Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Rénes Dániel

MAP/MAPEM ÜZENETSTRUKTÚRA AUTOMATIKUS GENERÁLÁSA JÁRMŰ-KOMMUNIKÁCIÓS ADATOK ELEMZÉSÉVEL

KONZULENS
Dr. Bokor László

BUDAPEST, 2019

Tartalomjegyzék

Köszönetnyilvánítás

Köszönöm az EFOP-3.6.2-16-2017-00013 és a 2018-1.3.1-VKE-2018-00021 Intelligens közlekedésirányítási rendszer fejlesztése projektek támogatását. Köszönöm konzulensemnek, Dr. Bokor Lászlónak, aki észrevételeivel és tanácsaival segítette a munkámat.

Összefoglaló

Manapság már látszik, hogy a jármű-kommunikációs rendszereket a közeljövőben be fogják vezetni, és kötelező tartozékká válnak minden járműben. Az egyetlen kérdés az, hogy ez mikor fog megtörténni. Néhány autógyártó azt mondja, hogy 2022-ben már jármű-kommunikációs képességekkel rendelkező új modelleket fognak forgalomba hozni [1]. Világszerte több kísérleti projekt működik, amelyek jármű-kommunikációs rendszereket használnak az autókban és a környező infrastruktúra egyes elemeiben [2].

A témával kapcsolatos tanulmányaim során nyilvánvalóvá vált számomra, hogy a járművek és az infrastruktúra közötti kommunikációban van jó néhány olyan lehetőség, amelyet a működés hatékonyabbá tételéhez ki lehetne aknázni. Érdeklődésem a járművek és az infrastruktúra közötti kommunikáción alapuló városi alkalmazások felé fordult, melyek többsége megköveteli a kereszteződések topológiai leírását az úgynevezett MAP (Európában MAPEM, MAP Extended Message) üzenetekben [3] [4]. A MAP üzeneteket nem kell megváltoztatni mindaddig, amíg a kereszteződést nem változtatják meg. A MAP üzeneteket el kell készíteni, mielőtt az út menti infrastruktúra elemek (Roadside Unit - RSU) teljes potenciálját ki lehet használni. A MAP üzenetek létrehozása történhet manuálisan vagy egy kereszteződéshez meglévő adatok felhasználásával és azok transzformációjával. Jelenleg nincs olyan felhasználási eset (use case) a járművek és az infrastruktúra közötti kommunikációra, amely a járművektől kapott dinamikus információkat (pl. Cooperative Awareness Message, CAM) használná fel a MAP üzenetek létrehozására az RSU egységeken. Úgy gondolom, hogy ez egy érdekes use case, amelyet érdemes megvizsgálni, hogy lássuk vajon alkalmas alternatíva lehet-e a MAP üzenetek hatékony létrehozására.

Munkám két fő dologra összpontosít. Az első egy olyan keretrendszer létrehozása, amely felhasználható különböző olyan algoritmusok könnyű implementációjára, cseréjére és kiértékelésére, amelyek potenciálisan képesek lehetnek MAP üzenetek létrehozására a járművek helyzetinformációjából. A keretrendszernek vizuális visszajelzést és numerikus elemzést kell nyújtania az algoritmusok teljesítményéről. A vizuális visszacsatolásra szükség van ahhoz, hogy gyorsan el tudjuk dönteni, hogy egy

algoritmust érdemes-e tovább kiértékelni. A numerikus elemzés olyan adatokat szolgáltat, amelyek felhasználhatók ugyanazon algoritmus különböző forgatókönyvekben vagy különböző algoritmusok ugyanabban a forgatókönyvben való összehasonlítására. A második rész a meglévő algoritmusok felkutatása vagy újak kitalálása, és ezek kiértékelése a keretrendszer felhasználásával. Ebben a részben végzem el az egyes algoritmusok összehasonlítását. A vizsgálatok főleg a különböző klaszterező algoritmusokra térnek ki, mint például a DBSCAN és a hierarchikus klaszterezés [5].

Abstract

Nowadays it is clear that vehicular communication systems will be widely deployed in the near future and they will be widely applied in all vehicular scenarios. The only question is when will it happen. Some car manufacturers say they will launch new models with vehicular communication capability (V2X) as early as 2022 [1]. There are multiple pilot projects around the world that use vehicular communication systems deployed in cars and in parts of the surrounding roadside infrastructure [2].

It has occurred to me that the vehicle-to-infrastructure communication may have some potential that has not been tried by anyone. Most of the urban applications based on the vehicle-to-infrastructure communication require the topology details of the intersections to be described in MAP (MAPEM, Map Extended Message in Europe) messages [3] [4]. The MAP messages do not have to be changed as long as the intersection is not changed. The MAP messages have to be created before the roadside units can be used to their full potential. The creation of the MAP messages can be done manually or using existing data of an intersection and transforming it into a MAP message. There is no existing use case for vehicle-to-infrastructure communication that uses the dynamic data received from the vehicles (e.g. Cooperative Awareness Message, CAM) to dynamically create the data structures to be used for generating the MAP messages on the roadside units. I think that is an interesting use case that requires some investigation to see whether it is a viable alternative when it comes to the creation/update of the MAP messages.

My work focuses on two main areas. The first is creating a framework that can be used to easily implement, swap and evaluate different algorithms that are potentially capable of creating MAP messages from the position data of the vehicles. The framework has to provide visual feedback and numerical analysis tools about the performance of the algorithms. The visual feedback is needed to be able to quickly decide whether an algorithm is worth further evaluation or not. The numerical analysis provides data that can be used to compare the same algorithm in different scenarios or compare different algorithms in the same scenario. The second domain of my interests in this work is

finding existing algorithms or implementing new ones and evaluating them using the above framework. I compare different algorithms in that part of my work. I focus mainly on evaluating different clustering algorithms, such as DBSCAN and hierarchical clustering [5].

1 Bevezetés

Az autonóm járművek fejlődéséhez nélkülözhetetlen a környezet egyre pontosabb digitális reprezentációja, melyhez szükséges lesz az úthálózat sávszintű szkennelésére és így sávszintű pontosságú digitális térképadatok előállítására, mely adatokat az autonóm járművek fel tudnak használni döntéseikhez. Az autonóm járművek természetesen más forrásokból is fel tudják térképezni valós időben az utat és tágabb környezetüket (pl. kamerarendszerek, Radar, Lidar segítségével), de ezeknek a megoldásoknak a megbízhatósága nagyban függ az úttest felfestésének láthatóságától, ami sok esetben kopott, elhalványult. Továbbá rossz látási viszonyok között (pl. ködben, havas úton) az úttest felfestései akár teljesen megfigyelhetetlenek is lehetnek. A mai, részben autonóm járművek által használt, gépi látásra épülő technológiák mellett tehát a sávszintű pontosságú digitális térképadatok megléte is elengedhetetlen.

Egy másik technológia, ami a jövő közlekedésében meghatározó lesz, a kooperatív jármű-kommunikáció vagy más néven V2X (Vehicle-to-Everything) kommunikáció. A V2X kommunikáció egyik típusa a V2V (Vehicle-to-Vehicle), a másik pedig a V2I (Vehicle-to-Infrastructure) és I2V (Infrastructure-to-Vehicle) kommunikáció. A V2V kommunikáció közvetlenül a járművek közötti, valós idejű adatátvitelt jelenti, amely során például a járművek a CAM (Cooperative Awareness Message) üzenetekben sugározzák a jelenlegi pozíciójukat, sebességüket, haladási irányukat, továbbá értesítik egymást olyan eseményekről, amiket egyénileg csak később tudnának megfigyelni, mint például egy vészfékezés a kocsisor elején.

A V2I és az I2V kommunikációban szereplő entitások a járművek és az infrastruktúra (autópálya, kereszteződés, stb.) jármű-kommunikációra felkészített elemei. A járművekben elhelyezett V2X kommunikációs eszközöket OBU-nak (On-board Unit), az infrastruktúrában lévőket RSU-nak (Roadside Unit) nevezik. A V2I és I2V kommunikációnak egy tipikus szituációja a városi kereszteződésekben elhelyezett RSU-k és a kereszteződést megközelítő OBU-k kommunikációja. Ezek az RSU-k a kereszteződés sávszintű geometriai leírását és a kereszteződésben lévő közlekedési jelzőlámpák státusz és időzítési információt közvetítik az OBU-knak. Ezen

információkkal a városi közlekedés hatékonyabbá tétele a cél, de ezen felül lehetőséget biztosíthat arra is, hogy egy megkülönböztetett járműben elhelyezett OBU elsőbbséget kérjen az RSU-tól, ami ezeket megpróbálja kielégíteni és a jelzőlámpák státuszát úgy megváltoztatni, hogy a megkülönböztetett jármű megállás nélkül át tudjon haladni a kereszteződésen.

Az autonóm járművek és a jármű-kommunikáció külön-külön is hasznosak, de együtt tudnak igazán jól működő és megbízható rendszert alkotni, amellyel a vezetést a jövőben teljesen rá lehet majd bízni a számítógépekre.

A teljesen autonóm közlekedéshez szükségesek lesznek sávszintű, és még pontosabb, digitális, úgynevezett HAD (High Definition Map) térképadatbázisok [6], melyek térbeli és időbeli felbontása és pontossága az önvezetés igényeinek van specifikálva, és amelyek létrehozása a jelenleg használatos módszerekkel igen költséges és időigényes. V2X kommunikációra képes járművek már jóval a teljesen autonóm közlekedés megvalósulása előtt járni fogják az utakat, és amelyik kereszteződésben telepítve van RSU, ott az OBU-val felszerelt járművek által sugárzott CAM üzeneteket fel lehetne használni a kereszteződések és a környező útszakaszok feltérképezésére akár sávszintű pontossággal.

Ennek lennének előnyei és hátrányai a manapság használatos módszerekkel szemben. Az egyik nagy előnye az lenne, hogy ez a folyamat teljesen automatikusan történne. Nem lenne szükséges emberi beavatkozásra és nem kéne drága úthálózat-szkennelésre alkalmas gépeket bérelni. Egy másik előny, hogy a környezet változásait dinamikusan lehetne kezelni, és a HAD térképeket frissíteni/módosítani, ami nyilván valamilyen késleltetéssel jár a környezet megváltozása és a környezet digitális reprezentációjának a frissítése között. Az eljárás hátránya, hogy a térkép elkészítésének idejére és az elkészített térkép pontosságára nehéz garanciát adni, mert mindez nagyban függ a forgalom méretétől és a forgalomban részt vevő járművek pozícióadatainak pontosságától.

A dolgozatom keretein belül egy olyan keretrendszert hoztam létre, amellyel a kereszteződésben telepített RSU és a járművekben lévő OBU-k közötti kommunikáció szimulálható, és az RSU-n futó különböző úttest feltérképezésre használható algoritmusok könnyen kipróbálhatók és kiértékelhetők. A dolgozatom célja annak megvizsgálása, hogy egy városi kereszteződést különböző forgalmi viszonyok között milyen megbízhatósággal lehet feltérképezni a járművek CAM üzeneteiből származó adatok alapján.

2 Téma elhelyezése

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

1. ábra: Jármű-automatizálás szintjei (SAE J3016)

Az autonóm járművekre a SAE (Society of Automotive Engineers) a J3016-ban egy hat szintű modellt állított fel a vezetés automatizálásának jellemzésére [7]. Jelenleg 2-es szintű automatizálásnál tartunk, bár a 2019-es Audi A8, amit már 2018-ban többször demóztak, a Traffic Jam Pilot rendszerrel már 3-as szintű automatizálásra képes, de ezt a rendszert sok országban a megfelelő, főleg a felelősségre vonatkozó, törvények hiányában nem fogják még engedélyezni [8]. Az 5-ös szintű automatizálás, tehát a teljesen autonóm vezetés, pedig valószínűleg legkorábban egy évtized múlva valósulhat meg.

A jármű-kommunikáció elterjedése ennél jóval korábban meg fog történni, mert ez a rendszer addig is jelentősen javíthatja a közlekedés biztonságát és hatékonyságát, amíg még nem érjük el a teljes autonóm közlekedést.

A dolgozatomban a V2X kommunikáció során gyűjthető adatokat mesterséges intelligencia algoritmusok segítségével dolgozom fel. A mesterséges intelligenciának és a V2X-kommunikációnak sok közös alkalmazási területe van, amiket ez a fejezet fog bemutatni.

2.1 Sebesség javaslat

A GLOSA (Green Light Optimized Speed Advisory) rendszer eredetileg a jármű-kommunikáció terén lett megvalósítva, ami a jelzőlámpa státusz és időzítési információi alapján egy optimális sebességet számol ki, amivel a piros lámpánál való megállás elkerülhető. Ezt az algoritmust a mesterséges intelligencia területén továbbfejlesztették, hogy többféle metrika (pl. átlagos üzemanyag-fogyasztás, átlagos várakozással töltött idő) alapján lehessen az optimális sebességet kalkulálni. A több szegmensű GLOSA szintén a mesterséges intelligencia területén lett kitalálva, ami a jármű útját több forgalmi jelzőlámpán keresztül írja le, és az optimális sebességet az útját alkotó összes jelzőlámpa információi alapján kalkulálja [9].

2.2 Dinamikus jelzőlámpa vezérlés

Tradicionálisan a dinamikus jelzőlámpa vezérlést fuzzy logikával és lineáris programozással oldottak meg, azonban neurális hálózatokkal valós idejű GPS adatokat felhasználva már sikerült jó eredményeket elérni a dinamikus jelzőlámpa vezérlés terén szimulált környezetben. Egy valós telepítésnél a valós idejű pozícióadatok szolgáltatására a jármű-kommunikáció kiválóan alkalmazható [9].

2.3 Hálózati torlódás vezérlés

A jármű-kommunikációs protokollok önmagukban is definiálnak olyan helyzeteket, amikor az üzenetek küldési frekvenciáját le kell csökkenteni. Például a CAM üzenetek esetében az alapértelmezett működés az, hogy a küldési frekvencia a sebesség függvényében változik 1 és 10Hz között. Maga a 802.11p protokoll is tartalmaz torlódás kezelési eljárásokat (pl. CSMA/CA, Carrier-sense multiple access with collision avoidance).

Klaszterezési algoritmusok használatával, szimulált környezetben, jobb eredményt sikerült elérni a CSMA/CA-nál. A módszer a kereszteződésekben méri a vezeték nélküli hálózat terheltségét. Az egyes V2X üzeneteket különböző jellemzők alapján (pl. üzent hossza, típusa és időtartama) csoportosítják, és az egyes csoportokra eltérő hálózati paramétereket definiál [9].

2.4 Autonóm járművek

Az autonóm járművek irányítása egy bonyolult feladat, ami a környezetében lévő többi jármű számának növekedésével egyre nehezebbé válik. Sok jármű viselkedését kell egyszerre megfigyelni, és ez alapján kell döntést hozni a megfelelő cselekvésről. A megfelelő döntés kiválasztására jó eredményeket értek például döntési fák alkalmazásával. A környező járművek megfigyelésében pedig a V2V kommunikáció segíthet, hogy ezeknek a járműveknek a pozícióját, sebességét és haladási irányát ne az autonóm járműnek kelljen kalkulálni a saját szenzorai alapján, hanem ezeket kész információként kapja meg [9].

2.5 Kooperatív járműkonvojok (Cooperative Platooning)

A platooning az egyik legfontosabb jármű-kommunikációs alkalmazás. A járművek egymást szorosan követve közlekednek, amit a gyors üzenetváltások segítségével biztonságosan meg tudnak tenni. Ezzel a szoros követéssel a követő járművek esetében a közegellenállást jelentősen csökkenteni lehet, aminek köszönthetően az üzemanyag-fogyasztásuk csökken. Ha a járművek szorosan tudják követni egymást, az a torlódások kialakulását meg tudja előzni, és az utazással töltött időt is csökkenteni tudja.

A platooning egyik problémaköre a klaszterek kialakítása a járművekből, a klaszter vezető kiválasztása, és a klaszter karbantartása. A mesterséges intelligenciát néhány munkában használták a klaszter vezető kiválasztására, amikben különböző paraméterek figyelembe vétele alapján (pl. a vezető várható élettartama) tanuló algoritmusok lépésről-lépésre találják meg a legjobb klaszter vezető kiválasztási stratégiát [9].

3 Cooperative Intelligent Transport Systems

A Cooperative Intelligent Transport Systems (C-ITS) fejlesztések elsődleges célja az utakat biztonságosabbá, a közlekedést pedig hatékonyabbá tenni. A rendszer a járművek közötti és a járművek és az infrastruktúra közötti gyakori adatcserére támaszkodik. A biztonsági alkalmazások számára 5875 és 5905 MHz, az egyéb alkalmazásoknak pedig 5855 és 5875 MHz közötti frekvenciasávot allokálták [10].

A C-ITS protokoll stack négy rétege az access, networking & transport, facilities és applications. Az access réteg támaszkodhat a 802.11p alapú ITS-G5-re vagy celluláris hálózatokra (4G, 5G). A networking & transport rétegben GeoNetworking és Basic Transport Protocol (BTP) található, valamint IPv6 és UDP vagy TCP. A networking & transport réteg fölött helyezkedik el a facilities réteg, amely az alkalmazások számára szükséges információkat szolgáltatják. Az európai szabványok legfontosabb facility rétegbeli protokollja a CAM, a DENM, a MAPEM és a SPATEM. A CAM protokoll jármű állapotáról szolgáltat információkat, amikkel a járművek nyomon tudják követni a többi jármű pozícióját és mozgását, amik a biztonsági és forgalom-hatékonysági alkalmazások számára nélkülözhetetlenek. A DENM protokoll eseményvezérelt biztonsági információkat terjeszt egy földrajzi régióban. A MAPEM protokoll az úttest geometriai leírására. A SPATEM jelzőlámpa időzítési és státusz információk leírására szolgál jelzőlámpás kereszteződésekben.

3.1 Áttekintés

A C-ITS célkitűzéseit öt fázisra bontották, amelyekről az alábbiakban rövid áttekintés található.

3.1.1 Phase 1

Ez a kezdeti fázis, amelyet a C-ITS fejlesztésekor kitűztek. Ebben a fázisban a járművek terjesztik a státuszinformációikat, így a többi jármű tudatában van a jelenlétének, és a detektált veszélyekről az úton [8].

3.1.2 Phase 2

Ezt a fázist érzékelő vezetésnek nevezik. Az érzékelő vezetés fázisában az autóban lévő szenzorok további információkat szolgáltatnak. Például kamerák és radarok adatait felhasználva olyan objektumokat is detektálni lehet, amik amúgy rejtve maradtak volna a jármű elől [8].

3.1.3 Phase 3

Ez a kooperatív vezetés fázisa, amiben a járművek megosztják a szándékaikat a közlekedés többi résztvevőivel. A szándék előrejelzésével a közlekedés résztvevői tulajdonképpen jövőbeli információk birtokában tudnak döntéseket hozni [8].

3.1.4 Phase 4

Ez a szinkronizált vezetés fázisa, ami akkor következhet be, amikor a járművek majdnem minden szituációt autonóm módon tudnak kezelni, és képesek haladási pályájukat egymással megosztani és egymáshoz igazítani [8].

3.1.5 Phase 5

Ezt a fázist balesetmentes vezetésnek nevezik. Ebben a fázisban teljesen autonóm a vezetés, és optimális forgalomáramlás érhető el [8].

3.2 V2X üzenettípusok

3.2.1 Cooperative Awareness Message (CAM)

A CAM egy periodikus üzenet, ami státuszinformációt szolgáltat a közelben lévő ITS állomásoknak. Az üzenet egy ITS PDU (Protocol Data Unit) fejlécből és néhány konténerből áll. Az ITS PDU fejléc a protokoll-verziót, az üzenettípust és a küldő címét hordozza. A Basic Container az állomás típusát és pozícióját tartalmazza. A High-Frequency Container a gyakran változó adatokat, úgy mint sebesség, gyorsulás, haladási irány, tartalmazza, és kötelezően el kell küldeni minden CAM üzenetben. A Low-Frequency Container azokat az adatokat tartalmazza, amik a biztonság szempontjából nem relevánsak vagy nagy méretűek, ezért nem küldik el minden CAM üzenetben. A

Special Vehicle Container opcionálisan hozzáadható az üzenethez, ha a jármű szerepéről tudnia kell a többi ITS állomásnak.

A CAM üzenetek küldési frekvenciája 1 és 10 Hz között változik attól függően, hogy az autó mekkora sebességgel halad, az adott alkalmazásnak milyen követelményei vannak, és mekkora a torlódás vezeték nélküli kommunikációs csatornán [11].

3.2.2 Decentralized Environmental Notification Message (DENM)

A DENM egy eseményvezérelt üzenet, ami biztonsági információkat tartalmaz. Amikor a jármű egy nem biztonságos szituációt észlel a DENM protokoll egy Action ID-t rendel hozzá, ami az ITS állomásra egyedi. A DENM üzenethez tartozik egy Relevance Area, amiben az összes ITS állomáshoz eljut az üzenet a GeoNetworking geo-broadcast módjával. A DENM üzenet a CAM-hez hasonlóan szintén kap egy ITS PDU fejlécet, és ez is konténerekből áll. A Management Container tartalmazza az Action ID-t, a detektálás idejét és pozícióját. Ez a konténer kötelező, a többi opcionális, ha az alkalmazás számára szükséges. A Situation Container mezői előre definiált kódokat tartalmaznak, amikkel a szituációt kiváltó esemény és az ezzel összefüggő események jellemezhetőek. A Location Container a leírt esemény haladási sebességét és irányát adja meg. Az Alacarte Container alkalmazás specifikus információkat szállít [12].

3.2.2.1 DENM Trigger

A DENM trigger a DENM üzenet generálásának és átvitelének a folyamatát jelenti, amikor a Decentralized Environmental Notification (DEN) Basic Service kap egy AppDENM_trigger típusú kérést az alkalmazástól.

A DEN Basic Service készít egy nem használt actionID-t a DENM triggernek [13].

3.2.2.2 DENM Update

Az ITS állomás érzékelheti az esemény fejleményeit a DENM trigger után. Ilyenkor az alkalmazás egy AppDENM_update kéréssel a DEN Basic Service rendelkezésére bocsájtja a frissített információkat.

A referenceTime paramétert minden DENM update esetén frissíteni kell, és az értékének nagyobbnak kell lennie minden megelőző DENM update referenceTime paraméterénél, aminek ugyanaz az actionID-ja.

Az actionID nem változik meg DENM update hatására, amíg a stationID változatlan [13].

3.2.2.3 DENM Repetition

A DEN Basic Service megismételheti a DENM üzenetet két egymást követő DENM update között, ha az alkalmazás kérésében szerepel a repetitionInterval és a repetitionDuration. Ha valamelyik mezőt nem adja meg az alkalmazás, akkor a DEN Basic Service nem ismétli meg a DENM üzenetet. Egy adott actionID esetén a DENM repetition a legfrissebb DENM-re vonatkozik [13].

3.2.2.4 DENM Termination

A DENM termination az érzékelt esemény végét jelzi. A DENM termination lehet Cancellation vagy Negation. A Cancellation DENM csak attól az ITS-től származhat, amelyik a DENM triggert küldte. A Negation DENM más ITS állomástól is származhat [13].

3.2.3 Signal Phase and Timing (SPAT/SPATEM)

A SPAT protokoll a jelzőlámpák státuszának és időzítésének leírására szolgál. A rendszer alapja az, hogy a kereszteződésben van egy Roadside Unit (RSU), ami ismeri a jelzőlámpa programját, és erre építve forgalomhatékonyság-növelő alkalmazások alapjául szolgálhat.

A kereszteződésben elhelyezett RSU, a jelzőlámpa jelenlegi fázisát és a következő fázisváltás időpontját SPAT üzenetekben kiküldve, a közeledő jármű OBU-ja ezt észleli, majd erre támaszkodó jármű oldali C-ITS alkalmazás javaslatot tud tenni, hogy mi az optimális sebesség a kereszteződés megközelítésére. Egy ilyen rendszer a városi közlekedést hatékonyabbá, tisztábbá és biztonságosabbá teheti.

A SPAT üzenet sávonként tartalmazza a jelzőlámpák jelenlegi fázisát és a fázis érvényességéből hátralévő időt.

3.2.4 Map Data (MAP/MAPEM)

A MAP protokoll egy üzenetformátumot ír le, amivel kereszteződések geometriai leírását lehet megadni. A MAP üzeneteket SPAT üzenetekkel együtt használják. A SPAT üzenetek a jelzőlámpák jelenlegi státuszát és időzítési információit közvetíti. A kettő együtt a kereszteződéshez közeledő forgalomnak olyan információkat tud adni, amivel a forgalomban haladó járművek optimális sebességet választva, hatékonyan tudnak áthaladni a kereszteződésen.

A MAP és SPAT protokollok használatával az elsődleges célok között szerepel a károsanyag-kibocsátás csökkentése, a biztonság növelése és prioritás biztosítása mentőautóknak és tömegközlekedési eszközöknek. MAP és SPAT üzenetekkel a forgalomáramlás javítható, a megállások száma csökkenthető, ami hatékonyabb közlekedést és kisebb károsanyag-kibocsátást eredményez. A kevesebb megállás következtében a ráfutásos balesetek száma is csökken, ami jelenleg az autós balesetek egyik leggyakoribb típusa.

A pozitív hatások még nem bizonyítottak a gyakorlatban, és potenciális problémák is lehetnek. Az egyes fedélzeti rendszerek máshogy dolgozhatják fel az adatokat, és más sebességet javasolhatnak a sofőrnek. Ez veszélyes lehet, ha a kereszteződéshez érkezők egy részének gyorsítást, a másiknak pedig lassítást javasol a fedélzeti rendszere. A másik probléma a javasolt sebesség és megállási instrukciók közvetítése a sofőrnek úgy, hogy egyértelmű és érthető instrukciókat kapjon, de a figyelmét ne vonja el a forgalomról.

Ezen potenciális problémák miatt még sok kutatómunka és szabványosítás szükséges egy éles rendszer bevetése előtt. Több kísérleti rendszer telepítése megtörtént már. Ezek közül a legnagyobb az európai Compass4D projekt volt, amely keretein belül Veronában és Berlinben telepítettek SPAT/MAP rendszert.

A kiépített rendszerek lehetnek lokálisak vagy centralizáltak. Lokális esetben minden kereszteződés forgalomirányító rendszere csak a saját információi alapján hoz döntéseket. Egy centralizált rendszerben minden kereszteződés forgalomirányító rendszere egy központi rendszerrel kommunikál, ami a beérkező információk alapján utasításokat küld a forgalomirányító rendszereknek.

Lokális rendszer kiépítése egyszerűbb, de hosszú távon a kereszteződésekbe érkező forgalmi adatok egy olyan centralizált forgalomirányítási szekvencia kialakítására ad lehetőséget, ami a megfigyelt forgalmi viszonyok mellett optimális.

A MAP és SPAT protokollok alkalmazása lehetséges segédrendszerként humán sofőrök számára vagy autonóm közlekedésben, C-ITS rendszer részeként. A MAP protokoll autonóm rendszerekben fontos lesz a kereszteződéseken kívül más akadályok, útgörbületek, útfelújítási munkálatok geometriájának leírására is.

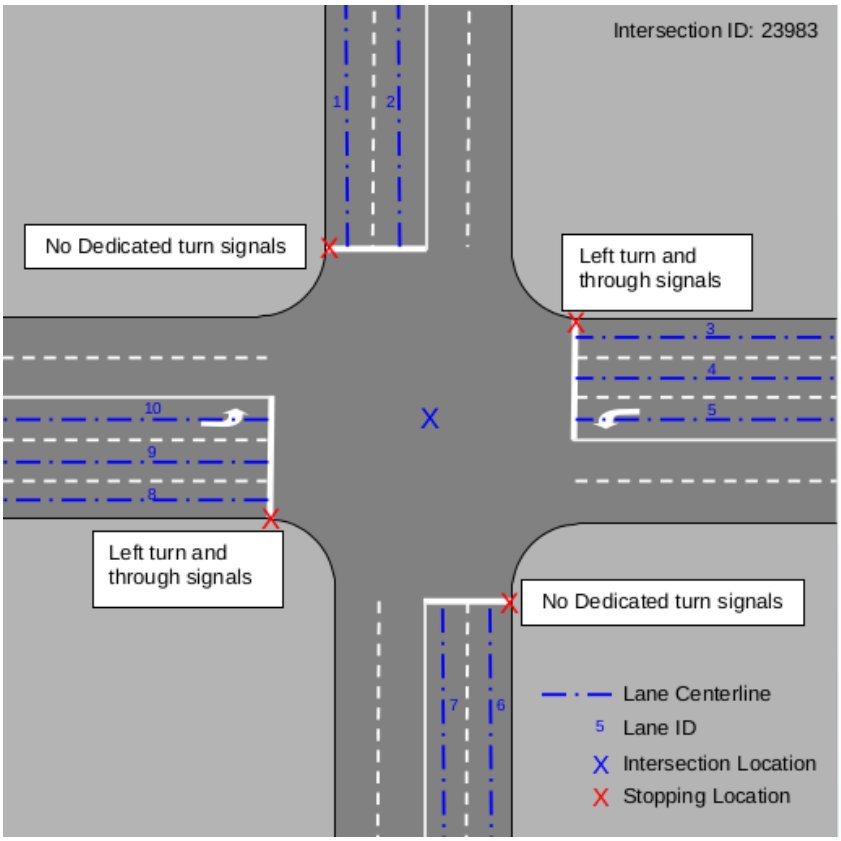
A MAP protokoll használatához először fel kell térképezni (mapping) az úttest geometriáját és releváns attribútumait úgy, mint járdaszegélyek, útburkolati jelek, táblák, sávok száma és szélessége. A mapping eljárásra többféle technológia is használható. Ezek közül a legrészletesebb adatokat a LIDAR (Light Detection and Ranging) szolgáltatja, de a berendezések drágák és az üzemeltetéséhez kiképzett személyzet kell. Más használható megoldás lehet légi felvételek készítése, építési tervek alapján történő MAP adatok előállítása vagy crowdsourcing [14].

A MAP protokoll az úttest és az úttest közvetlen környezetének geometriai leírására szolgál. A protokoll elsődleges alkalmazási területe kereszteződések geometriai/topológiai jellemzése.

Egy MAP üzenetben lévő geometriai leírás mindig egy referenciaponthoz képest van megadva. Egy üzenet több kereszteződést írhat le, de nem kell mindegyikhez külön referenciapontot megadni. Amíg nincs újabb referenciapont megadva, addig a legutóbbit használja.

Minden leírt kereszteződéshez megadhatja a hozzá tartozó sávok geometriáját és azok engedélyezett megközelítéseit. Továbbá tartalmazhat akadályokra vonatkozó leírásokat is. Például gyalogos-átkelőhelyek, vasúti sínek, kerékpáros sáv.

Ezen túlmenően a MAP üzenetben leírhatók a sávok szomszédsági viszonyai és a szomszédos sávok között megengedett manőverek is. Egy adott sávra vonatkozó korlátozásokat is le lehet írni vele. Ilyen korlátozás lehet például, hogy egy sávból milyen irányba lehet tovább haladni vagy milyen járművek használhatják.



2. ábra: Egyszerű példa a MAP protokollra

A képen látható kereszteződés bemutatja a MAP alapjait, de a szabvány ennél sokkal többre képes. Egy kereszteződés leírása egy Intersection objektummal valósítható meg. A példában az Intersection objektum azonosítója 23983 (IntersectionID). Az Intersection objektumhoz hozzárendelt ApproachObject meghatározza a sávonként megengedett haladásokat. A sávok azonosítói (LaneNumber) 1-től 10-ig vannak kiosztva. Az 5-ös és a 10-es sávra korlátozva van a továbbhaladás iránya, a többire nincsenek korlátozások megadva. A korlátozások a VehcileLaneAttributes mezőben adhatók meg. Ebben az esetben, mivel mindkét korlátozott sávból csak balra lehet fordulni, a VechicleLaneAttributes értéke 4 (maneuverLeftAllowed).

3.3 A MAP és SPAT alkalmazásai: C-ITS vezetéstámogató alkalmazások

A vezetéssegítő alkalmazások vagy ADAS (Advanced Driver Assistance Systems) célja a sofőr segítése, hogy a vezetés biztonságosabb (és kényelmesebb) legyen.

A C-ITS segítségével megszerezhető adatokkal olyan alkalmazásokat lehet létrehozni, amelyek már az első fázisoktól kezdve használhatók. Ezek az alkalmazások figyelmeztetni tudják a sofőrt kritikus eseményekre vagy javaslatot tehetnek neki például az ideális sebességre vonatkozóan.

A következő alfejezetekben néhány MAP és/vagy SPAT üzenetekre épülő C-ITS vezetéstámogató alkalmazást fogok bemutatni.

3.3.1 Curve Speed Warning

A Curve Speed Warning központjában egy RSU van, ami hirdeti a közeledő járműveknek, hogy az úton egy veszélyes kanyar van és javaslatot tesz a kanyar bevételének a sebességére.

Ez a figyelmeztetés nagyon fontos, mert sok beláthatatlan kanyar van, aminél a sofőrök hajlamosak túlbecsülni azt a sebességet, amivel azon biztonságosan át lehet haladni. Ebből nagyon könnyen baleset lehet, mert amikor a jármű vezetője észleli, hogy túl gyorsan kezdte meg a kanyart, és fékezni kezd, könnyen elveszítheti az irányítást a jármű felett. Ez éjszaka vagy rossz látási viszonyok között még nagyobb problémát jelent.

Amennyiben több száz méterrel a kanyar előtt már tudja, hogy milyen sebességet érdemes választania, ez a probléma könnyen elkerülhető [15].

3.3.2 Emergency Vehicle Priority

Egy megkülönböztetett jármű (pl. rendőr, mentő, tűzoltó) egy jelzőlámpás kereszteződés felé haladva a kereszteződésben telepített RSU-tól kérhet elsőbbséget a kereszteződésen való áthaladásra.

Az RSU ilyenkor, ha teljesíteni tudja a kérést, módosítja a jelzőlámpák időzítésére vonatkozó leírókat, amiket elküld a jelzőlámpákat vezérlő egységnek, majd módosítja az általa sugárzott SPAT üzeneteket. Ezzel a közlekedés résztvevői már a módosított jelzőlámpa státusz és időzítési információkat kapják meg. Végül visszajelzi a megkülönböztetett járműnek, hogy az elsőbbségi kérését ki tudja szolgálni [14].

3.3.3 Wrong Way Driving Warning

A kijelölt haladási iránnyal ellentétes irányban közlekedő járműről figyelmezteti azokat a járműveket, amelyek az érintett út irányába közelednek. A figyelmeztetés célja, hogy az ilyen szituációkból következő frontális ütközéseket el lehessen kerülni.

Ez egy korlátozott ideig fennálló figyelmeztetés, amit a jármű periodikusan ismétel ezalatt az idő alatt. A minimális ismétlési frekvencia erre a figyelmeztetésre vonatkozóan 10 Hz és a késleltetés maximálisan 100 ms lehet, mivel ez egy időkritikus esemény [15].

3.3.4 Signal Violation Warning

Egy észlelő ITS állomás jelzi a többi forgalomban résztvevőnek, hogy egy jármű tiltott jelzés alatt haladt át egy kereszteződésen. Ez az ITS állomás valószínűleg egy RSU, mivel az úgyis ott van a kereszteződésben elhelyezve, hogy a kereszteződésre vonatkozó SPAT és MAP üzenteket küldje a közeledő járműveknek.

A minimális ismétlési frekvencia erre a figyelmeztetésre vonatkozóan 10 Hz és a késleltetés pedig maximálisan 100 ms lehet [15].

3.3.5 Green Light Optimal Speed Advisory

A szolgáltatás sebesség javaslatokat nyújt egy jelzőlámpás kereszteződés felé közeledő járművek számára. A javasolt sebesség olyan, hogy az adott kereszteződésen megállás nélkül át tudjanak haladni a járművek vagy olyan, hogy több egymást követő kereszteződésen optimálisan haladjanak át a járművek.

Az alkalmazás célja a forgalom-áramlás javítása, a megállások és a megállások utáni gyorsítások csökkentése, amivel a károsanyag-kibocsátás és a jelzőlámpás kereszteződésekben történő balesetek száma jelentősen csökkenthető [16].

4 Trajektóriák klaszterezése

A kereszteződésekben megfigyelt járművek pozícióadataiból minden járműhöz egy-egy trajektóriát lehet rendelni. A kereszteződés környezetében lévő úttest feltérképezéséhez, majd MAP/MAPEM struktúrába történő rendezéséhez és V2X kommunikációval történő kiküldéséhez ezeket a trajektóriákat kéne klaszterezni. A klaszterezés során a trajektóriák kereszteződésbe bemenő és a kereszteződésből kimenő részeinek az irányaiból a trajektóriák az irányuk szempontjából megkülönböztethetők. A trajektóriák egymástól való távolságát figyelembe véve pedig az ugyanolyan irányú trajektóriákat sáv szinten lehet megkülönböztetni egymástól.

Ennek a fejezetnek az a célja, hogy röviden bemutassa a trajektóriák klaszterezésére használt már létező módszereket [5]. Az ezek közül általam implementált algoritmusok részletesen be vannak mutatva a 9.2.3 Algoritmusok fejezetben.

4.1 Felügyelet nélküli tanulás

A felügyelet nélküli tanulás olyan algoritmusok csoportja, amelyek címkézetlen adatok közötti kapcsolatok jellemzésére használhatók.

4.1.1 Sűrűség alapú klaszterezés

A sűrűség alapú klaszterezés olyan klasztereket alakít ki, melyek sűrű területeket jelölnek ki az adathalmazban, és amiket ritkább területek választanak el egymástól. Az egyik jól ismert sűrűség alapú klaszterező a DBSCAN [17].

4.1.2 Hierarchikus klaszterezés

A hierarchikus klaszterezés klaszterek hierarchiáját alakítja ki, ami egy fa-struktúrára hasonlít, és dendrogramnak hívják. A hierarchikus klaszterezés lehet összevonó (agglomerative) vagy felosztó (divisive).

Az összevonáson alapuló algoritmus elején minden elemet külön klaszterbe sorol, majd minden lépésben a két legközelebbi klasztert összevonja, és a végén egyetlen klasztert

kapunk, ami az összes elemet tartalmazza. Az összevonáson alapuló algoritmust alulról felfele (bottom-up) eljárásnak is nevezik.

A felosztáson alapuló algoritmus kiindulási állapotában minden elemet egy klaszterbe sorol. Az algoritmus minden lépésében egy új klasztert hoz létre úgy, hogy a klasztereken belüli távolságok minimálisak legyenek. Az eljárás végén minden elem külön klaszterben van. A felosztáson alapuló algoritmust felülről lefele (top-down) eljárásnak is nevezik [18].

4.1.3 Spektrális klaszterezés

A spektrális klaszterezés gyökerei a gráfelméletből erednek, ahol ez a módszer használható csúcspontok csoportosítására az őket összekötő élek alapján. A módszer használható más adatok klaszterezésére is, nem csak gráfokra. A spektrális klaszterezésben a klaszterek létrehozása a sajátvektorok alapján történik [19].

4.2 Felügyelt tanulás

A felügyelt tanulás csoportjába olyan algoritmusok tartoznak, amelyek címkézett tanuló adatokon történő betanítási fázis után használhatók címkézetlen teszt adatok felcímkézésére.

4.2.1 Legközelebbi szomszéd analízis

A legközelebbi szomszéd analízis olyan algoritmusokat tartalmaz, ahol valamilyen szavazási rendszer határozza meg egy új entitás kategóriáját. Ilyen algoritmus például a k-NN, amiben az új entitás kategóriájának megállapítása a k legközelebbi szomszédjának szavazatai alapján történik [20].

4.2.2 Statisztikai modellek

A statisztikai modellek közül a trajektória klaszterezés körében gyakran használják a keverék modelleket és a Bayes-osztályozókat.

A keverék modellek úgy tekintenek az adatokra, mint különböző valószínűségi eloszlások keverékeként előálló megfigyelésekre. A keverék modellek közül az egyik leggyakrabban használt a Gaussian Mixture Model (GMM) [21].

A Bayes-osztályozók alapja a Bayes-tételt. A tanulási folyamat során a kategóriaváltozók és a többi attribútum értékei közötti feltételes valószínűségeket tanulja meg, amit utána a tesztadatokon úgy alkalmaz, hogy az attribútumokhoz legnagyobb feltételes valószínűséggel előálló kategóriaváltozót rendeli [22].

4.2.3 Neurális hálózat

A neurális hálózatok egy bemeneti, egy kimeneti, és több rejtett rétegből állnak. Az egyes rétegek neuronokból állnak. Az egyes neuronok összeköttetésben állnak az előtte és az utána lévő réteg neuronjaival. Minden összeköttetés egy-egy súlyt hordoz, így a kimeneti rétegben egy lineáris egyenletrendszert kapunk. A tanulási fázisban a súlyokat úgy módosítja, hogy a globális hibát minimalizálja. A tesztelési fázisban pedig a neurális hálózat összeköttetéseiben hordozott súlyok felhasználásával létrehozott lineáris egyenletrendszer megoldását kapjuk kimenetként [23].

5 Helymeghatározási módszerek

A jármű-kommunikációs rendszerekre épülő alkalmazások számára elengedhetetlen egy bizonyos pontosságú helymeghatározási pontosság elérése annak érdekében, hogy a feladatukat megfelelően el tudják végezni. A MAP/MAPEM struktúrák (és egyéb nagy felbontású térkép információk) előállítására is használható CAM üzenetekbe szintén ilyen helymeghatározási módszerekkel kerülnek az aktuális pozícióadatok. A helymeghatározási módszer kiválasztásánál a legfontosabb szempont az, hogy a helymeghatározás pontossága az összes támogatott alkalmazás számára megfelelő legyen.

A kereszteződések feltérképezése tekintetében is fontosak a nagy pontosságú pozícióadatok, hiszen egy sáv szélessége általában 2,5-3,5 méter szokott lenni, tehát pontatlan pozícióadatok használatával a trajektóriák sáv szintű csoportosítása nagyon nehéz feladat lenne.

A következő alfejezetekben a lehetséges helymeghatározási megoldásokat fogom röviden bemutatni.

5.1 Global Positioning System (GPS)

Az 1970-es évek elején az Amerikai Védelmi Minisztérium (Department of Defense – DoD) egy robusztus, stabil navigációs rendszert akart kiépíteni. A javasolt megoldásukban műholdak támogatták a helymeghatározást. A GPS rendszerük neve Navigation System with Timing and Ranging (NAVSTAR) lett. Az első műholdjukat 1978-ban állították pályára, és 1993-ra a 24 műholdból álló rendszer teljesen működőképes lett. [24]

A legutóbbi, 2008-as, teljesítmény standard mérései szerint a lakossági GPS-szel elérhető horizontális pontosság 10-20 méter, a vertikális pontosság pedig 20-40 méter tipikusan [18]. Egy másik forrásból származó mérésben a GPS átlagos pontossága három dimenziós helymeghatározás esetén nagyjából 25 méter, két dimenziós esetben pedig körülbelül 20 méter [26].

5.2 Differential GPS (DGPS)

A DGPS bázisállomások által sugárzott korrekciós jelek segítségével próbálja javítani a GPS helymeghatározási pontosságát. A bázisállomások egy előre meghatározott helyen vannak, a pozíciójuk nem változik. A bázisállomás az ismert pozíciójából és a saját GPS vevőjén keresztül megszerzett pozícióadatból ki tudja számolni a helymeghatározás hibáját. Ez alapján sugározza a korrekciós jelet, amivel a hatósugarában lévő DGPS vevők a műholdaktól kapott pozícióadatot pontosítani tudják.

A DGPS pontosságát is megvizsgálták a GPS-nél említett mérésben, ahol azt találták, hogy a DGPS átlagos pontossága három dimenziós helymeghatározás esetén nagyjából 5 méter, két dimenziós esetben pedig körülbelül 4 méter [26].

5.3 Real-time Kinematic (RTK)

Az RTK rendszerben a DGPS-hez hasonlóan telepített földi bázisállomások szolgáltatnak korrekciós adatokat a felhasználóknak. Az RTK a GPS vivőjel fázismérésére támaszkodik a korrekciós adatok meghatározásában. Elméleti szinten az RTK rendszerrel 1-2 centiméter pontosságú helymeghatározás is elérhető.

Az RTK pontosságát több publikációban is megvizsgálták, és a mérések alapján valós körülmények között is nagyon pontos helymeghatározásra képes a rendszer. A pontossága átlagosan 10-20 centiméter [27] [28], ami már nagyon jónak mondható.

6 Keretrendszer

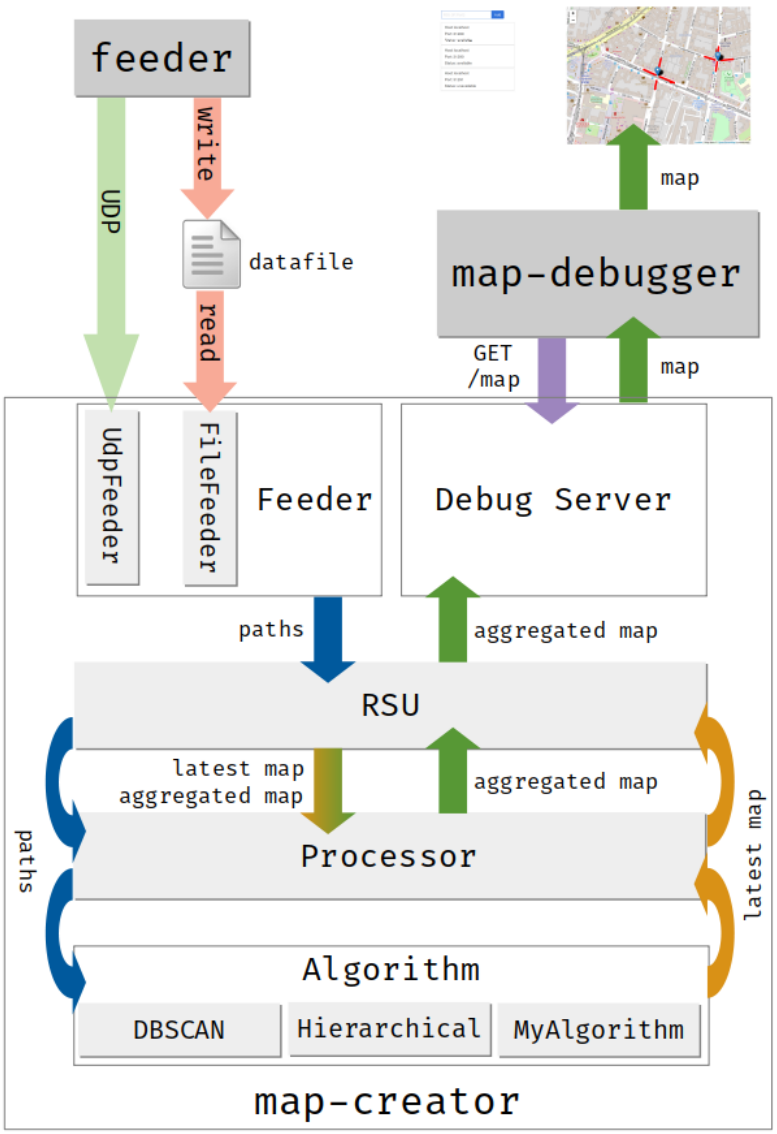
A kereszteződésben elhelyezett RSU folyamatosan fogadja a kereszteződés környezetében lévő OBU-val felszerelt járművektől a CAM üzeneteket, amiben a megtalálható a jármű földrajzi koordinátája. A javasolt megoldás a járművek útvonalai alapján próbálja meg leírni a kereszteződést az útvonalak klaszterezésével.

Az algoritmusok összehasonlításához és a legjobb algoritmus kiválasztásához szükség van egy olyan rendszerre, amiben ezeket könnyen és gyorsan ki lehessen próbálni, változatos adatokkal meg lehessen hajtani, és egyértelműen el lehessen dönteni, hogy az adott algoritmus megfelelően működik-e. Ebben a fejezetben annak a keretrendszernek a felépítéséről lesz szó, amit erre a célra megterveztem és implementáltam.

A keretrendszer négy nagy komponensből áll, amiket különálló programokként készítettem el. Ez egy tervezési döntés volt. Meg lehetett volna valósítani a négy komponenst egy nagyobb komplexitású programban is, de abban az esetben vagy a flexibilitásából veszített volna vagy a konfiguráció feldolgozása túlságosan bonyolult lett volna. A kialakított keretrendszer rugalmasságát az adja, hogy tetszőleges számú map-creator példányt lehet indítani, amik különböző kereszteződésekkel és különböző algoritmusokkal tudnak dolgozni, és mindezt egyszerűen lehet konfigurálni.

Ha mindezt egy nagy alkalmazás keretein belül lenne megvalósítva, akkor ugyanezt a funkcionalitást úgy lehetne megvalósítani, hogy az alkalmazásnak folyamatosan figyelnie kéne a konfigurációs fájlt, hogy történtek-e benne módosítások, és ha szintaktikailag megfelelő változás történt, akkor kéne csinálnia valamit (például egy új map-creator példányt elindítani). Mindez felesleges komplexitást vinne a rendszerbe, és a működés lényegi részének, az algoritmusok a rendszerbe való egyszerű csatolásának, szempontjából teljesen irreleváns, ezért döntöttem a négy nagy komponens szeparálása mellett.

A keretrendszer négy komponense a következők: feeder, map-creator, map-debugger, map-validator.



3. ábra: A megtervezett és implementált keretrendszer felépítése

A feeder komponens feladata különböző adatforrásokból egy olyan adatstruktúrában létrehozni és továbbítani az adatokat, amiben minden lényeges információ benne van, amiből a map-creator elő tudja állítani a MAP üzeneteket. A feeder lehetőséget biztosít az adatok valósidejű továbbítására UDP socketen keresztül vagy azok fájlba írására, utólagos felhasználásra.

A map-creator komponens a négy közül a legösszetettebb komponens, a legtöbb feladattal és felelősséggel. Egyrészt biztosítania kell egy feldolgozó egységet, amiben a használt algoritmus könnyen cserélhető. Kommunikálnia kell a feeder komponenssel (valósidejű mód esetén) és az attól érkező adatokat továbbítania kell a feldolgozó

egység felé. Valamint elérhetővé kell tennie a létrehozott MAP üzeneteket, hogy azokat további alkalmazások fel tudják használni.

A map-debugger komponens a MAP üzenetek megjelenítésére szolgál. Ez a komponens lényegében arra szolgál, hogy az algoritmusok jóságáról egy gyorsan és könnyen értelmezhető visszajelzést adjon. A map-debugger a map-creator által létrehozott MAP üzenetekkel dolgozik, valamint képes a map-validator által leírt kereszteződés megjelenítésére is, ami egy kicsit eltér a MAP struktúrától.

A map-validator a létrehozott MAP üzenetek helyességének eldöntéséért felel. Két bemenetet kell adni neki: a referencia és az ellenőrzendő MAP struktúrákat. Ez a komponens önmagában nem képes valósidejű működésre. Ez nem is volt cél.

A következő alfejezetekben a négy komponens részletes leírása található.

6.1 feeder

A feeder egy Pythonban megírt program, ami kétféle adatforrásból tud dolgozni: CSV fájlokból és SUMO szimulációkból. CSV fájlok esetén a fejléc megléte szükséges, és meg kell adni a hozzárendelést, hogy melyik oszlop milyen információt tartalmaz, valamint a CSV fájl elérési útját. SUMO szimuláció esetén egyedül a SUMO konfigurációs fájl elérési útját kell megadni.

A feeder olyan adatstruktúrában hozza létre az adatokat, amiben van egy azonosító, valamint a szélességi és a hosszúsági koordináta. Az azonosító a trajektóriák egyértelmű azonosítására szolgál, a szélességi és a hosszúsági koordináta pedig annak a trajektóriának egy pontját adja meg. A feeder képes UDP socketen keresztül továbbítani az adatokat vagy azokat fájlba írni.

Az UDP socketet használó működés esetén a létrehozott adatok bekerülnek egy queue-ba. A queue-ba került adatok továbbítására egy UDP szerver szolgál. Az UDP szerverre „connect” üzenettel lehet feliratkozni és „disconnect” üzenettel lehet leiratkozni. Az UDP szerver több kapcsolatot tud kezelni, és az összes feliratkozónak egy megadott periodicitással elküldi a queue-ban lévő adatokat, majd a queue-t kiüríti.

A fájlba írást használó működési mód lényegesen egyszerűbb, mint az UDP socket alapú működés. Értelemszerűen socket helyett fájlba dolgozik, így nincs szükség a queue-ra és a kapcsolatok kezelésére se.

A feeder UDP socketes működését használva nagyjából olyan szituáció valósítható meg, mintha egy RSU a járművektől érkező CAM üzeneteket kapná meg, ami alapján valós időben próbál tud számításokat végezni.

A feeder fájlba írásos működésével nagy mennyiségű adat előre legyártódik, amivel a map-creator számításait lehet lényegesen felgyorsítani. Ez főleg abban az esetben jelentős, amikor az adatforrás egy SUMO szimuláció, mert így a szimulációs lépések közötti késleltetés kiküszöbölhető. Ez a működési mód javasolt, ha sok mérést kell elvégezni ugyanazzal a szimulációval. Például, ha a map-creatorban implementált algoritmusokat, azokat különbözőképpen felparaméterezve, össze akarjuk hasonlítani ugyanazon a szimuláción, akkor csak egyszer kell legyártani az adathalmazt fájlba, és utána ugyanazt a fájlt fel lehet használni az összes mérésben.

A feeder komponens felépítése UML osztálydiagram formájában a függelék 8.1 részében látható.

A konfiguráció egy JSON fájlban adható meg. A konfigurációban meg kell adni a működési módot, ami lehet a UDP socketen (*feed*) keresztül vagy fájlba írva (*generate*). Továbbá meg kell adni az adatforrást, ami lehet egy CSV fájl (*csv*) vagy egy SUMO szimuláció (*sumo*).

A CSV fájlok esetén meg kell adni az azonosító (*identifier*), a szélesség (*latitude*) és a hosszúság (*longitude*) összerendelését a CSV fájl oszlopainak fejléceihez, és meg kell adni a fájl elérési útját (*filepath*).

A SUMO szimuláció esetén csak a SUMO konfigurációs fájl elérési útját kell megadni (*sumocfg_path*).

A feeder konfigurációs sémája CSV adatforrás esetén a függelék 8.2 részében, SUMO adatforrás esetén pedig a függelék 8.3 részében látható.

6.2 map-creator

A map-creator egy Pythonban megírt program, egy példánya egy RSU-nak feleltethető meg, aminek van egy referencia pontja és egy hatótávolsága. Az ellátott feladatok szempontjából négy részre lehet elkülöníteni.

Az egyik rész felelőssége az adatok megszerzése valamilyen forrásból, azok szűrése az RSU referencia pontja és hatótávolsága alapján, és az adatok továbbítása a feldolgozó egység felé. Az adatok jöhetnek feeder komponenstől UDP socketen keresztül vagy fájlból beolvasva.

A második rész a feldolgozó egység és a hozzá tartozó adatmodellek, ami a feldolgozás általános viselkedését valósítják meg, és a hozzá csatolt algoritmust felhasználva a feldolgozás specializálható.

A harmadik részbe sorolnám az algoritmus interfészt és az azt implementáló konkrét algoritmusokat.

A negyedik rész pedig a debug szerver, ami egy HTTP szerver. A debug szerver tárolja, és elérhetővé teszi a legfrissebb MAP üzenetet, amit a map-creator előállított. Továbbá van egy ping útvonal, amin a debug szerver elérhetőségét lehet tesztelni. A ping útvonalat a map-debugger használja az RSU elérhetőségének megjelenítéséhez.

A map-creator felépítése UML osztálydiagram formájában a függelék 8.4 részében látható.

A feederhez hasonlóan a map-creator konfigurációja is egy JSON fájlban adható meg, de ennek a komponensnek a működéséből eredően a konfigurációja is sokkal összetettebb.

A *log_level* megadásával szabályozható a program logolása, aminek a Python standard könyvtárbeli logger által is elfogadott szinteket lehet megadni. A *dist_func* megadásával állítható be, hogy a map-creator feldolgozó egysége milyen távolságfüggvényt használjon. Az *algorithm* mezőben adható meg, hogy a feldolgozó egység melyik algoritmust használja klaszterezésre. A *debug_server* IP címe és portja is megadható a konfigurációban. A *feeder* felkonfigurálása fájlból olvasó vagy UDP socketen keresztül kommunikáló módban lehetséges. A feldolgozó egység vagy *processor* konfigurálásánál kell megadni az RSU hatótávolságát vagyis azt, hogy mekkora az a maximális távolság,

amiből eljutnak hozzá a járművektől érkező üzeneteket. Az RSU-nak szüksége van egy referencia pontra, amihez képest a MAP üzeneteket leírja offszetekkel, de a map-creator csak arra használja ezt a pontot, hogy ezt tekinti a kereszteződés középpontjának. Ezt a *reference_point* értékkel lehet megadni. A szimulált RSU felkonfigurálása az *rsu* paraméter alatt történik. Megadható neki a *time_window*, ami azt adja meg, hogy hány másodperces csúszóablakban jegyezze meg a trajektóriák pontjait. Memóriakezelési szempontból nem lenne praktikus az összes pontot örökké tárolni, ezért a régi pontokat ennek a csúszóablaknak a segítségével törli ki. Az RSU másik paramétere az *update_time*, ami azt adja meg, hogy a feldolgozó egység hány másodpercenként fusson le. Ezzel a periodicitással fog frissülni a létrehozott MAP adatstruktúra.

A konfigurációs séma a függelék 8.5, a DBSCAN konfigurálása a 8.6, a Hierarchial konfigurálása a 8.7, a MyAlgorithm konfigurálása a 8.8, a File Feeder konfigurálása a 8.9, az UDP Feeder konfigurálása pedig a 8.10 részében látható.

A map-creator egy Feeder objektumon keresztül éri el az adatokat. A Feeder lehet FileFeeder vagy UdpFeeder. A FileFeeder objektum egy előre legenerált fájlból olvassa az adatokat. Az UdpFeeder objektum pedig a feeder komponenstől UDP socketből olvassa az adatokat. A Feeder objektum a beolvasott adatokból időbélyeggel ellátott Point objektumokat készít, amiket egyesével továbbít az RSU objektumnak, ha azok az RSU hatósugarában vannak. Az RSU objektum a beérkező pontokat útvonalakban, egy Path objektumokból álló listában, tárolja. Az RSU objektumban periodikusan lefut egy frissítés művelet, amiben frissíti a tárolt útvonalakat és újraszámolja a MAP-ot. A tárolt útvonalak frissítése annyit jelent, hogy az összes olyan pontot törli az útvonalakból, ami a csúszóablakon kívül esik, és ha egy útvonalnak már az összes pontja törölve lett, akkor magát az útvonalat is eltávolítja.

A MAP újraszámolása során először az RSU továbbítja a tárolt útvonalait a feldolgozó egységnek. A feldolgozó egység elvégzi rajtuk az előfeldolgozást, ami az összes útvonalat lecseréli olyan útvonalra, amiben az eredeti útvonalnak csak a kulcspontjai szerepelnek. Egy útvonal kulcspontjait a pontjaik közötti haladási irány változásával határoz meg. Ha egy bizonyos határérték feletti az irányváltozás egy adott pontban, akkor az a pont kulcspontként lesz kiértékelve. Minden útvonal első és utolsó pontja kulcspontok lesznek, hogy még egy egyenesen haladó útvonalon is legyen legalább két

kulcspont. Az előfeldolgozás után a feldolgozó egység a már csak kulcspontokból álló útvonalakat fogja a használt algoritmusnak továbbítani klaszterezésre, ami után előáll a MAP adatstruktúra. Az RSU miután megkapta a feldolgozótól az újonnan előállt MAP-ot, elküldi az általa tárolt jelenlegi MAP-ot és az újonnan előállt MAP-ot a feldolgozónak utófeldolgozásra. Az utófeldolgozás lényege, hogy a jelenlegi MAP-ot kiegészítése azokkal a részletekkel az újonnan előállított MAP-ból, amik a jelenlegi MAP-ban nincsenek benne. Az RSU minden frissítés végén elkészíti a tárolt MAP adatstruktúra JSON reprezentációját, amit továbbít a debug szervernek.

6.2.1 Távolságfüggvények

A megfelelő távolságfüggvény nagyon fontos ahhoz, hogy két trajektóriát jól tudjunk összehasonlítani. Ha rossz a távolságfüggvény, akkor semmilyen klaszterező eljárástól nem várhatjuk el, hogy jó eredményt adjon, hiszen már eleve rossz távolság adatokkal kell dolgoznia. Éppen ezért nagyon fontos, hogy a feladathoz jól illeszkedő távolságfüggvényt használjunk.

6.2.1.1 Dynamic Time Warping (DTW)

A Dynamic Time Warping főleg az idősor elemzésben használt algoritmus, ahol két időbeli sorozat közötti hasonlóság meghatározására használják, amik sebességben eltérhetnek egymástól.

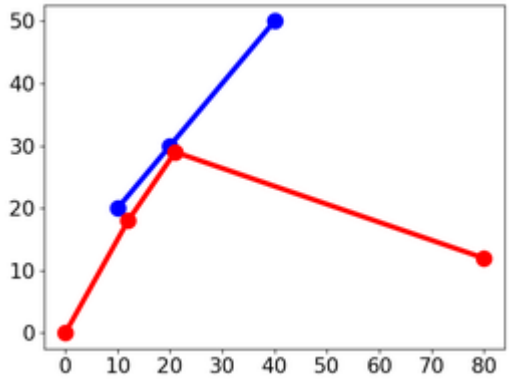
Általánosságban a Dynamic Time Warping egy olyan módszer, ami két sorozat közötti legjobb egyezést számítja ki.

Az algoritmus alapvetően nem túl bonyolult. Van két bemeneti sorozat: X, Y. Az X sorozat hossza legyen L(X), az Y sorozaté pedig L(Y). Kezdetben inicializálunk egy (L(X) + 1) x (L(Y) + 1) méretű táblázatot, amit most jelöljünk T-vel, és a T[0][0] elemét beállítjuk 0-ra. Minden i = [1, L(X)] és j = [1, L(Y)] párokra a táblázat T[i][j] elemét beállítja a távolság(X[i], Y[i]) + minimum(T[i - 1][j], T[i][j - 1], T[i - 1][j – 1]) értékre. A két sorozatra meghatározott távolságot pedig a táblázat utolsó sorának utolsó oszlopából olvashatjuk ki, tehát a távolságuk a T[L(X) + 1][L(Y) + 1] lesz.

Például adott az alábbi két pontsorozat:

X = [(10, 20), (20, 30), (40, 50)]

Y = [(0, 0), (12, 18), (21, 29), (80, 12)]



	0	1	2	3	4
0	0.00	inf	inf	inf	inf
1	inf	14.14	16.97	32.53	131.52
2	inf	42.43	25.46	18.38	103.24
3	inf	98.99	65.05	45.25	74.95

4. ábra: DTW példa számítás (X: kék, Y: piros)

6.2.1.2 Euklideszi távolság

Két pont közötti Euklideszi távolság a két pontot összekötő szakasz hossza. A Descartes-féle koordináta rendszerben ez úgy írható fel, ha van egy p = (p_1, p_2, …, p_n) és egy q = (q_1, q_2, …, q_n) pont, akkor a p és q közötti távolság felírható a következő formulával:

$$d(p,q)=d(q,p)=\sqrt{(p_1-q_1)^2+(p_2-q_2)^2+...+(p_n-q_n)^2}=\sqrt{\sum_{i=1}^{n}(p_i-q_i)^2}$$

6.2.2 Algoritmusok

6.2.2.1 DBSCAN

A DBSCAN egy sűrűség alapú klaszterező algoritmus. Az algoritmust 1996-ban találták ki, és a térnek azon pontjait csoportosítja egy klaszterbe, amik szorosan helyezkednek el. Az algoritmus kiugró értékként kezeli azokat a pontokat, amik alacsony sűrűségű régiókban helyezkednek el, vagyis a tér egy olyan részén helyezkednek el, ahol a pontok ritkábbak, és a legközelebbi szomszédjaik túl messze vannak tőlük.

Az algoritmus a térben lévő pontokat alapvetően két paraméter alapján csoportosítja. Az eps paraméter megadja, hogy a pontoknak mekkora sugaró körén belül kell a

szomszédokat keresni. A minPts pedig meghatározza, hogy az algoritmus minimum hány szomszédot vár el egy ponttól az eps sugarú környezetében.

A tér pontjait négy kategóriába sorolja.

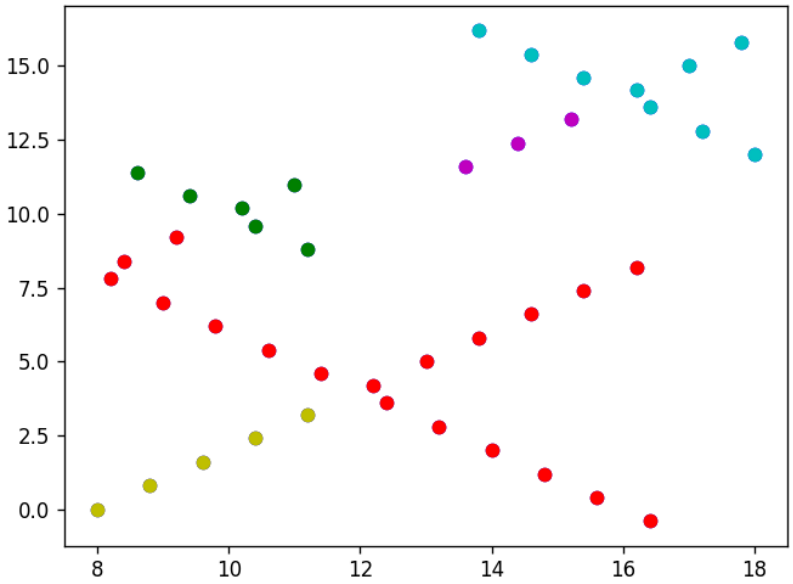
- core point
- directly reachable
- reachable
- outlier/noise

Egy p pont core point, ha legalább minPts pont van az eps sugarú környezetében (beleértve p-t is).

Egy q pont directly reachable p-ből, ha a q pont a p pont eps sugarú környezetében van és a p core point.

Egy q pont reachable p-ből, ha létezik p_1, p_2, …, p_n útvonal, ahol $p_1 = p$ és $p_n = q$ úgy, hogy minden p_{i+1} directly reachable p_i-ből. Ebből következik, hogy az útvonalon minden pontnak core point-nak kell lennie, kivéve q-t.

Minden pont, ami nem reachable egyik pontból sem, azok outlier vagy noise pontok.



5. ábra: Példa a DBSCAN klaszterezésre (eps=1.25, minPts=3)

6.2.2.2 Hierarchical

A hierarchikus klaszterezés egy olyan klaszterezési módszer, amely klaszterek hierarchiáját építi fel. Kétféle stratégia szerint lehet végrehajtani: top-down vagy bottom-up.

A top-down megközelítésben a hierarchia legfelső szintjéről indul, ahol minden pont egy klaszterben van. A hierarchiában lefelé haladva minden lépésben egy klasztert vág ketté, tehát lefelé haladva minden hierarchia szinten egyel több klaszter van, mint a felette lévő hierarchia szinten.

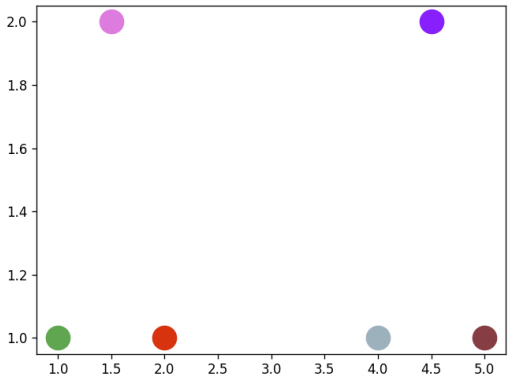
A bottom-up megközelítésben a hierarchia legalsó szintjéről indul, ahol minden pont külön klaszterben van. A hierarchiában felfelé haladva minden lépésben két klasztert összevon. Ebben az esetben is igaz, hogy minden hierarchia szinten egyel több klaszter van, mint a felette lévő szinten.

Elméletileg a hierarchikus klaszterezés nem zárja ki több klaszter összevonását vagy több klaszter szétválasztását egy lépésen belül, ha több ugyanolyan jó megoldás van, így nem igaz minden esetben, hogy egy adott hierarchia szinten pontosan egyel több klaszter van, mint a felette lévő szinten, de a legtöbb implementáció ilyen esetekben kiválasztja az egyik megoldást a sok ugyanolyan jó közül, és csak egy összevonást vagy szétvágást végez el egy lépésben.

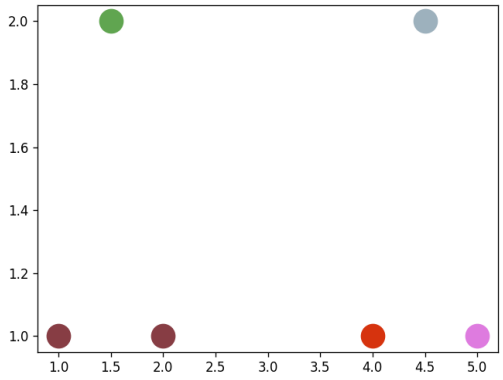
A hierarchikus klaszterezés a klaszterek hasonlósága alapján dönti el, hogy melyik kettőt kell összevonni vagy szétválasztani. A hasonlóság megállapításához egy távolságfüggvény és egy úgynevezett „linkage criteria” szükséges. A „linkage criteria” mondja meg, hogy két halmaznak mi a távolsága az elemeik távolságának a függvényében. A „linkage criteria” nagyon sokféle lehet, én ezekből hármat implementáltam: single linkage, complete linkage, average linkage.

A single linkage függvény két halmaz távolságának a legközelebbi pontjaik távolságát veszi. A complete linkage függvény két halmaz távolságának a legtávolabbi pontjaik távolságát veszi. Az average linkage függvény pedig két halmaz távolságának a pontjaik átlagos távolságát veszi.

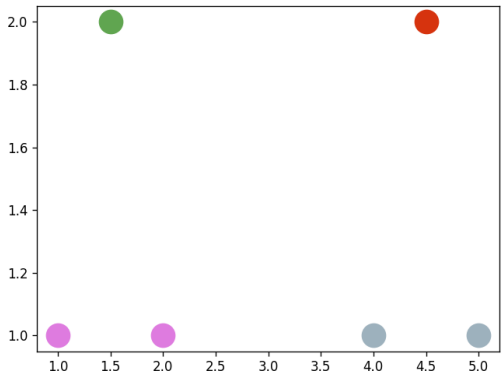
A következő példában két azonos oldalhosszúságú egyenlő szárú háromszög pontjaira nézzük meg a hierarchikus klaszterezést a bottom-up módszerrel. A pontok a következők: [(1, 1), (2, 1), (1.5, 2), (4, 1), (5, 1), (4.5, 2)]. A klaszterezés eredménye alább látható lépésről-lépésre.



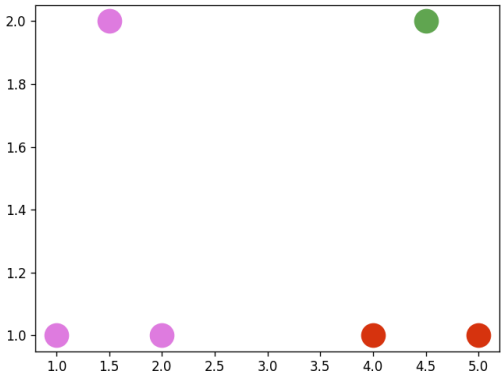
6. ábra: Hierarchikus klaszterezés bottom-up példa: 1. lépés



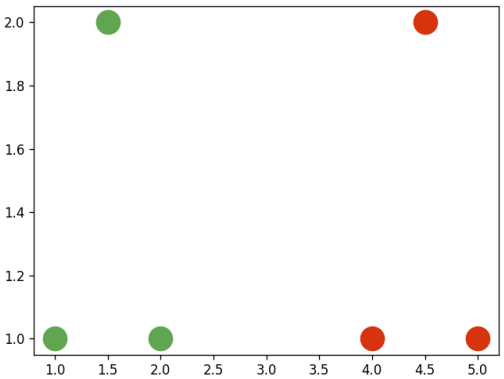
7. ábra: Hierarchikus klaszterezés bottom-up példa: 2. lépés



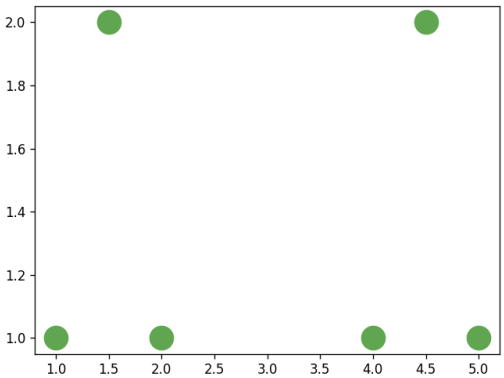
8. ábra: Hierarchikus klaszterezés bottom-up példa: 3. lépés



9. ábra: Hierarchikus klaszterezés bottom-up példa: 4. lépés



10. ábra: Hierarchikus klaszterezés bottom-up példa: 5. lépés



11. ábra: Hierarchikus klaszterezés bottom-up példa: 6. lépés

6.2.2.3 MyAlgorithm

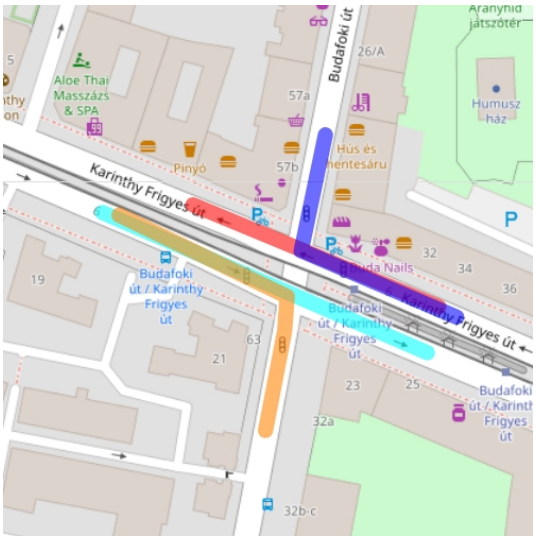
A MyAlgorithm az általam kitalált heurisztikus algoritmus, amit specifikusan trajektóriák csoportosítására lett megvalósítva. Az algoritmus elképzelése mögött vizuális megközelítések vannak: ha ránézek egy térképre rajzolt két trajektóriára, akkor a közelségük és a formájuk alapján meg tudom róluk mondani, hogy azok ugyanazon az útvonalon haladtak-e. Esetleg még azt is, hogy ugyanabban a sávban vagy sem.

Az algoritmus két paramétere a diff_distance és a diff_heading. A diff_distance azt adja meg, hogy két trajektória pontjainak mi lehet a maximális átlagos távolsága, a diff_heading pedig azt adja meg, hogy két trajektória átlagos haladási irányának mekkora lehet a maximális különbsége ahhoz, hogy a két trajektóriát megegyezőnek tekintse az algoritmus.

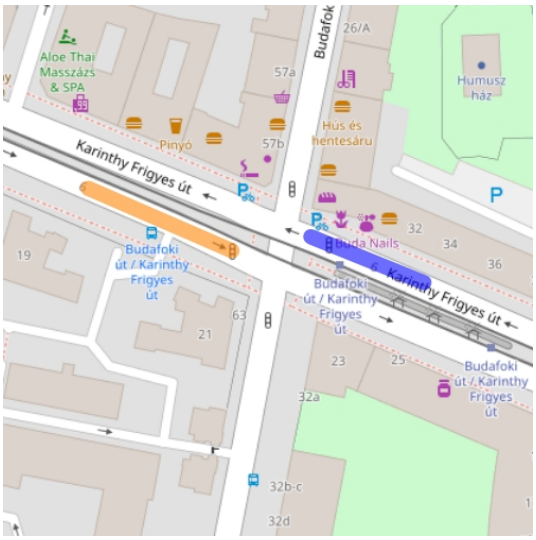
Az algoritmus először minden trajektóriát a kereszteződés referencia pontjához legközelebbi pontjánál kettéválaszt ingress és egress részekre. Ezután az ingress és az egress trajektóriákat egymástól elkülönítve vizsgálja meg, és megnézi minden trajektória párra, hogy azok megegyeznek-e. Először az ingress trajektóriákat vizsgálja meg, és ha talál köztük megegyező párt, akkor a pár egyik elemét megtartja, és annak az egress trajektóriáihoz hozzáadja a pár másik elemének az egress trajektóriáit, és a pár másik elemét törli. Utána elvégzi a vizsgálatokat az egress trajektóriákra, ami annyiban különbözik az előzőtől, hogy ebben az esetben adatokat nem kell mozgatni, csak a pár egyik elemét törölni kell.

A következő egyszerű példában az alábbi négy trajektóriát akarjuk klaszterezni az algoritmussal.

Az algoritmus először az összes trajektóriát felbontja ingress és egress részekre a kereszteződés középpontjához legközelebbi pontjánál. Utána az ingress részeket klaszterezi, aminek az eredményeként előállnak az egyedi ingressek. Végül külön-külön klaszterezi az egyes ingressekhez tartozó egresseket. Alább láthatók a két ingresshez tartozó egressek.



12. ábra: MyAlgorithm példa: a négy klaszterezendő trajektória



13. ábra: MyAlgorithm példa: a klaszterezett ingressek

14. ábra: MyAlgorithm példa: a kék ingresshez tartozó klaszterezett egressek

15. ábra: MyAlgorithm példa: a narancs ingresshez tartozó klaszterezett egressek

6.3 map-debugger

A map-debugger egy Vue.js alkalmazás, amivel egy időben több map-creator példány által létrehozott MAP üzeneteket lehet megjeleníteni.

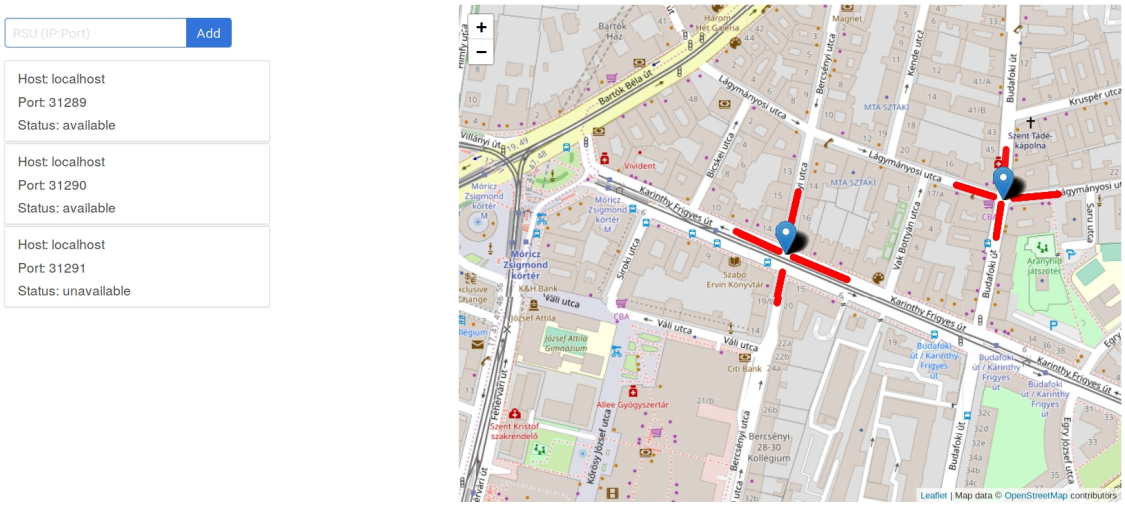
Az alkalmazás kétféle adatforrásból tud MAP adatstruktúrát megjeleníteni: JSON fájlból beolvasva vagy egy map-creator példány debug szerverétől kapott adatokból.

Az alkalmazásban meg lehet adni a map-creator példányok debug szerverének az IP-címét és portját, amikhez csatlakozni szeretnénk. Az alkalmazás monitorozza a debug

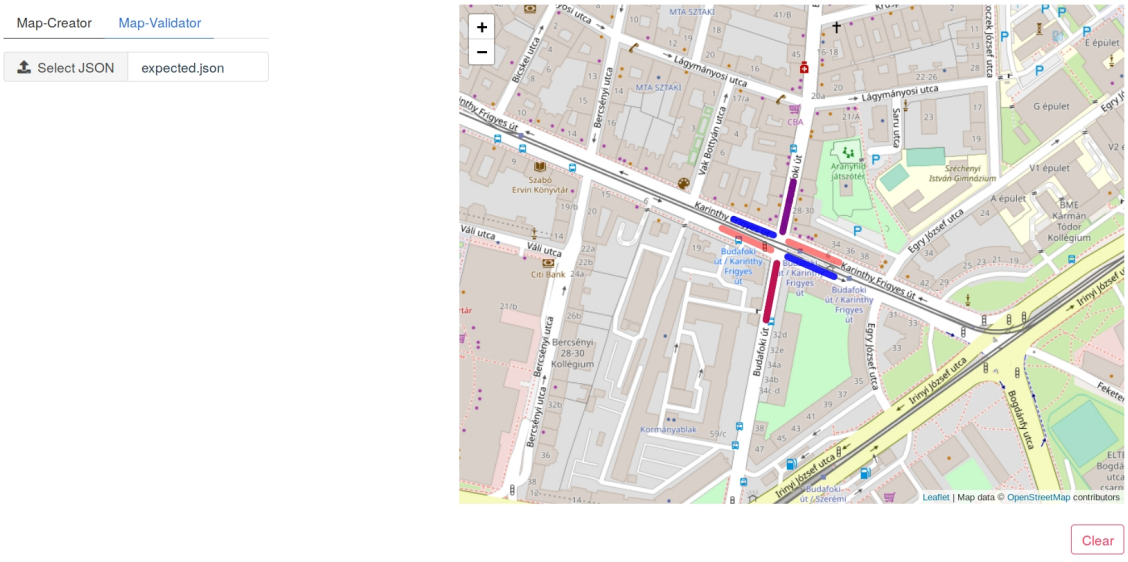
szerver elérhetőségét a ping útvonalon, és periodikusan lekéri a MAP üzenetet, amit térképen megjelenít.

Egy formon lehet beállítani a létrehozni kívánt kapcsolatot, amin egy beviteli mező van, amiben a map-creator példány debug szerverének az IP címét és a portját kell megadni kettősponttal elválasztva.

A másik lehetőség a map-validator által létrehozott JSON fájlból beolvasva megjeleníteni a MAP-ot, amit egy fájlfeltöltő dialóguson keresztül lehet megtenni.



16. ábra: A map-debugger több map-creator példánnyal kommunikálva



17. ábra: Egy map-validator JSON fájl megjelenítése map-debuggerrel

6.4 map-validator

A map-validator egy Kotlin nyelven írt konzolos alkalmazás, aminek a feladata az elkészített MAP adatstruktúra összehasonlítása egy referencia MAP adatstruktúrával, és ez alapján az elkészített MAP jóságának jellemzése különböző mérőszámokkal.

A MAP adatstruktúra egy listát tartalmaz, amik a bemenő és kimenő sávok közötti kapcsolatokat írják le. A két bemeneti MAP adatstruktúra összehasonlításához az egyes kapcsolatokat kell összehasonlítani. Tökéletes egyezés esetén elvárás, hogy a két listában pontosan ugyanannyi kapcsolat legyen leírva, és az egyik listában lévő minden kapcsolathoz legyen azzal megegyező kapcsolat a másik listában. Két kapcsolat akkor tekinthető megegyezőnek, ha a két kapcsolatot leíró bemenő és kimenő sávok megegyeznek.

A bemenő és a kimenő sávok is egy-egy lista, amik a sávot alkotó pontokból állnak. A bemenő sávoknál a lista utolsó pontja és a kimenő sávoknál pedig a lista első pontja a kereszteződéshez legközelebbi pont. Két bemenő vagy két kimenő sáv megegyezőségét a következő algoritmus dönti el:

- A több pontot tartalmazó sávból a távolságok kinyerése.
- A másik útvonal módosítása úgy, hogy a kereszteződéshez legközelebbi pontját meghagyja. Az sáv többi pontját újraszámolja úgy, hogy a legutóbbi ponthoz képest a sáv eredeti pontjai által meghatározott irányban és a másik sáv pontjai között lévő távolságban veszi fel a következő pontot.
- A két útvonalnak itt már ugyanannyi pontja van, és azok között ugyanakkora a távolság. A két útvonal megegyezőségének megállapításához egyszerre végigmegy mindkét útvonal pontjain, és az egymással párban lévő pontok között megnézi a távolságot. Ha az összes párban lévő pont között egy adott, maximálisan elfogadható, távolságnál kisebb a távolság, akkor a két útvonal megegyezőnek tekintett. Egyébként a két útvonal különbözőnek lesz kiértékelve.

7 Mérési eredmények

Az összes mérés ugyanazon a térképszeleten történt. A mérésnek két része van, ami konfigurálható. Az egyik az adathalmaz legenerálása, ami ebben a mérésben egy SUMO szimuláció felkonfigurálását jelenti. A másik pedig a klaszterezésre használt algoritmus kiválasztása, és a kiválasztott algoritmus paramétereinek a megadása.

Az autók száma és a szimuláció hossza voltak a változó értékek az adathalmaz legenerálásakor. A forgalom generálása a SUMO randomTrips.py programja segítségével történt, ami úgy volt konfigurálva, hogy az autók egyenletes időközönként lépjenek be a szimulációba. Itt az autók számában lehetnek eltérések a beállított értéktől, mivel a randomTrips.py hajlamos érvénytelen útvonalakat létrehozni, amik eldobásra kerülnek. A szimulációt legeneráló szkript úgy van beállítva, hogy a beállított autószámhoz képest a szimulációban lévő autók száma ±10%-on belül legyen.

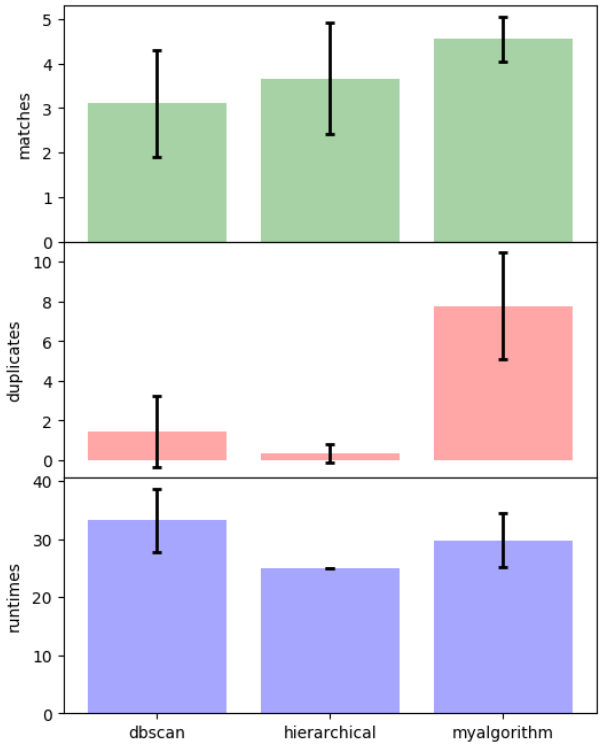
A mérés kiértékeléséhez előre el kell készíteni a térképen lévő kereszteződéshez egy referencia MAP adatstruktúrát, amivel a mérés során kapott eredményeket össze lehet hasonlítani. A mérés kiértékelését a map-validator program végzi, ami két metrikával jellemzi a kapott eredményt: *matches*, *duplicates*. A *matches* azt adja meg, hogy hány ingress-egress párt sikerült megtalálni referencia MAP-ban. A *duplicates* pedig azt mondja meg, hogy hány olyan ingress-egress pár van a mérés eredményeként kapott MAP-ban, ami egy másik benne lévő párral megegyezik. Duplikátumok például akkor jöhetnek létre, amikor az RSU frissítési fázisának a végén az adott periódusban létrehozott MAP-ból és az aktuális MAP-ból a feldolgozó egység létrehozza az új aktuális MAP-ot.

A map-validator azt nem jellemzi, hogy az eredményben van-e, és ha van, akkor hány darab, olyan ingress-egress pár, ami a referencia MAP-on nincs feltüntetve. A SUMO egy OpenStreetMap térképszelettel dolgozik, amin lehet, hogy nincsenek felvéve olyan korlátozások, amik a valóságban viszont léteznek. Ebből adódóan a szimulációból lehet, hogy más ingress-egress párok is előfordulnak, mint amik a referencia MAP-on fel vannak véve.

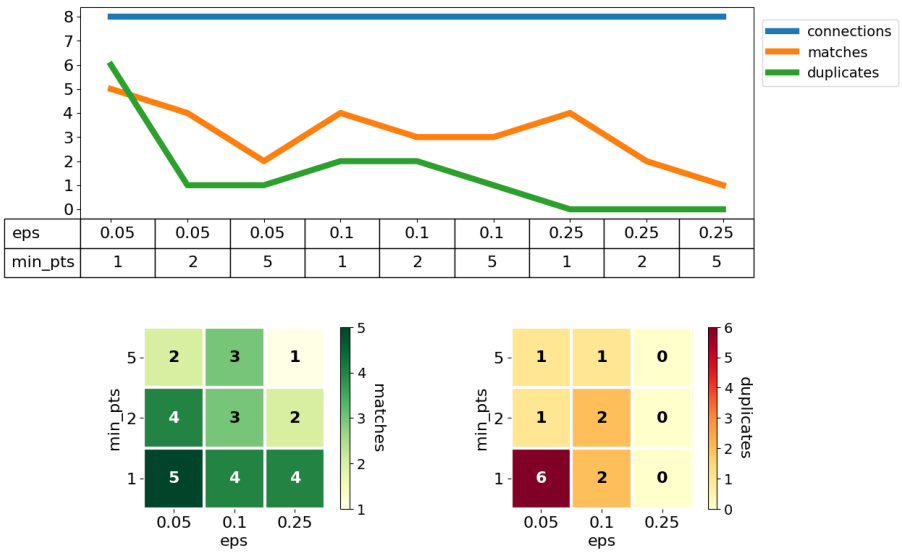
A mérésben szereplő kereszteződéshez létrehozott referencia MAP-ban nyolc darab ingress-egress pár van, amik az alábbi ábrán láthatók.

18. ábra: Az elvárt kimeneti útvonalak
(piros: ingress, kék: egress)

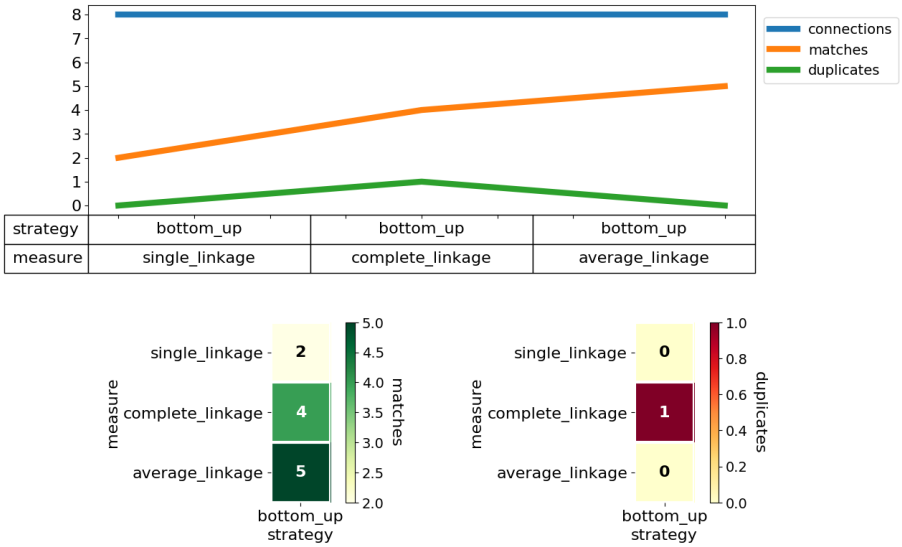
7.1 Autók száma: 100, Szimuláció ideje: 1000



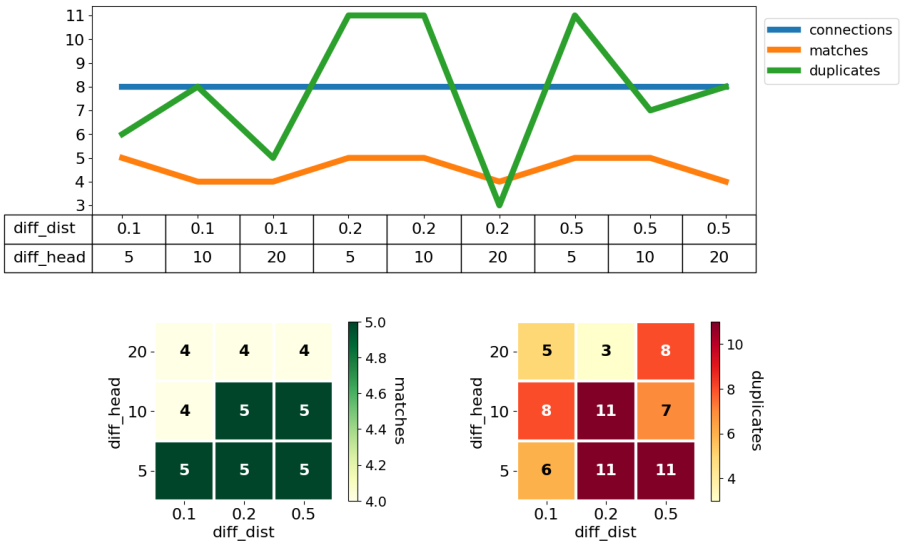
19. ábra: Algoritmusok összehasonlítása (autók: 100, szimuláció: 1000)



20. ábra: DBSCAN eredmények (autók: 100, szimuláció: 1000)



21. ábra: Hierarchical eredmények (autók: 100, szimuláció: 1000)



22. ábra: MyAlgorithm eredmények (autók: 100, szimuláció: 1000)

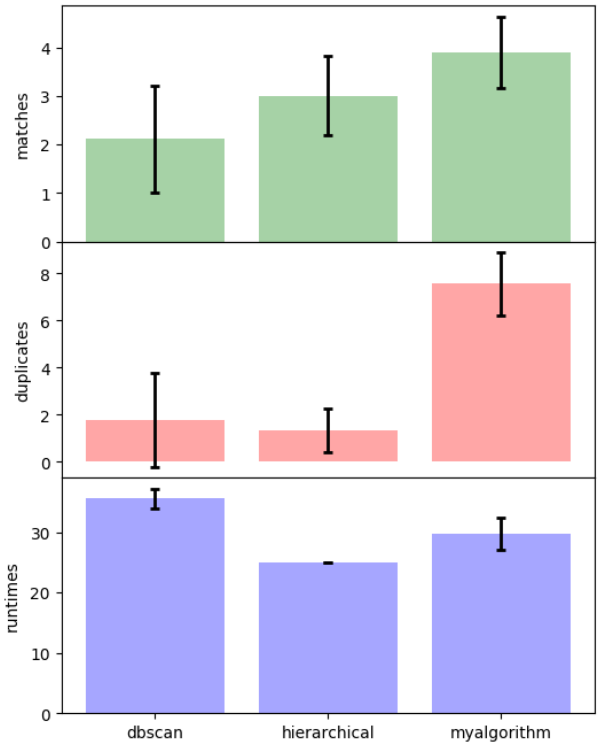
Az algoritmusok összehasonlítására szolgáló ábrán három diagram található, amik a párosítások számát, a készített duplikátumok számát és a futási időt jellemzik. Ezeken a diagramokon az adott jellemző átlagos értéke és szórása van ábrázolva.

A következő három ábrán a három algoritmus (DBSCAN, Hierarchical, MyAlgorithm) eredményei láthatók az adott mérési paraméterek mellett (autók száma, szimuláció ideje).

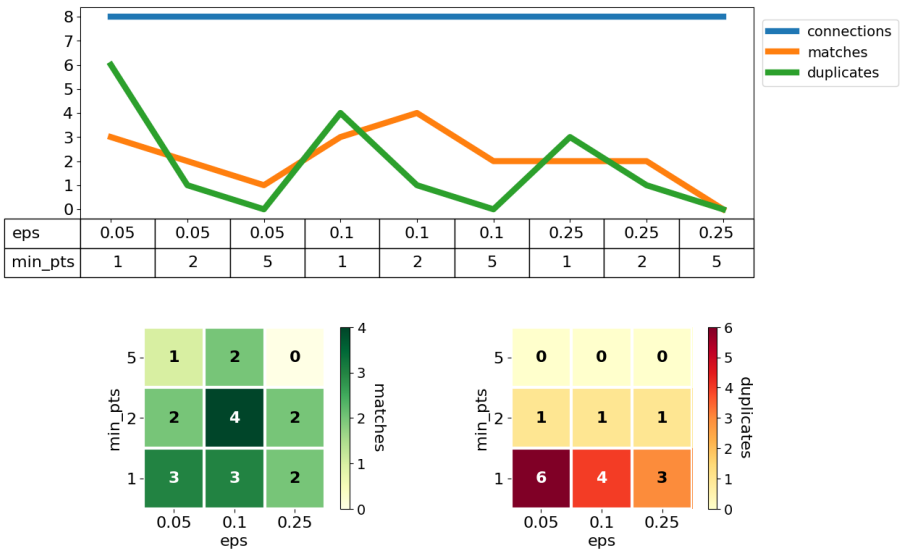
Az algoritmus eredményeit bemutató ábrán a megtalált párosítások és a készített duplikátumok száma van jellemezve az algoritmus paraméterei mellett. A felső vonaldiagramon a párosítások és a duplikátumok száma egy diagramon láthatók. Ezen a diagramon egy vízszintes vonal jelzi, hogy összesen hány párosítást (*connections*) kellett volna megtalálni. Az alsó ábrán két hőtérkép látható, amelyekről könnyebben le lehet olvasni, hogy az algoritmus adott paraméter értékek mellett hány párosítást talált és hány duplikátumot készített.

Ebben a mérésben mindhárom algoritmus 5 párosítást talált meg. A Hierarchical teljesített a legjobban a duplikátumok minimalizálásában. Az 5 párosítást úgy találta meg, hogy közben egyetlen duplikátum sem volt. A DBSCAN és a MyAlgorithm is 6-6 duplikátumot csinált, amikor megtalálta az 5 párosítást. Futási időben a Hierarchical a legjobb, mögötte a DBSCAN és a MyAlgorithm nagyon hasonló a futási idejeik átlagában és szórásában is. A DBSCAN a legtöbb paraméter mellett kevés duplikátumot készített, de a legjobb párosítás elérésekor ez nem sikerült.

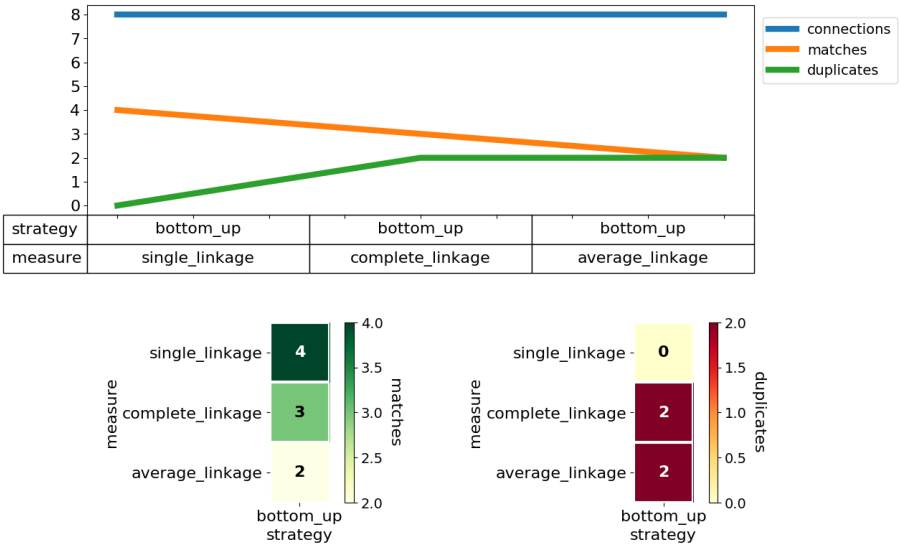
7.2 Autók száma: 100, Szimuláció ideje: 10000



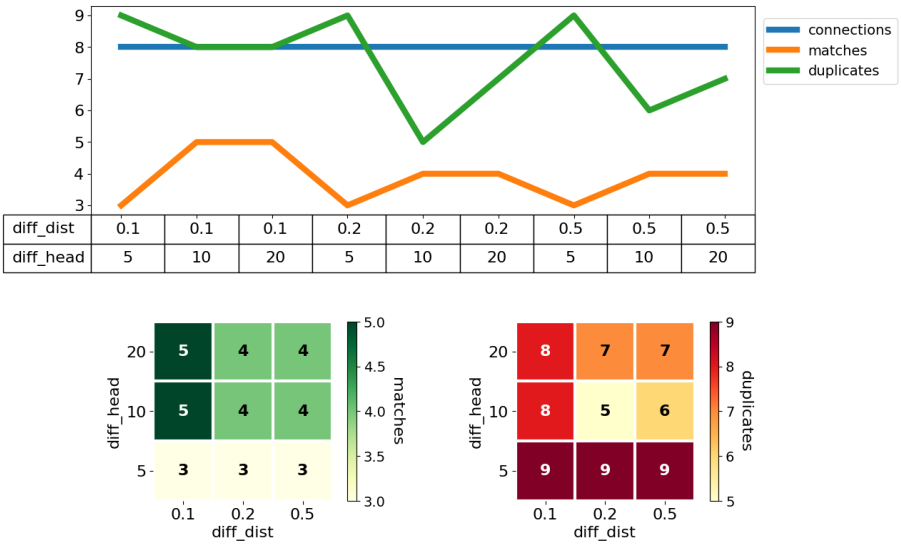
23. ábra: Algoritmusok összehasonlítása (autók: 100, szimuláció: 10000)



24. ábra: DBSCAN eredmények (autók: 100, szimuláció: 10000)



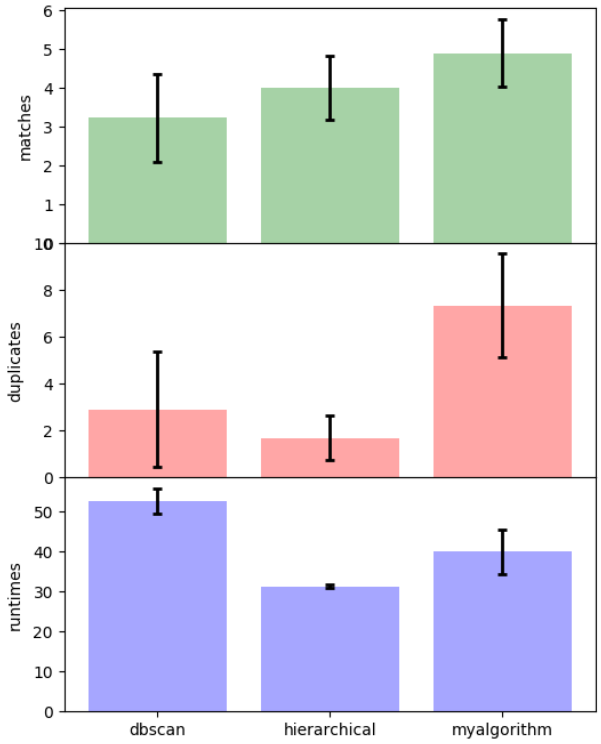
25. ábra: Hierarchical eredmények (autók: 100, szimuláció: 10000)



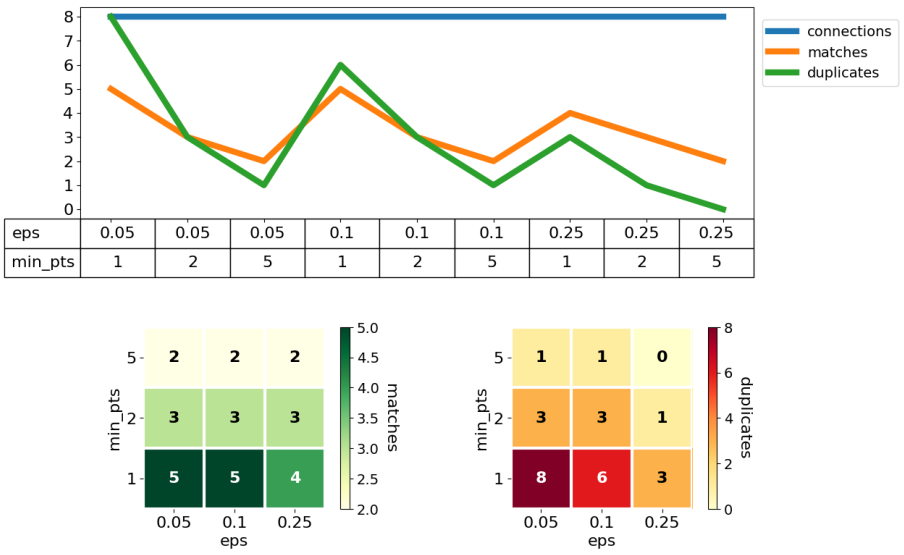
26. ábra: MyAlgorithm eredmények (autók: 100, szimuláció: 10000)

A legtöbb párosítást, 5-öt, a MyAlgorithm talált meg. A DBSCAN és a Hierarchical 4-4 párosítást találtak meg. Duplikátumok tekintetében a Hierarchical minimuma 1 volt és a szórása is alacsony. A DBSCAN minimuma 2 duplikátum volt, de a szórása is nagyobb, mint a Hierarchical esetében. A MyAlgorithm esetében a duplikátumok száma 5 és 9 között mozgott, jóval nagyobb, mint a másik kettőnél. A futási idő a Hierachical esetében a legkisebb és a szórás is itt a legkisebb. Mögötte a MyAlgorithm a második futási időben.

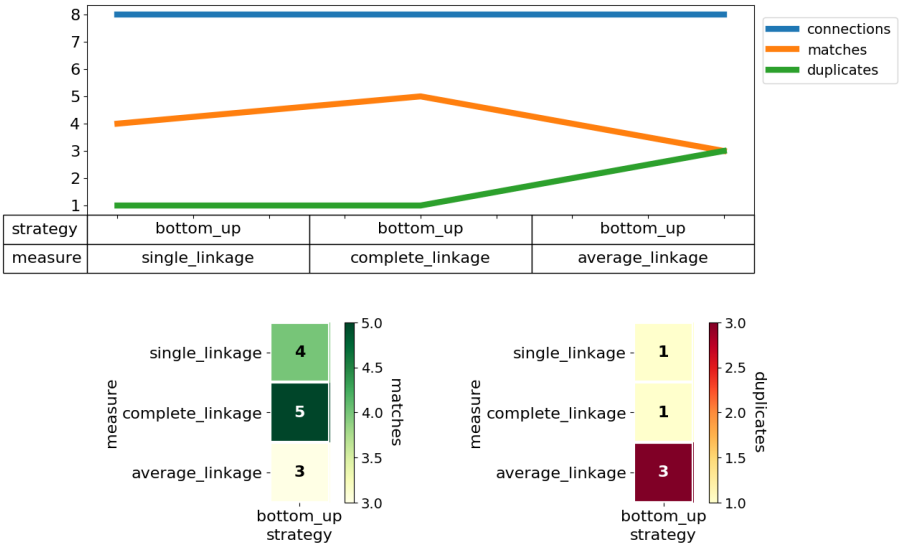
7.3 Autók száma: 100, Szimuláció ideje: 100000



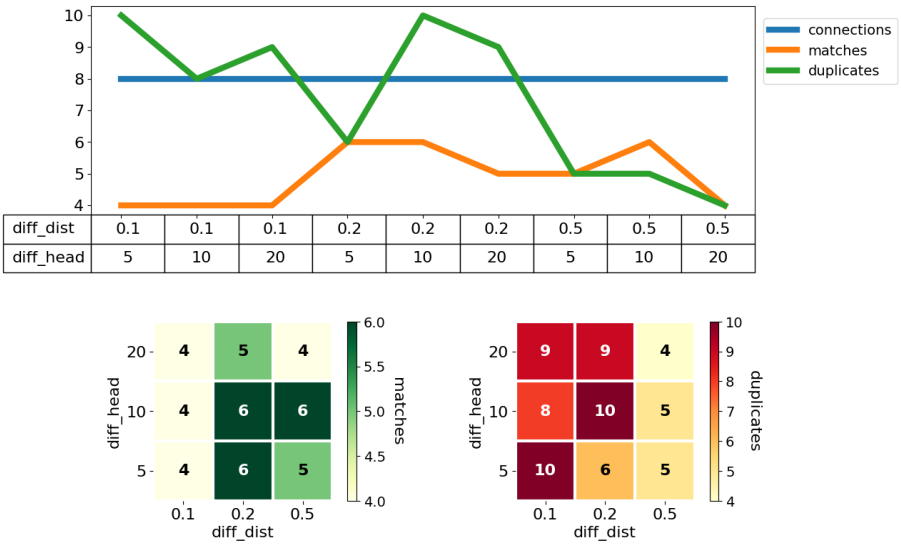
27. ábra: Algoritmusok összehasonlítása (autók: 100, szimuláció: 100000)



28. ábra: DBSCAN eredmények (autók: 100, szimuláció: 100000)



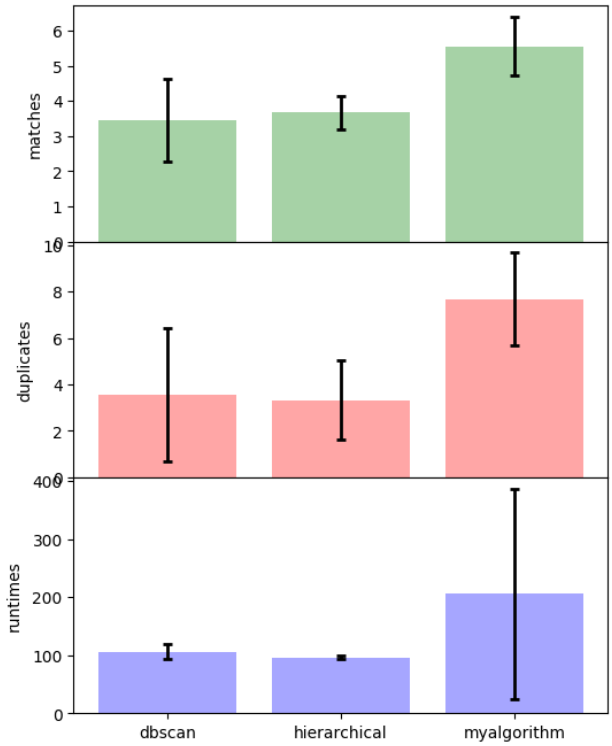
29. ábra: Hierarchical eredmények (autók: 100, szimuláció: 100000)



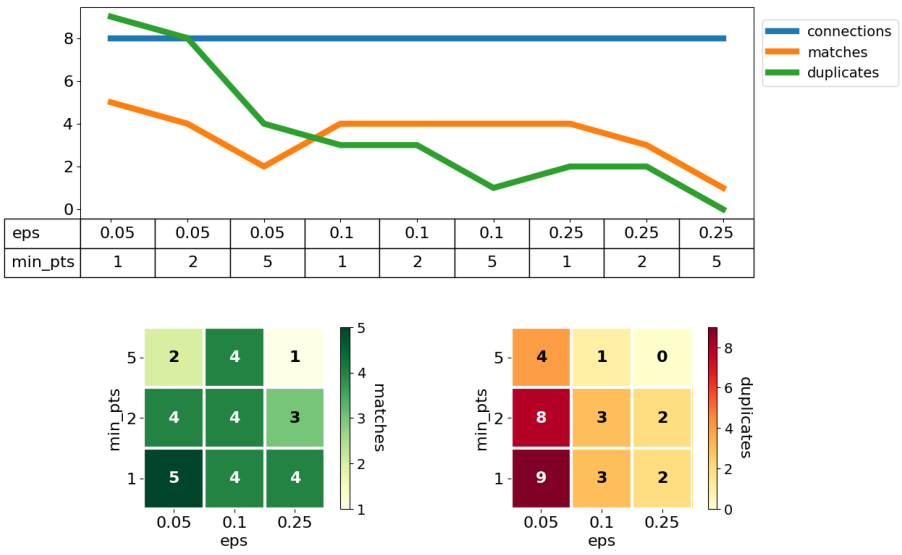
30. ábra: MyAlgorithm eredmények (autók: 100, szimuláció: 100000)

A MyAlgorithm 6, a DBSCAN és a Hierarchical pedig 5-5 párosítást talált meg. A MyAlgorithm esetében a duplikátumok minimuma 5, és ebben az esetben is 6 párosítást talált. A DBSCAN 6 duplikátumot készített a legjobb párosítás megtalálásakor. A minimuma 0 volt, de ekkor csak 2 párosítást talált meg. A Hierarchical, amikor 5 párosítást talált, mindössze 1 duplikátumot készített, és ez is volt a minimuma. Futási időben a Hierarchical a legjobb, a második a MyAlgorithm és a DBSCAN a leglassabb.

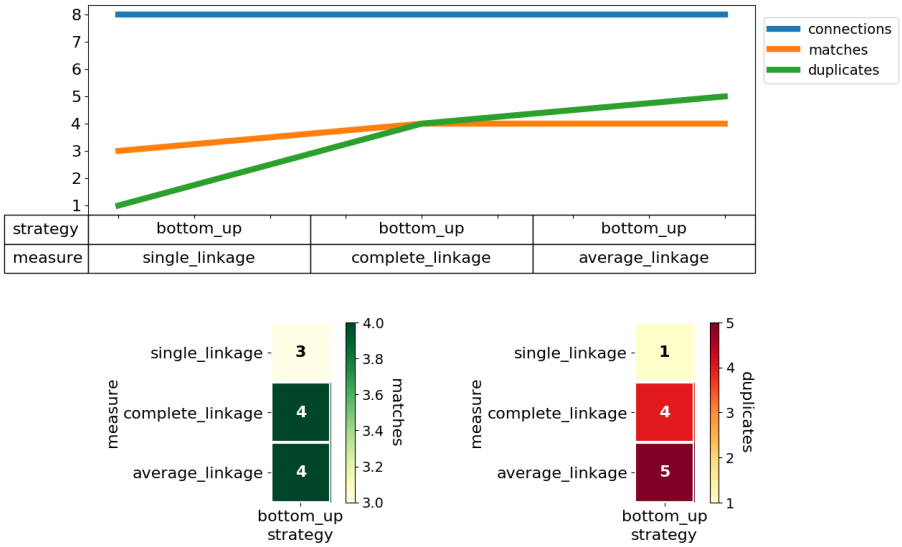
7.4 Autók száma: 250, Szimuláció ideje: 1000



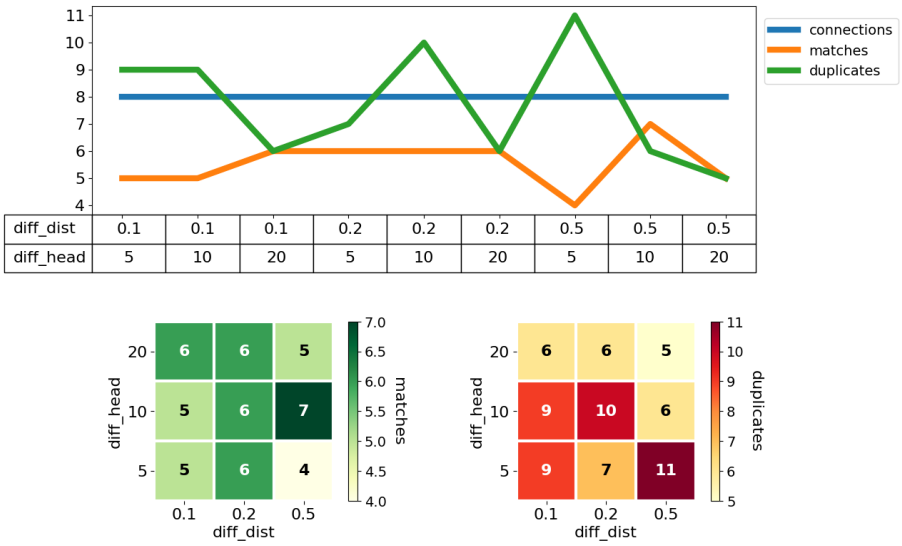
31. ábra: Algoritmusok összehasonlítása (autók: 250, szimuláció: 1000)



32. ábra: DBSCAN eredmények (autók: 250, szimuláció: 1000)



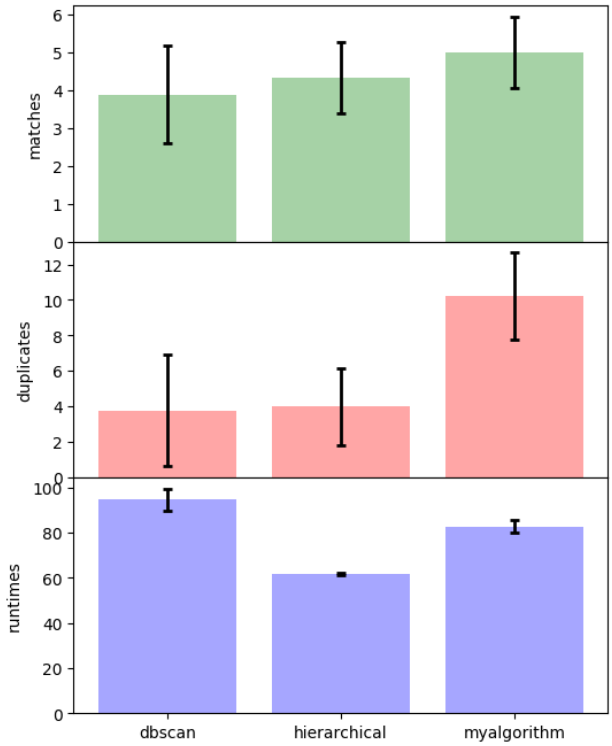
33. ábra: Hierarchical eredmények (autók: 250, szimuláció: 1000)



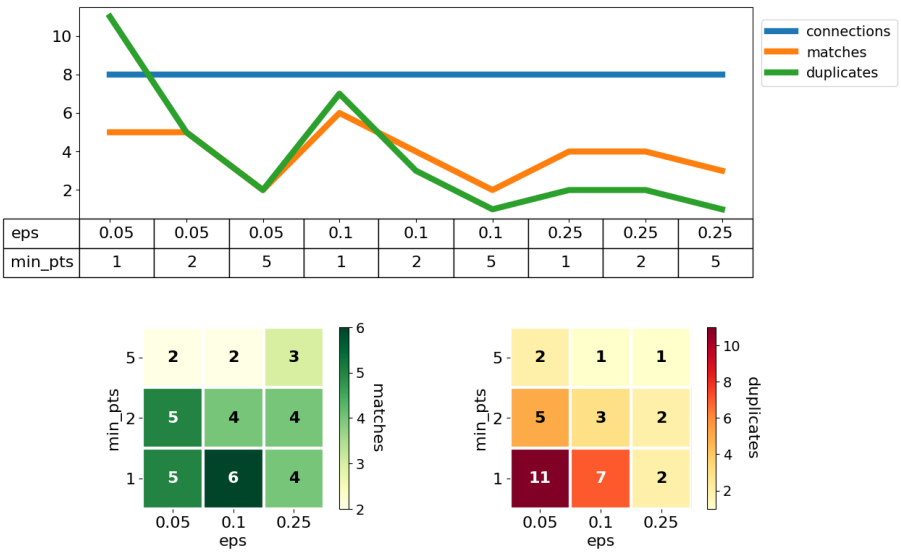
34. ábra: MyAlgorithm eredmények (autók: 250, szimuláció: 1000)

A MyAlgorithm találta meg a legtöbb párosítást, pontosan 7-et. A DBSCAN 5, a Hierarchical pedig 4 párosítást talált. A duplikátumok tekintetében a DBSCAN és a Hierarchical nagyon hasonló átlagot hoztak, de a DBSCAN esetében a szórás nagyobb, és a legjobb párosításaik esetében is a Hierarchical kevesebb duplikátumot készített. A legjobb párosításuk megtalálásakor a DBSCAN 9, a Hierarchical pedig 5 duplikátumot készített. A MyAlgorithm átlagosan több duplikátumot csinált mindkettőnél, de a legjobb párosítása esetében „csak” 6 duplikátumot készített. Futási időben a DBSCAN és a Hiearchical átlaga és szórása is hasonló, míg a MyAlgorithm átlagban és szórásban is sokkal rosszabban teljesített náluk.

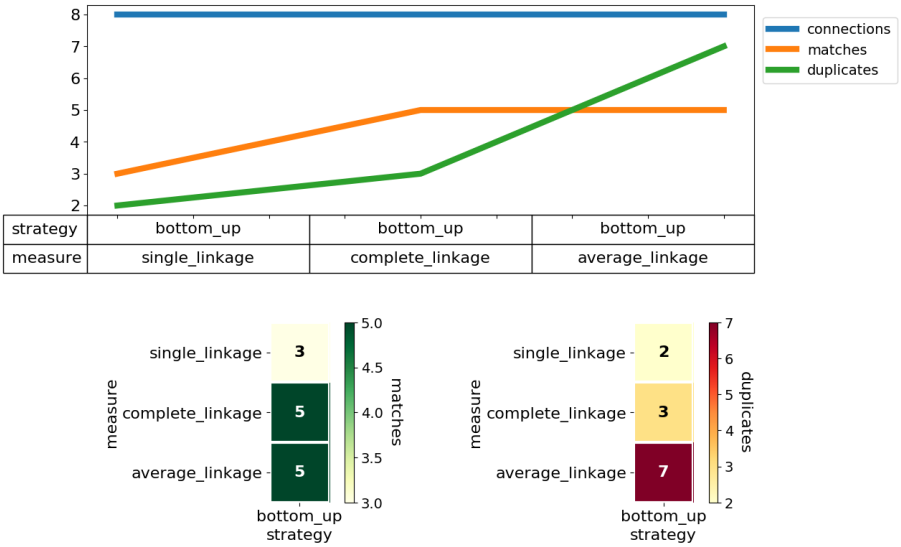
7.5 Autók száma: 250, Szimuláció ideje: 10000



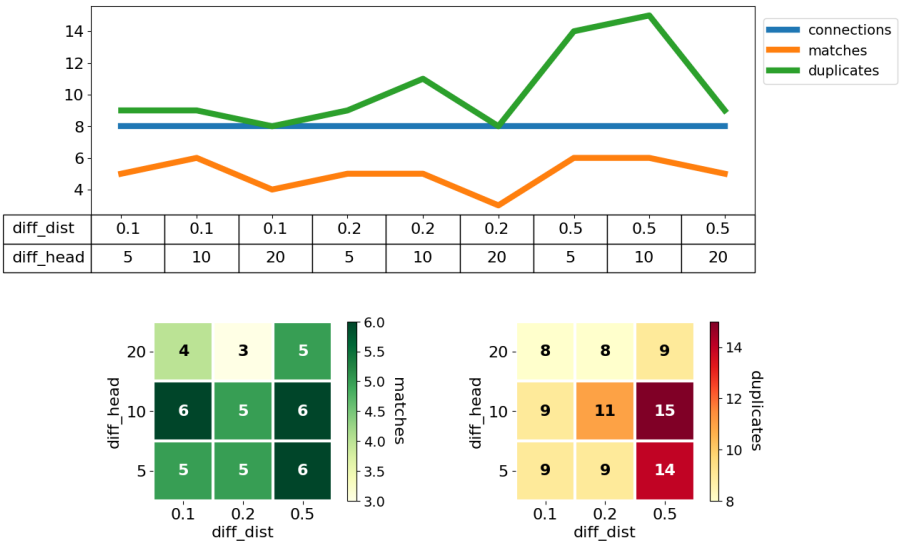
35. ábra: Algoritmusok összehasonlítása (autók: 250, szimuláció: 10000)



36. ábra: DBSCAN eredmények (autók: 250, szimuláció: 10000)



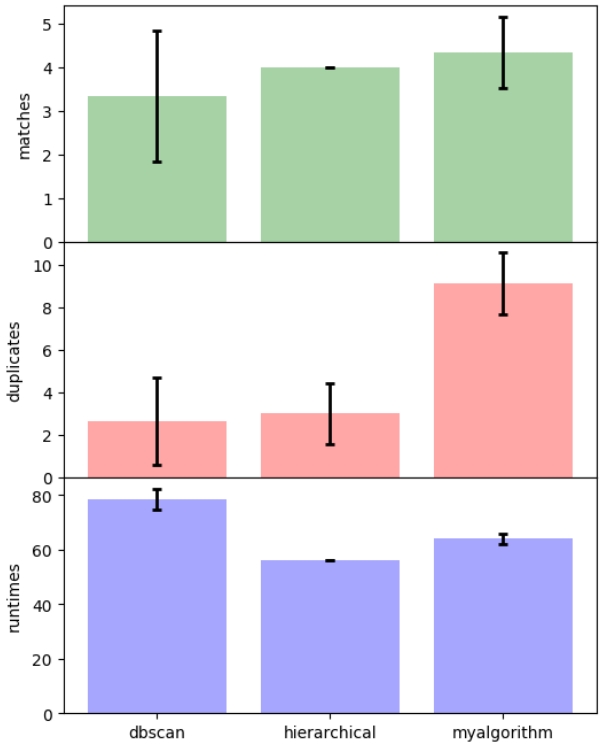
37. ábra: Hierarchical eredmények (autók: 250, szimuláció: 10000)



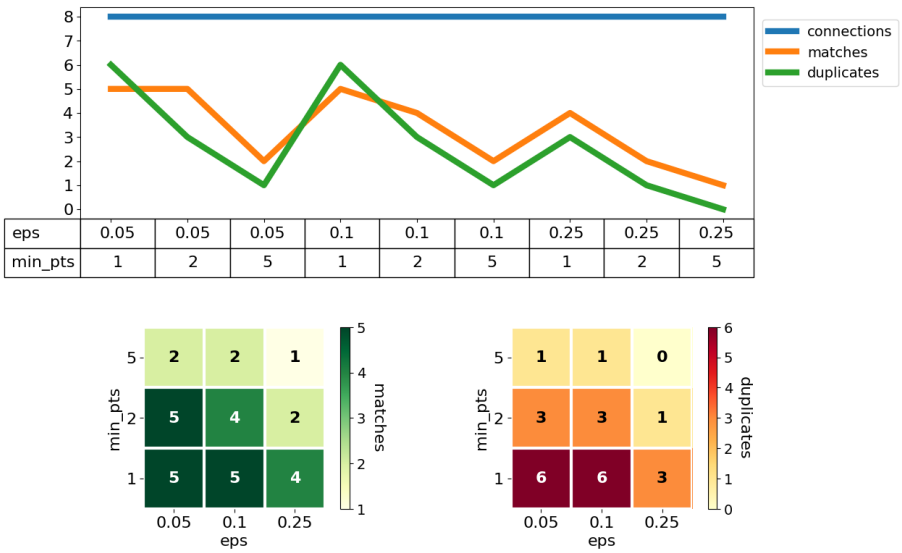
38. ábra: MyAlgorithm eredmények (autók: 250, szimuláció: 10000)

A DBSCAN és a MyAlgorithm 6-6 párosítást találtak meg, a Hierarchical pedig 5-öt. A DBSCAN minimum 1 duplikátumot készített, de a legjobb párosításakor 7-et. A MyAlgorithm esetében a minimum 8 és a legjobb párosításakor pedig 9. A Hierarchical 3 duplikátumot talált az 5 párosítás megtalálásakor. Futási időben a Hierarchical a legjobb, mögötte a MyAlgorithm, és a leglassabb a DBSCAN.

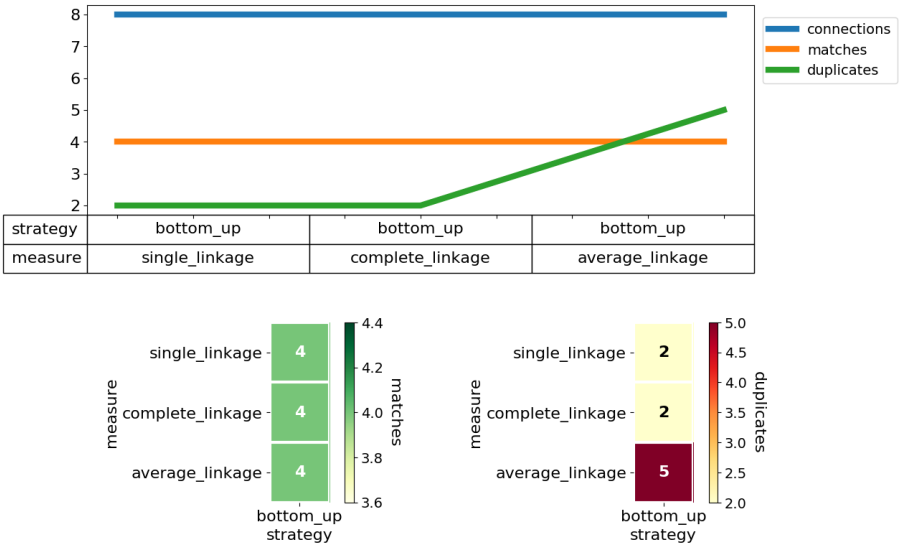
7.6 Autók száma: 250, Szimuláció ideje: 100000



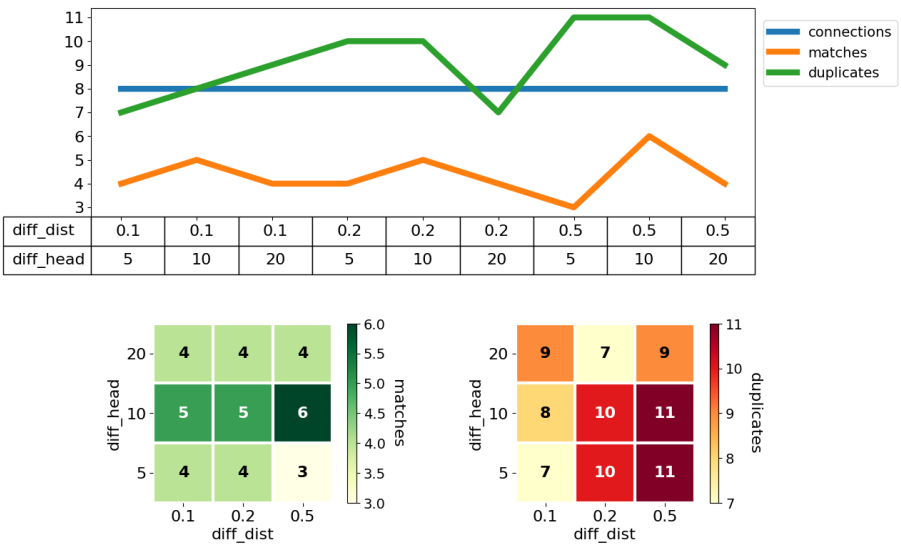
39. ábra: Algoritmusok összehasonlítása (autók: 250, szimuláció: 100000)



40. ábra: DBSCAN eredmények (autók: 250, szimuláció: 100000)



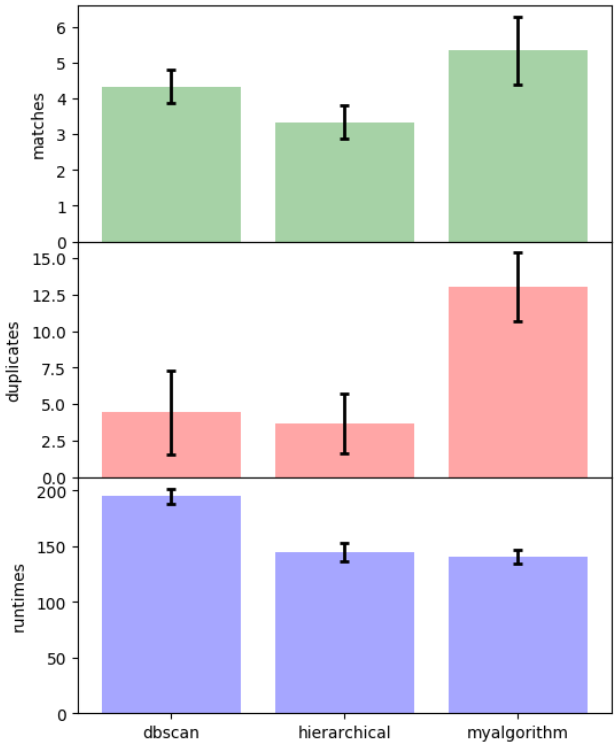
41. ábra: Hierarchical eredmények (autók: 250, szimuláció: 100000)



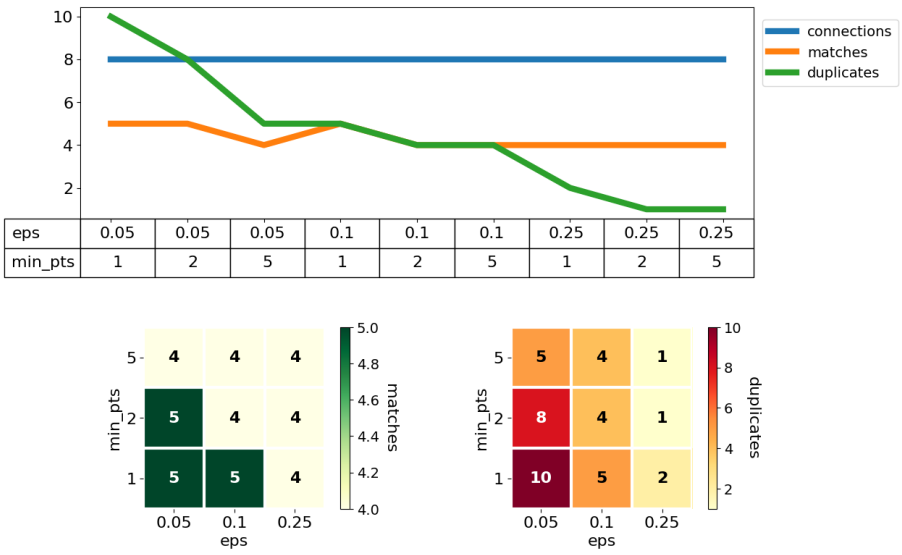
42. ábra: MyAlgorithm eredmények (autók: 250, szimuláció: 100000)

A MyAlgorithm 6, a DBSCAN 5, a Hierarchical pedig 4 párosítást találtak meg maximálisan. A duplikátumok száma ezekben az esetekben 11, 3 és 2 voltak. A DBSCAN és a Hierarchical is átlagosan sokkal kevesebb duplikátumot készítettek, mint a MyAlgorithm. A futási idő átlagában Hierarchical, MyAlgorithm, DBSCAN sorrend van, és mindegyiknél alacsony a szórás.

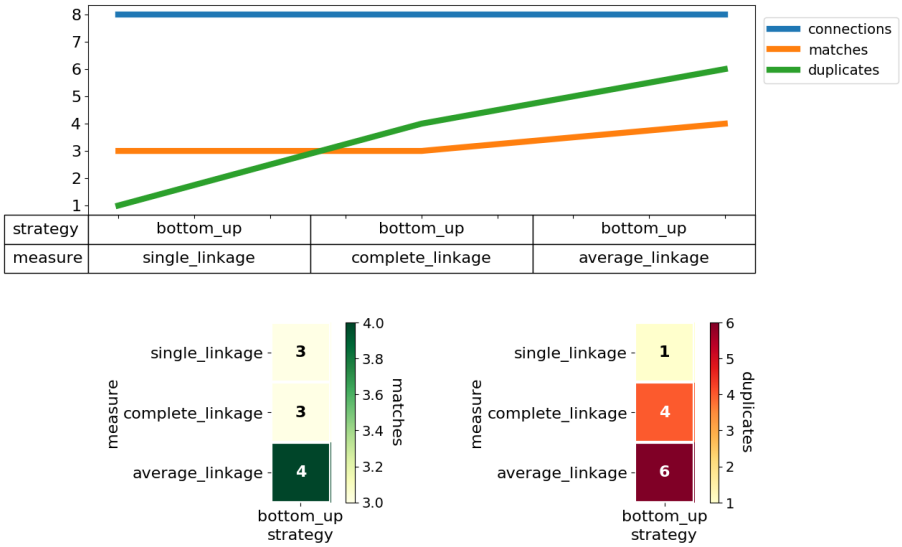
7.7 Autók száma: 500, Szimuláció ideje: 1000



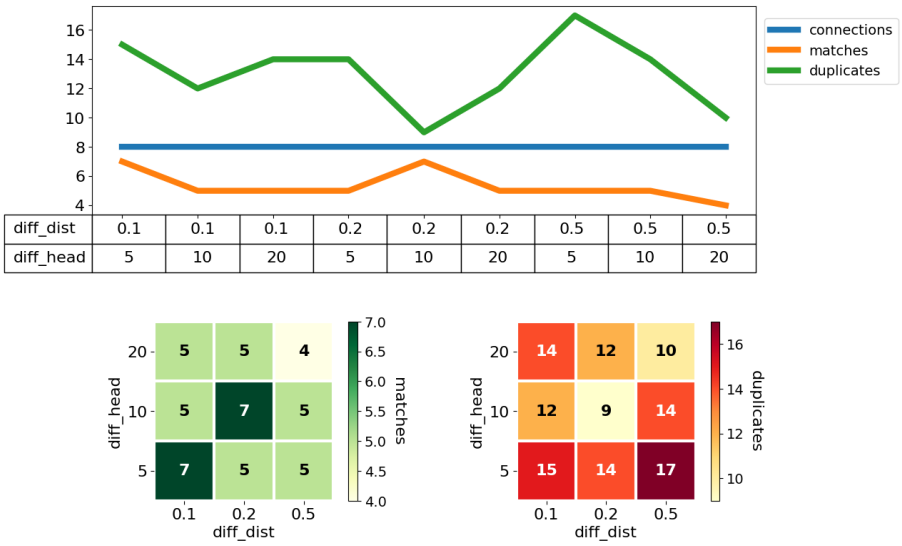
43. ábra: Algoritmusok összehasonlítása (autók: 500, szimuláció: 1000)



44. ábra: DBSCAN eredmények (autók: 500, szimuláció: 1000)



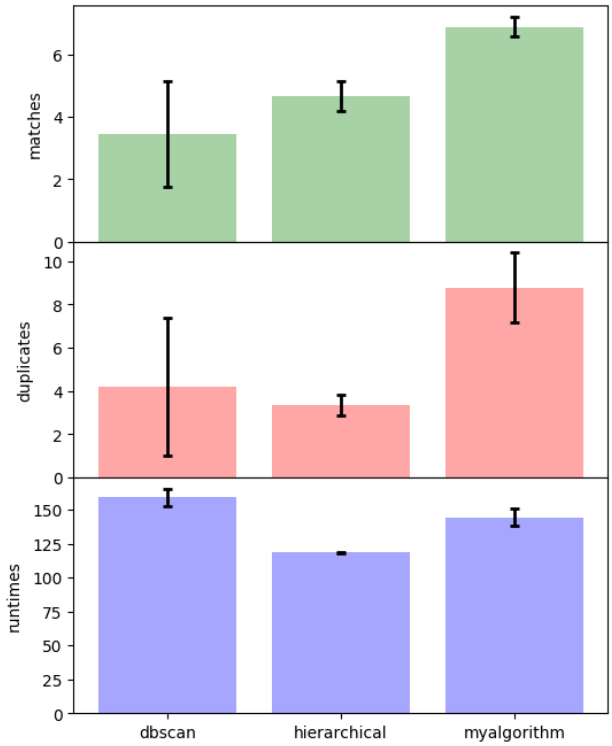
45. ábra: Hierarchical eredmények (autók: 500, szimuláció: 1000)



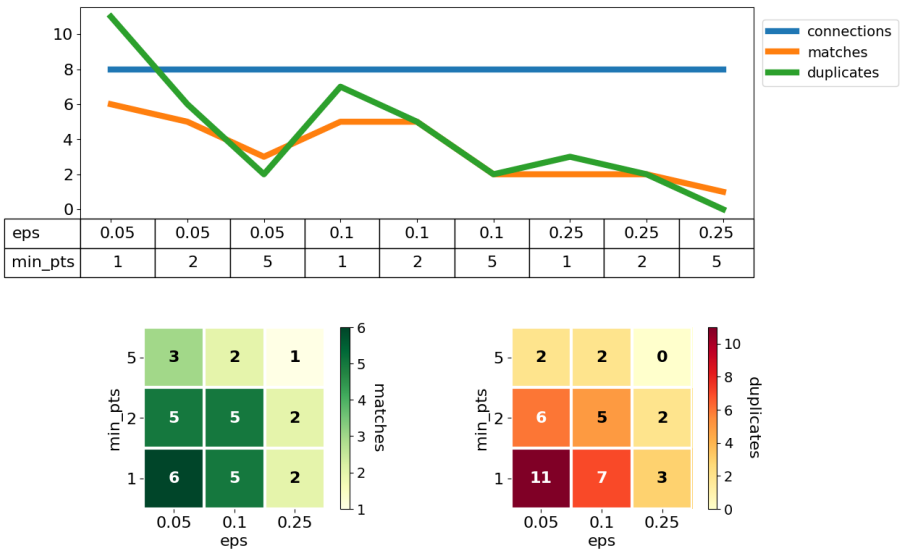
46. ábra: MyAlgorithm eredmények (autók: 500, szimuláció: 1000)

A MyAlgorithm eredményezte a legtöbb párosítást, maximálisan 7-et. A DBSCAN több paraméter mellett is 5 párosítást talált meg. A Hierarchical esetében a maximális párosítás 4 volt. A MyAlgorithm 9, a DBSCAN 5, a Hierarchical pedig 6 duplikátumot készített a legjobb párosításuk megtalálásakor. Az átlagos duplikátumok számának tekintetében a DBSCAN és a Hierarchical hasonló értékeket ad. A MyAlgorithm átlagosan több, mint kétszer annyi duplikátumot készített a másik kettőnél. A futási időben a MyAlgorithm kicsivel jobb volt, mint a Hierarchical, és a DBSCAN volt a leglassabb.

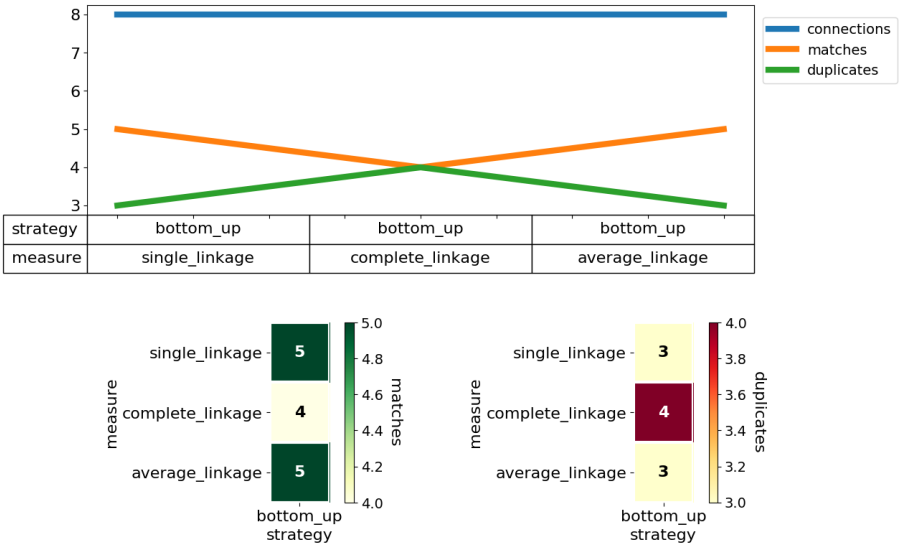
7.8 Autók száma: 500, Szimuláció ideje: 10000



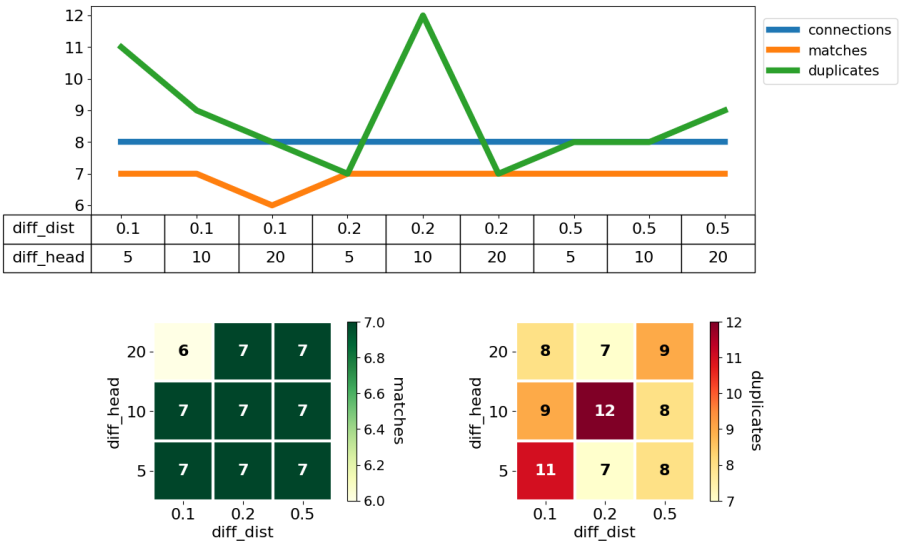
47. ábra: Algoritmusok összehasonlítása (autók: 500, szimuláció: 10000)



48. ábra: DBSCAN eredmények (autók: 500, szimuláció: 10000)



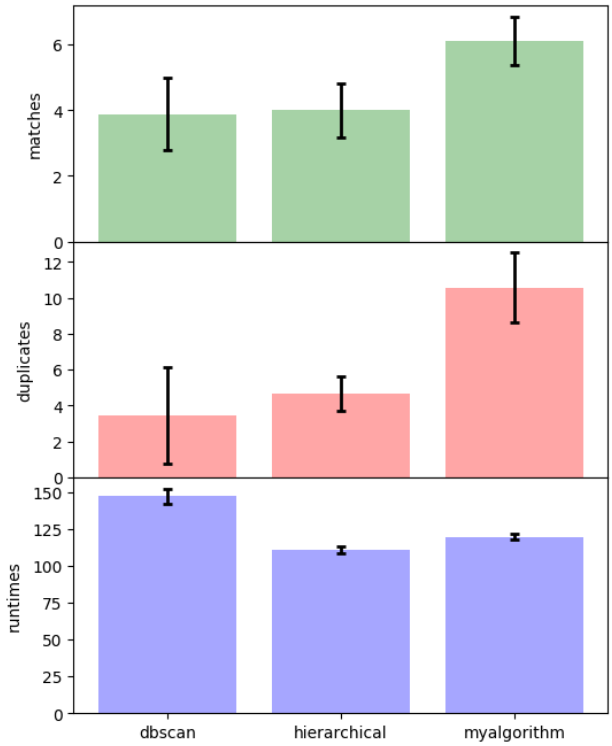
49. ábra: Hierarchical eredmények (autók: 500, szimuláció: 10000)



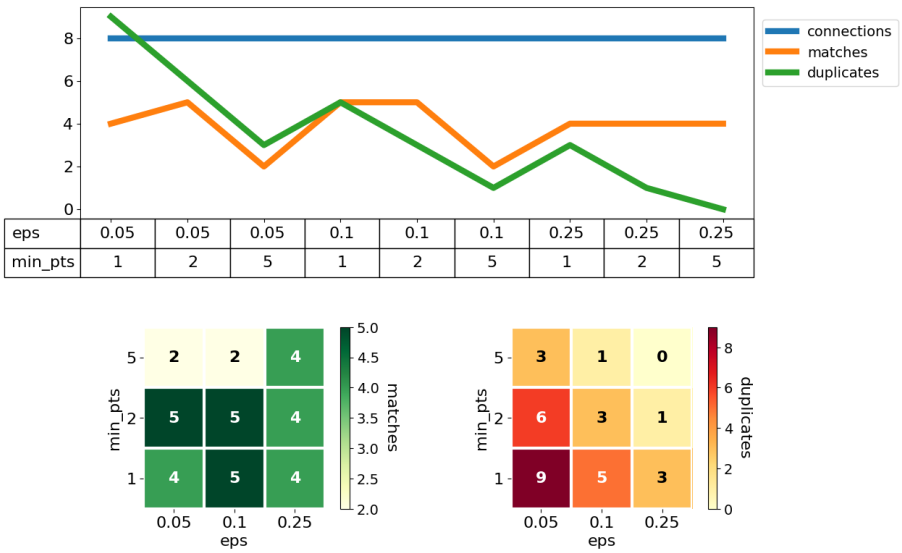
50. ábra: MyAlgorithm eredmények (autók: 500, szimuláció: 10000)

A MyAlgorithm találta meg a legtöbb párosítást. Egy kivételével minden paraméter mellett 7-et talált meg. A DBSCAN legjobb eredménye 6, a Hierarchical esetében pedig 5 párosítás volt. A duplikátumok tekintetében a Hierarchical a legjobb, és ennél van a legkisebb szórás is. A DBSCAN a duplikátumok átlagának tekintetében jobb, mint a MyAlgorithm, de a legjobb párosításaik esetén a MyAlgorithm 7, a DBSCAN pedig 11 duplikátumot készített. Futási időben a Hierarchical a legjobb, mögötte a MyAlgorithm, és a leglassabb pedig a DBSCAN. A futási idő szórása mindegyiknél alacsony volt.

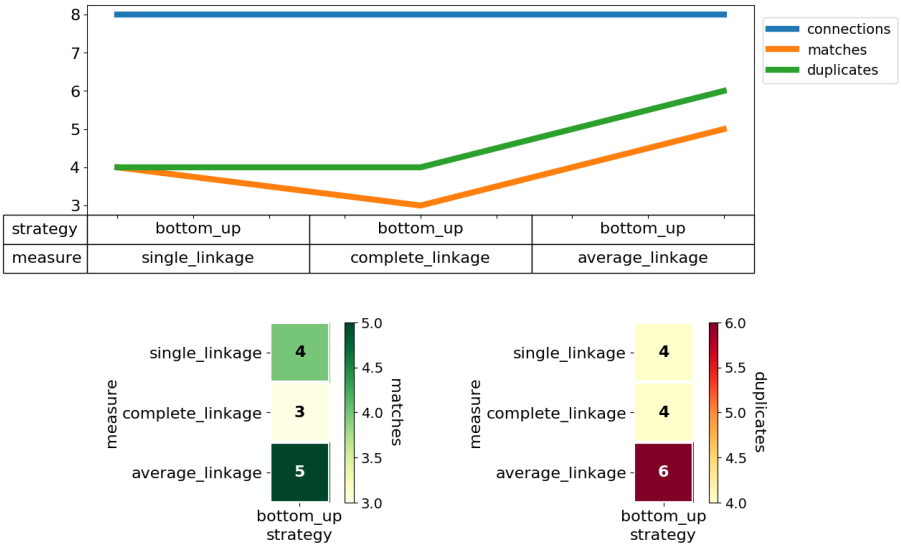
7.9 Autók száma: 500, Szimuláció ideje: 100000



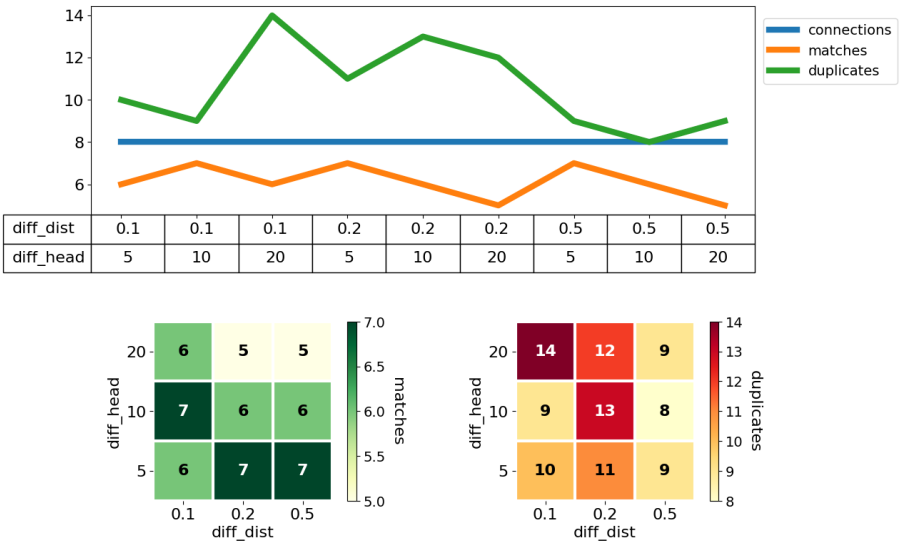
51. ábra: Algoritmusok összehasonlítása (autók: 500, szimuláció: 100000)



52. ábra: DBSCAN eredmények (autók: 500, szimuláció: 100000)



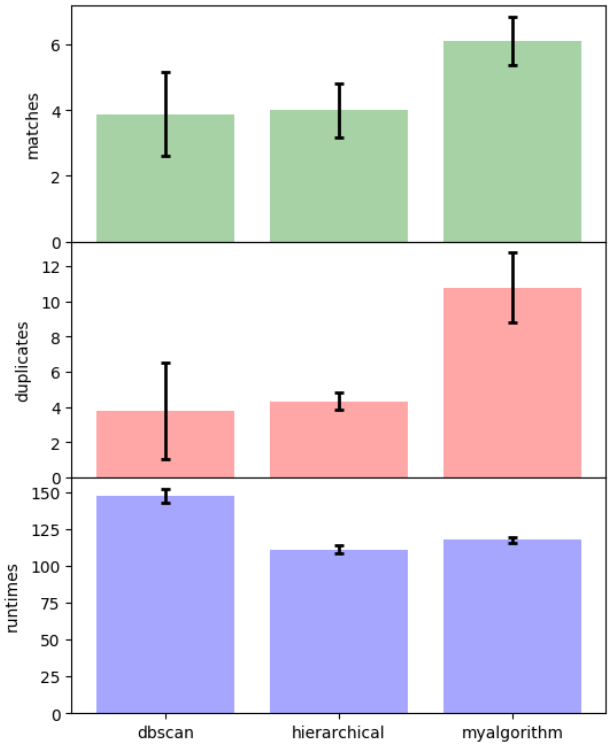
53. ábra: Hierarchical eredmények (autók: 500, szimuláció: 100000)



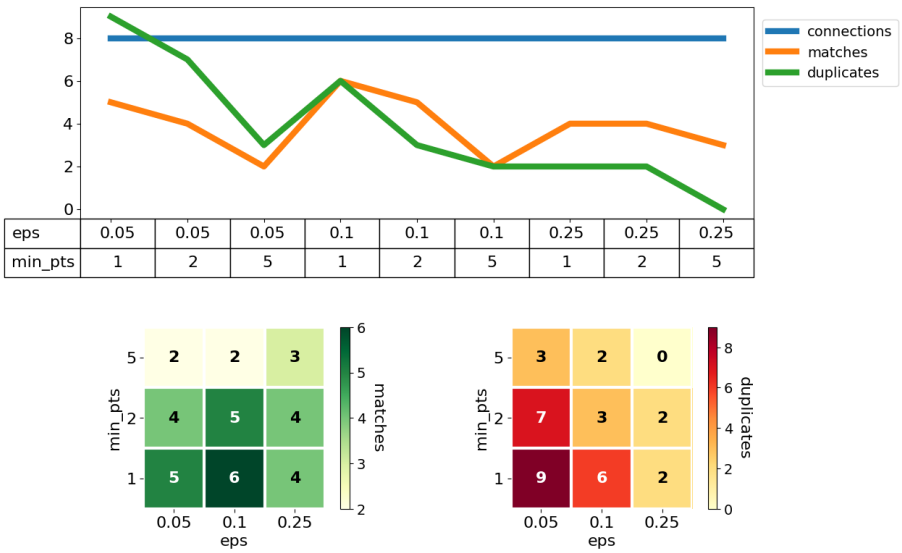
54. ábra: MyAlgorithm eredmények (autók: 500, szimuláció: 100000)

A legtöbb párosítást a MyAlgorithm találta, 7-et. Mögötte 5-5 párosítással van a DBSCAN és a Hierarchical. A duplikátumok száma a legjobb párosításuk elérésekor a MyAlgorithm esetében 9, a DBSCAN esetében 3, a Hierarchical esetében pedig 6 volt. A duplikátumok átlagos értékének tekintetében a DBSCAN egy kicsit jobb a Hierarchical eredményénél, és mindkettő sokkal jobb, mint a MyAlgorithm eredménye. A duplikátumok szórása a Hierarchical esetében a legkisebb. A futási idő tekintetében mindegyiknek alacsony a szórása. A futási idő átlagában a Hierarchical a legjobb, mögötte a MyAlgorithm, és a leglassabb a DBSCAN.

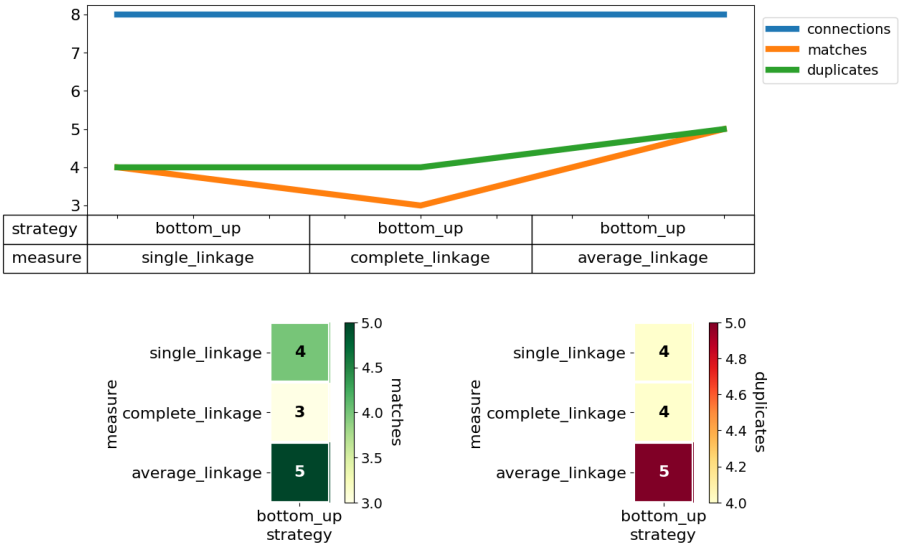
7.10 Autók száma: 1000, Szimuláció ideje: 1000



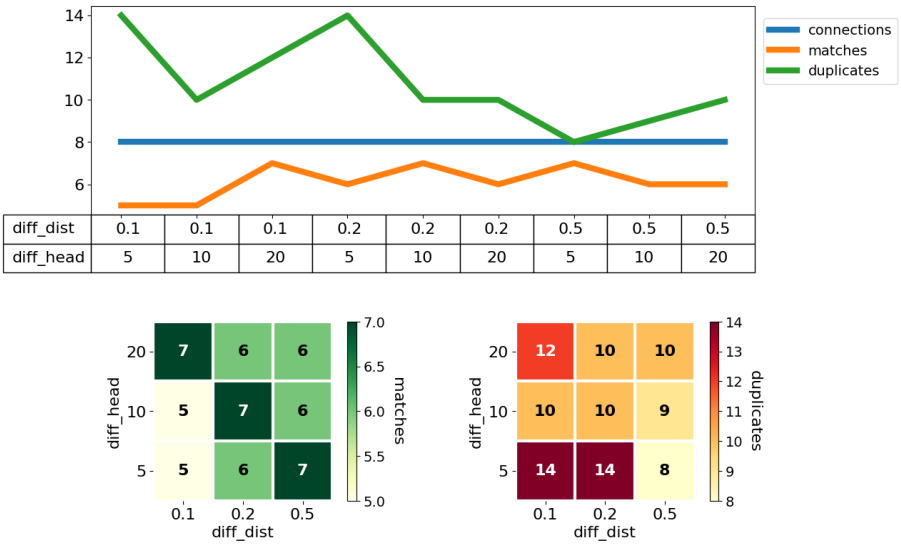
55. ábra: Algoritmusok összehasonlítása (autók: 1000, szimuláció: 1000)



56. ábra: DBSCAN eredmények (autók: 1000, szimuláció: 1000)



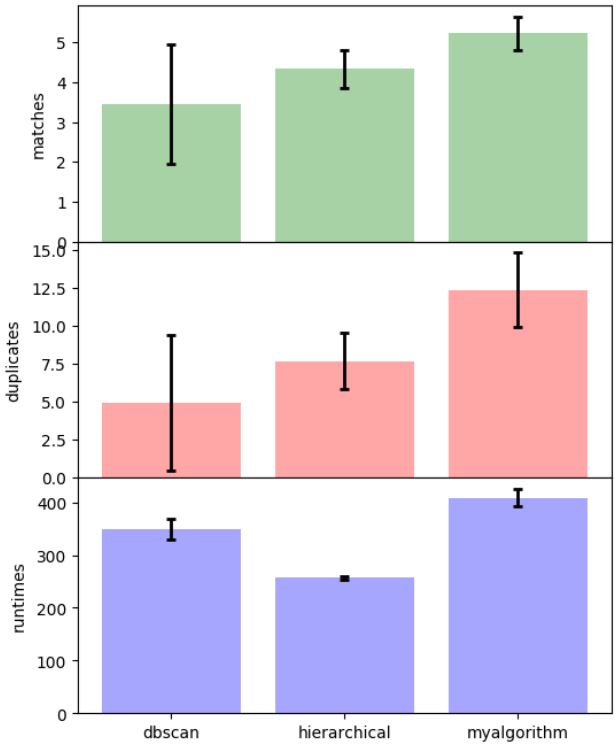
57. ábra: Hierarchical eredmények (autók: 1000, szimuláció: 1000)



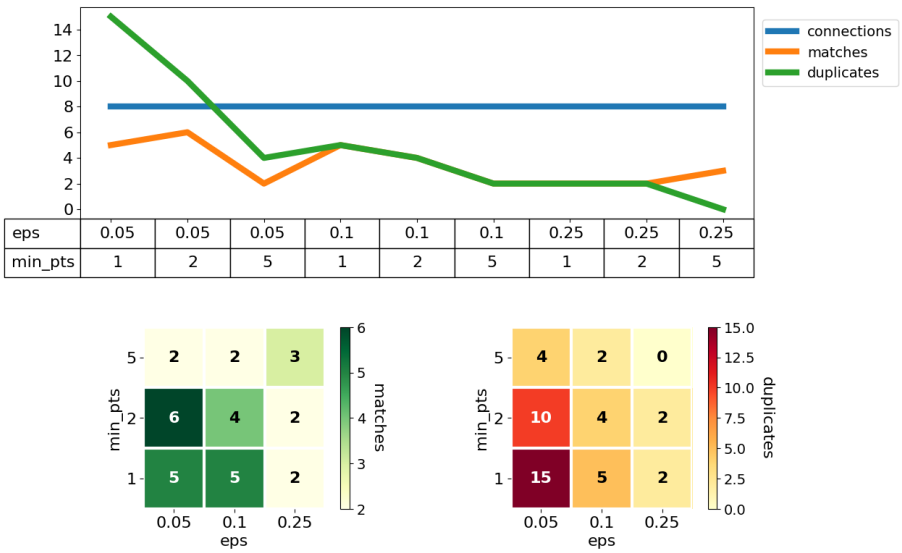
58. ábra: MyAlgorithm eredmények (autók: 1000, szimuláció: 1000)

A megtalált párosítások maximumában a MyAlgorithm volt a legjobb 7-tel, mögötte a DBSCAN 6-tal, és a legrosszabbul a Hierarchical teljesített 5-tel. A MyAlgorithm 8, a DBSCAN 6, a Hierarchical pedig 5 duplikátumot készített a maximális párosításuk elérésekor. A duplikátumok átlagának tekintetében a DBSCAN és a Hierarchical jóval alacsonyabb értéket eredményezett, mint a MyAlgorithm, de a szórása tekintetében csak a Hierarchical teljesített jól. Futási időben a Hierarchical a legjobb, mögötte a MyAlgorithm, és a DBSCAN a leglassabb. A futási idő szórása mindegyiknél alacsony volt.

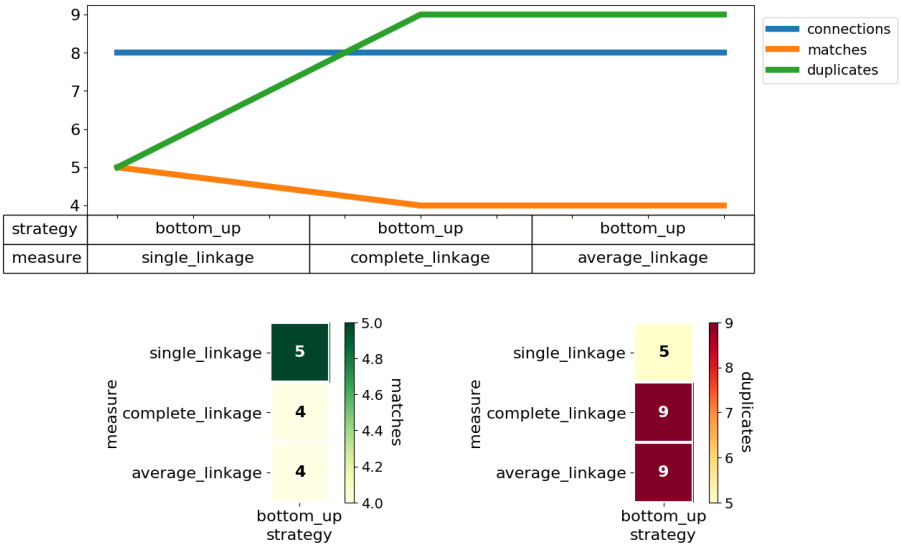
7.11 Autók száma: 1000, Szimuláció ideje: 10000



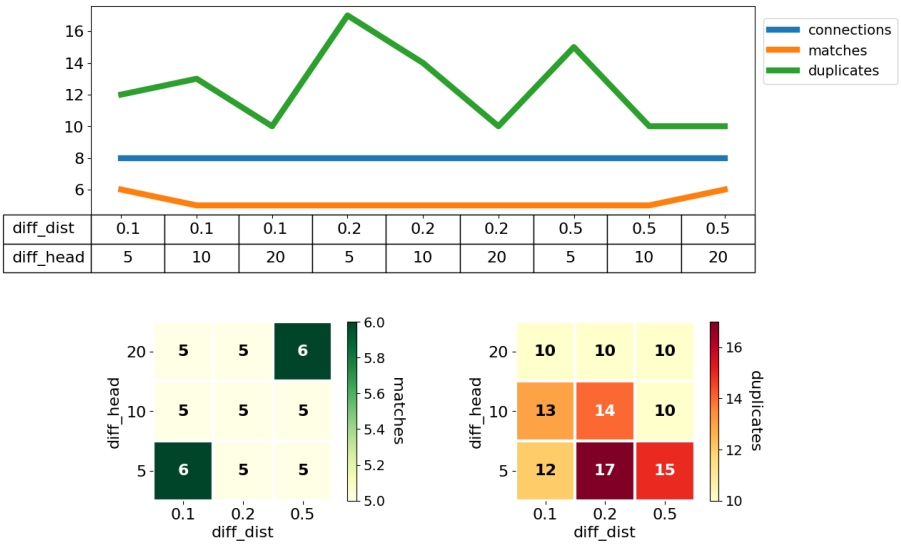
59. ábra: Algoritmusok összehasonlítása (autók: 1000, szimuláció: 10000)



60. ábra: DBSCAN eredmények (autók: 1000, szimuláció: 10000)



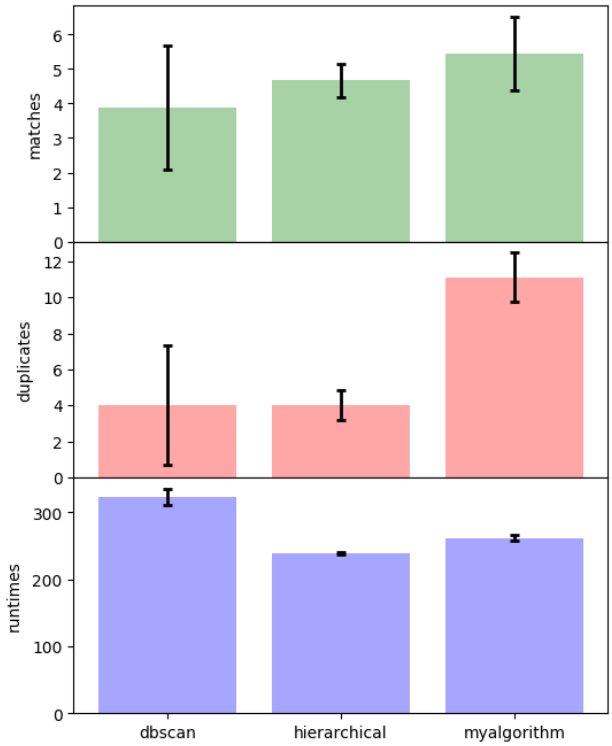
61. ábra: Hierarchical eredmények (autók: 1000, szimuláció: 10000)



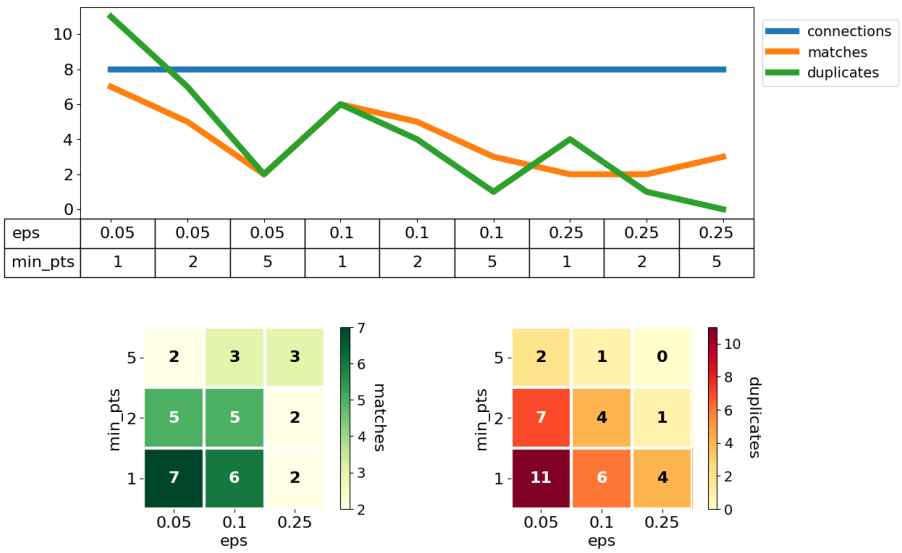
62. ábra: MyAlgorithm eredmények (autók: 1000, szimuláció: 10000)

A DBSCAN és a MyAlgorithm 6-6, a Hierarchical pedig maximálisan 5 párosítást talált meg. A megtalált párosítások szórása a MyAlgorithm és a Hierarchical esetében jóval alacsonyabb volt, mint amit a DBSCAN elért. A legtöbb párosítás elérésekor a DBSCAN és a MyAlgorithm 10-10, a Hierarchical 5 duplikátumot készített. A készített duplikátumok átlaga a DBSCAN esetében volt a legjobb, de viszont a szórása is annál volt a legnagyobb. A futási idő átlaga a Hierarchical esetében a legkisebb, mögötte a DBSCAN a második, és a MyAlgorithm volt a leglassabb. A futási idő szórása a Hierarchical esetében a legalacsonyabb, a DBSCAN és a MyAlgorithm esetében kicsit rosszabb volt annál.

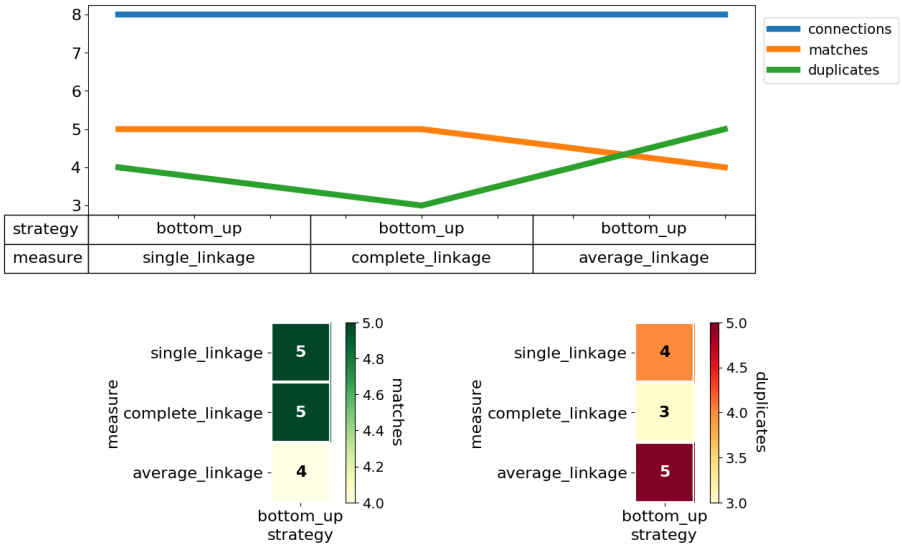
7.12 Autók száma: 1000, Szimuláció ideje: 100000



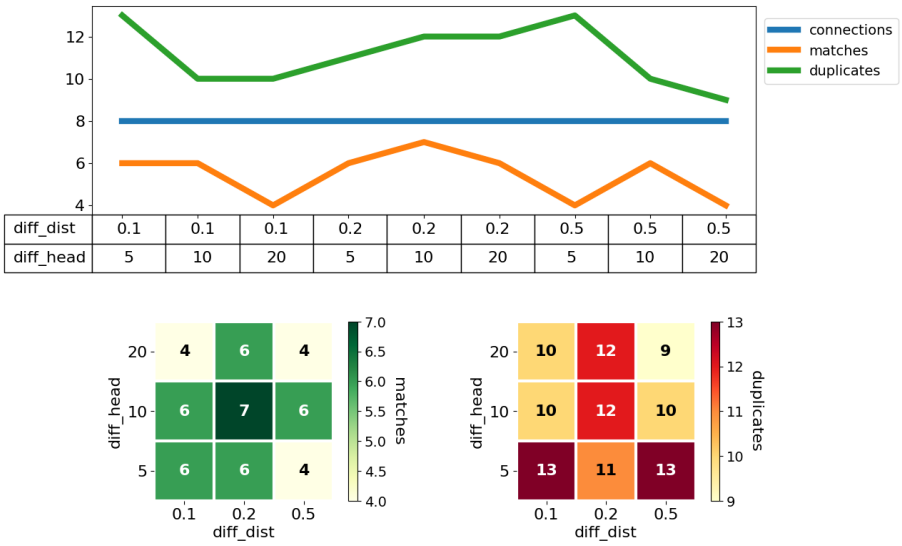
63. ábra: Algoritmusok összehasonlítása (autók: 1000, szimuláció: 100000)



64. ábra: DBSCAN eredmények (autók: 1000, szimuláció: 100000)



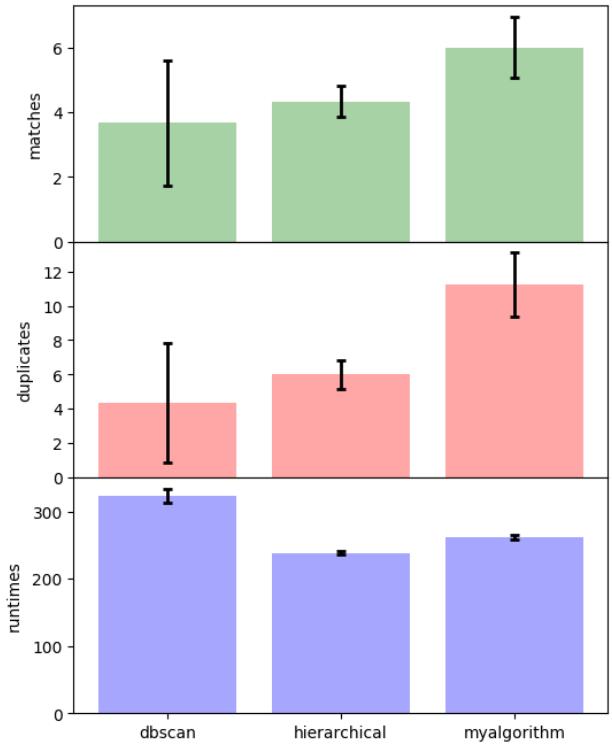
65. ábra: Hierarchical eredmények (autók: 1000, szimuláció: 100000)



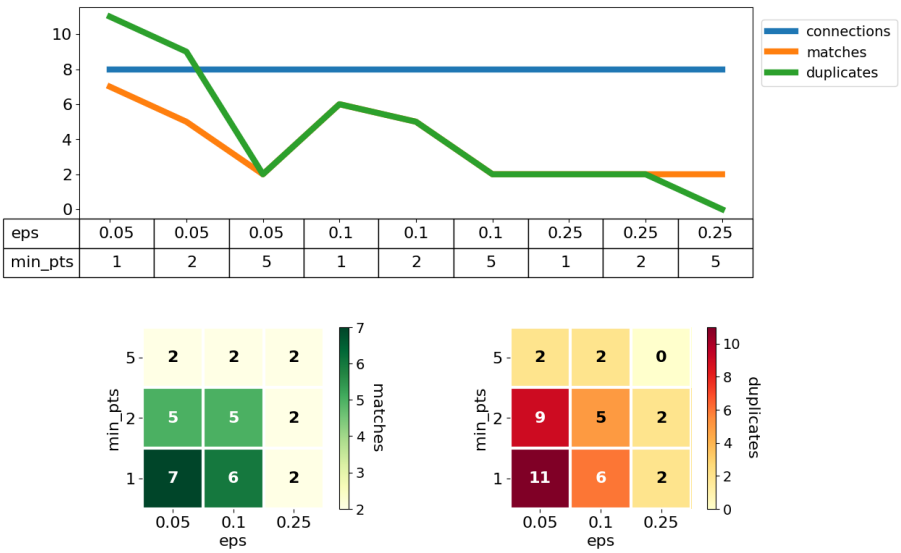
66. ábra: MyAlgorithm eredmények (autók: 1000, szimuláció: 100000)

A MyAlgorithm és a DBSCAN 7-7, a Hierarchical pedig 5 párosítást talált meg. A DBSCAN 11, a MyAlgorithm 12, a Hierarchical pedig 3 duplikátumot készített a legjobb párosításuk elérésekor. A duplikátumok átlagában a DBSCAN a Hierarchical szintjén van, de a szórása sokkal nagyobb. A MyAlgorithm pedig átlagosan több, mint kétszer annyi duplikátumot készít, mint a másik kettő. Az átlagos futási időben a Hierarchical a legjobb, mögötte a MyAlgorithm, és a DBSCAN volt a leglassabb. A futási idő szórása a Hierarchical és a MyAlgorithm esetén nagyon alacsony, a DBSCAN esetében egy kicsit nagyobb azoknál.

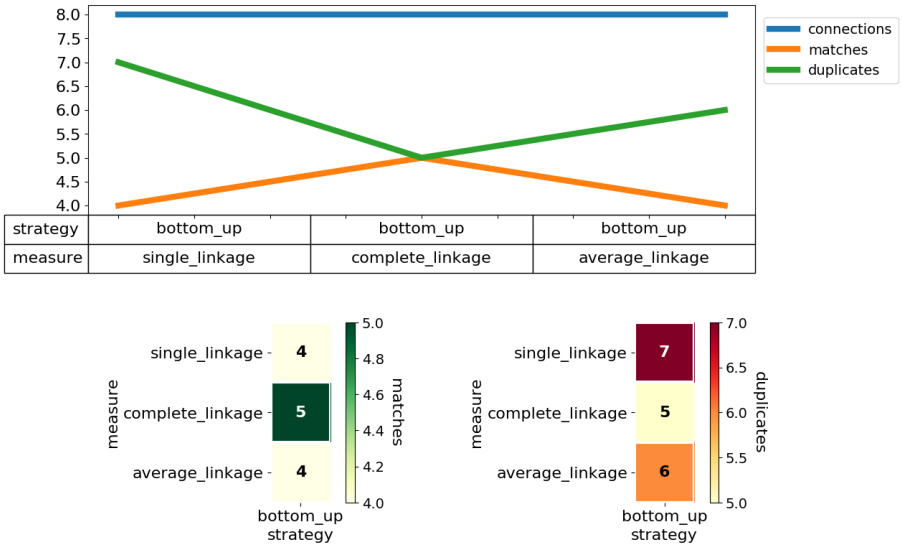
7.13 Autók száma: 5000, Szimuláció ideje: 1000



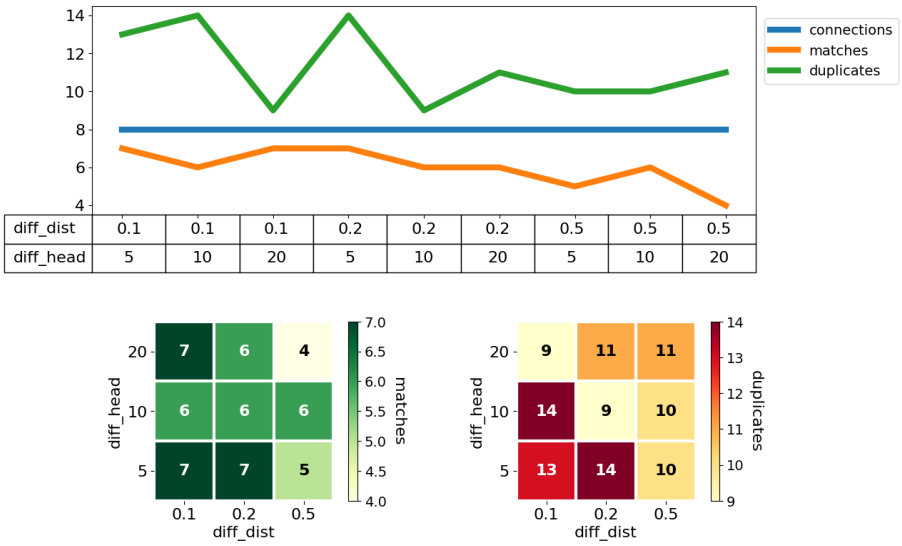
67. ábra: Algoritmusok összehasonlítása (autók: 5000, szimuláció: 1000)



68. ábra: DBSCAN eredmények (autók: 5000, szimuláció: 1000)



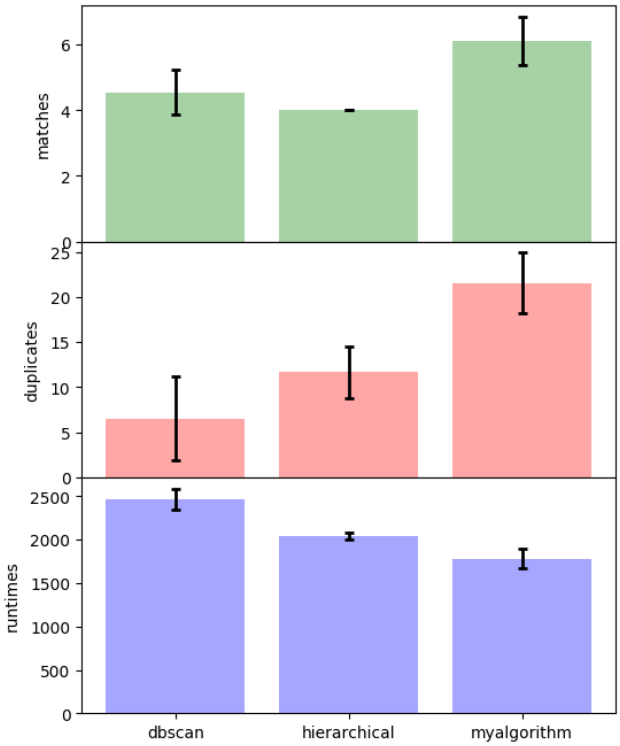
69. ábra: Hierarchical eredmények (autók: 5000, szimuláció: 1000)



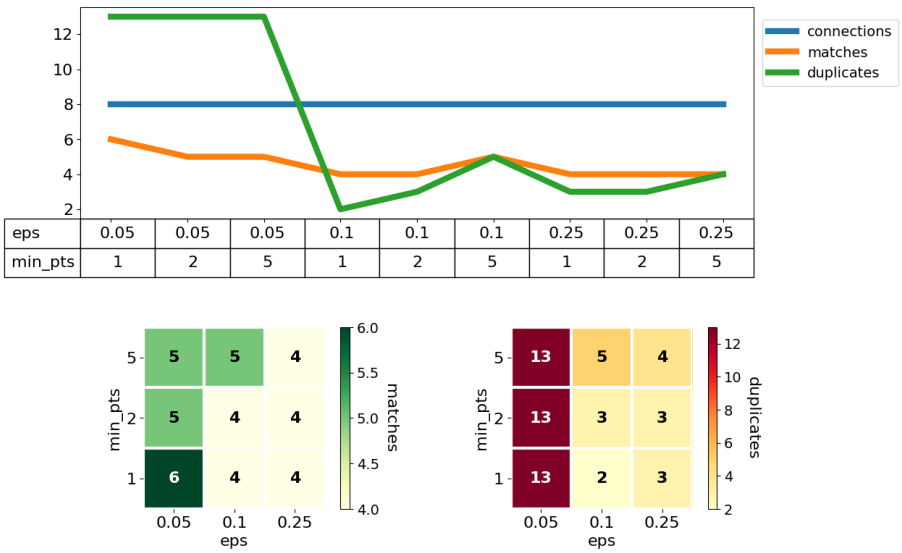
70. ábra: MyAlgorithm eredmények (autók: 5000, szimuláció: 1000)

A DBSCAN és a MyAlgorithm 7-7, a Hierarchical pedig 5 méretű legnagyobb párosítást talált. A párosítás átlagos mérete ennek ellenére DBSCAN esetében a legalacsonyabb. A MyAlgorithm 9, a DBSCAN 11, a Hierarchical pedig 5 duplikátumot készített a legjobb párosításuk elérésekor. A duplikátumok átlaga a DBSCAN esetében a legalacsonyabb, mögötte van a Hierarchical, és a MyAlgorithm volt a legrosszabb, de a DBSCAN esetén a legnagyobb a duplikátumok számának a szórása. Az átlagos futási időben a Hierarchical a legjobb, mögötte nem sokkal a MyAlgorithm, és a DBSCAN a legrosszabb. A futási idő szórása egyiknél se volt nagy, de a DBSCAN esetén egy kicsit nagyobb volt, mint a másik kettőnél.

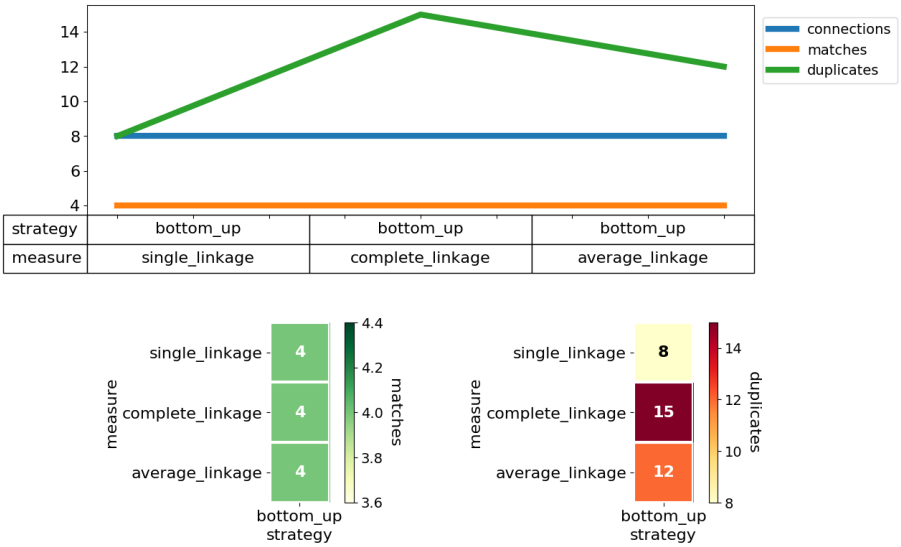
7.14 Autók száma: 5000, Szimuláció ideje: 10000



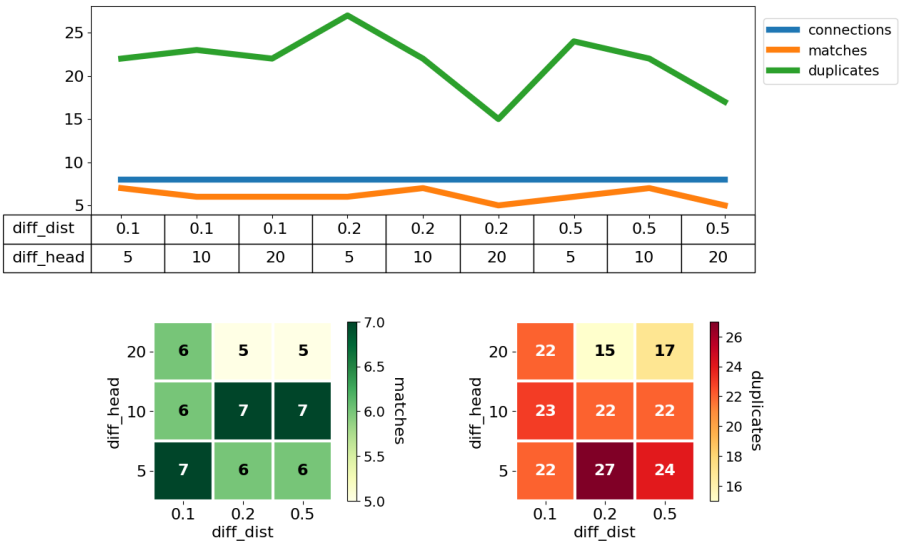
71. ábra: Algoritmusok összehasonlítása (autók: 5000, szimuláció: 10000)



72. ábra: DBSCAN eredmények (autók: 5000, szimuláció: 10000)



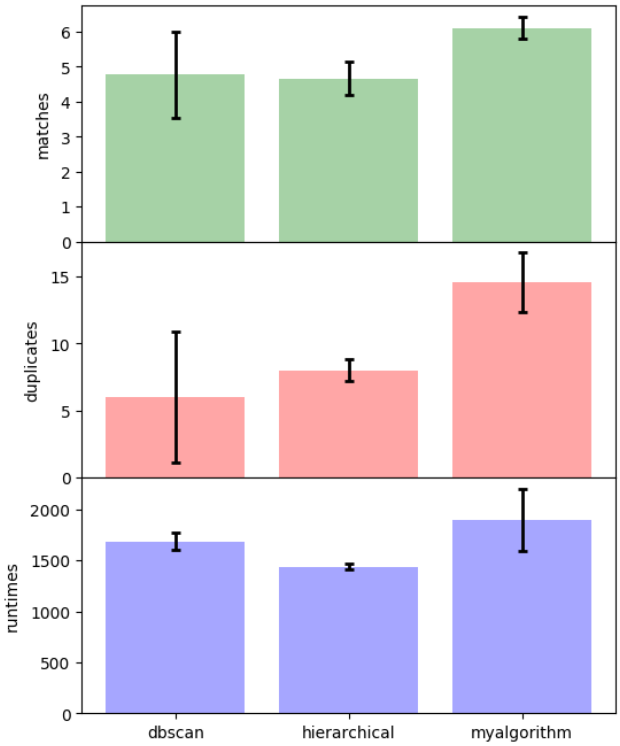
73. ábra: Hierarchical eredmények (autók: 5000, szimuláció: 10000)



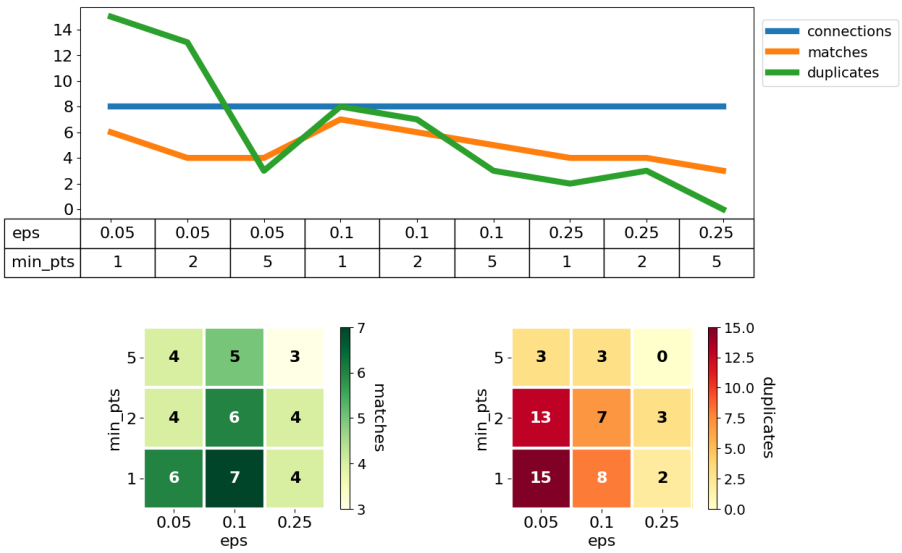
74. ábra: MyAlgorithm eredmények (autók: 5000, szimuláció: 10000)

A MyAlgorithm 7, a DBSCAN 6, a Hierarchical pedig 4 méretű legnagyobb párosítást ért el. A duplikátumok száma ezeknél a 22, 13 és 8 volt. A DBSCAN átlagosan kevesebb duplikátumot készített, mint a Hierarchical, és mindkettő jobb ezen a téren, mint a MyAlgorithm, de a DBSCAN esetében a legnagyobb a szórás a duplikátumok számában. Az átlagos futási időben a MyAlgorithm a legjobb, mögötte a Hierarchical van, és a DBSCAN a leglassabb. A futási idő szórásában a DBSCAN és a MyAlgorithm hasonlóan teljesítettek, de még ezeknél is alacsony volt. A Hierarchical viszont a futási idő szórásában mindkettőnél alacsonyabb értéket ért el.

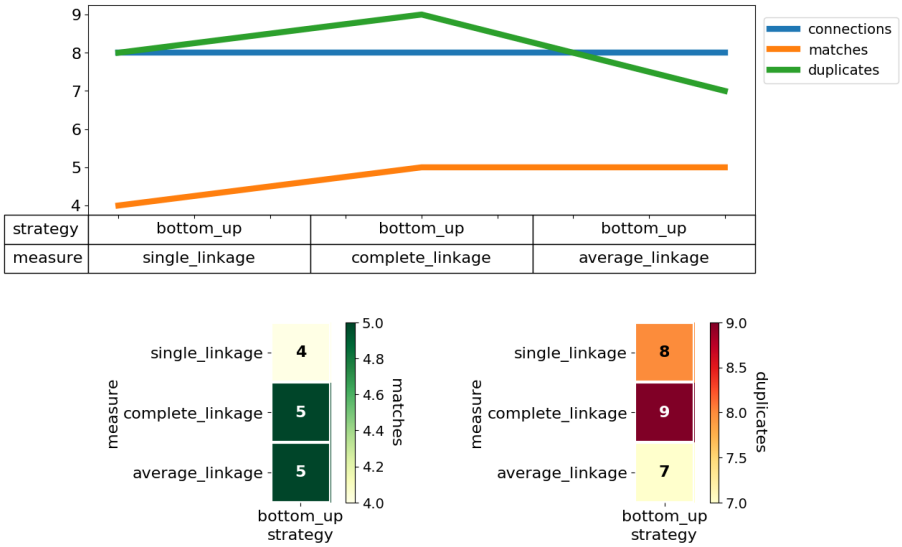
7.15 Autók száma: 5000, Szimuláció ideje: 100000



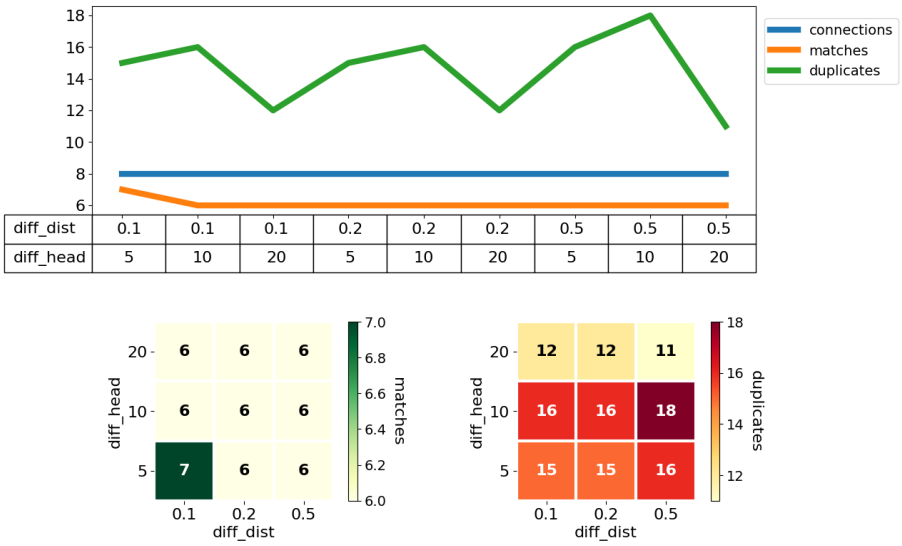
75. ábra: Algoritmusok összehasonlítása (autók: 5000, szimuláció: 100000)



76. ábra: DBSCAN eredmények (autók: 5000, szimuláció: 100000)



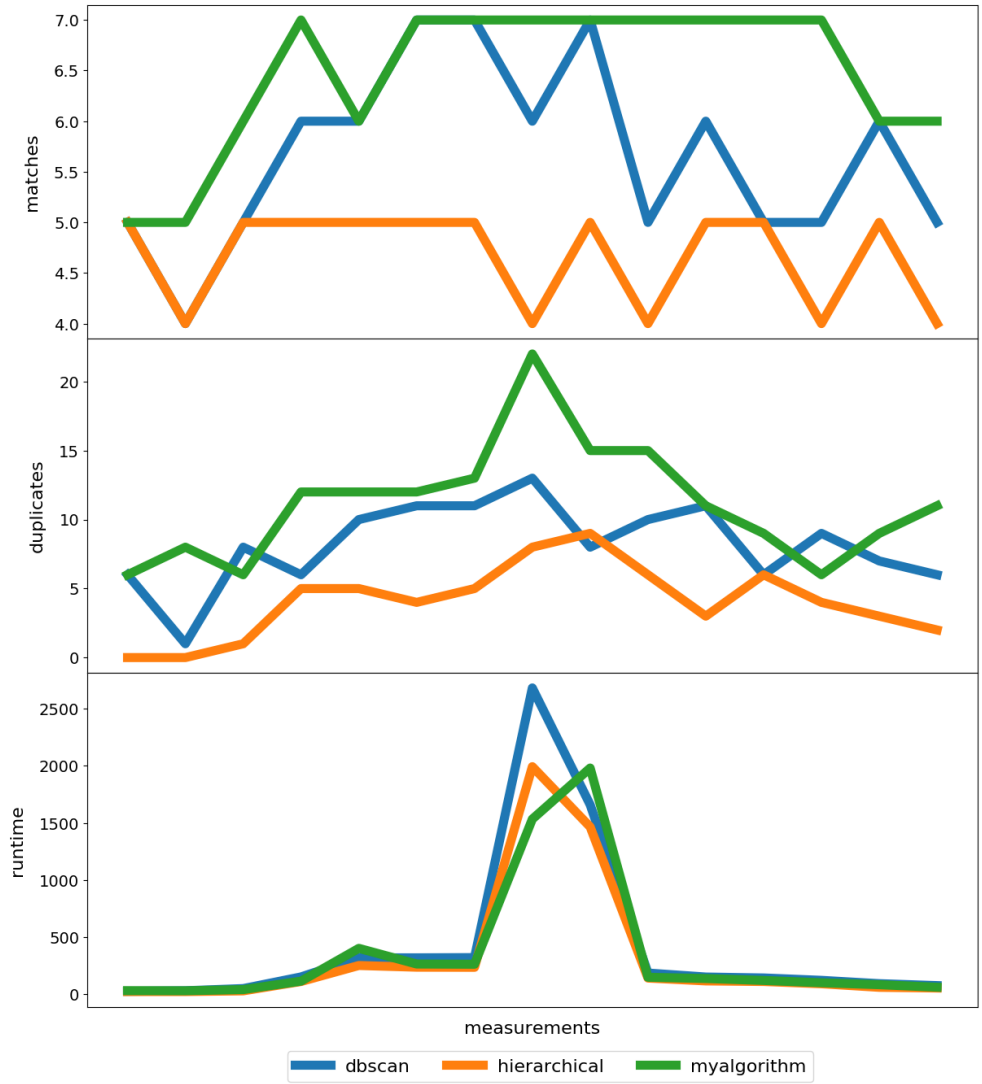
77. ábra: Hierarchical eredmények (autók: 5000, szimuláció: 100000)



78. ábra: MyAlgorithm eredmények (autók: 5000, szimuláció: 100000)

A DBSCAN és a MyAlgorithm is 7-7 méretű legnagyobb párosítást ért el. A Hierarchical esetén a legnagyobb párosítás csak 5 volt. Ezen párosítások elérésekor a DBSCAN 8, a MyAlgorithm 15, a Hierarchical pedig minimálisan 7 duplikátumot készített. A duplikátumok átlagában a DBSCAN érte el a legalacsonyabb eredményt, mögötte a Hierarchical, és a legrosszabb pedig a MyAlgorithm, de a duplikátumok szórásában viszont a DBSCAN a legrosszabb. Az futási idő átlagában és szórásában is Hierarchical, DBSCAN, MyAlgorithm sorrend született. A MyAlgorithm kifejezetten nagy szórást mutatott a futási időben.

7.16 Összefoglalás



79. ábra: A mérési eredmények összehasonlítása

Az ábrán az X-tengelyen láthatók az egyes mérések. Az nincs feltüntetve, hogy az egyes mérések milyen paraméterekkel (autók száma, szimuláció ideje) lettek elkészítve. Ez az ábra a trendek illusztrálására szolgál, ezért ez nem is lényeges. A három diagram a párosítások számát, a duplikátumok számát és a futási időt ábrázolják. Minden mérésben a legjobb párosítást találó paraméter készlet lett kiválasztva az algoritmusokhoz, és az azzal a paraméter készlettel elért duplikátumok és futási idő van ábrázolva a diagramon.

A mérések legnagyobb részében a MyAlgorithm találta meg a legtöbb párosítást, de általában ez készítette a legtöbb duplikátumot is. A DBSCAN a legnagyobb párosítások tekintetében néhány mérésben hozta a MyAlgorithm eredményét, de a mérések többségében inkább 1-gyel vagy 2-vel kevesebbet talált meg. Cserébe a duplikátumok száma is kevesebb volt. Ugyan a legtöbb mérésben a legjobb párosítást elért paraméterekkel a duplikátumok száma megközelítette a MyAlgorithm értékét, de átlagosan lényegesen kevesebb volt. A Hierarchical abból a szempontból érdekes, hogy minden mérésben 4 vagy 5 párosítást talált csak meg a 8-ból, de a duplikátumok számában néhány kivétellel ez volt a legjobb. A futási időben a Hierarchical volt a legjobb a mérések döntő többségében, és általában a DBSCAN volt a leglassabb, a MyAlgorithm pedig a kettő között volt valahol, de nagyságrendi különbség nem volt közöttük.

8 Továbbfejlesztési lehetőségek

A feeder egy elég egyszerű, és kevés felelősséggel rendelkező komponens, ezért abban nem látok nagy továbbfejlesztési lehetőséget.

A map-debugger egy vizualizációs eszköz, ezért abban igazából ízlés kérdése, hogy kinek milyen jellegű megjelenítés adja a legtöbb és legkönnyebben feldolgozható információt. A map-debugger estén el lehetne készíteni többféle megjelenítést, amik vagy különböző információkat jeleníthetnének meg egyszerre vagy ugyanazokat az információkat jeleníthetnék meg különbözőképpen, és a felhasználónak adná meg a választást, hogy ő melyiket szeretné használni.

A map-validator esetén két területen van lehetőség a fejlesztésre. Az egyik az útvonalak összehasonlítása, a másik pedig a metrikák bővítése. Az útvonalak összehasonlítására használt eljárást lehetne finomítani vagy esetleg többféle összehasonlítási módszert használni, és a végén többségi döntés alapján megállapítani, hogy két útvonal megegyezik-e. Új metrikák felvételével több tulajdonság alapján lehetne összehasonlítani az algoritmusokat, amivel több információt lehetne megtudni a működésükről.

A map-creator a legösszetettebb komponens, ezért ebben látom a legtöbb továbbfejlesztési lehetőséget. A map-creator először az UDP socket alapú működésre volt elkészítve, és ebből lett átalakítva úgy, hogy fájlból is tudjon adatokat kapni, mivel a gyorsítás szükséges volt ahhoz, hogy a mérések értelmes időkeretben elvégezhetők legyenek. Az UDP socket alapú működés esetében két időzítési paraméter volt, ami a fájl alapú működésben is fel lett használva: *update_time* és *time_window*. Az *update_time* megadja, hogy milyen gyakran fusson le a klaszterezési algoritmus az útvonalakra. A *time_window* pedig megadja, hogy milyen idős lehet maximálisan egy időbélyeggel ellátott pont. Ezeknek a paramétereknek az állítása nagyban befolyásolja az elért eredményeket, de főleg akkor lehet vele problémát okozni, amikor az adatok fájlból vannak beolvasva, mert ilyenkor nagyon kis idő telik el két pont beérkezése között. A fájlból beolvasásba tettem 1 ms késleltetést minden pont továbbítása elé, hogy

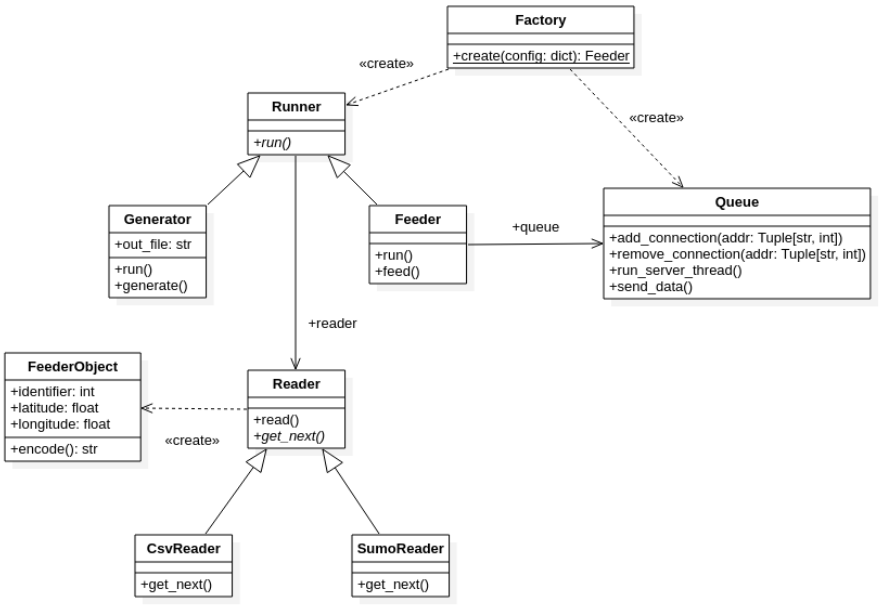
ehhez lehessen igazítani az *update_time* és a *time_window* értékét, de ez még így sem egy ideális megoldás.

A másik dolog, amiben a map-creator fejlesztésre szorul, szintén a frissítés környékén van, de ez nem az időzítéssel kapcsolatos. A frissítési fázis végén az aktuálisan tárolt MAP-ból és az újonnan előállított MAP-ból elő kell állítani az új aktuális MAP-ot. Ezt praktikusan azzal a klaszterező algoritmussal kéne megcsinálni, amivel a map-creator fel lett konfigurálva. Ez viszont sajnos nagyon lassúnak bizonyult, ami miatt a méréseket nem lehetett volna belátható idő alatt elvégezni, ezért a két MAP fúziójában az útvonalak összehasonlítása most egy DTW alapú távolság számolással van megoldva. Ez a mérésekben látható sok duplikátum keletkezésének az egyik forrása.

A duplikátumok számát csökkenteni kéne, hogy a módszer jól használható legyen a gyakorlatban is. Erre egy jó megoldás lehet például, a sávok szélességének megadásával, a trajektóriák sávszintű egyediségének ellenőrzése.

9 Függelék

9.1 A feeder felépítése



80. ábra: A feeder komponens felépítése

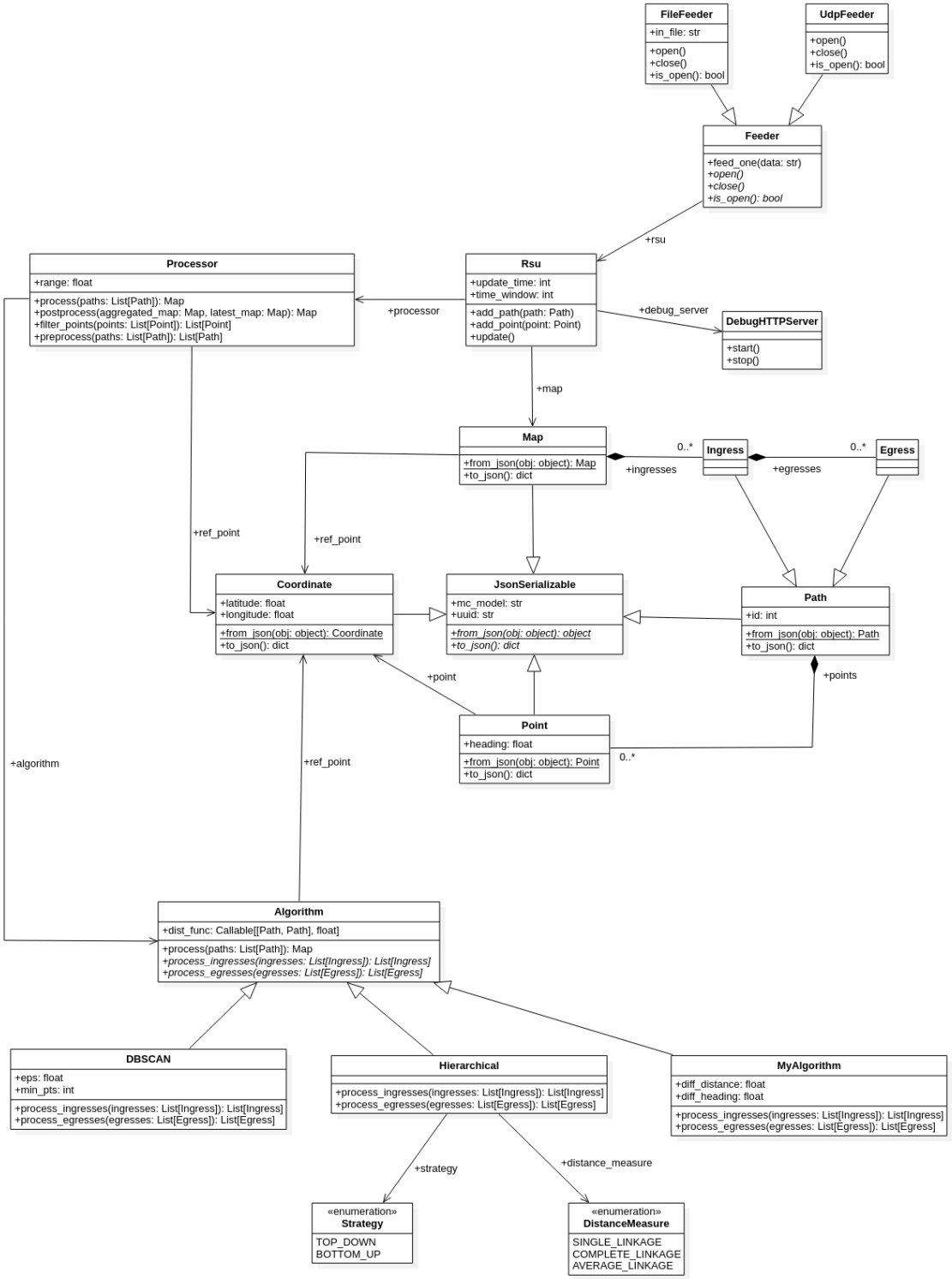
9.2 A feeder konfigurációja CSV adatforrással

```
{
    "mode":             <feed / generate>,
    "source":           "csv",
    "mapping": {
        "identifier":   <azonosító oszlop fejléce>,
        "latitude":     <szélességi fok oszlop fejléce>,
        "longitude":    <hosszúsági fok oszlop fejléce>
    },
    "filepath":         <fájl elérési útja>
}
```

9.3 A feeder konfigurációja SUMO adatforrással

```
{
    "mode":           <feed / generate>,
    "source":         "sumo",
    "sumocfg_path":   <SUMO konfigurációs fájl elérési útja>
}
```

9.4 A map-creator felépítése



81. ábra: A map-creator komponens felépítése

9.5 A map-creator konfigurációs sémája

```
{
    "log_level":        <CRITICAL / ERROR / WARNING /
                         INFO / DEBUG / NOTSET>,
    "dist_func":        <euclidean / dtw>,
    "algorithm": {
        "type":         <dbscan / hierarchical / myalgorithm>,
        ...
    },
    "debug_server": {
        "host":         <debug szerver IP címe>,
        "port":         <debug szerver portja>
    },
    "feeder": {
        "type":         <file / udp>,
        ...
    }
    "processor": {
        "range':        <RSU hatótávolsága>
    },
    "reference_point": {
        "latitude":     <RSU referencia pont szélességi koordinátája>,
        "longitude":    <RSU referencia pont hosszúsági koordinátája>
    },
    "rsu": {
        "time_window": {
            "enabled":  <true / false>,
            "value":    <a csúszóablak ideje másodpercben>
        },
        "update_time": {
            "enabled":  <true / false>,
            "value":    <a frissítési periódus ideje másodpercben>
        }
    }
}
```

9.6 A map-creator konfigurálása: DBSCAN

```
"algorithm": {
    "type":       "dbscan",
    "eps":        <a szomszédság sugara a pont körül>,
    "min_pts":    <a szomszédok számának minimálisan
                  elvárt száma eps sugáron belül>,
    "ref_point":  <RSU referencia pontja>
}
```

9.7 A map-creator konfigurálása: Hierarchical

```
"algorithm": {
    "type':       "hierarchical",
    "strategy":   <top_down/bottom_up>,
    "measure":    <single_linkage/complete_linkage/average_linkage>,
    "ref_point":  <RSU referencia pontja>
}
```

9.8 A map-creator konfigurálása: MyAlgorithm

```
"algorithm": {
    "type':       "myalgorithm",
    "ref_point":  <RSU referencia pontja>,
    "diff_dist":  <két azonosnak tekintett trajektória közötti
                   átlagos távolság maximuma>,
    "diff_head":  <két azonosnak tekintett trajektória átlagos
                   haladási iránya különbségének a maximuma>
}
```

9.9 A map-creator konfigurálása: File Feeder

```
"feeder": {
    "type":  "file",
    "path":  <előre legenerált adatforrás fájl elérési útja>
}
```

9.10 A map-creator konfigurálása: UDP Feeder

```
"feeder": {
    "type":         "udp",
    "host":         <a map-creator UDP socketének IP címe>,
    "port":         <a map-creator UDP socketének port száma>,
    "server_host":  <a feeder példány UDP socketének IP címe>,
    "server_port":  <a feeder példány UDP socketének port száma>
}
```

Irodalomjegyzék

[1] “Ford Breaks With Auto Rivals By Committing To C-V2X Vehicle Communications Tech,” *Forbes*. [Online]. Available: https://www.forbes.com/sites/samabuelsamid/2019/01/07/ford-becomes-first-automaker-to-commit-production-c-v2x-communications/. [Accessed: 27-Sep-2019].
[2] “Intelligent Transportation Systems - ITS Deployments.” [Online]. Available: https://www.its.dot.gov/pilots/tampa_datahub.htm. [Accessed: 27-Sep-2019].
[3] “Welcome to ROSA P | Vehicle-to-vehicle communications : readiness of V2V technology for application. - 27999 | US Transportation Collection.” [Online]. Available: https://rosap.ntl.bts.gov/view/dot/27999. [Accessed: 27-Sep-2019].
[4] Etsi, “ETSI TS 103 301 V1.2.1 (2018-08) Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Facilities layer protocols and communication requirements for infrastructure services,” 2018.
[5] J. Bian, D. Tian, Y. Tang, and D. Tao, “A survey on trajectory clustering analysis,” *ArXiv180206971 Cs*, Feb. 2018.
[6] András C., “Helyettesítheti a gép az embert a közlekedésben?,” p. 20.
[7] Sae, “Summary of SAE International’s Levels of Driving Automation for On-Road Vehicles,” vol. 2, p. 3.
[8] K. Sjöberg, P. Andres, T. Buburuzan, and A. Brakemeier, “Cooperative Intelligent Transport Systems in Europe Current Deployment Status and Outlook,” 2017.
[9] V. V. Vehicle-to-Vehicle and V. P. Vehicle-to-Pedestrian, “Artificial Intelligence for Vehicle-to-Everything: A Survey,” *IEEE Access*, vol. 7, 2019.
[10] Etsi, “EN 302 663 V1.2.0 (2012-11) European Standard Intelligent Transport Systems (ITS); Access layer specification for Intelligent Transport Systems operating in the 5 GHz frequency band,” 2012.
[11] Etsi, “ETSI TS 102 637-2 V1.2.1 (2011-03) Technical Specification Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service,” no. 2011–2003, 2011.
[12] A. Festag, “AUTOMOTIVE NETWORKING AND APPLICATIONS Cooperative Intelligent Transport Systems Standards in Europe,” vol. 2, no. 610542, p. 166, 2015.
[13] Etsi, “EN 302 637-3 V1.2.1 (2014-09) EUROPEAN STANDARD Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 3: Specifications of Decentralized Environmental Notification Basic Service,” no. 2014–2009, 2014.
[14] Amsterdam Group, “Signal Phase and Time (SPAT) and Map Data (MAP),” 2015.
[15] Its, “TR 102 638 - V1.1.1 - Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Definitions,” no. 2009–2006, 2009.
[16] R. Bodenheimer, D. Eckhoff, and R. German, “GLOSA for adaptive traffic lights: Methods and evaluation,” in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2015, pp. 320–328.

[17] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” 1996, pp. 226–231.
[18] Nisha and P. J. Kaur, “Cluster quality based performance evaluation of hierarchical clustering method,” in *2015 1st International Conference on Next Generation Computing Technologies (NGCT)*, 2015, pp. 649–653.
[19] T. Xiang and S. Gong, “Spectral clustering with eigenvector selection,” *Pattern Recognit.*, vol. 41, pp. 1012–1029, 2008.
[20] R. Güting, T. Behr, and J. Xu, “Efficient k-nearest neighbor search on moving object trajectories,” *Vldb J. - VLDB*, vol. 19, Feb. 2010.
[21] M. Á. Bautista *et al.*, “A Gesture Recognition System for Detecting Behavioral Patterns of ADHD,” *IEEE Trans. Cybern.*, vol. 46, no. 1, pp. 136–147, Jan. 2016.
[22] H. Wang and C. O’Sullivan, “Globally Continuous and Non-Markovian Crowd Activity Analysis from Videos,” 2016, vol. 9909.
[23] K. Cho and X. Chen, “Classifying and Visualizing Motion Capture Sequences using Deep Neural Networks,” *VISAPP 2014 - Proc. 9th Int. Conf. Comput. Vis. Theory Appl.*, vol. 2, 2013.
[24] T. Mai, “Global Positioning System History,” *NASA*, 05-May-2015. [Online]. Available: http://www.nasa.gov/directorates/heo/scan/communications/policy/GPS_History.html. [Accessed: 12-Oct-2019].
[25] “GPS.gov: Performance Standards & Specifications.” [Online]. Available: https://www.gps.gov/technical/ps/. [Accessed: 12-Oct-2019].
[26] M. Matosevic, Z. Salcic, and S. Berber, “A Comparison of Accuracy Using a GPS and a Low-Cost DGPS,” *IEEE Trans. Instrum. Meas.*, vol. 55, no. 5, pp. 1677–1683, Oct. 2006.
[27] D. Manandhar, K. Honda, and S. Murai, “Accuracy assessment and improvement for level survey using real time kinematic (RTK) GPS,” in *IEEE 1999 International Geoscience and Remote Sensing Symposium. IGARSS’99 (Cat. No.99CH36293)*, 1999, vol. 2, pp. 882–884 vol.2.
[28] V. Ho, K. Rauf, I. Passchier, F. Rijks, and T. Witsenboer, “Accuracy Assessment of RTK GNSS based Positioning Systems for Automated Driving,” in *2018 15th Workshop on Positioning, Navigation and Communications (WPNC)*, 2018, pp. 1–6.

Ábrajegyzék

Rövidítésjegyzék

ADAS	Advanced Driver Assistance Systems
C-ITS	Cooperative Intelligent Transport Systems
CAM	Cooperative Awareness Message
CSMA/CA	Carrier-sense multiple access with collision avoidance
CSV	Comma-separated values
DBSCAN	Density-based spatial clustering of applications with noise
DEN	Decentralized Environmental Notification
DENM	Decentralized Environmental Notification Message
DGPS	Differential GPS
DoD	Department of Defense
DTW	Dynamic Time Warping
GLOSA	Green Light Optimzed Speed Advisory
GMM	Gaussian Mixture Model
GPS	Global Positioning System
HAD	High Definition Map
I2V	Infrastructure-to-Vehicle
JSON	JavaScript Object Notation
LIDAR	Light Detection and Ranging
MAP	Map Data
NAVSTAR	Navigation System with Timing and Ranging
OBU	Onboard Unit
PDU	Protocol Data Unit
RADAR	Radio Detection and Ranging

RSU	Roadside Unit
RTK	Real-time Kinematic
SAE	Society of Automotive Engineers
SPAT	Signal Phase and Timing
SUMO	Simulation of Urban Mobility
UDP	User Datagram Protocol
UML	Unified Modeling Language
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
V2X	Vehicle-to-Everything