

# TDK dolgozat

---

Fekete Marcell



**Eötvös Loránd Tudományegyetem**

**Informatikai Kar**

**Információs Rendszerek Tanszék**

# Torlódásdetektálás nagy mennyiségű trajektoriális adatokban

---

TDK dolgozat

Készítette: **Fekete Marcell**

II. évf. Bsc Programtervező Informatikus, B szakirány

Témavezető: **Dr. Vincellér Zoltán**

Műszaki tanár, Információs Rendszerek Tanszék

Budapest, 2013.05.03

|         |                                                    |    |
|---------|----------------------------------------------------|----|
| I)      | Absztrakt .....                                    | 4  |
| II)     | Adatbányászat.....                                 | 4  |
| II.1)   | OSZTÁLYOZÁSOK .....                                | 5  |
| II.2)   | ITEMSETS.....                                      | 6  |
| II.2.1) | FCI bányászat és megközelítései.....               | 8  |
| a)      | Elemek bejárési sorrendje .....                    | 9  |
| b)      | Feltételes adatbázis ábrázolása .....              | 9  |
| c)      | Feltételes adatbázis konstrukciós stratégiái ..... | 9  |
| d)      | Fa bejárása.....                                   | 10 |
| III)    | Adatbányászat forgalmi adatokon .....              | 10 |
| III.1)  | TRAJEKTÓRIÁK .....                                 | 11 |
| III.2)  | PATTERN MINING .....                               | 11 |
| III.3)  | PERIODICITÁS .....                                 | 17 |
| III.4)  | VÁROSTERVEZÉSI SZEMPONTOK.....                     | 18 |
| IV)     | Get_Move algoritmus és keretrendszere.....         | 20 |
| IV.1)   | ELŐFELDOLGOZÁS .....                               | 21 |
| IV.2)   | OSZTÁLYOZÁS.....                                   | 22 |
| IV.3)   | FCI BÁNYÁSZAT ÉS PATTERN MINING .....              | 24 |
| a)      | Adatmodell.....                                    | 24 |
| b)      | LCM algoritmus .....                               | 25 |
| c)      | Get_Move és Pattern mining .....                   | 27 |
| IV.3.2) | Utófeldolgozás, megjelenítés .....                 | 27 |
| IV.4)   | EREDMÉNYEK .....                                   | 28 |
| IV.4.1) | FCI bányászat.....                                 | 28 |
| IV.4.2) | torlódásdetektálás.....                            | 29 |
| V)      | Értékelés, fejlesztési tervek .....                | 29 |
| V.1)    | FCI BÁNYÁSZ ALGORITMUS FEJLESZTÉSE .....           | 30 |
| V.2)    | HATÉKONYABB IMPLEMENTÁCIÓ .....                    | 30 |
| V.3)    | PERIODIKUS MINTÁK .....                            | 30 |
| V.4)    | PÁRHUZAMOSÍTÁS.....                                | 31 |
| V.5)    | HATÉKONYABB KERETRENDSZER .....                    | 31 |
| VI)     | Irodalomjegyzék.....                               | 31 |

## I) Absztrakt

A mai világ forgalmi hálózatai messze túl nagyok és bonyolultak ahhoz, hogy hagyományos módszerekkel vizsgáljuk őket és ezzel egy időben a *GPS* adatok rögzítésére és feldolgozására képes készülékek gyors térhódításával egyre több vizsgálható adat áll a rendelkezésünkre.

Fontos lehet várostervezési szempontból egy-egy hálózat, vagy részhálózat valamilyen karakterisztikus tulajdonságát meghatározni, mint például a kapacitását, vagy kihasználtságát. Az utóbbi időben több különböző megközelítés született a valamilyen séma alapján együtt mozgó objektumok beazonosítására. A talált minták periodicitását vizsgálva fel lehetne mérni földrajzi alapon jól meghatározott közösségek (város lakói, lakótelep lakói) tipikus viselkedését, szokásaikat. Ezek a trajektoriális adatok származhatnak különböző forrásokból, ezért bevezetek egy egységes adatábrázolási modellt, melyben az adatokat tárolom és anonimizáltan kezelem. Az adatok anonimizálása a személyiségi jogok védelme mellett, a különböző típusú objektumok egységes kezelését is elősegíti.

Dolgozatom célja nagy mennyiségű *trajektoriális* adatokban különböző mintákat megtalálni képes algoritmusok keresése és hatékony implementálása, azok eredményeinek megjelenítése, értelmezése. A feldolgozandó adatok masszív jellege miatt vizsgálom a feladat oszthatóságát és párhuzamosíthatóságát is. Első lépésben a nyers adatokat <azonosító, hosszúság, szélesség, mérési idő> formátumú adatsorrá konvertálok. Ezeket egy klaszterezést hajtok végre (*DBScan*), majd egy saját adatstruktúrában tömörítve tárolom. A *Frequent Closed Itemset*-ek megkeresésére az *LCM* megközelítés egy módosított változatát implementáltam, majd ezekből nyerem ki a keresett mintákat. Az elő és utófeldolgozást, megjelenítést R-ben, az eljárást magát pedig C++-ban oldottam meg.

## II) Adatbányászat

Manapság rengeteg forgalmi, helyrajzi adatot tesznek közzé az emberekről, vagy akár ők magukról. Egyes országokban már az új telefonos előfizetők hozzájárulhatnak ahhoz, hogy rögzítsék helyrajzi adataikat, míg mások a különböző közösségi oldalakon önként teszik azokat közzé. Az adatbányász eszközök feladata az ilyen - és más jellegű -, de nagy mennyiségű adatok hatékony feldolgozása, illetve az adatok olyan konverziója ami nem jár releváns információ elvesztésével, de az eljárások számára könnyebben értelmezhető, azok futása erőforrás kímélőbbek a konverzió által. Az adatbányászat szerves részét képezi az adatbázisokban végzett tudásfeltárás (*Knowledge Discovery in Databases*) folyamatának, amely a nyers adatoknak hasznos információvá való átalakításának a teljes folyamata, amint azt a *II-1. ábra* mutatja. Ez a folyamat transzformációs lépéseknek az adatok előfeldolgozásától az adatbányászati eredmények utófeldolgozásáig terjedő sorozatából áll. Az előfeldolgozás során történnek meg a transzformációs lépések, melyekkel az adatokat normalizálják. Az adatok mennyisége miatt praktikus dimenziócsökkentő, illetve metsző eljárásokat is alkalmazni. Majd ezeken, a már előkészített adatokon dolgoznak az adatbányász eljárások, mint például az osztályozások, klaszterezések, vagy mintakereső eljárások. Az utófeldolgozás során a kinyert információ szűrhető, ember számára könnyen értelmezhető formára hozható, megjeleníthető.

A gépi tanulásnak három típusa van:

- Felügyelt tanulás,
- megerősítéses tanulás
- felügyeletmentes tanulás

A fő különbség köztük az ember szerepe az eljárás során. A felügyeletmentes tanuló eljárások esetében a tanuló technikák előre implementálva vannak, emberi beavatkozás nem szükséges. A megerősítéses tanulás, általában egy zárt rendszeren belül határoz meg viselkedésformákat és azok szerint cselekszik. Ezeknek a viselkedéseknek a meghatározásához a felhasználó megerősítését kéri. Az utolsó típus a felügyelt tanulás. A felügyelt tanulás olyan eljárás, ami előre ismeri egy úgynevezett tanulóhalmaz elemeire a megoldást és megpróbál összefüggéseket találni, majd azok alapján következtetni. Ez a típus képes absztrakt fogalmak megfogására, de általában nehéz betanítani, jó adatokat találni (egyáltalán egy adathalmaz jóságát meghatározni). Az adatbányász eszköz eredményének jellegétől függően két típust különböztetünk meg: osztályozást és előrejelzést. Ha az eredmény folytonos előrejelzésről beszélünk, egyébként osztályozásról.



II-1. ábra

## II.1) Osztályozások

Az objektumok valamilyen jól meghatározott tulajdonságai szerint csoportokba sorolását osztályozásnak nevezzük. Különböző problémák különböző megközelítéseket igényelnek. Az alapján, hogy az egyes eljárások milyen osztályokkal dolgoznak, két fajta csoportot különböztetünk meg. Egyes eljárások rögzített osztályokkal dolgoznak, a vizsgált elemeket azokba sorolják be. Ilyen eljárás például az általam is implementált *DBScan*. Meghatározhatunk ilyen osztályokat alakjuk alapján (kör, ellipszis), sűrűségük alapján, vagy valamilyen címkehalmaz alapján is. Előfordul azonban, amikor absztrakt, vagy előre nem látható osztályokba szeretnénk az elemeket sorolni, ekkor azok meghatározása az osztályozó eljárás feladata (*K-means* család, *ISOData*). Ilyen feladat, ha például egy üzlet relációs adatbázisából próbálunk vásárlói csoportokat meghatározni, illetve azok szokásait feltérképezni.



II-2. ábra

A képzett osztályok egymáshoz vett viszonyát tekintve két féle osztályozás típust különböztetünk meg: *Crisp* és *Fuzzy*. A *Fuzzy Clustering* eljárások megengedik, hogy egy objektum egyszerre több osztályban is szerepeljen, míg a *crisp* jellegű eljárásoknál az eredményként kapott osztályok diszjunktak, ahogy azt a II-2. ábra mutatja.

## II.2) Itemsets

Az adatbányászatban alapvető probléma az elemek nagy halmazai között összefüggéseket keresni. Általában egy-egy elemhalmaz valamilyen feltétel melletti előfordulásait keresik a teljes adatbázisban (ennek az elemhalmaznak nem kell szerepelnie, az adatbázisban). Azt az elemhalmazt, aminek a vizsgált elemhalmaz részhalmaza, előfordulásnak nevezzük, a relációjukat pedig tranzakciónak. Mindez formálisan:

Legyenek,

$$\hat{I} = \{1..n\}$$

egy itemset (vagy elemhalmaz)

$$\hat{I}(S) = \cap T : T \in S$$

egy elemhalmaz megszorítása S halmazra

$$T = \{t_1..t_m\} : t_i \in \hat{I}$$

transaction set (tranzakciós halmaz)

$$P \subseteq \hat{I}$$

egy elem, vagy halmaz  $\hat{I}$ -ből

Ekkor,

$$P' \subseteq T$$

P egy előfordulása

$$T(P)$$

P előfordulásainak halmaza

$$\text{Frq}(P) = |T(P)|$$

P gyakorisága (frequency)

P sűrű, ha egy előre rögzített számnál nagyobb a gyakorisága. Fontos még bevezetni a zártság fogalmát:

$$P \text{ zárt} \Leftrightarrow [P \subseteq T(P) \rightarrow \neg(\exists P') : T(P) = T(P') \wedge P \subset P']$$

Sűrű, ha  $\text{Frq}(p) > 3$

| P Elemhalmaz | Frq(P) | Gyakori | Zárt |
|--------------|--------|---------|------|
| {a}          | 4      | igen    | nem  |
| {b}          | 5      | igen    | igen |
| {c}          | 3      | nem     | nem  |
| {a,b}        | 4      | igen    | igen |
| {a,c}        | 2      | nem     | nem  |
| {b,c}        | 3      | nem     | igen |
| {a,b,c}      | 2      | nem     | nem  |

### II-3. ábra

A II-3. ábra egy példa, a zártság és sűrűség tulajdonságok bemutatására. Szemléletesen, nem lehet új elemet hozzávenni a vizsgált elemhalmazhoz a gyakoriság számértékének változása nélkül.

A *GeT\_Move* gyakori, zárt elemhalmazok keresésére szolgál (*FCI – Frequent Closed Itemset*). Azt, hogy ilyen típusú adatokon mit jelentenek a következő fejezetben tárgyalom. Ahhoz, hogy meg tudjuk vizsgálni egy elemhalmaz zártságát, be kell vezetni a lezárt fogalmát:

$$\text{Clo}(P) = \hat{I}(T(P)) = \bigcap T : T \in T(P) \quad P \text{ lezártja}$$

Ekkor kimondhatjuk a következő tételt:

$$P \text{ zárt} \Leftrightarrow \text{clo}(P) = P$$

Biz.:

→ irány:

$$P \text{ zárt} \rightarrow P \subseteq \text{clo}(P)$$

mert nem létezik bővebb  $P'$  úgy, hogy ugyanazt az előfordulás halmazt generálná

$$\text{clo}(P) \subseteq P$$

a definícióból következik

← irány:

$$P = \text{clo}(P) \rightarrow \neg(\exists P') : P \subset P' \wedge P' \subseteq \text{clo}(P)$$

pedig a  $\text{Clo}(P)$  definíciójából következik. Általában igaz, hogy

$$P \subset P' \rightarrow T(P') \subseteq T(P)$$

mivel csak megszorítással élünk. Ekkor nem állhat fent egyenlőség, mert  $P$  az a halmaz, amely  $T(P)$  minden elemének részhalmaza és az első állítás szerint  $P'$  nem lehet szintén ilyen.

□

Gyakori kérdés, egy adatbázis elemhalmazainak az összes zárt, gyakori elemhalmazainak (FCI), az összes gyakori elemhalmazainak meghatározása (FI), illetve az összes maximális gyakori elemhalmazainak (FMI) a meghatározása. Az FCI és FI fogalma már be lett vezetve. Egy FI maximális, ha ő maga gyakori, de egyetlen részhalmaza sem az. A II-3. ábra példájában FMI a  $\{b,c\}$  és a  $\{a,b\}$  elemhalmazok. A példából is látszik, hogy

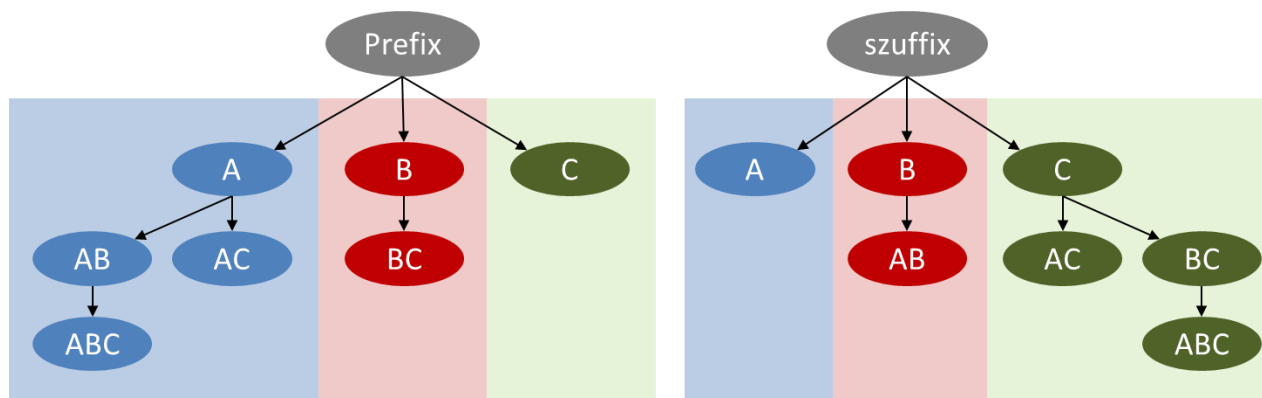
$$\{f \mid f \text{ az FMI} \} \subseteq \{c \mid c \text{ az FCI} \} \subseteq \{i \mid i \text{ az FI} \}$$

### II.2.1) FCI bányászat és megközelítései

A IV.3)b) pontban részletesen tárgyalom az általam is implementált LCM megközelítést. Egy másik hatékony algoritmus az AFOPT. A sűrű elemhalmazokat bányászó algoritmusok több választott stratégia szerint is osztályozhatóak:

- 1) elemek bejárásának sorrendjében
- 2) a feltételes adatbázis ábrázolásában
- 3) a feltételes adatbázis konstruálásának módjában
- 4) a IV.3)b) pontban definiált fa bejárásának módjában

Az AFOPT egy adaptív eljárás, ami megpróbálja mindig a legnagyobb haszonnal járó ábrázolási technikákat alkalmazni egy tetszőleges elemhalmaz adatbázisban. Ezt a bővítő megközelítést alkalmazó eljárások - pattern growth approach - különböző stratégiáinak helyes megválasztásával teszi. Minden megtalált sűrű elemhalmazhoz felépít egy feltételes adatbázist, és a gyerek osztályokat ebben a megszorított adatbázisban keresi, ezzel csökkentve a keresési teret. A két fő művelet az ilyen típusú eljárásoknál, a sűrűség meghatározása és a feltételes adatbázis felépítése. Ezeknek a hívásoknak a száma erősen függ az állapottér bejárásának stratégiájától. A bejárás és a feltételes adatbázis felépítésének költsége pedig  $z$  eredeti adatbázis méretétől, a reprezentáció módjától (fa-szerű ábrázolás, vagy blokkyszerű) és a konstrukciós stratégiától (fizikai vagy pseudo). Az adatok bejárásának sorrendje is számít. Prefix szemlélet, ha egy  $\{a,b,c,d,e\}$  adatbázist 5 diszjunkt feltételes adatbázisra bontunk, a következő módon: a-t tartalmazó elemhalmazok, b-t tartalmazó, de a-t nem tartalmazó elemhalmazok, c-t igen, de a, b-t nem tartalmazó elemhalmazok, d-t igen, de a, b, c-t nem tartalmazó elemhalmazok és végül a csak e-t tartalmazó elemhalmaz. A konstrukciós szabály egyszerű, a kisebb elemeket kihagyjuk az elemhalmazok képzéséből. Ez a módszer nem csak a bejárást gyorsítja meg, de segítségével nincs szükség annyi metszet vizsgálatra. A prefix szemlélet párja a suffix szemlélet, ami ugyanez az algoritmus a rendezéssel ellentétes irányban, ahogy II-4 ábra mutatja.



II-4 ábra

#### a) Elemek bejárési sorrendje

Az adatbázis felosztásához szükség van az elemek közt valamilyen rendezést definiálására, ami meghatározza, hogy az egyes feltételes adatbázisokba mely elemek kerüljenek. Egy lehetséges statikus rendezés a lexikális rendezés. Az egyes iterációk során ez a sorrend nem változik. Másik megközelítés a frekvencia szerinti dinamikus rendezés (emelkedő, vagy fogyó bejárás). A dinamikus rendezés minden iterációnál újrendezi a feltételes adatbázist és a stratégiájának megfelelően járja be. A növekvő bejárás előnye, hogy mindig azt az elemet vizsgálja, ami (potenciálisan) a legkisebb számítási időt igényli, ezzel minimalizálva a feltételes adatbázisok méretét minden iterációban, illetve a szükséges konstrukciók számosságát.

#### b) Feltételes adatbázis ábrázolása

Az ábrázolás meghatározza a feltételes adatbázisok bejárásának és létrehozásának költségét, ezért fontos megvizsgálni, hogy az egyes technikáknak mik az előnyei, hátrányai. A fa ábrázolás csökkenti a bejárás költségét, mert képes a redundáns elemhalmazokat összevonni, de drasztikusan növeli az építés költségét a szükséges vizsgálatok miatt (főleg, ha az adatbázis nem sűrű). Ennek oka, hogy azokat az elemhalmazokat, amiknek a metszete nem üres lehetséges egy memóriaterületen tárolni. Az ilyen jellegű memóriatakarékosság persze drága lehet<sup>[7]</sup>. A blokkszerű ábrázolás nagyon gyorsan felépül, de a redundáns elemek látogatása megnöveli a bejárás költségét. Következésképpen sűrű adatokon a fa-ábrázolás az optimális, míg ritkán a blokkszerű.

#### c) Feltételes adatbázis konstrukciós stratégiái

Az adatbázisokat fizikailag tárolni, költséges lehet, főleg ha az egyes iterációkban lassan fogynak a bővítéshez a jelöltek. Megoldás lehet a pszeudó konstrukciós stratégia: használhatunk mutatókat, amik egy felsőbb absztrakciós szinten lévő adatbázisban tárolt adatokra mutatnak. Ezzel a tároláshoz szükséges tárhely mérete csökkent, a bejárás kárára. Ha egy iterációban úgy ítéli az eljárás, hogy a felépítendő feltételes adatbázis elég kicsi, akkor stratégiát válthat, ezzel gyorsítva annak az ágnak a kiértékelési költségét.

#### d) Fa bejárása

Ezeket a fákat a gyökérből a levelekig a legolcsóbb bejárni, a *IV.3)b)* pontban definiált szülő-gyerek kapcsolatokat, mint él relációkat használva. Ennek a relációnak a kiértékelése persze költséges, de így nem születnek felesleges ágak.

Az *AFOPT* eljárás a fent említett módszereket vegyesen alkalmazza a helyzetnek megfelelően, ezzel elérve a felvetett helyzetekben az optimális költségeket, a vizsgálatok árán. Továbbá, a sűrűség alapú rendezés konzisztens használatával nincs szükség az elemek újrendezésére, ezért nem kell fizikailag létrehozni az új adatbázisokat, csupán az utolsó elem indexét kell tárolni. Ezt az algoritmust választottam referenciának (az eredmények ellenőrzésére), mert sok potenciális fejlesztési irányt mutat és egy elég távoli, de elérhető célt futási időben és memóriahasználatban egyaránt.

### III) Adatbányászat forgalmi adatokon

Egy városban több eszköz is szolgáltat adatokat, amikből hasznos információk bányászhatók. Ilyenek a különböző forgalomszámlálásra alkalmas útba süllyesztett eszközök, illetve a *GPS* készülékek. Ezek a források, vagy eleve anonim, vagy utólag anonimizált adatokat szolgáltatnak. Egy jármű szerepelhet két különböző azonosítóval is különböző helyzetekben, ami feleslegesen növeli tovább az egyébként is tetemes adatmennyiséget. Problémát okozhat továbbá, hogy a különböző eszközök különböző természetű adatokat szolgáltatnak, esetleg különböző időközönként, ezért ezeket aggregált formában egy közös modellben kell ábrázolni és feldolgozni. Például rögzíthetik járművek rendszámát, ha kilépnek vagy belépnek egy településre, egy másik forrásból pedig kaphatunk járművekben lévő *GPS* készülékből származó eredményeket. Ezen a ponton nehéz megállapítani, hogy redundánsak-e az adatok, illetve vélhetően a *GPS* készülékek sűrű adatot szolgáltatnak, míg a város szélén rögzített adatok nem. A feladat, hogy egy régió forgalmas városközi útvonalait térképezzük fel. Először meg kell határozni, mit is jelent a forgalmas és nem forgalmas. Célszerűnek tűnik valamilyen sűrűséget, vagy távolságot kihasználó *Crisp* osztályozó eljárás alkalmazása (hasonló megfontolásból, általában ilyen osztályozásokkal lehet a forgalmi adatokat tovább tömöríteni), mely zajszűrőként is funkcionál. Zajnak tekintjük a ritkás, nem forgalmas útszakaszokat, azok forgalmával az adott feladat szempontjából felesleges foglalkozni. Más fajta forgalmat jellemezhet egy olyan osztályozás, ami egy-egy fix pont körül egy alakzatban (kör, ellipszis) vizsgálja az objektumok mozgását. Például ha egy országúti kereszteződést vizsgálunk, forgalmasnak tekinthetjük, ha  $n$  darab egymást követő mért időben  $m$ -nél több jármű volt jelen, ha pedig ez az  $m$  darab jármű mindig azonos az  $n$  időben, akkor megfontolandó lehet elterelő építése a torlódás feloldása végett. Egy másik féle megközelítés, ha nem egy pontot tekintünk fixnek, hanem  $m$  objektumokat, amik együtt mozognak  $n$  időn keresztül. Ez a módszer nem kereszteződések, sokkal inkább utak, útvonalak vizsgálatára alkalmas.

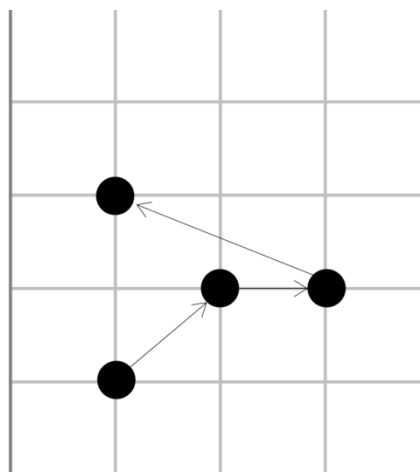
Általában a részlet gazdag hálózatokat, ahol az egyes objektumok viselkedése hangsúlyos, mikro-hálózatoknak nevezzük, ahol pedig a topológia a hangsúlyos, az egyes objektumok viselkedése konstansnak tekinthető, makro-hálózatnak. Mikro-hálózatok olyan kérdésekre adhatnak választ, hogy egy város egy pontján történt baleset milyen hatással van a hálózat egészére, még a makro-hálózatok például arra lehetnek alkalmasak, hogy a helyközi forgalom természetét vizsgáljuk. Természetesen a példaként említett feladat, mikro-hálózatokkal is értelmes, viszont egyes individuumok sok - a feladat

szempontjából - zajszerű információt hordoznak. Aki ismeri a várost, kis kerülőutakat választhat, aki pedig először jár arra, a nagyobb utakat, vagy a toronyiránti utakat választja. A részletesebb úthálózat miatt, az útvonalválasztás is sokkal komplexebb feladat lehet.

### III.1) Trajektóriák

Ha a modellben szerepel egy objektum azonosítója, pozíciója és a mérés ideje, akkor képezhetők ún. trajektóriák. Trajektóriának azokat az irányított (gráfelméleti) utakat nevezzük, amiknek pontjai az objektum pozíciói, és két pont akkor és csak akkor van összekötve, ha a mérési időt tekintve az egyik rákövetkezője a másiknak és a későbbi idő felé van irányítva az él. Jellemzően, az egy időben mért adatok mennyisége elhanyagolható az összes mért adathoz képest, ezért célszerű az egyes síkszeleteket külön-külön feldolgozni, osztályozni, majd egy újabb osztályozással időben is vizsgálni az adatokat. Világos, hogy a síkokban azonos objektumokból képzett osztályok továbbra is trajektóriát alkotnak. Az általam implementált eljárás is ezt a módszert alkalmazza: először sűrűség alapú osztályokat képez minden időben, majd ezekben az objektum halmazokban keres zárt, sűrű osztályokat.

| Time | X | Y |
|------|---|---|
| 1    | 1 | 1 |
| 2    | 2 | 2 |
| 3    | 3 | 2 |
| 4    | 1 | 3 |

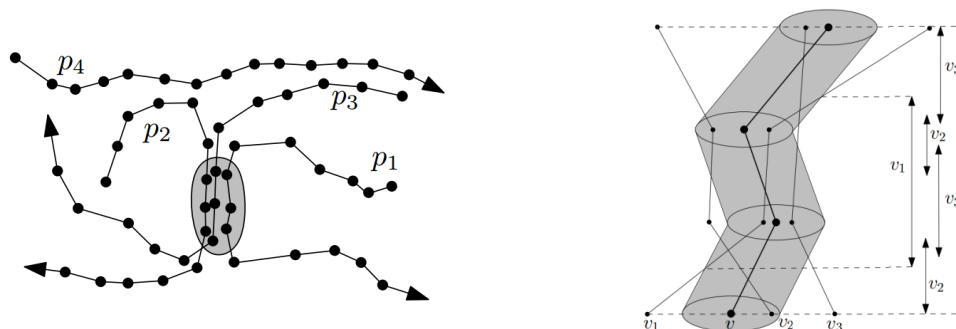


III-1 ábra

### III.2) Pattern Mining

A forgalmi adatok elemzésének egy fontos kérdése, hogy milyen feltételek mellett, hogyan mozognak együtt objektumok a hálózatokban. Az ilyen együttes mozgások a mintáknak, vagy angolul *pattern*-ek. A korai megközelítések a síkon vett pozíciójuk alapján kerestek csoportokat (olyan kérdésekre válaszoltak, mint „hány autó volt a Kossuth téren kedden?”). Ezek a módszerek viszont korlátozottan, vagy nem képesek tükrözni az objektumok általános viselkedését, azok típusosztályait megkülönböztetni. Egy másik értelmezése a problémának, ha az objektumok elhelyezkedése helyett a keresés az objektumok viselkedéseinek a hasonlóságát hivatott megtalálni. A fent említett három-dimenziós trajektória értelmezésben egy-egy síkra, mint az őt reprezentáló mérési idő időszelétére fogok hivatkozni. A következő két minta, ezekben az időszakokban próbál olyan köröket meghatározni, melyeknek az időben vett mozgása előre rögzített.

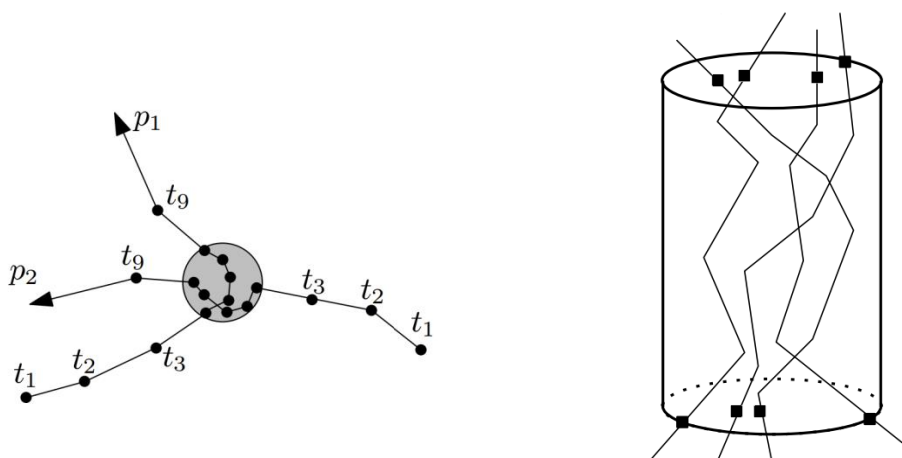
Flock(m, k, r): Adott  $n$  darab trajektória a síkon, egyenként  $T$  darab ponttal  $I$  időintervallumon. Ha  $I$  legalább  $k$  időegységből áll és minden időben legalább  $m$  darab pont van egy  $r$  sugarú kör belsejében, akkor ezek a trajektóriák  $I$  felett Flock-ot alkotnak. Ha minden időegységben ugyanaz az  $m$  pont alkotja a Flock-ot, akkor Fixed-Flock-ról beszélünk, ellenkező esetben Varying-Flock-ról. A III-2 ábrán a Flock minta síkvetülete, illetve térbeli projekciója látható.



III-2 ábra: Flock<sup>[3]</sup> síkvetülete és térbeli projekciója

Meet(m, k, r): Adott  $n$  darab trajektória a síkon, egyenként  $T$  darab ponttal  $I$  időintervallumon. Ha  $I$  legalább  $k$  időegységből áll és minden időben legalább  $m$  darab pont van egy rögzített  $r$  sugarú kör belsejében, akkor ezek a trajektóriák  $I$  felett Meet-et alkotnak. Ha minden időegységben ugyanaz az  $m$  pont alkotja a Flock-ot, akkor Fixed-Meet-ről beszélünk, ellenkező esetben Varying-Meet-ről. A III-3 ábra a Meet minta síkvetületét, illetve térbeli projekcióját mutatja.

Ezek a *minták* a térben hengeres testekként reprezentálódnak. A henger sugara  $r$ , magassága legalább  $k$  és minden síkmetszetben legalább  $m$  pont van. A [3] cikk, ennek a két alakzatnak, adott  $r$ ,  $k$ ,  $m$  paraméterek mellett, az időben leghosszabb előfordulását keresi. Azonban a pontos eredmény kinyerése éppen olyan nehéz feladat, mint egy gráfban a legnagyobb klikket megtalálni, ezért csak közelítő algoritmusokat ad.



III-3 ábra: Meet<sup>[3]</sup> síkvetülete és térbeli projekciója

Egy másik “családja” a mintáknak, ahol az objektumok csoportos mozgására következtetünk. Az előző megközelítés magába foglalt egyfajta osztályozást (a kör maga), ez viszont már csak a képzett objektumosztályokkal dolgozik.

Legyen:

$$O_{DB} = \{o_1, \dots, o_z\}$$

Az objektumok listája

$$T_{DB} = \{t_1, \dots, t_n\}$$

Az időegységek listája

| $O_{DB}$ | $T_{DB}$ | x   | y   |
|----------|----------|-----|-----|
| $O_i$    | $T_j$    | ... | ... |
| ...      | ...      | ... | ... |

Sík-idő adatbázis (Spatio-temporal database)

$$\varepsilon, \min_t, \min_c$$

Előre rögzített természetes számok

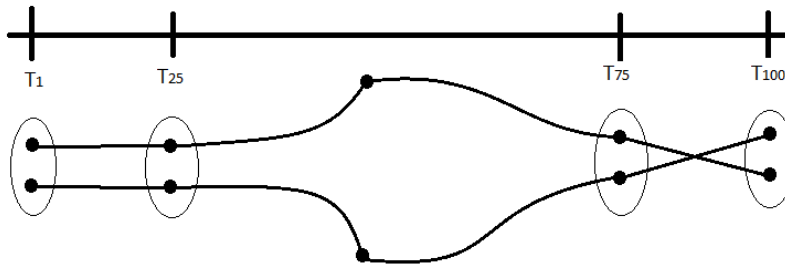
$$\min_{wei}$$

Előre rögzített valós szám

Ekkor értelmezhetjük a következő minták definícióit:

Swarm( $O \subseteq O_{DB}, T \subseteq T_{DB}$ ): O objektumok T időkbén Swarm-ot alkotnak, ha

1.  $\forall t_{ai} \in T : \exists c \geq 0$
2.  $|O| \geq \varepsilon$
3.  $|T| \geq \min_t$



III-4 ábra: Swarm mintázat

Closed Swarm( $O \subseteq O_{DB}, T \subseteq T_{DB}$ ): O objektumok T időkbén Closed Swarm-ot alkotnak, ha

1. (O,T) swarm
2.  $\neg \exists (O', T) : (O', T) \text{ swarm} \wedge O \subset O'$
3.  $\neg \exists (O, T') : (O, T') \text{ swarm} \wedge T \subset T'$

Convoy( $0 \subseteq O_{DB}, T \subseteq T_{DB}$ ):  $O$  objektumok  $T$  időkbén Convoy-t alkotnak, ha

1.  $(O, T)$  swarm
2.  $\forall i \in [1..|T|]$ -re  $t_{ai} \in T$ -nek  $t_{ai+1} \in T$  rákövetkezője

Értelmezzük ekkor a következő jelölést:

$$T_s = \{s_1, \dots, s_l\} : \forall s_i \text{ convoy}$$

Group Pattern( $O, T_s$ ):  $s_i$  convoyok Group Pattern-t alkotnak, ha

1.  $(O, T_s)$  Closed Swarm
2.  $|T_s| \geq \min_c$
3.  $\forall s \in T_s : 0 \subseteq s$
4.  $\frac{\sum_{i=1}^{|T_s|} |s_i|}{|T_{DB}|} \geq \min_{wei}$

Látszik, hogy mind a Meet, mind a Flock egy-egy speciális esete a Convoy-nak. A definíciókból az is látszik, hogy elég a Closed Swarmokat megtalálni és azokból „kikeresni” a többi mintát.

| Object_ID | $t_1$ | $t_2$    | $t_3$ |
|-----------|-------|----------|-------|
| 1         |       |          |       |
| 2         |       | Nem zárt |       |
| 3         |       |          |       |

| Object_ID | $t_1$ | $t_2$ | $t_3$ |
|-----------|-------|-------|-------|
| 1         |       |       |       |
| 2         |       |       |       |
| 3         |       |       | Zárt  |

III-5 ábra

A II.2) pontban definiált zárt, sűrű elemhalmazokból könnyen adódik a fent definiált swarm minta, lévén hogy a két definíció gyakorlatilag megegyezik. Legyen ekkor a reprezentáció:

$$C_{DB} = \{C_1, \dots, C_k\} \quad , \text{ ahol}$$

$$C_i = \{c_{i_1} t_i, \dots, c_{i_g} t_i\} \quad , \text{ ezeknek az osztályoknak egy halmaza } t_i \text{ időben}$$

$$\vartheta = \{c_{t_{a1}}, \dots, c_{t_{ap}}\} \quad , \text{ elemhalmaz aminek}$$

$$T_\vartheta = \{t_{a1}, \dots, t_{ap}\} \quad \text{az élettartama}$$

Bevezetünk két segédfüggvényt, melyek rendre

$$O(\vartheta) = \bigcap_{i=0}^p c_{t_{ai}} \quad \vartheta \text{ elemeinek a metszete}$$

$$\sigma(\vartheta) = |O(\vartheta)|$$

Ezekkel a jelölésekkel már leírható a fenti négy Pattern, illetve a sík-idő adatbázis a következőképpen:

|          |       |           |     |     |
|----------|-------|-----------|-----|-----|
| $T_{DB}$ |       | $t_i$     |     | ... |
| $C_{DB}$ |       | $C_{k,l}$ | ... | ... |
| $O_{DB}$ | $o_j$ | ...       | ... | ... |
|          | ...   | ...       | ... | ... |

Cluster mátrix

A zártság tulajdonságai jól láthatóak Cluster mátrixos ábrázolással, ahogy azt a III-5 ábra is érzékelteti.

Swarm( $O(\vartheta)$ ,  $T_\vartheta$ ):

1.  $\sigma(\vartheta) \geq \varepsilon$
2.  $|\vartheta| \geq \min_t$

**Bizonyítás:**

Konstrukció szerint  $\sigma(\vartheta) \geq \varepsilon$  és  $\sigma(\vartheta) = |O(\vartheta)|$ , tehát  $|O(\vartheta)| \geq \varepsilon$ , illetve  $|\vartheta| \geq \min_t$  és  $|\vartheta| = |T_\vartheta|$ , tehát  $|T_\vartheta| \geq \min_t$ . Továbbá  $\forall T_{aj} \in T_\vartheta$ ,  $O(\vartheta) \subseteq c_{t_{aj}}$ , tehát minden mérési időben van egy osztály, ami tartalmaz minden elemet  $O(\vartheta)$ -ből. Ezek alapján  $(O(\vartheta), T_\vartheta)$  kielégíti a *Swarm* definícióját.

□

Closed Swarm( $O(\vartheta)$ ,  $T_\vartheta$ ):

1.  $(O(\vartheta); T_\vartheta)$  swarm
2.  $\neg \exists \vartheta' : O(\vartheta) \subset O(\vartheta'); T_{\vartheta'} = T_\vartheta \wedge (O(\vartheta'); T_\vartheta)$  swarm
3.  $\neg \exists \vartheta' : O(\vartheta') = O(\vartheta); T_\vartheta \subset T_{\vartheta'} \wedge (O(\vartheta); T_{\vartheta'})$  swarm

**Bizonyítás:**

A konstrukció szerint  $(O(\vartheta), T_\vartheta)$  swarm, illetve ha  $\neg \exists \vartheta' : O(\vartheta) \subset O(\vartheta')$ , akkor  $T_\vartheta = T_{\vartheta'}$  és  $(O(\vartheta'); T_\vartheta)$  swarm teljesül,  $(O(\vartheta); T_\vartheta)$  nem bővíthető új elemmel. Ez alapján teljesíti az elemekre való zártság tulajdonságát. Továbbá, ha  $\neg \exists \vartheta' : O(\vartheta') = O(\vartheta)$ , akkor  $T_\vartheta \subset T_{\vartheta'}$  és  $(O(\vartheta); T_{\vartheta'})$  swarm teljesül,  $(O(\vartheta); T_\vartheta)$  élettartama sem bővíthető. Ezzel teljesíti az időben való zártságot is. Következésképp,  $(O(\vartheta), T_\vartheta)$  olyan swarm, hogy időben és elemeire nézve is zárt, ami éppen azt jelenti, hogy  $(O(\vartheta), T_\vartheta)$  closed swarm.

□

### Convoy( $O(\vartheta)$ , $T_\vartheta$ ):

1.  $(O(\vartheta); T_\vartheta)$  swarm
2.  $\forall i \in [1..|T|]$ -re  $t_{a_i} \in T$ -nek  $t_{a_{i+1}} \in T$  rákövetkezője

### Bizonyítás:

Konstrukció szerint  $(O(\vartheta), T_\vartheta)$  swarm. Továbbá, ha  $\vartheta$  kielégíti a 2. feltételt, akkor  $\vartheta$  élettartama megszakítás nélküli. Tehát,  $(O(\vartheta), T_\vartheta)$  convoy.

□

### Group Pattern( $O(\vartheta)$ , $T_s$ ):

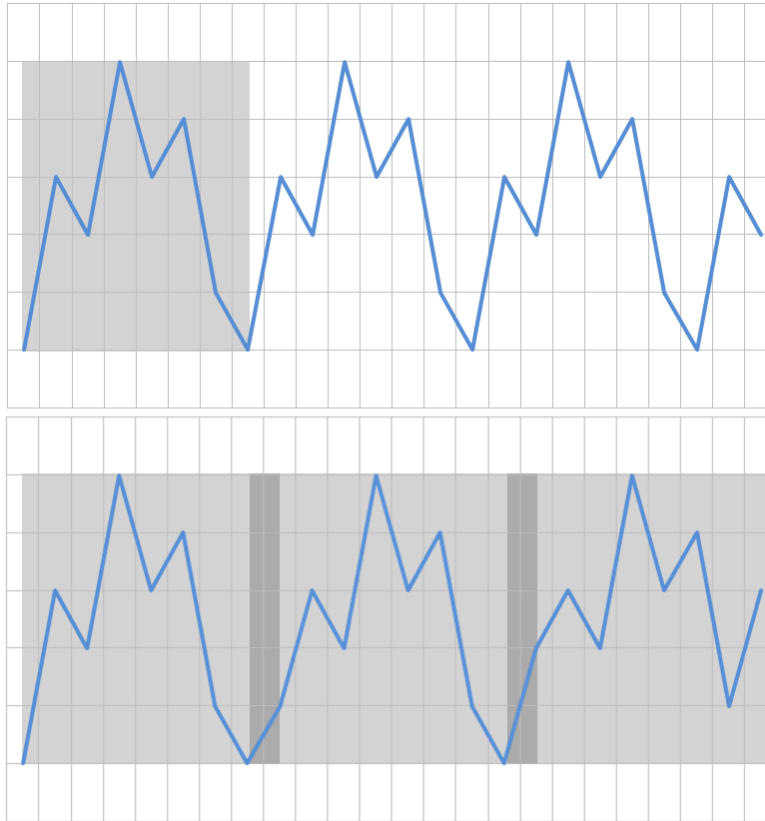
1.  $|T_s| \geq \min_c$
2.  $\forall s_i : s_i \subseteq T_\vartheta \wedge |s_i| \geq \min_t$
3.  $\bigcap_{i=1}^n s_i = \emptyset \wedge \bigcap_{i=1}^n O(s_i) = O(\vartheta)$
4.  $\forall s \notin T_s$ , ahol  $s$  convoy  $\wedge O(\vartheta) \not\subseteq O(s)$
5.  $\frac{\sum_{i=1}^{|T_s|} |s_i|}{|T|} \geq \min_{wei}$

### Bizonyítás:

Ha  $|T_s| \geq \min_c$  akkor legalább  $\min_c$  darab egymást követő  $s_i$  intervallum van  $T_s$ -ben. Ha  $\forall s_i : s_i \subseteq T_\vartheta$ , akkor  $O(\vartheta) \subseteq O(s_i)$ , illetve ha  $|s_i| \geq \min_t$  is teljesül, akkor  $(O(\vartheta), s_i)$  convoy. Ezek alapján  $T_s$  convoy-ok halmaza  $O(\vartheta)$ -ból ráadásul, ha  $\bigcap_{i=1}^n s_i = \emptyset$  akkor diszjunkt convoy-ok halmaza. Továbbá, ha  $\forall s \notin T_s : s$  convoy és  $O(\vartheta) \not\subseteq O(s)$ , akkor  $\neg \exists T_s$ , hogy  $T_s \subset T_{s'}$  és  $\bigcap_{i=1}^{|T_{s'}|} O(s_i) = O(\vartheta)$ . Tehát,  $(O(\vartheta), T_s)$  nem bővíthető további convoy-okkal. hasonlóan, ha  $\bigcap_{i=1}^n O(s_i) = O(\vartheta)$  akkor  $(O(\vartheta), T_s)$  további objektumokkal sem bővíthető. Következésképp,  $(O(\vartheta), T_s)$  diszjunkt convoy-ok closed swarm-ja, mert  $|O(\vartheta)| \geq \epsilon$ ,  $|T_s| \geq \min_c$  és  $(O(\vartheta), T_s)$  nem bővíthető (zárt). Végül, ha  $(O(\vartheta), T_s)$  teljesíti az 5. feltételt is, akkor érvényes group pattern.

□

### III.3) Periodicitás



III-6 ábra

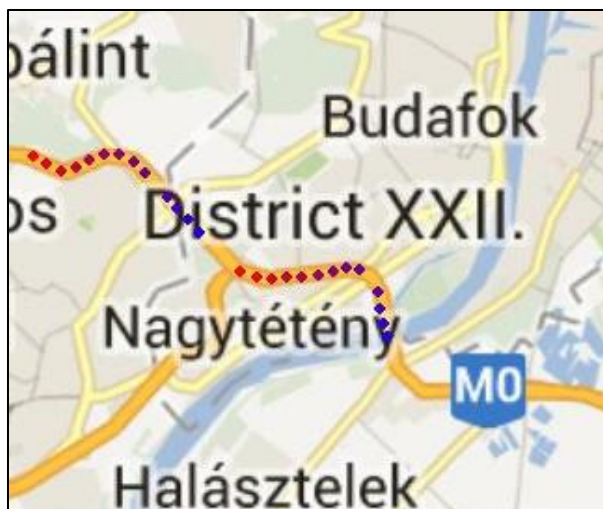
Perióduskeresésnek a ciklikusan előforduló mintákat hívjuk. Az egyes minták minden eleme hasonló, a többi minta megfelelő elmével valamilyen osztályozás alapján. A probléma kezdeti megközelítései tökéletes és teljes ciklikusságot voltak képesek találni. Tökéletes abban az értelemben, hogy időben közvetlenül követik egymást. Például egy vizsgált objektum, minden nap azonos órában indul azonos úton ugyanoda, mint máskor. Ekkor az objektum minden napja tökéletesen megegyezik. Ez a viselkedés általában nem életszerű. Még ha időben és térben is megengedett némi tévedés és csak órára pontosan vizsgáljuk az objektum minta ciklusának 100 méteres körzetét, akkor is elrontja a ciklikusságot, ha egyszer kitérőt tesz. Ezt egy folytonos függvény viselkedésével mutatja a *III-6 ábra*. Az első függvény teljesen ciklikus, minden pontját egy periódussal eltolva, a függvényre esik. A második függvénynek viszont van egy nem ciklikus része, ami bár periodikusan fordul elő, mégis elrontja a mintát. Számos algoritmus képes ezt a problémát úgy megoldani, hogy az adott napot nem ítéli ciklikusnak, viszont a környezetét igen, így feleslegesen két részre bontva az eredményeket. A problémásabb eset viszont, ha a vizsgált objektum például egy futár járműve, ami egy intervallumban minden nap máshogy viselkedik. Azokat már nem képesek a teljes ciklikusságot vizsgáló algoritmusok megtalálni, a futár viselkedését kaotikusnak ítélnék. Hasonló példák motiválták a *szegmens alapú periodicitás* fogalmának bevezetését. Ekkor azzal az engedménnyel élünk, hogy a vizsgált időintervallumban (a példában egy nap) lehetnek ciklikus és nem ciklikus szegmensek is. Bár néhány megközelítés módosítható az ilyen típusú periódusok

detektálására, de azok jellemzően reménytelenül elbonyolítják az eljárást, futási idejük négyzetes. Ilyen megközelítés, ha a vizsgált időszaknak egyszerre csak egy részét vizsgáljuk, és azokat tekintjük potenciális periódusoknak. Manapság már léteznek hatékony algoritmusok, amik képesek a szegmens alapú periódusokat hatékonyan megtalálni adatkockák, bit-map-ek, illetve apriori eljárások alkalmazásával, de még nem találkoztam olyannal, ami képes lett nem előre rögzített intervallumokat hatékonyan vizsgálni.

Az általam javasolt eljárás nem általános, a sík-idő adatokon kívüli jelentése nem szándékolt. Ebben a témakörben jellemző megoldás a már megtalált minták vizsgálata, vagy a periódusok közvetlen bányászata. Ekkor viszont el kell döntenie egy-egy periódusról, hogy ciklikus-e. Ehelyett a trajektóriákat egyesével vizsgálva és azokat osztályozva, a probléma jelentősen redukálható. Ha találunk olyan  $r_{sup}$  sugarú kört, amiben legalább  $c_{sup}$  darab előfordulása van. Minden pontot, amit nem tudtunk ilyen körökbe sorolni mellőzünk, ciklikusság szempontjából nem hordoznak információt. Ez általában jelentős csökkentése az input adatbázis méretének. Az ezekben az adatokban talált minták periodikusak és előfordulásuk nem függ semmilyen előre meghatározott paramétertől (bár annak eldöntése, hogy van-e benne ilyen speciális periódus, az előfordulások számával arányos feladat). A jövőben még vizsgálat tárgyát képezheti, hogy a szűréskor használt köröket, mint új hiperpontokat felhasználva a későbbi mintakeresésben, a talált minták miben különböznek, ha különböznek. A körök lényegesen kevesebben lehetnek a szűrt adatoknál is, tovább gyorsítva az eljárás költséges fázisát.

### III.4) Várostervezési szempontok

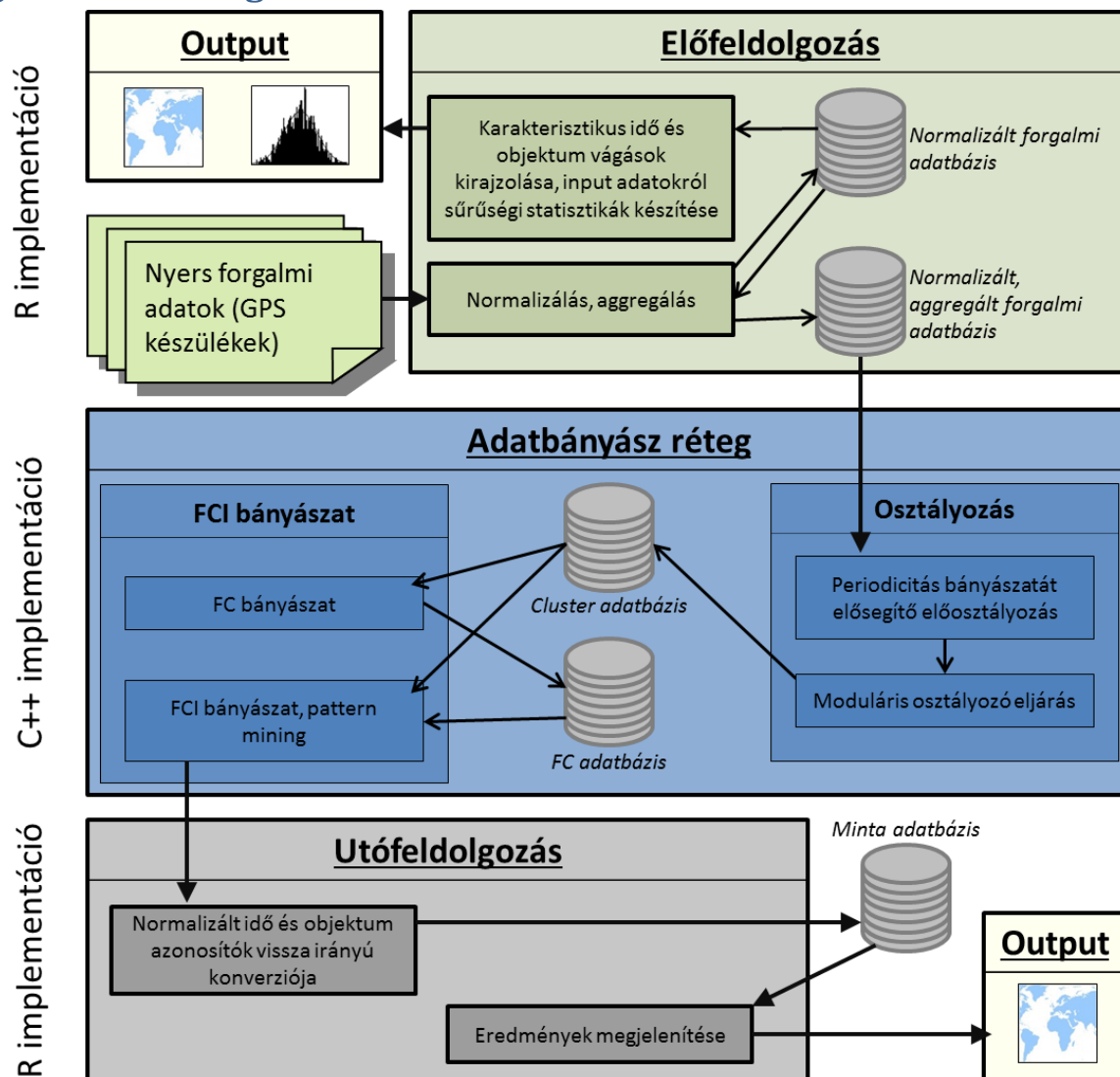
A III.2) részben tárgyalt *meet*, illetve *flock* minta alkalmas torlódások reprezentálására egy város hálózatában. Ez az információ –amellett, hogy egy hálózat kis kapacitású, de nagy terhelésű pontjait megtalálja- betekintést adhat váratlan helyzetek (balesetek), hatásainak a vizsgálatára a hálózaton. A trajektoriális ábrázolás miatt ez az információ vektoriális, tehát a torlódás irányát is meg lehet határozni. A *meet*, illetve a *flock* is a *convoy* minta egy-egy speciális esete, tehát az is alkalmas reprezentációja a torlódást okozó utaknak, csomópontoknak. A *closed swarm* minta, azonos objektumokból álló *convoy*-ok sorozata az időben. Segítségével kimutathatóak olyan jelenségek, hogy egy kialakult torlódás a város mely pontjain okoz további torlódásokat. Tegyük fel, hogy a vizsgált város egy kereszteződésében baleset történt és torlódás alakult ki. Ekkor dönthetnek úgy a torlódás résztvevői, hogy a zsúfolt, vagy egy alternatív útvonalat választanak céljuk eléréséhez. Ha elég sokan választanak alternatív, kis kapacitású utakat, azok is torlódni kezdenek. A *closed swarm* épp az ilyen jellegű „görgetett” torlódások detektálására szolgálhat. Ha a vizsgált időszak elég hosszú, akkor sok hasznos információ kinyerhető, a minták periodicitásából. Ha egy csomópont rendszeresen torlódik és kimutatható, hogy a város több pontján további torlódásokat okoz (például, mert délután egy népszerű bevásárlóközpontból sokan próbálnak távozni), fontos várostervezési szempont lehet, az adott útszakasz tehermentesítése. Makro hálózatok esetében, ha az országba egyszerre sok tehergépjármű lép be, torlódást okozhatnak, jelentősen gátolva az adott terület átlagos forgalmát. Ez a probléma könnyen orvosolható elkerülő építésével.



**III-7 ábra: Példa két torlódásra, az első torlódás ideiglenesen megszűnt, majd újra torlódás állt elő**

A torlódás fogalmának általánosításával, a bányászott minták reprezentálhatják az emberek szokásait. Legyen a torlódásnak felső határa is (eddig az egy helyen elég sok objektumot hívtuk torlódásnak, ezentúl elég kevesen is kell lenniük). Ha azokat a mintákat vizsgáljuk, amik „egész sokan” vannak, de nem okoznak fennakadást a hálózatban, olyan útvonalak kezdenek kirajzolódni, mint például lakónegyed-munkahely, munkahely-bevásárlóközpont. Ha ezekben megtaláljuk a periódusokat, akkor szokásokra utalhatnak. Tegyük fel, hogy egy megtalált minta a következő: 1 kilométeres körzetben 20 jármű reggel nyolckor egyszerre halad egy lakónegyedből egy gyárba, majd délután négytől az előző példában említett torlódást okozó távoli bevásárlóközpontba. Feltehető, hogy a 20 objektum vezetője, utasai a lakónegyed lakói és a gyárban dolgoznak, vagy ahhoz közel. Feltehető továbbá, hogy délutánonként az adott távoli bevásárlóközpontban vásárolnak. Ha elég sok ilyen jellegű információ összegyűlt, meg lehet határozni egy új bevásárlóközpont optimális helyét, úgy hogy az új bevásárlóközpont vonzáskörzetében már adott egy minimum forgalom (feltéve, hogy az emberek tudnak róla és számukra ugyanolyan előnyös), ezzel enyhítve a régi bevásárlóközpont által okozott torlódást. Ha e mögött az eljárás mögött lenne egy adatbázis arról, hogy az egyes minták közelében milyen jellegű helyek vannak (iskolák, stb.), akkor automatikusan bányászhatóak lennének a város lakóinak szokásai, illetve javaslatokat adhatna építkezési beruházásokra. Különösen hasznos lehet egy hasonló eljárással támogatott tanácsadó rendszer az okos városok irányítására. Bár a jelenlegi kutatások inkább célkitűzések, alapelvek meghatározására irányulnak, megfontolandó lehet felmérni a ma is elérhető eszközök által nyújtott lehetőségeket.

#### IV) *Get\_Move* algoritmus és keretrendszere



IV-1 ábra: Az eljárás architektúrája

Az eljárás megírásának motivációja a III.4) pontban tárgyalt szokáselemzés, automatikus forgalomelemző és várostervezési döntéseket elősegítő program készítése. Az eljárás jelenleg a nem általánosított torlódásdetektálásra képes, a periodikus minták keresése rövidtávú cél. A keretrendszert R-ben valósítottam meg, mert sok hatékony és elismert statisztikai és megjelenítő eljárást implementáltak benne. Megjelenítésre a *ggmap* és *ggplot2* csomagokat használom. Az eljárást magát C++-ban implementáltam. A teljes eljárás architektúrája a IV-1 ábrán látható.

## IV.1) Előfeldolgozás

Az előfeldolgozás több opcionális lépésből áll:

1. Az input adatok aggregálása, egységesítése
2. Az egységesített adatokról alap statisztikák készítése
3. Adott paraméterek közti sík és időmetszetek kirajzolása
4. Az utófeldolgozást gyorsító map file-ok előállítása

Az 1. Pont adott időszeletekre szorítja a különböző sűrűséggel mért adatokat, majd azok időit természetes számokra vetíti (ezt a megfeleltetést egy map\_file-ban tárolja). A map\_file-ra azért van szükség, mert a kód lehetőséget ad az aggregációs lépés elhagyására. Ezután a különböző objektumazonosítókat is a természetes számokra vetíti.

```
1 ##### Features
2 mapping=TRUE
3 analyze=TRUE
4 plots=TRUE
5
6 cname="masik_budapest"
7 input_file="fcd_20100901_075416_budapest.txt"
8 separator=";"
9 plot_by_id_TH=20
10 plot_by_time_TH=30
11 zoom_=10
12 town="Budapest"
13 ##### End of features
14
15 library(ggplot2) # Plotting maps.
16 library(ggmap) # Google maps.
17 FCD <- read.csv(input_file, header=FALSE, sep=separator)
18 start__time__ <- Sys.time()
```

### IV-2 ábra

A 2. Pontban hisztogramok készülnek az egy időben egyszerre mozgó objektumok számáról, illetve az egy objektum élettartamáról a hálózatban. Ezekről átlagok, minimumok és maximumok is számolódnak. A tesztelés során használt adatok eleje és vége (időben) ritkás volt, nem volt bennük releváns adat, de az eljárást érezhetően visszafogta. Ezek a statisztikák elősegítették az adatok szűrésének és az aggregálás során használt idő optimálisabb meghatározását.

A 3. Pontot is hasonló indokok motiválták, mint a 2.-at. Térképen megjeleníti azokat az objektumokat, amik vagy elég sokan voltak egy időben, vagy elég hosszú ideig éteztek (objektumonként, illetve időnként egy-egy képként menti el).

A 4. Pont elmenti az objektumok egyes időpillanatban vett koordinátáit, az idő és objektumazonosítók map\_file-jait és végül létrehozza az eljárásnak megfelelő formátumú input fájlt.

Az R-ben megvalósított előfeldolgozás paraméterei a IV-2 ábrán láthatóak. A mapping az utófeldolgozást gyorsítja, az analyze alap statisztikákat készít, a plot pedig plot\_by\_id\_TH és plot\_by\_time\_TH feletti előfordulású időszakokat és objektumokat rajzolja ki a térképre, amit a town és zoom paraméterek határoznak meg.

## IV.2) Osztályozás

```

1 #include <math.h>
2 #include <string>
3 #include <vector>
4 #include <iostream>
5
6 using namespace std;
7
8 class _DBScan{
9     vector < vector < double > > O_i; // Osztalyozando adatbazis matrixa (sikkeli koordinatakk) [id][long][lat][visited]
10    double Eps; // A vizsgalt pont koruli kereses kornyezet sugara
11    unsigned int MinPts; // A kereses kornyezet minimalis surusege
12    bool DEBUG; // Debug uzenetek
13    int DISTANCE; // Euclidian - 1, Polar - 2
14
15    // Euklideszi tavolsagfuggveny
16    double Euclidian_distance(const double &x1, const double &x2, const double &y1, const double &y2){ ... }
17
18    //polaris tavolsagfuggveny
19    double polar_distance(const double &x1, const double &x2, const double &y1, const double &y2){ ... }
20
21    double polar_distance2(const double &x1, const double &x2, const double &y1, const double &y2){ ... }
22
23    // Egy adott objektumnak a matrixban elfoglalt helye szerinti indexet adja vissza ID alapjan
24    int Get_index(const int &id){ ... }
25
26    // Visszaadja egy adott pont kornyezeteben levo pontokat
27    vector <int> RegionQuery (const double &id, const double &x, const double &y){ ... }
28
29    vector <int> ExpandCluster(const vector <double> &P, const vector <int> &nei){ ... }
30
31    public:
32    // A C_I-ben tarolodik el az osztalyozas eredmenye
33    vector < vector < int > > C_i;
34
35    _DBScan(const vector < vector < double > > &O, const double &E, const int &M, const int &Dis, const bool &D=false){ ... }
36
37    void Scan(const int &time){ ... }
38
39 };

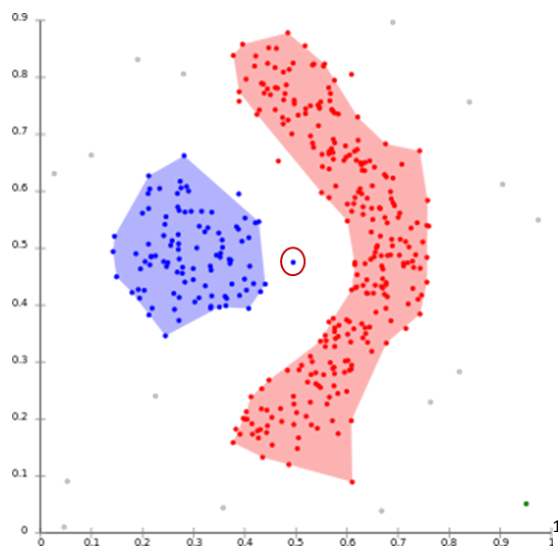
```

### IV-3 ábra: A DBscan implementációja. két távolságot (polár, euklideszi) implementáltam.

Az *Fuzzy C-Means* algoritmusnak, mint hatékony osztályozó eljárásnak, több *Crisp* változata is létezik. Működésük hasonló, egy jól meghatározott mértékig közelíteni próbálj az optimális *centroidokat*. Az eljárás egy gyengesége, hogy előre meg kell adni az osztályok számát. A család tagjai között a fő különbség a közelítés gyorsasága mellett, az optimális megoldás megtalálásának képessége. Rosszul lerakott kezdő *centroidok* esetén előfordulhat, hogy egy-egy *centroid* nem tud elég gyorsan, vagy egyáltalán a „helyére találni”. A Fuzzy változathoz képest erős megszorítás, hogy az osztályok diszjunktak, így az osztályok számának rossz választása különösen pontatlan eredményeket adhat, még helyesen működő osztályozás mellett is. Egy másik Crisp osztályozó algoritmus a *DBScan*<sup>[5]</sup>, melynek két

előnye van a *K-means* családdal szemben: nem kell előre meghatározni az osztályok számát és jóval egyszerűbb. A *DBScan* két paramétert vár: MinPts és Eps, melyek rendre egy pont környezetében lévő azonos osztálybeli pontok minimális száma, illetve az adott pont környezetének sugara. Azok a pontok tartoznak egy osztályba, melyek Eps sugarú környezetében, legalább MinPts darab azonos osztálybeli pont van. Az eljárás (futás szempontjából), nem érzékeny az osztályozó algoritmusra, egyaránt működik crisp és fuzzy osztályozókkal, de az osztályozások jellegének különbségei miatt az eredmények mást jelenthetnek.

A *DBScan*-t C++-ban implementáltam, érdemi módosítást nem végeztem rajta. Egyszerre egy időszeletet dolgoz fel, ahol aránylag kevés adattal kell dolgoznia, így ez a része az eljárásnak az adatbázis időbeli hosszával arányos. Kimenetként egy általam kitalált, a memóriában felépített adatmodellben tárolja. Az adatmodell célja, hogy a már egyszer osztályozott adatokban gyorsabban lehessen műveleteket végezni, tulajdonságaikat hatékonyan meghatározni. Három osztályt használ: Object, Cluster, Time\_stamp. Ezek ábrázolására valamilyen formában szükség van, mert az általuk hordozott információkban keres az eljárás. Mindhárom láncolt, megkönnyítve az olyan típusú statisztikák előállítását, mint Cluster osztályok számossága az idő (objektumok) függvényében. Random access elérésre ritkán van szükség, az eljárás sorban vizsgálja az adatokat. Az egyes osztályok megvalósítását a következő pontban tárgyalom.



IV-4 ábra

A DBscan – működéséből adódóan – időben osztott és eredményei diszjunktak. Az különböző időkhöz tartozó feldolgozások párhuzamosan futtathatók. Ez az algoritmus gyors és egyszerű, de esetenként lehet pontatlan. A működésére és a pontatlanságára látható egy példa a IV-4 ábra. A bekarikázott pont egy kék osztálybeli pont környezetében van, de ő maga nem rendelkezik elég szomszédal, ezért távol esik a legtöbb ponttól.

<sup>1</sup> kép forrása: [en.wikipedia.org/wiki/File:DBSCAN-density-data.svg](https://en.wikipedia.org/wiki/File:DBSCAN-density-data.svg)

### IV.3) FCI bányászat és pattern mining

Inputként a korábban feltöltött adatmodell szolgál, az kimenetet pedig fájlban menti el az eljárás. Az output fájl az egyes FCI-ket az azokban résztvevő objektumokkal és az előfordulási időkkal reprezentálja. Az eljárás időben osztható, ha elmentünk minden zárt elemhalmazt. A különböző futások eredményei összefűzhetők. Az eltérő osztályok bekerülnek az eredményhalmazba, míg az azonos osztályok az időbeni előfordulásaiknak uniójával kerülnek be. A képzett eredményhalmaz minden eleménél meg kell vizsgálni, hogy sűrű-e majd csak ez után kell a mintákat kinyerni belőlük.

#### a) Adatmodell

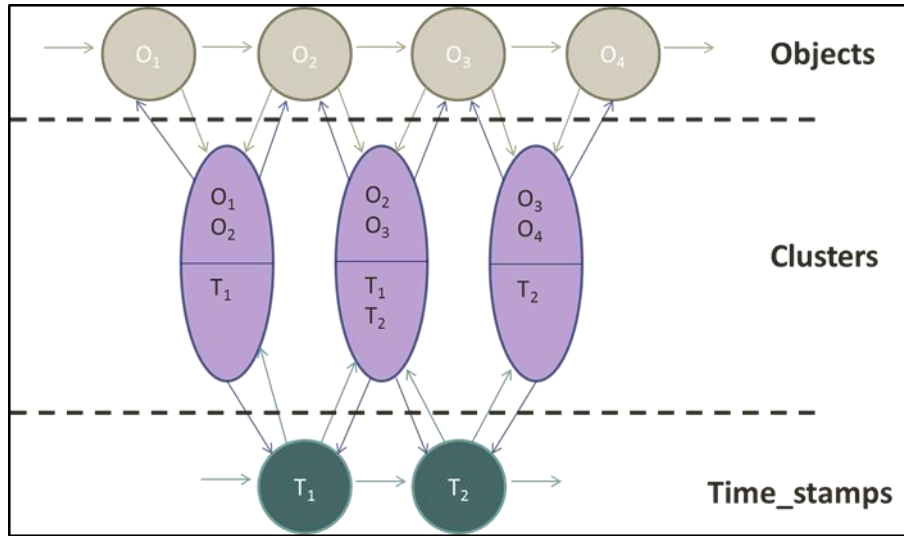
Time\_stamps osztály az egyes mérési idők láncolt listájának és feldolgozását elősegítő függvények gyűjteménye. Egy mérési időt egy struktúraként ábrázoltam, ami tartalmazza az azonosítóját, a rákövetkezőjére egy mutatót, illetve egy `std::unordered_set` típusú mutató tömböt, ami azokra az Cluster osztályokra mutat, amik az adott időben előfordultak. Egy Time\_stamps osztály adatai: egy mutató az első és az utolsó idő struktúrára, illetve egy előjel nélküli egész, ami az idők számosságát jelöli. Függvényei a konstruktorokon és a felépítést és bejárást segítő eljárásokon kívül:

- Check() - bejárja az idők láncolt listáját és megszámolja, hogy Length sokan vannak-e
- Clean() - Kitörli azokat az időket, amikben nem fordul elő Cluster

Az Object osztály – nevétől és az alap struktúrától eltekintve – teljesen megegyezik a Time\_stamps osztállyal. Az egy osztályt reprezentáló struktúra is csak nevében tér el az egy időt reprezentáló struktúráétól.

A Clusters osztály eltér az előző kettőtől, de szintén egy alap struktúra láncolt listájával dolgozik. Az alap Cluster struktúrának nincs azonosítója, viszont tartalmaz egy-egy `std::unordered_set` konténert, amiben azokat az idő és objektum mutatókat tárolja, amik és amikor reprezentálják az adott clustert. Ezek után a Clusters osztály tartalmaz egy mutatót az első és utolsó osztályra, egy hosszú tároló előjel nélküli egészet és egy logikai változót, ami azt a tényt tárolja, hogy korábban optimalizálva lett-e az osztály. Tagfüggvényei –szintén eltekintve a felépítést, bejárást elősegítő, megvalósító függvényektől – a következők:

- Tau(...) - Visszaadja egy Cluster lista előfordulásait és lezártját, oly módon, hogy a mutatókon keresztül bejárja a lezártban potenciálisan szereplő objektumokat és azok előfordulásainak (Cluster mutató listáinak) metszete a lezárt. A vizsgált clusterek előfordulásainak (Time\_stamp listájának) uniója pedig az előfordulás
- Organize() - Ez az eljárás tölti fel az idő és objektum osztályok cluster listáit, majd meghívja azok Clean() eljárásait
- Check() - Bejárja a Clusterek mutatólistáit és ellenőrzi, hogy szerepel-e bennük visszamutató, illetve hogy Length számosságú Cluster van-e



IV-5 ábra: Az implementált adatmodell

#### b) LCM algoritmus

Az [5] cikkben két LCM megközelítést tárgyalnak, amelyből az eredeti *Get\_Move* algoritmusban nem azt implementálták, mint amit én. Ennek okait ebben a fejezetben tárgyalom. Az LCM algoritmusok közös vonása, hogy definiálnak egy gyerek-szülő relációt a zárt, sűrű elemhalmazok közt. Ez a reláció egy fa szerkezetet határoz meg, amin mélységi bejárást alkalmaznak. Legyenek

$$C_{DB} = \{C_1, \dots, C_k\}$$

, a III.2) fejezetben definiált halmaza az összes vizsgált elemhalmaznak (osztálynak)

$$X = \{x_1, \dots, x_n\}$$

, egy elemhalmaz, ahol

$$\forall i \in \{1..n\}: x_n \geq x_i \geq x_1$$

$$\text{Tail}(X) = x_n$$

$$X(i) = X \cap \{1, \dots, i\}$$

$$X = Y(i)$$

,  $Y$  az  $X$  prefixe és  $i = \text{Tail}(X)$

$i(X)$  olyan minimális elem, hogy

$$T(X) = T(X(i)), \text{ de } T(X) \neq T(X(i-1)).$$

Ha  $P$  egy gyerek-szülő reláció, úgy hogy  $X = P(Y)$ , akkor és csak akkor, ha  $X = Y / \text{Tail}(X)$ . Ez a reláció az *FCI*-ken meghatároz egy prefix fát (trie),  $P$  él relációval. Legyen  $P$  reláció a következő:

$$P(X) = \hat{I}(T(X(i(X)-1)))$$

Bármilyen  $X \in C_{db}$ -re és  $Y$  szülőjére igaz a  $Y \subset X$  tartalmazás, mert

$$T(X(i(X)-1)) \subset T(X).$$

Tehát a  $P$  által képzett gráf körmentes és összefüggő, ahol a gyökér a  $\hat{I}(T(\emptyset))$  a legkisebb elemhalmaz. A módszer előnye, hogy a fa minden pontját egyszer kell bejárni és nem kell az egész fát a memóriában tárolni. Ha az eljárás egy levélhez ért, akkor visszalép a szülőjére. A következő lemma:

Legyen  $X[i]=X \cup H$ , ahol  $H = \{j \mid j \in \hat{I}(T(X \cup \{i\})) \wedge j \geq i\}$ . Ekkor:

$X'$  gyereke  $X$ -nek, akkor és csak akkor, ha teljesülnek a következő feltételek:

- (cond1)  $X' = X[i]$ , ahol  $i > i(X)$
- (cond2)  $X' = \hat{I}(T(X'))$  ( $X'$  zárt)
- (cond3)  $X'$  sűrű

**Bizonyítás:**

Tegyük fel, hogy  $X' = X[i]$  kielégíti mindhárom feltételt. Ekkor,  $X' \in C_{db}$ , mert  $T(X(i)-i) = T(X)$  és  $T(X(i-1) \cup \{i\}) = T(X')$ , tehát  $i(X') = i$ . Emiatt  $X'$  gyereke  $X$ -nek. Tegyük fel, hogy csak (cond2) és (cond3) teljesül.  $i(X')$  definíciója alapján  $T(X(i(X')) \cup \{i(X')\}) = T(X')$ . Tehát  $X' = X[i(X')]$ . Tudjuk, hogy  $i(X') > i(X)$ , mert  $T(X'(i(X')-1)) = T(X)$ . Tehát  $(\text{cond2}) \wedge (\text{cond3}) \rightarrow (\text{cond1})$ .

□

Ennek a lemmának az alkalmazásával megadható a következő eljárás, ami kiszámolja egy elemhalmaz összes leszármazottját:

**LCM** ( $X : \text{FCI}$ )

1. **Out**  $X$
2. **Foreach**  $i > i(X)$  **do**
3.   **If**  $\text{cond2}(X[i])$  **and**  $\text{cond3}(X[i])$  **then**
4.     **LCM**( $X[i]$ )
5. **End foreach**

Mivel a fa minden csúcsát egyszer járja be az eljárás, bármelyik FCI lineáris időben megkapható. Az eljárás műveletigénye az összes FCI (a fa csúcsainak száma) számosságától függ. Költséges lenne minden iterációban újraszámolni az előfordulásokat és minden lehetséges elemmel bővíteni, ezért a [5] cikk több módszert is kínál a számítási idő csökkentésére. A módszer, amit a *Get\_Move* eredeti implementációja használ az *Occurance deliver*, ami párhuzamosan számítja ki egy FCI és annak összes gyerekének az előfordulás halmazát az elemhalmazok egyetlen végigolvasásával. Ez lerövidíti a  $T(X[i])$  számítását, ami a (cond3) kiértékeléséhez kell. Ez a módszer akkor hatékony, ha  $|T(X[i])|$  jóval kisebb, mint  $|T(X)|$ . A másik megoldás a *Diffset*, ami akkor hatékony, ha az említett két számosság közel azonos. Az eljárás lényege, hogy az előfordulás konstruálása közben kigyűjti azokat

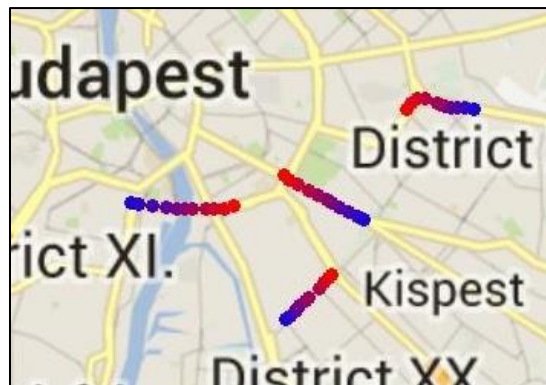
az elemeket, amikkel kiegészítve  $X$ -et, új  $FCI$ -k jöhetnek létre. Ezáltal nem kell minden iterációban minden elemet megvizsgálni.

Azért döntöttem a *Diffset* megközelítés mellett az *Occurance deliver* rovására, mert az előbbi sűrű input adatokon gyorsabb és az általam vizsgált adatok általában ilyen jellegűek voltak. Eltérés a [2] cikkben közölt javasolt implementáció és az én megvalósításom közt, hogy minden iterációban mohóbb lépésekkel halad a fában, illetve az imént közölt adatszerkezet miatt az előfordulás halmazt meghatározó eljárás nem az összes elemhalmaz számosságával arányos. Amikor a *Diffset*-et használó LCM algoritmus egy iterációja elvégzi a feltételek kiértékelését, ha csak a zártság tulajdonságát vizsgáljuk, akkor elveszhetnek azok az elemhalmazok, amelyek azonosan viselkednek az adott előfordulás halmazban. Ezt elkerülendő, összehasonlítom a megszorított előfordulásokat, és ha ilyen találok, mindet hozzáveszem a gyerek  $FCI$ -hez. Ez a lépés akkor különösen hasznos, ha sok egymáshoz hasonló útvonalú objektumot vizsgálunk. Ekkor nagy agresszív lépésekkel halad a fában.

### c) *Get\_Move* és *Pattern mining*

A *Get\_Move* algoritmus magába foglalja az előzetes osztályozást (jelen esetben *DBScan*), az  $FCI$ -k keresését, illetve az egyes  $FCI$ -kből a minták felismerését. Az algoritmus (az adatmodellnek is köszönhetően) a különböző objektumok számától függ és nem a vizsgálat idejétől, ezzel lehetővé téve, hogy egy fix hálózat, nagy intervallumot vizsgált forgalmát gyorsan elemezze. A mintakeresés gyakorlatilag csak a III.2) pontban definiált minták, elemhalmazként értelmezett definícióinak feltételeit vizsgálja. A kapott  $FCI$  definíció szerint *closed swarm*. Ennek elég nagy összefüggő részegységei a *convoy*-ok, illetve ezek sorozatai a *moving cluster*-ek. Ha pedig a *convoy* nagyobb, mint a paraméterként megadott súly, akkor *group pattern*. A *Get\_Move*-nak egy változata, az osztott *Get\_Move*, ami nem vizsgálja a frekvenciát, csak a zártságot. Ezeket a  $CI$ -ket egy egyesítő lépésben összefűzi és a frekvenciával kiegészített eljárás újra feldolgozza. Ideális esetben ezek az előfeldolgozások drasztikus időmegtakarítással járnak.

#### IV.3.2) Utófeldolgozás, megjelenítés



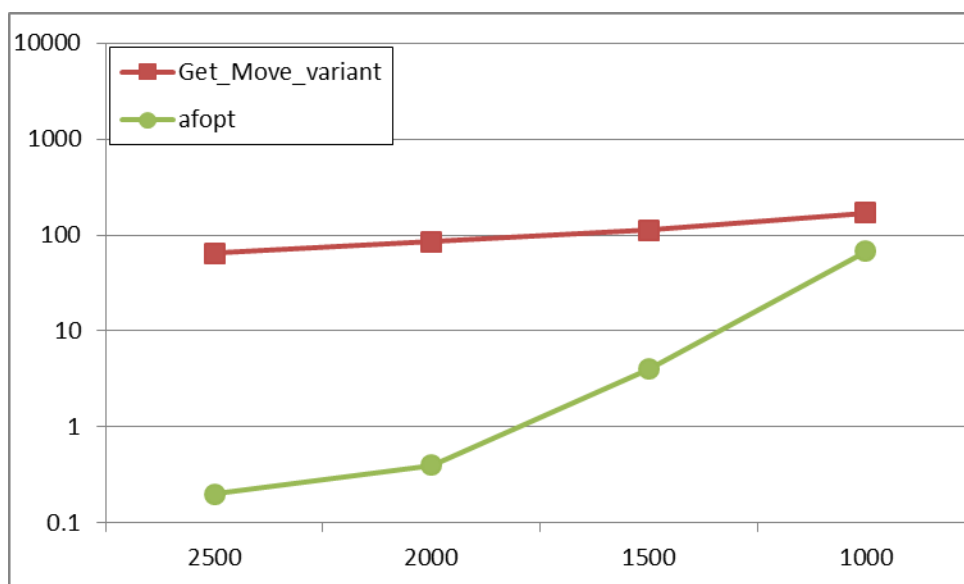
IV-6 ábra

Az utófeldolgozás, az előfeldolgozás során létrehozott map\_file-ok alapján visszaalakítja az objektumok azonosítóit és az idők formátumait, majd az eljárás során talált zárt, sűrű elemhalmazokat egy-egy térképen megjeleníti. A megjelenítés során az időben haladva a pontok színe fokozatosan pirosból, kékbe vált így érzékeltetve a haladás irányát. Az irányított torlódások grafikus megjelenítésére példa a IV-6 ábra. Az adatok ritkák voltak mind időben, mind térben. Ezért az egyes járművek egy város hálózatához képest messze vannak egymástól.

## IV.4) Eredmények

### IV.4.1) FCI bányászat

Referencia algoritmusként az AFOPT eredeti implementációját használtam. Az eredmények azt mutatják, hogy az általam implementált megoldás lassabb, viszont lassabban lassul. Sajnos az osztás implementálásáig a memória szab határokat, aminek a felhasználásában közel azonosak. Az implementációs nehézségek mellett a lassabb működés indokolt, mert az általam implementált eljárás az előfordulás kiszámítása mellett a tényleges előfordulásokat is tárolja, eredményként visszaadja. Tovább lassítja az eljárást az adatmodell felépítése, ami futási időben a fizikai tárolással szemben lassú. A teljes eljárás szempontjából az adatmodell felépítése kis plusz költséggel jár, mert az osztályozó algoritmus optimálisan adja át az új osztályokat. Ez a lépés jelentősen gyorsítható egy ideiglenes `std::unordered_map` bevezetésével. Az objektumok megkeresésére az adatbázisban a random access képes tárolókkal hatékonyabb. Referencia adatnak a *Frequent Itemset Mining Dataset Repository*<sup>2</sup> weblapján elérhető Chess.dat állományt használtam.



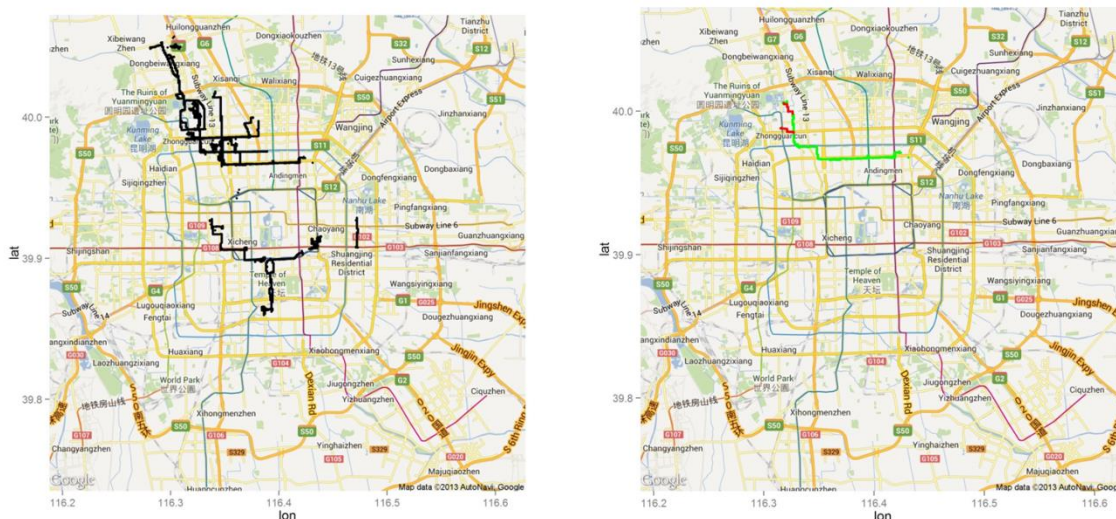
IV-7 ábra:

teszt környezet: Intel Core i5-3470 CPU (3.2 GHz), 8.00 GB memória, Ubuntu 12.04 64 bit

<sup>2</sup> <http://fimi.ua.ac.be/data/>

#### IV.4.2) torlódásdetektálás

Torlódásdetektálásra a Microsoft által közzétett, *Geolife*<sup>3</sup> adatokat használtam. A *Geolife* Pekingben mozgó objektumok különböző sűrűséggel mért földrajzi pozícióit tartalmazza. Az adatok erősen szűrtek, bennük igazi összefüggéseket találni nehéz. A vizsgált terület összes pontja, az összes mért időben és a két talált FCI a *III-7 ábrán* látható. A *DBScan* paraméterei: 2 objektumnak kellett, 250 méteres távolságra lennie a vizsgált ponttól. A *Get\_Move* paraméterezése szerint pedig a talált *FCI*-k legalább 4 eleműek és frekvenciájuk legalább 10. A mérési időközök egy másodperctől akár 10 percekig is terjednek, ezért az aggregáció után a *pattern mining* nem talált *convoy*-t, illetve *group pattern*-t.



IV-8 ábra

#### V) Értékelés, fejlesztési tervek

Bár még nem sikerült elég nagy adatokon kipróbálni az eljárást, illetve összevetni a valóságban fellelhető szokásokat a következtetett szokásokkal, de a kezdeti eredmények biztatóak. Ha sikerülne hatékonyan (ideális esetben, valós időben) kiértékelni az eredményeket és frissíteni a talált szokásokat, számos lehetőség nyílna meg. Például, ha a talált szokásokat relevánsnak ítéltük, egy keretrendszer elemezheti a trendeket és idejekorán észlelheti a trendekben mutatott változásokat. Egy város életében ez jelentheti azt, hogy egy beruházás felesleges, vagy pont hogy hamarabb, nagyobb igény várható rá. A szokások földrajzi vándorlását, terjedését vizsgálva az is felmérhető, hogy egy új szolgáltatást hol érdemes tesztelni, elindítani, úgy hogy a legnagyobb érdeklődést váltsa ki. Ha több, hasonló rendszerrel ellátott város, megosztja az információit, osztályozhatóvá válnak a városok, városrészek a lakók szokásai alapján. Felhasználva ezt az osztályozást egy-egy jelenség megjósolhatóvá válna valamilyen valószínűséggel, arra fel lehetne készülni, vagy korábban ki lehetne váltani. Ha rendelkezésre állnának adatok a városközi forgalomról, akkor nyomon követhető lenne az urbanizációs folyamat. Ez már akár

<sup>3</sup> <http://research.microsoft.com/en-us/downloads/b16d359d-d164-469e-9fd4-daa38f2b2e13/>

országos döntéseket, beruházások allokálásánál is releváns információ lehet. Megtalálhatóak lennének továbbá (szezonálisan), a népszerű üdülőhelyek, azok forgalmának eloszlásában trendeket lehetne keresni (azonos módszerekkel). Látszik, hogy az eljárás elég általános, hogy sok hasznos információt találjon, viszont ahhoz hogy mindegyik emberi beavatkozás nélkül képes legyen valamilyen szemantikus adatbázisból ki kellene nyernie a hálózat egyes pontjainak funkcióját (iskola, lakónegyed). A szokások meghatározása még ekkor sem feltétlenül triviális, nem egyértelmű esetekben egy ember tekintete is elfuthat felettük. Az eljárásnak nem az imént felvetett kérdésre választ találni, de egy alkalmas rendszernek megfelelő adatokat állíthat elő.

### V.1) FCI bányász algoritmus fejlesztése

Az eljárás legköltségesebb lépése a zárt, sűrű elemhalmazok keresése. Az egész folyamat úgy van felépítve, hogy a legjobban elősegítse annak működését. A jövőben a fejlesztések fő célja a hatékonyság növelése. Az irodalomban számos megoldást találtam a központi algoritmus gyorsítására. Ha például bevezetünk egy olcsón kiértékelhető feltételt, ami meghatározza a potenciális gyerek osztályok előfordulásainak halmazairól, hogy azok sűrűk, vagy ritkásak. Ekkor a kiértékelés függvényében döntene az eljárás, hogy IV.3)b) pontban vázolt két technika közül várhatóan melyik hatékonyabb és ennek megfelelően lépne tovább. Másik fejlesztési a feldolgozás sorrendjének jobb megválasztása. Legyen például rendezés az egyes osztályok közt az előfordulásuk számossága szerint. Ekkor minden iterációban a lehető legkisebb gyereket dolgozzuk fel és az őt reprezentáló elemmel már nem kell foglalkozni, ezáltal csökkentve a későbbi iterációk költségét.

### V.2) Hatékonyabb implementáció

A gráf felépítése jelenleg lassabb, mint indokolt lenne. Ennek fő oka, hogy láncolt listában tárolom az objektumokat, az eljárás pedig véletlen eléréssel lenne optimális. Ennek a problémának egy lehetséges megoldása, ha ideiglenesen egy `std::unordered_map`-ben tárolnám a mutatókat úgy, hogy kulcsként az azonosítót használnám. A V.1) pontban említett bejárési sorrend hatékony megvalósítására egy lehetőség, ha az említett rendezést implementálom és a Clustereket `std::set`-ben tárolom. Ezáltal nő az egyes elemek elérésének költsége, de a rendezettség minden előnye felhasználhatóvá válik.

### V.3) Periodikus minták

A fejlesztési terv egy másik iránya – a III.4) pontban említett motivációk miatt -, a periodikus minták keresése. Ha egy trajektóriában van periodikus résztrajektória, az a síkban egymáshoz (elég) közeli, de időben (elég) távoli pontokként jelenik meg. A III.2 pontban tárgyalt *meet* minta, egyetlen síkon reprezentál egy ilyen periódust, ha a belső pontok mérési időit vizsgáljuk. Világos, hogy a probléma jóval egyszerűbb, mint aminek a megtalálására az eredeti cikkben közölt módszer hivatott, mert nem kell az időben leghosszabb hengert megkeresni. Azok a pontok, amiket nem sikerült ilyen módon osztályokba sorolni, zajnak tekinthetők, nincs bennük periódus. A *DBScan*, vagy más klaszterező eljárás tovább szűri ezeket az adatokat, megkeresve a csoportos periódusokat. A *Get\_Move* algoritmus pedig visszaadja a releváns eredményeket, ezzel megoldva a feladatot. Annak a kérdésnek a megválaszolása, hogy ez a módszer minden periódust megtalál-e, még további kutatásra szorul, de vélhetően gyorsabb az a módszer mintha minden zárt, sűrű elemhalmazban keresnénk a periódusokat.

#### V.4) Párhuzamosítás

Bár az eljárás megszerkesztésekor fontos szempont volt, hogy kevés munkával több eszköz párhuzamosan is feldolgozhassa az adatokat, de ez elsősorban az oszthatóság garantálásában merült ki. A legtöbb részfolyamatnál tárgyaltam az osztási és egyesítési lépéseket, de megvalósítva nincsenek. Ezek megírása drasztikusan növelheti a futási sebességet. Ahhoz hogy párhuzamosan kezelhetők legyenek a részfolyamatok, szükség van egy globális adatbázisra, ami nem csak a korábban feldolgozott adatokat képes tárolni, de egyes részeredményeket is.

#### V.5) Hatékonyabb keretrendszer

Az eljárás keretrendszere az egyszerűség kedvéért lett R-ben megvalósítva, de létezik nála sokkal hatékonyabb megoldás is. Egy út lehet C++-ban implementálni az elő és utófeldolgozás lépéseit és egy kompakt, grafikus felületű szoftvert készíteni. Cél továbbá a megjelenítésre a dinamikus elemek igényes megjelenítését támogató *WorldWind*-et használni.

### VI) Irodalomjegyzék

- [1] Phan Nhat Hai, Pascal Poncelet, Maguelonne Teisseire, „GeT\_Move: An Efficient and Unifying Spatio-Temporal Pattern Mining Algorithm for Moving Objects”, IDA'12 Proceedings of the 11th international conference on Advances in Intelligent Data Analysis
- [2] T. Uno, T. Asai, Y. Uchida, H. Arimura, „An efficient algorithm for enumerating closed patterns in transaction databases”, Proceedings of the 7th International Conference on Discovery Science, Padova, Italy, pp. 16-31, 2-5 October 2004.
- [3] Gudmundsson J, van Kreveld M., „Computing longest duration Focks in trajectory data”, In: GIS 06, New York, NY, USA, pp.35-42.
- [4] by Jiawei Han , Wan Gong , Yiwen Yin, „Mining Segment-Wise Periodic Patterns in Time-Related Databases”, Proc. Int. Conf. on Knowledge Discovery and Data Mining
- [5] Ester M., Kriegel H.-P., Sander J., Xu X. A, „Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise”, KDD '96, Portland, pp. 226-231.
- [6] T. Uno, M. Kiyomi, H. Arimura, "LCM ver.2: Efficient Mining Algorithms for Frequent/Closed/Maximal Itemsets", In Proc. IEEE ICDM'04 Workshop FIMI'04, 2004.
- [7] G. Liu, H. Lu, J. X. Yu, W. Wang, X. Xiao. AFOPT: An Efficient Implementation of Pattern Growth Approach. The 1st International Workshop on Frequent Itemset Mining Implementations (FIMI'03), Melbourne, Florida, USA, 2003.
- [8] Bart Goethals, Mohammed J. Zaki, „FIMI'03: Workshop on Frequent Itemset Mining Implementations”, volume 90 of CEUR Workshop Proceedings, CEUR-WS.org, (2003)
- [9] Yu Zheng, Xing Xie, Wei-Ying Ma, „GeoLife: A Collaborative Social Networking Service among User, location and trajectory”