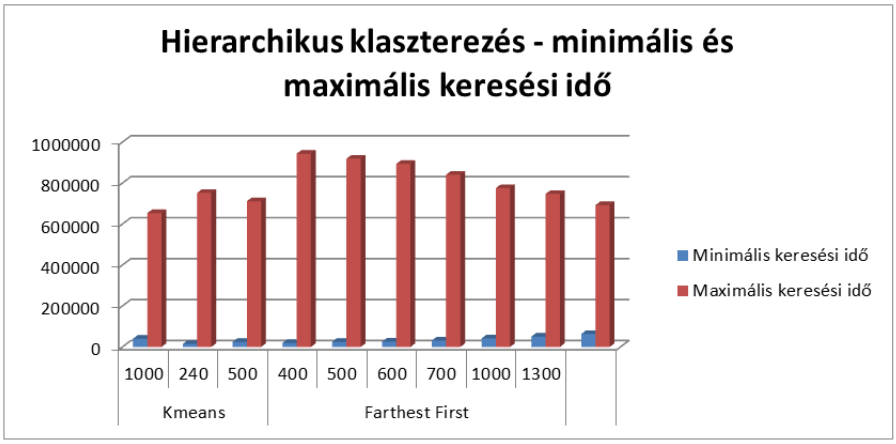
Sűrűségalapú klaszterezések értékelése. A sűrűség alapján történő klaszterezést elvégeztük az RSSI térben és valós koordináták alapján is a DBSCAN algoritmus segítségével. A klaszterek térbeli eloszlása látható a 22. ábrán és a 23. ábrán. Látható, hogy helytálló az állítás, miszerint az RSSI tér leképezhető valós koordinátákra, mivel közel azonos lett a klaszterek elhelyezkedése, mérete. Ennek oka, hogy bizonyos ponttól távolodva a pontban vett jelerősségek is változnak, például: ha A pontban helyezkedik el az A1 AP és B pontban a B1 AP, akkor, ahogy távolodunk A ponttól, úgy csökkent az A1 AP és nő B1 AP jelerőssége és fordítva. Az ábrákon megfigyelhető, hogy sikerült olyan paraméterezést találni, amellyel sűrűség alapon felbontható a tér és megtalálhatók a csomósodások. Ennek oka valószínűleg, hogy a DBSCAN algoritmus érzékeny arra, ha a különböző klaszterek sűrűsége eltérő és ebben az esetben nem biztos, hogy lehet olyan paraméterezést találni, amellyel a klaszterezés eredményes lesz. Ennek érdekében az OPTICS algoritmussal - mely nem készít külön



18. ábra. Hierarchikus klaszterezéssel elért átlagos keresési idő



19. ábra. Hierarchikus klaszterezéssel elért keresési idők szórása

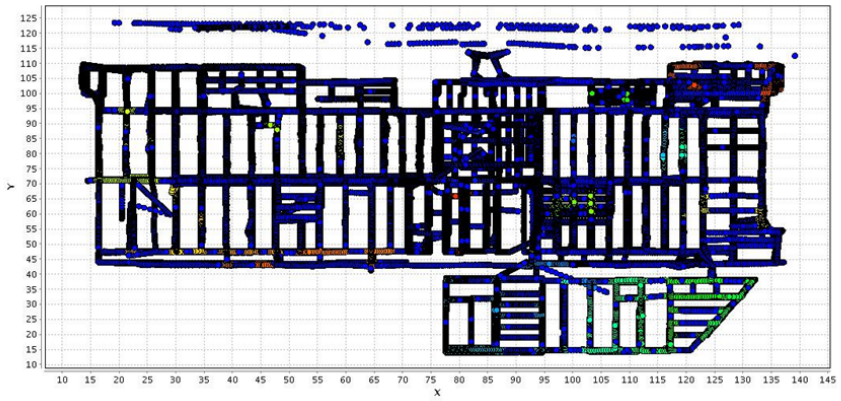


20. ábra. Hierarchikus klaszterezéssel elért maximális és minimális keresési idő

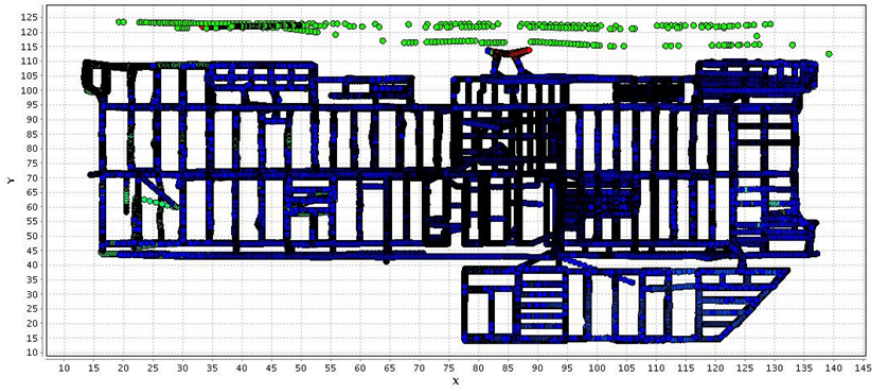
klasztereket - megvizsgáltuk az egyes elemek középponttól való távolságát. Azt tapasztaltuk, hogy nincs olyan vágóegyenes, mellyel az OPTICS által kirajzolt grafikont elvágva megfelelő klasztereket kapnék, mivel bemeneti sorrend szerint az adathalmaz eleje sűrűbb (a core distance kisebb), mint az adathalmaz végén található adatpontok közti távolság (core distance nagyobb). Ezért arra a kö-

21. ábra. Hierarchikus klaszterező eljárással elért legrosszabb sebességnövekedés mértéke

vetkeztetésre jutottunk, hogy se RSSI, se valós koordináták alapján nem bontható fel megfelelően az adathalmaz klaszterekre. Ennek ellenére megkerestem a legbővebb (legtöbb klasztert tartalmazó) klaszterezést és megvizsgáltam, hátha sebességnövekedéshez vezet az alkalmazása.



22. ábra. Sűrűségalapú klaszterezés az RSSI térben



23. ábra. Sűrűségalapú klaszterezés valós koordináták alapján

A sűrűségalapú klaszterező eljárások mérési eredményeit az 1. táblázat tartalmazza. Megfigyelhető, hogy az RSSI térben történt klaszterezést használva a rendszerrel nagyobb gyorsítást sikerült

elérnünk, mint a valós koordináták alapján. Ennek oka valószínűleg, hogy az RSSI térben jobb paraméterezést sikerült találnunk,1így valamivel egyenletesebben osztottuk fel a teret, mint valós koordináták alapján. Az elért eredmények valószínűleg azért hoztak némi javulást, mert a keresést nem az egész adathalmazon kellett elvégezni, hanem legrosszabb esetben is csak a legnagyobb klaszteren, amelynek mérete körülbelül 70%-a az eredeti adathalmazának, tehát nem maga a klaszterezés hozott eredményt, hanem az adatszerkezet amiben tároltam az adatot. Ha a legrosszabb esetet, tehát a maximális keresési időt nézzük, akkor is sikerült ezzel a módszerrel gyorsítani a rendszert. Az így elért keresési idők szórása kiemelkedően magas, így ezzel a módszerrel nagyon szaggatott, ugráló képet kaphatunk a megfigyelt objektumról.

	XYZ	RSSI
Átlagos idő	1 863 734,086	1 409 487,551
Gyorsítás	32,67%	49,08%
Szórás	536 446,51	369 713,096
Medián	2 024 283	1 490 137
Minimum keresési idő	231 106	276 150
Maximális keresési idő	2 596 935	2 125 919
Klaszterezési idő (perc)	186	2 172
„Worth case” gyorsítás	6,18%	23,19%

1. táblázat. Sűrűség alapú klaszterezés eredményei

4.3.3. KD-fa

Az adathalmaz pontjait indexeltük az RSSI vektor szerint k-d fa segítségével. Az egyes vágási síkokat az RSSI vektorok koordinátái adják. A választott indexelő eljárás, a k-d fa által elért eredményeket a 2. táblázat tartalmazza. Mivel a k-d fától nagyobb gyorsítást vártunk, ezért megvizsgáltuk a fa felépítését. Tízezer elemű adathalmaz esetén a fa magassága tizenhárom, hetvenezer elemű adathalmaz esetén a fa magassága, a gyökérpont megválasztásától függően huszonegy, illetve huszonhárom között mozgott. Ebből látható, hogy a felépített bináris fa nem kiegyensúlyozott. Egy n szintű, kiegyensúlyozott bináris fa 2^n pontot tartalmazhat maximum. Jól látható, hogy az általam épített fa sokkal több pontot is tartalmazhatna, ezért a fa nem kiegyensúlyozott, így egy elem vizsgálata során nem zár ki elég sok elemet a kereső algoritmus.

Méréseink alapján az elemek átlagosan 30%-át vizsgálja meg a rendszer egyetlen legközelebbi szomszéd megtalálása esetén, ami azt jelenti, hogy a jelenlegi rendszernél lassabb a k-d fával indexelt adathalmazban való keresés, mivel a táblázatban látható eredmény nem az összes legközelebbi szomszéd megtalálását reprezentálja. Egyes források szerint a k-d fa akkor optimális, ha $N \gg 2^k$, amennyiben N az adathalmaz található pontok száma és k az indexelendő adathalmaz dimenziója, tehát a rendszerünkben az RSSI vektor dimenziója. Ennek értelmében az adathalmazunknak az optimális működéshez sokkal több adatpontot kellene tartalmaznia. Problémát jelenthet még a fa kiegyensúlyozatlansága, de a k-d fák nem egyensúlyozhatók ki a bináris fáknál ismert faforgatásokkal, ezért a fát építés után már nem volt lehetőségünk kiegyensúlyozottá tenni. A fa építéséhez olyan heurisztikát kerestünk, mellyel a fa kiegyensúlyozottabb lesz. Ennek érdekében megkerestem az RSSI vektorok szerint sorba állított adathalmaz középső elemet. Ezzel a megoldással sikerült a fa magasságát pár szinttel csökkenteni, de a táblázatban ezzel a megoldással elért eredmények láthatóak. Ennek oka valószínűleg, hogy az adathalmaz túl kevés elemet tartalmaz ahhoz, hogy optimális k-d fát lehessen rá építeni.

	Lineáris (k=13)	k-d fa (k=1)
Átlagos idő	2 767 873,99	1 685 824,06
Gyorsítás	0%	39,09%
Szórás	16 248	311 601,05
Medián	2 766 846	1 641 518
Minimum keresési idő	2 734 424	1 168 126
Maximális keresési idő	2 824 493	2 474 259
„Worth case” gyorsítás	-2,05%	10,61%

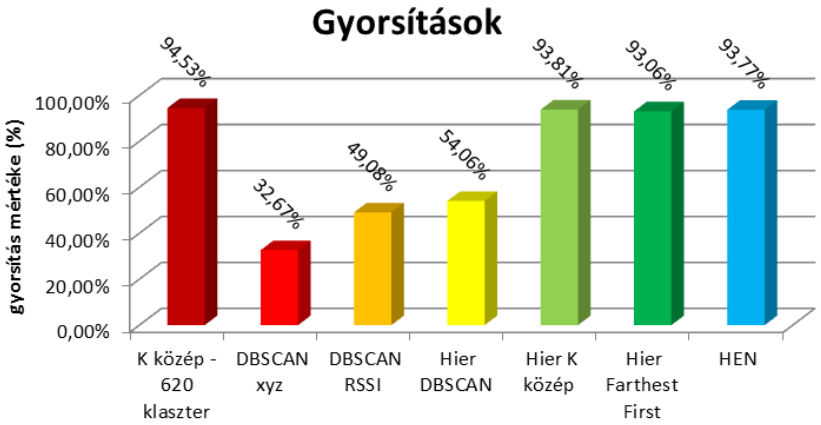
2. táblázat. K-d fa indexelő eljárás által elért eredmények

5. Összefoglalás

Ebben a fejezetben azokat a módszereket hasonlítjuk össze, amelyekkel sikerült gyorsítani a rendszert. Amint azt az előző fejezetekben már részletesen bemutattuk, azon módszerek, melyeknél a hierarchikus adatszerkezetet használtuk, mind gyorsítottak a keresésen. Természetesen volt köztük olyan, amely csupán csak azért ért el némi gyorsítást, mert több kisebb részre osztotta az adathalmazt és nem azért, mert megfelelő csoportokat hozta létre. Az mindenképpen igaz, hogy hierarchikus adatszerkezetek alkalmazásával, a keresési tér felbontásával a rendszer gyorsítható, az adathalmazban való keresési idő lerövidíthető. A diagramokon az egyes módszereket színnel és a nevük rövidítésével jelöltem, melyek a következők:

- **K közép - 620 klaszter:** 620 klaszter, melyet k közép algoritmussal készítettem a diagramokon bordó színnel jelölve.
- **DBSCAN xyz:** Az xyz, tehát valós térben elvégzett sűrűség alapú klaszterezéssel készített klaszterekkel való vizsgálat eredménye a diagramokon piros színnel jelölve.
- **DBSCAN RSSI:** Az RSSI térben elvégzett sűrűség alapú klaszterezéssel készített klaszterekkel való vizsgálat eredménye a diagramokon narancssárga színnel jelölve.
- **Hier DBSCAN:** Hierarchikus klaszterezőbe ágyazott sűrűség alapú klaszterezéssel készített klaszterekkel való vizsgálat citromsárga színnel jelölve.
- **Hier K közép:** Hierarchikus klaszterezőbe ágyazott k közép algoritmussal készített klaszterekkel való vizsgálat világoszöld színnel jelölve.
- **Hier Farthest First:** Farthest First algoritmussal készített klaszterekkel való vizsgálat sötét zöld színnel jelölve.
- **HEN:** Kétszintű hierarchia alkalmazása kék színnel jelölve.

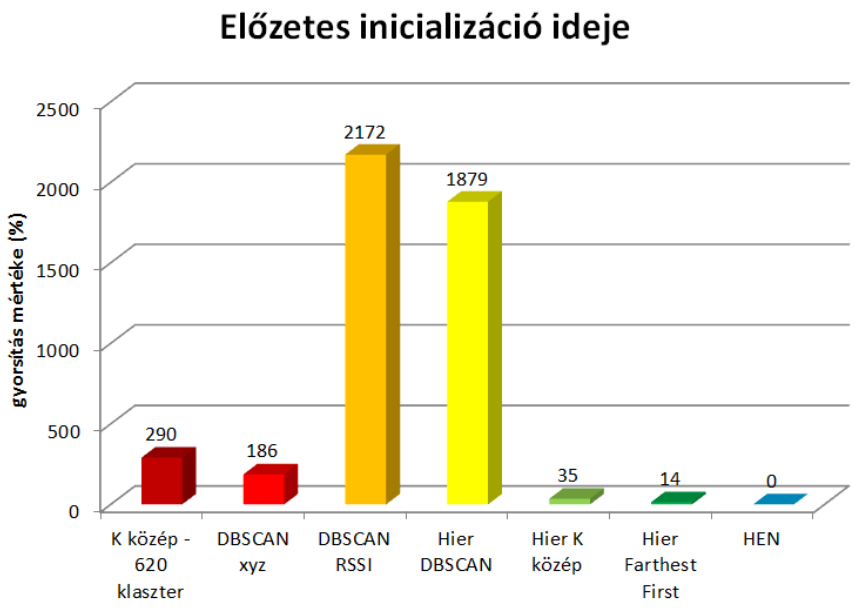
A 24. ábrán megfigyelhető, hogy a legnagyobb gyorsítást az egyik legegyszerűbb klaszterező algoritmus segítségével, a k közép algoritmussal értük el, amikor a klaszterek száma 620 volt. A Hierarchikus keresések közül a Farthest First és a Top-Down algoritmusba ágyazott k közép algoritmus, illetve egyszerűen csak a kétszintű hierarchia használatával is hasonló eredményeket sikerült elérnünk, így kijelenthetjük, hogy az adathalmaz kisebb részekre történő felbontásával a rendszer jelentős mértékben gyorsítható, így több adatot lehet gyűjteni az objektumok mozgásáról. Az egyes klaszterező módszerek valószínűleg csak azért gyorsították ilyen mértékben a rendszer keresési sebességét, mert maga az adatszerkezet, amiben az eredeti megoldással ellentétben a rendszer felosztva tárolta az adatokat, megfelelő kis részekre volt bontva. Ez a kijelentés legfőképpen a sűrűség alapú megoldásokra igaz, melyeken látható, hogy alkalmazásuk meg se közelítette azt a gyorsaságot, amit a kétszintű hierarchia alkalmazásával sikerült elérni. Ennek oka, hogy egyik sűrűség alapú klaszterező eljárás se bontotta fel a teret se egyenletesen kis részekre, se a kívánt csomósodásokat kiszűrve. A 28. ábra mutatja, hogy még a legrosszabb esetben, ha minden egyes keresés a lehető leglassabban következik be, is jelentős, akár több mint 76%-os sebességnövekedés is lehetséges. Ez azt jelenti, hogy a 26 ábra látható keresési időket és a rendszer keresési idejét figyelembe véve akár 18-szor annyi pontot lehet meghatározni, mint eredetileg.



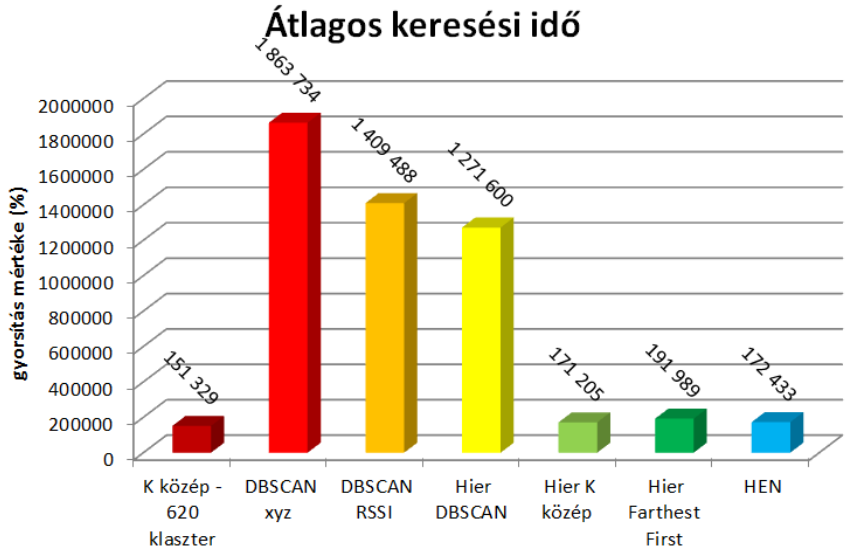
24. ábra. Legjobb gyorsítások

A módszertől függően a csoportok létrehozásának, illetve a középpontok meghatározásának az ideje látható a 25. ábrán. Látható, hogy a sűrűség alapú klaszterező eljárások előzetes inicializációs

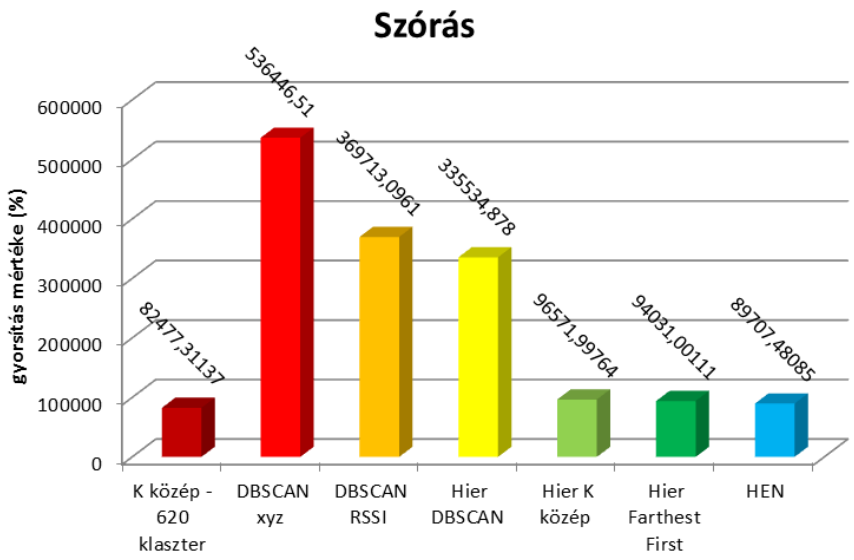
ideje is jelentősen több, mint a többi módszer esetén. A kettős hierarchia a diagramon ezért 0 idővel szerepel, mert ez a megoldás nem igényel előzetes csoportosítást, az csupán a pontok „érkezés" sorrendjétől függően választja ki a csoportvezetőket, tehát a csoportok középpontjait. Az előzetes inicializáció után a program adatstruktúra-létrehozási ideje méréseink alapján lineárisan változott a csoportok számának függvényében. Megfigyelhető, hogy szintén a 620 klaszterrel rendelkező k közép algoritmus által létrehozott csoportosítás érte el a legkisebb szórást, ami azt jelenti, hogy az ezzel a megoldással készített csoportokban történő keresési idők a legstabilabbak. Meg kell jegyezni, hogy ez rosszabb, mint az eredeti megoldásé, mivel ott a keresési idők közel azonosak voltak, de az így mért ingadozás mértéke még mindig kisebb, mint az eredeti megoldás által mért keresési idő, ami azt jelenti, hogy ha bizonyos esetekben, lassabban is találja meg a keresett pontot, mégis gyorsabban, mint a k legközelebbi szomszéd esetén, melyet eredetileg használtunk.



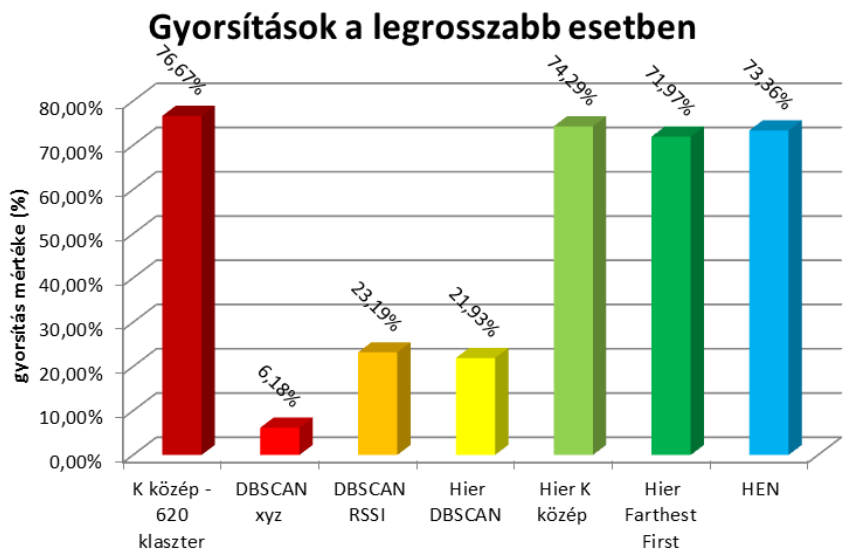
25. ábra. Csoportok meghatározásának ideje



26. ábra. Átlagos keresési idők



27. ábra. Módszerek szórása



28. ábra. Gyorsítások a legrosszabb esetben

Irodalomjegyzék

[1] Mihael Ankerst – Markus M. Breunig – Hans peter Kriegel – Jörg S: Optics: Ordering points to identify the clustering structure. In *ACM SIGMOD: International Conference on Management of Data* (konferenciaanyag). 1999, ACM Press, 49–60. p.

[2] Jon Louis Bentley: Multidimensional binary search trees used for associative searching. 18. évf. (1975. September), *Commun. ACM*, 509–517. p. ISSN 0001-0782.

[3] Sanjoy Dasgupta – Philip M. Long: Performance guarantees for hierarchical clustering. 70. évf. (2005. June), *J. Comput. Syst. Sci.*, 555–569. p. ISSN 0022-0000.

[4] Martin Ester – Hans peter Kriegel – Jörg S – Xiaowei Xu: A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)* (konferenciaanyag). 1996, AAAI Press, 226–231. p.

[5] Seymour Lipschutz: *Adatszerkezetek*. Budapest, Magyarország, 1993, Panem-McGraw-Hill.

[6] Hugo Steinhaus: Sur la division des corp materiels en parties. 1. évf. (1956), *Bull. Acad. Polon. Sci*, 801–804. p.