Adatbányászat fehérjéken

Hubenkó Elemér szakdolgozata

Témavezető: Grolmusz Vince

2007 január

Kivonat

A modern gyógyszerkutatásban egyre fontosabb szerephez jut a gyógyszerjelölt molekulák számítógépes tervezése és tesztelése, az ismert térszerkezetű fehérjék adatbázisának rohamos bővülése elősegíti a bioinformatikai alkalmazások térnyerését. Ezek segítségével, növekvő pontosságú, illetve érvényességű előrejelzések adhatók az egyes hatóanyag-molekulák viselkedésére.

Egy ligand és egy receptor-molekula – dokkolásnak nevezett – interakciójának végeredménye, a két molekula által a térben felvett, összességében energetikailag minimális helyzet segítségével adható meg.

Dokkolást „végrehajtó" – tehát egy energiafüggvény meghatározásához, minimalizálásához, összetett kvantummechanikai, molekuláris-dinamikai számításokat végző – szoftverekből több is létezik. S bár a szoftverek többsége „kompromisszumos" számítási módokat is támogat – ez alatt a számítási idő ésszerű korlát alá szorítása mellett, a dokkolás használható minőségének a megtartása értendő –, a rendelkezésre álló, nagy adathalmazok feldolgozása mégis kivitelezhetetlen a napjainkban elérhető hardware eszközökkel.

Például tekintsünk egy – a dolgozatban később felhasznált – 4200 elemből álló fehérje-kötőhely halmazt (bizonyos „aktív" terület, a fehérjefelület közelében). Tegyük fel, hogy egy 100000 elemből álló ligandhalmaz elemeivel szeretnénk az összes kötőhelyet páronként dokkolni. Egy dokkolás időigénye, számítási módtól és szoftvertől függően, akár perceket is igénybe vehet, de mindenképpen másodpercekben mérhető egy PC-n futtatva, a ma elérhető technológiákat figyelembe véve. Ha csak egy másodperccel számolunk dokkolásonként, a számítás 4861 napot venne igénybe. Itt megjegyzendő, hogy a dolgozatban végrehajtott „merev" dokkolás – ami körülbelül háromszor kevesebb számolást igényel –, átlagosan 10 másodpercet igényelt ahhoz, hogy determinisztikus-közeli eredményt érjünk el. Tehát az egy másodperc, valószínűleg, nagyon gyenge alsó becslése az átlagos dokkolási időnek, és akár valamennyi technológiai fejlődést is figyelembe véve, érvényes marad a közeljövőben.

A fent említett, és az ehhez hasonlatos kísérletek, tehát, közel kivitelezhetetlenek rövidtávon, vagy megvalósításuk nagyon komoly, költséges számítástechnikai infrastruktúrát igényel.

Dokkolás szempontjából hasonlóan „viselkedő" fehérje-kötőhely csoportok megtalálásával esély mutatkozik a számítási idő rövidítésére, ugyanis egy adott ligandhoz, csoportonként elegendő lehet annak csak néhány reprezentáns elemmel való dokkolása a dokkolási energiák jó becsléséhez a csoport többi elemével.

Dolgozatomban a fehérje-kötőhelyek bizonyos jellemzőjének – ligandok egy 1838 elemű csoportjával való dokkolásából származó, valós komponensű, 1838 dimenziós vektorok – klaszterezése segítségével próbálom meghatározni a – dokkolás szempontjából – hasonló fehérje-kötőhelyek csoportjait.

A dolgozat első részében fehérjék szerkezetéről, felépítéséről, annak tárolási módjairól következik egy rövid bevezető. Ez után az energiavektorok meghatározásához is használt, AutoDock 3.0 szoftver ismertetése következik. Az implementált és felhasznált klaszterezési algoritmusok leírásával folytatom a dolgozatot: K-means, sűrűség alapú klaszterezés (DBSCAN, OPTICS).

Ezek után a futtatási körülmények (paraméterek, paraméterfájlok) és eredmények ismertetése következik, továbbá az eredmények értékelése.

Tartalomjegyzék

rész I

A felhasznált elmélet és a felhasznált eszközök

0.1. Fehérjék

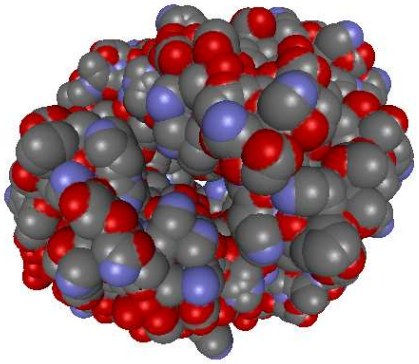
0.1.1. Fehérjékről általában

A dolgozatban központi fogalom (bemeneti adat) a fehérje molekula: rendeltetését és tulajdonságait, felépítését ebben a fejezetben vázolom.

Fehérjék, többek között, az élő sejt építőelemei, molekuláris gépei és fegyverei, szabályozzák a sejtekben végbemenő folyamatokat, és lehetővé teszik a sejtek közötti kommunikációt, raktározzák és szállítják a sejtek működéséhez szükséges anyagokat, a sejtek helyes elhelyezkedését és szerkezeteinek megtartását is fehérjék szabályozzák.

A fehérjék működése a velük kapcsolatba lépő – feldolgozandó vagy kapcsolódó – molekulákkal való jól meghatározott kölcsönhatáson alapul, amihez – viszonylag – merev térbeli szerkezetet szükséges. Ezért a fehérjék, viszonylag, tömör felépítésűek, jól kitöltik a rendelkezésre álló teret. Funkciójuk szorosan összefügg térbeli szerkezetük meghatározottságával, mivel akár egy kisebb szerkezeti változás is a fehérje működésének gyengülését vagy annak megszűnését is okozhatja.

A fehérje a génekben kódolt információ „jelentése", amely a transzlációnak nevezett folyamat útján jön létre, így bizonyos fehérjék evolúciós rokonságban vannak (mivel közös az eredetük). Az evolúció, mutációk és szelekciók révén, vezetett el gyakran fehérje-óriásmolekulákhoz, amelyek egészen parányi atomok, molekulák vagy egyéb részecskék nagyon specifikus feldolgozását végzik.



A leggyakoribb humán hemoglobin fehérje. Molekuláris képlete $C_{2952}H_{4664}N_{812}O_{832}S_8Fe_4$, tehát több mint 9200 atomból áll és mindössze négy oxigénmolekulát szállít. A fehérje négy aminosav-láncból tevődik össze.
[Forrás: RCSB-PDB, kód: 1BZ0]

A fehérjék tehát általános felhasználású építőelemek, és a genetikailag különböző szervezetekben, az egyébként azonos funkciókat ellátó fehérjék, jellemzőek arra a szervezetre, amelyben előfordulnak. Ezért a fehérjék szerkezetének meghatározása, működésének megértése, illetve más molekulákkal való kölcsönhatásuk prognosztizálása, alapvető fontosságú – többek között – a gyógyszerkutatás területén.

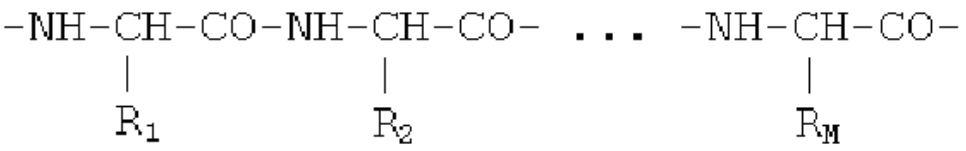
0.1.2. Aminosavak, fehérjék szerkezete

Fehérjének, általában, egy vagy több speciálisan összetapadó (nem kovalens kötéssel összekapcsolt) aminosavmaradék-láncot neveznek. Szerkezetileg, tehát, a fehérje feltekeredett lineáris polimer, máshogy: aminosavakból (aminosav-maradékokból) álló polipeptidlánc. Gyakran a funkciók ellátásához szükséges további, nem aminosav eredetű komponenseket is beleértve, amik ugyancsak hozzátapadnak. A dolgozat modelljében fehérjének szintén ezt a biológiai egységet (komplexet) definiálom.

Az aminosav-láncban az aminosavak sorrendjét gének kódolják, ez a sorrend

egyértelműen meghatározza a fehérje térszerkezetét, és ez biztosítja a fehérjék szerkezeti és méretbeli változatosságát.

Húszféle aminosav létezik, így a lánc is húszféle aminosavból (aminosav-maradékból) tevődhet össze, és akár több tízezer elemből is állhat. Egy (szabad) aminosav $H_2N - (CH - COOH) - R_i$ szerkezetű, az aminosavak csak az R_i aminosav-oldalláncokban különböznek egymástól. Miközben egy aminosav belép az aminosav-láncba, megválik egy oxigén és két hidrogén atomjától.
A lánc gerince így a – kémiailag reguláris – minden aminosavban azonos ($NH-CH-CO$) csoport ismétlődő szakaszaiból áll, amelytől R_i aminosav-oldalláncok ágaznak el.

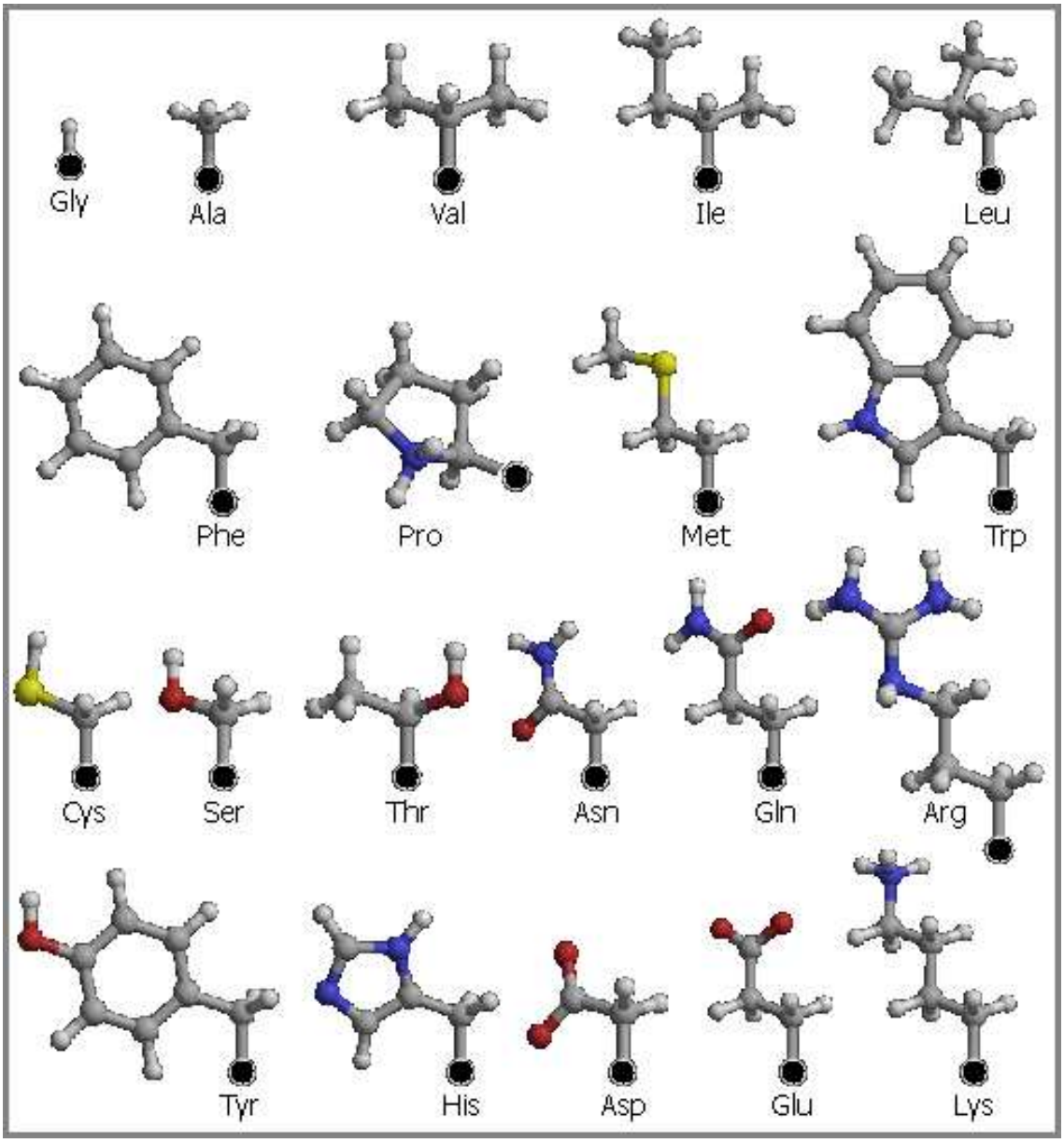


1. ábra. Polipeptidlánc kémiai szerkezetének vázlata.

Az alábbi táblázat tartalmazza az aminosavak összetételét: a nevek mellett a háromjegyű rövidített jelölés és a molekuláris tömeg is szerepel.

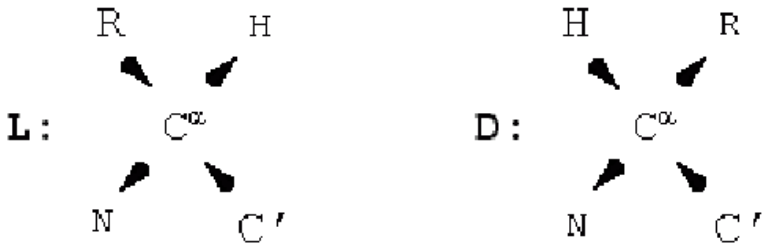
Aminosav neve	Rövidítés	Molekula képlete	molekuláris tömeg
Alanine	ALA	C3 H7 N1 O2	89.09
Arginine	ARG	C6 H14 N4 O2	174.20
Asparagine	ASN	C4 H8 N2 O3	132.12
Aspartic acid	ASP	C4 H7 N1 O4	133.10
Cysteine	CYS	C3 H7 N1 O2 S1	121.15
Glutamine	GLN	C5 H10 N2 O3	146.15
Glutamic acid	GLU	C5 H9 N1 O4	147.13
Glycine	GLY	C2 H5 N1 O2	75.07
Histidine	HIS	C6 H9 N3 O2	155.16
Isoleucine	ILE	C6 H13 N1 O2	131.17
Leucine	LEU	C6 H13 N1 O2	131.17
Lysine	LYS	C6 H14 N2 O2	146.19
Methionine	MET	C5 H11 N1 O2 S1	149.21
Phenylalanine	PHE	C9 H11 N1 O2	165.19
Proline	PRO	C5 H9 N1 O2	115.13
Serine	SER	C3 H7 N1 O3	105.09
Threonine	THR	C4 H9 N1 O3	119.12
Tryptophan	TRP	C11 H12 N2 O2	204.23
Tyrosine	TYR	C9 H11 N1 O3	181.19
Valine	VAL	C5 H11 N1 O2	117.15

A húszféle R_i aminosav-oldallánc kapcsolódási helyét (az egyik szénatomot) a későbbi hivatkozásokhoz C_α-val jelölik, az aminosav másik szénatomját C'-vel. Az aminosav-oldalláncok térbeli szerkezete is rögzített, a következő ábrán látható a szerkezeti információ.



2. ábra. Az R_i aminosav-oldalláncok térbeli szerkezete. A kapcsolódási C_α szénatom fekete körlappal van jelölve.

Az oldalláncok kétféleképpen, L és D szimmetrikus, módon kapcsolódhatnak a főlánc szén (C_α) atomjához, helyet cserélve a hidrogén (H) atommal. Emellett csak az L-alak jön létre közvetlenül a transzláció útján. A Glycine aminosav oldalláncának nincs két kapcsolódási lehetősége, mivel oldallánca egyetlen hidrogén atomból áll. A következő, 3. ábra szemlélteti ezeket a kapcsolódási módokat.

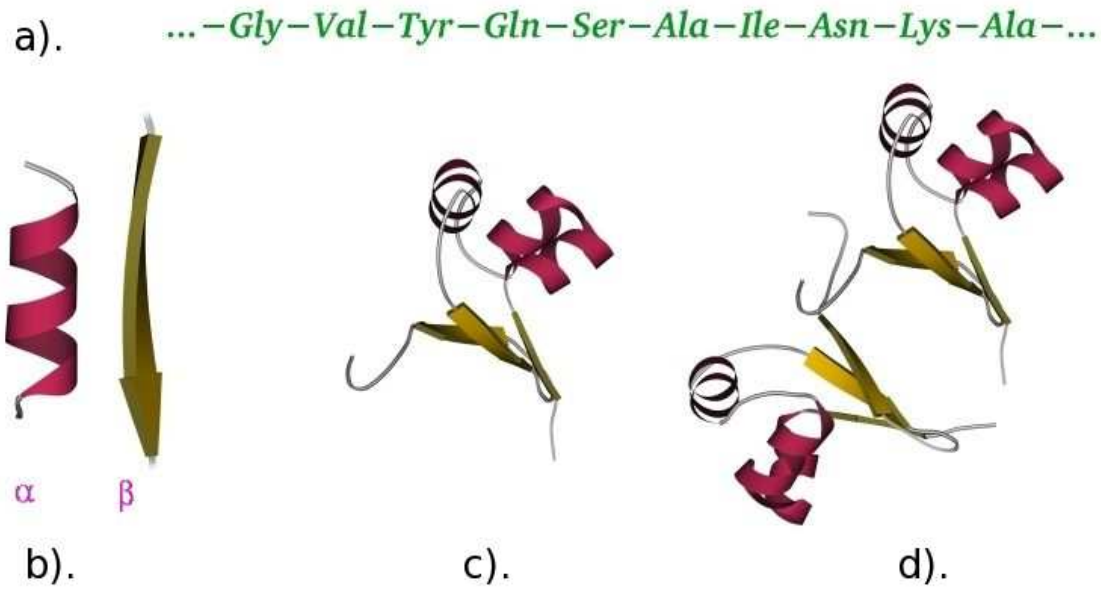


3. ábra. Az L és a D szerkezetű oldallánc-kapcsolódás

A fehérje *elsődleges szerkezetének* nevezzük az aminosavak sorrendjét a láncban. Az aminosavakat háromjegyű vagy egyjegyű kódokkal azonosítják. A szomszédos aminosavak kovalens kötésben vannak, C' és N atomok peptidkötése köti össze őket, ami sokkal erősebb kötés, mint a fehérjelánc szerkezetét összetartó nem kovalens kötés.

A fehérjék térbeli szerkezete, általában, bonyolult és sokféle lehet, mégis előfordulhatnak bennük reguláris részletek, ezek alkotják a fehérjék *másodlagos szerkezetét*. Az α-spirál és a β-lemez ilyenek, előbbit gyakran hengerrel, utóbbit nyíllal ábrázolják (lsd. 4. ábra).

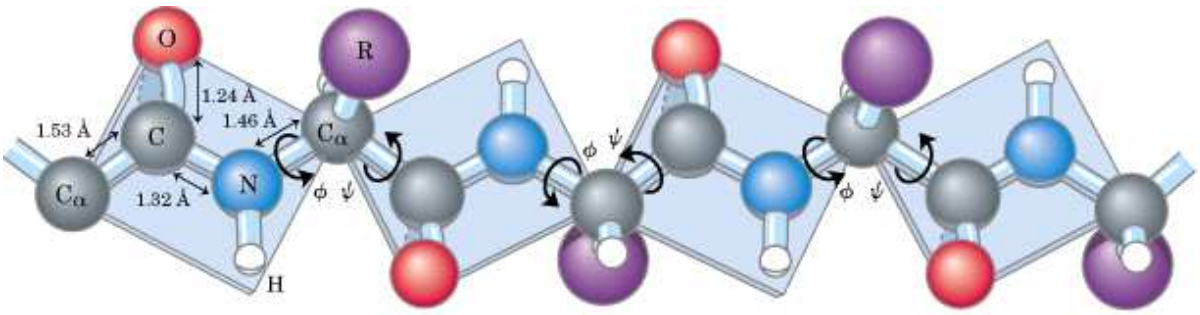
A másodlagos szerkezetek elhelyezkedése egy aminosav-láncon belül a fehérje *harmadlagos szerkezete*. Végül, az aminosav-láncok, harmadlagos szerkezetek, összessége alkotja egy fehérje *negyedleges szerkezetét*.



4. ábra. A fehérje szerkezeteinek részletességi szintjei: a). elsődleges, b). másodlagos, c). harmadlagos, d). negyedleges.

Szabadsági fokok

Az aminosav-láncban az oldalláncok, a már említett, merev szerkezetben vannak. A C_α szénatomhoz kapcsoló kovalens kötésük körül "foroghatnak" csupán. A főláncban is vannak mindig szigorúan egy síkban lévő részletek: az előző aminosav-maradék C_α atomja és az azt követő aminosav-maradék O-C'-NH-C_α része. A lánc, összességében, csak a C_α szénatomhoz kapcsolódó kovalens kötések körül "foroghat", az - adott környezetben - energetikailag minimális konformációt felvéve. Az atom-párok közötti kovalens kötések meghatározott erősségűek és távolságúak. Ezt a távolságot Angströmben ($\mathring{A}$) mérik, ahol $1\mathring{A} = 0.1$ nm $= 10^{-10}$ m.



5. ábra. A fehérjelánc gerincén belül egy síkban lévő atomok, a szabadsági fokok és kötéstávolságok.

0.1.3. Fehérjék és egyéb molekulák megadása

A fehérjék – és általában a molekulák – szerkezetének leírása a fehérje-atomok helyzetének megadásával történik a háromdimenziós valós térben. A szerkezet meghatározását röntgenkrisztallográfiai vagy más eljárásokkal végzik. Ezeknek a módszereknek az eredményessége és érvényessége sok tényezőtől függ és gyakran hiányos, vagy korlátozott pontosságú adatokhoz vezetnek. Így önmagában az atomok koordinátái — és akár a kötések felsorolása is – nem mindig elegendőek a molekula helyes számítógépes reprodukálásához, feldolgozásához. Emiatt az általánosan használt fájlformátumok további információt is tartalmaznak: ilyen lehet az aminosav-láncok mérete és az aminosavak sorrendjének megadása (fehérjékben, segítségével illeszthetőek az ismert aminosav-szerkezetek), illetve az, hogy egy adott atom az aminosavnak pontosan melyik atomja (fehérjékben, a hiányzó atomok ezek után pótolhatók).

Lelőhelyek

Számos fehérje térszerkezeti és szekvencia adatbázis létezik, szabadon elérhetően az Interneten. Ezek közül a legáltalánosabban használt a PDB (Protein Data Bank), ami a kísérleti úton felfedezett fehérjeszerkezetek adatbázisa. Minden feltöltött molekulának egy

négyjegyű kódot feleltet meg, PDB formátumban letölthetők a molekulák (lsd. következő fejezet).

A 2006. november 29. állapotnak megfelelően a PDB adatbázis 40354 molekula szerkezeti információit tartalmazta: ezekből 36991 fehérje.

0.1.4. Fájlformátumok

Ebben a fejezetben, a dolgozatban kísérletében előforduló fájlformátumok áttekintése következik.

PDB

A PDB a Protein Data Bank fájlformátuma. Minden sor valamilyen kulcsszóval kezdődik, és ennek függvényében értelmezhetőek az azt követő adatok. Minden kulcsszót követő adatsornak megvan a maga, karaktermezőnként definiált, formátuma (ettől néha eltérnek). A fájl végződése .pdb .

A fejléc (Title Section) a következő mezőket tartalmazhatja: HEADER, OBSLTE, TITLE, CAVEAT, COMPND, SOURCE, KEYWDS, EXPDTA, AUTHOR, REVDAT, SPRSDE, JRNL, REMARK, REMARK 1, REMARK 2, REMARK 3, REMARK 4 – 999. Ebben különböző adatok, dátumok, megjegyzések találhatók a publikációról, a szerzőkről, kulcsszavak. Automatizált számítógépes feldolgozás szempontjából kevésbé lényeges információk.

Az elsődleges szerkezet (Primary Structure Section) rész kulcsszavai: DBREF, SEQADV, SEQRES, MODRES. A szekvencia-kapcsolatok más adatbázis elemekkel, az atomkoordináta és szekvencia adatokban megtalált ellentmondások, illetve az aminosavak sorrendje lánconként, a megtett változtatások. A SEQRES kulcsszó tartalmazza tehát a szekvencia-információt, karakterenként:

(1 - 6) a sor elején az azonosító: "SEQRES",

(9 - 10) az adott lánchoz tartozó SEQRES sor sorszáma (1-től indulva, 1-el inkrementálva),

(12) a láncot azonosító karakter,

(14 – 17) az aminosavak száma a láncban (ez az érték ismétlődik minden, az adott lánchoz tartozó sorban),

(20 – 22) - ... $(20 + 4 * K$ – $22 + 4 * K)$ ... – (68 – 70) az aminosav-maradékok háromkarakteres kódjai (tehát, legfeljebb 13 lehet egy sorban).

Példa:
SEQRES 1 A 141 VAL LEU SER PRO ALA ASP LYS THR ASN VAL LYS ALA ALA
SEQRES 2 A 141 TRP GLY LYS VAL GLY ALA HIS ALA GLY

Az eltérő komponensek része (Heterogen Section) kulcsszavai: HET, HETNAM, HETSYN, FORMUL. Az előforduló nem aminosav komponensek kódjai, nevei, alternatív nevei és képletei.

Másodlagos struktúra rész (Secondary Structure Section) kulcsszavai: HELIX, SHEET, TURN. Az α, β struktúrák kezdő és utolsó aminosavakkal, azonosítóval és

sorszámmal ellátva, továbbá az ezeket összekötő esetleges hurkokról rendelkezésre álló információ.

Összekapcsolódásra vonatkozó rész (Connectivity Annotation Section) kulcssavai: SSBOND, LINK, HYDBND, SLTBRG, CISPEP. Ebben a részben a lánc(ok) különböző „összetapadásai", kapcsolódási helyei találhatók típusonként, a résztvevő atomok, aminosavak megjelölésével.

Különböző tulajdonságok rész (Miscellaneous Features Section): SITE kulcsszó. Ezzel a kulcsszóval a fehérje aktív központja adható meg.

Krisztallográfiai és koordináta transzformációs rész (Crystallographic and Coordinate Transformation Section) kulcsszavai: CRYST1, ORIGXn, SCALEn, MTRIXn, TVECT.

Koordináta rész (Coordinate Section) kulcsszavai: MODEL, ATOM, SIGATM, ANISOU, SIGUIJ, TER, HETATM, ENDMDL. A MODEL és ENDMDL kulcsszavak több modell előfordulása esetén szerepelnek. Az ATOM kulcsszó a hagyományos aminosavakban előforduló atomokról tartalmaz koordináta és egyéb információt, karakterenként:

(1 - 6) a sor elején a kulcsszó: "ATOM",

(7 - 11) az atom sorszáma,

(13 - 16) atom neve,

(17) alternatív elhelyezkedés,

(18 - 20) aminosav neve,

(22) lánc azonosítója,

(23 - 26) aminosav szekvencia-sorszáma,

(27) aminosavak beszúrásának kódja,

(31 - 38) valós(8.3) szám: x koordináta Angstromben,

(39 - 46) valós(8.3) szám: y koordináta Angstromben,

(47 - 54) valós(8.3) szám: z koordináta Angstromben,

(55 - 60) valós(6.2) szám: occupancy,

(61 - 66) valós(6.2) szám: hőmérséklet faktor,

(73 – 76) szegmens azonosítója,

(77 - 78) atom szimbóluma,

(79 - 80) atom töltése.

Példa:

```
ATOM    109  O   GLY A  15      39.273  23.533  31.034  1.00 33.39           O
ATOM    110  N   LYS A  16      38.649  22.768  33.063  1.00 28.13           N
```

A SIGATM a standard deviancia értékeit tartalmazza az ATOM és HETATM mezőkben lévő atomokhoz, hasonló formátumban. Az ANISOU, SIGUIJ mezők hőmérsékletfaktorokra vonatkozó információt tartalmaznak. A TER kulcsszó egy adott

lánc ATOM/HETATM atomjai felsorolásának végét jelenti. Végül, a rész HETATM kulcsszavai után a molekula nem szabványos, vagyis nem aminosavba tartozó atomjainak koordinátái és egyéb adatai következnek, az ATOM megadásához hasonló alakban.

Összekapcsolódás rész (Connectivity Section) kulcsszava: CONECT. A korábban előre jelzett kapcsolatok megadása immár atomok sorszámaival.

A befejező elszámoló rész (Bookkeeping Section) kulcsszavai: MASTER, END. A MASTER kulcsszó után a bevitt – különböző – mezők száma szerepelhet, utólagos teljesség-ellenőrzéshez. A PDB fájlt END kulcsszó zárja.

PDBQ

A PDBQ fájlformátum a PDB-ből származik, a Q betű a résztöltések jelenlétére utal: az ATOM sorok 71-76. mezőiben a résztöltések értékeit tartalmazza. A fájlok végződése .pdbq .

Tartalmazhat továbbá különböző flexibilitásokra utaló kulcsszavakat, mint ROOT, TORSION, BRANCH. A fájlformátumot és a kulcsszavakat az AutoDock szoftverhez használom, a ligandok megadásához.

PDBQS

A PDBQS formátum szolvatációs paramétereket is tartalmaz a résztöltések mellett, a makromolekulák megadásához használatos fájlformátum. A fájlok végződése .pdbqs .

A felsorolt fájlformátumok egymásba konvertálhatók a később vázolt módon.

0.2. Ligandok

A ligandok nem kovalens kötéssel kötődő (tapadó) „kismolekulák”, a definíciójuk ezen belül flexibilis.
Adott esetben, egy molekulahalmaz elemeit nevezzünk ligandoknak, amelyeket többek között gyógyszerkutatásokhoz szelektáltak ki olyan módon, hogy azok páronként minél különbözőbbek legyenek. Így adatbányászati alkalmazásokhoz kiválóan alkalmas reprezentáns elemek. Ez a halmaz szabadon elérhető: NCI Diversity Set [6].

A molekulák ebben az esetben is az atomok koordinátáit tartalmazó fájlokkal vannak megadva. A ligandok szén (C), aromás szén (A), nitrogén (N), oxigén (O), kén (S), fluor (F), klór (c), bróm (b) és hidrogén (H) atomokból állnak, néhány kivételtől eltekintve.

0.3. Dokkolás

0.3.1. A Dokkolás

0.3.1. Definíció. *A dolgozatban **dokkolásnak** egy kismolekula és egy receptor-molekula (adott esetben fehérjemolekula) interakciójának szimulált végeredményét, vagyis a két molekula által a térben felvett, összességében energetikailag minimális helyzetet nevezzük.*

A szimulációnál, tehát, van egy energiafüggvény, aminek a globális minimumát (közelített globális mnimumát) keressük. Ezt az értéket később felhasználjuk. Az energiafüggvény a két molekula egymáshoz viszonyított térbeli helyzetétől függ. Általános esetben a molekulák bizonyos deformációira is kiterjeszthető.

A dokkolási eljárásoknak, egyrészt, a lehető leggyorsabban kell megtalálniuk az energiafüggvény minimumát, másrészt alaposnak kell lenniük, és lehetőleg az összes szabadságfok figyelembevételével kell meghatározniuk az energiafüggvény globális minimumát. Ez gyakorlatilag két ellentétes követelmény, melyeknek a dokkolási programok igyekeznek megfelelni.

A dolgozat kísérleteiben dokkoláshoz az AutoDock 3.0 szoftvert használom.

0.3.2. AutoDock

Az AutoDock program egy flexibilis ligand és egy másik molekula (komplexum), adott esetben fehérje, interakciójának előrejelzéséhez használható. A ligand flexibilitása azt jelenti, hogy a felhasználó bizonyos kötések körüli forgásokat is megengedhet a dokkolás során. A fehérjemolekula mindig merev az AutoDock szoftver 3.0 verziójában.

A fehérje egy részét, vagy egészét egy téglatestbe ágyazhatjuk és a program a ligand téglatesten belüli konformációira minimalizálva az energiafüggvényt, végrehajtja a dokkolást. A téglatest élein, azt egyenlő szakaszokra felosztó, rácspontokat definiálhatunk, amelyek a téglatesten belül meghatározzák a számítások „felbontását” meghatározó háromdimenziós rácsot. Így az energiaminimum keresése egy előre definiált téglatestre korlátozódik, amely megegyezik a rácspontok konvex burkával. Rögzített téglatest esetén a rácstávolság (tehát a rácspontok száma) alapvetően befolyásolja a számítások időigényét és pontosságát.

Két fő részből és további segédalkalmazásokból áll a szoftver: az AutoGrid előzetes energiakiértékelésekkel meghatározza az energiapotenciálok rácstérképeit (grid maps), amiket felhasználva az AutoDock elvégzi a dokkolást.

Rácstérképekre a később következő energiafüggvény-kiértékelések gyorsításához van szükség. Az AutoGrid a ligandmolekulában előforduló összes atomtípushoz előre kiszámolja az energiapotenciálok térképeit. Az alkalmazás ezt a következőképpen végzi: a téglatesten belüli rácspontokban minden előforduló atomtípushoz kiszámolja az energiapotenciált és az elektrosztatikus potenciált. Ezek után, egy tetszőleges ligand-konformáció atomjainak energiapotenciálját tri-lineáris interpolációval számolja az AutoDock program.

Maga a dokkolás az AutoDock-ban implementált genetikai algoritmus segítségével történik.

A következő fejezetben az AutoDock és AutoGrid szoftverekben alkalmazott energiafüggvény rendkívül vázlatos ismertetése következik.

0.3.3. A számítások elméleti alapjai

Az további hivatkozásokhoz megjegyzendő, hogy az AutoDock szoftver bemenete, lefutásonként, egy PDBQ formátumban megadott ligand, egy PDBQS formátumban megadott fehérjemolekula, továbbá a DPF paraméterfájl és az energiapotenciálok rácspontokban számolt értékei minden egyes ligandatomra. Utóbbi fájlokat az AutoGrid állítja elő, ezért a végrehajtási sorrendben megelőzi az AutoDock lefutásait. Az AutoGrid paraméterfájlja GPF, a rácspontok számát dimenziónként, a rácstávolságot, továbbá a rácspontok konvex burkaként létrejövő téglatest középpontját tartalmazza.

Most tehát egy adott E fehérje, és I ligand molekula dokkolásának szimulációját tekintjük, oldatban. Az oldószer molekulái hajlamosak a nagyobb molekulákhoz és molekula-komplexekhez tapadni, és ezzel befolyásolják az energiafüggvény értékét. Ugyanis, dokkoláskor sok oldószer-molekula felszabadul ezekből a nem kovalens kötésekből. Ezt a körülményt az AutoDock szoftver energiafüggvényében egy entrópiás tagként veszik figyelembe, melynek paramétereit kísérleti úton határozták meg.
A következő képlet mutatja az összefüggést a vákuumban és oldószerben történő kötések (dokkolás) között, Hess törvényét felhasználva.

$$\Delta G_{kotes\ oldoszerben} = \Delta G_{kotes\ vakuumban} + \Delta G_{szolvatacio(EI)} + \Delta G_{szolvatacio(E+I)}$$

A vákuumban történő dokkolást a szoftver szimulálja, a szolvatációs paramétereket kísérleti úton adták meg. Így közelíthető az oldatban történő dokkolás $\Delta G_{kotes\ oldoszerben}$ energiaváltozása is. A paraméterek a következőképpen néznek ki (a GPF és a DPF fájlok esetében egyaránt):

```
1  #
2  # Free energy model 140n coefficients:
3  #
4  FE_vdW_coeff = 0.1485
5  FE_estat_coeff = 0.1146
6  FE_hbond_coeff = 0.0656
7  FE_tors_coeff = 0.3113
8  FE_desol_coeff = 0.1711
```

A fentiek sorrendben: van der Waals, ϵ energia súlyozási paraméterei, továbbá, a hidrogénkötések és a nehézatomokat tartalmazó, kötéseik körül szabadon forgónak definiált ligandbeli részek, és a ligandmolekulák deszolvációjának súlyozási paraméterei (ezeknek az értékeknek a módosítása nem ajánlott).
A kötéseik körül szabadon forgónak definiált ligandbeli részeket a felhasználó adhatja meg, vagy az *autotors* alkalmazás segítségével készíthetők el. Az első esetben a PDBQ fájlformátumban használatos kulcsszavak segítségével definiálhatók a szabadsági fokok (Mivel a dolgozat kísérletében merev dokkolást hajtok végre, *ROOT* és *ENDROOT* kulcsszavak közé zárom a ligandmolekulákat, ami azt jelenti, hogy a molekulában nincsenek kötéseik körül forgó részek). Az alkalmazások az aromás és nem aromás szénmolekulákat megkülönböztetik, az előbbit A szimbólummal kell jelölni a bemeneti fájlokban.

0.4. Klaszterező algoritmusok

0.4.1. Bevezető, a klaszterező algoritmusokról általában

A klaszterezés feladata egy objektumhalmaz olyan partícionálása, vagy halmazrendszerrel való lefedése, amelyben a halmaz két különböző objektuma akkor kerül ugyanabba a részhalmazba (klaszterbe), ha *hasonlóak*, és akkor kerülnek különböző részhalmazokba, ha kevésbé *hasonlóak*. Ennek a hasonlóságnak a pontos meghatározásához egy hasonlóság- vagy `távolság-függvény`t definiálhatunk (tehát metrikát), amellyel két objektum annál inkább hasonló, minél kisebb a köztük lévő távolság. Ekkor az objektumok hasonlósága közelségükkel lesz azonos. Ugyanakkor előfordulhat, hogy a tényleges klaszterszerkezet meghatározásához, az objektumhalmaz két különböző objektumának ugyanabba a részhalmazba kerülésének kritériumaként nem adhatunk meg egy globális, az egész halmazon érvényes, távolságkorlátot, hanem azt régiónként változtatva kell meghatározzuk, az objektumok eloszlásához igazodva, vagy más, a halmaz lokális sajátosságainak figyelembevételét lehetővé tevő módszert kell választanunk, amely ettől különböző kritériumok alapján határozza meg az objektumok részhalmazokba sorolását.

Néhány klaszterező módszer áttekintése következik, a teljesség igénye nélkül, a dolgozatban felmerülő objektumhalmaz, a közel 2000 dimenziós, valós vektorok, klaszterszerkezetének felderítéséhez használt eljárásokból.

0.4.2. K-means

A k-közép algoritmus (k-means algorithm, [2]) egy iteratív partícionáló algoritmus. Vektortérben alkalmazható egy ponthalmaz csoportokra bontására.

A bemeneti paraméterei: egy n pontból álló halmaz – a partícionálandó adatbázis, és k természetes szám, ahol k a leendő részhalmazok számát jelenti.

Az algoritmus kezdetben választ k pontot a halmazból, véletlenszerűen. Az összes többi pontot besorolja a legközelebbi kiválasztott ponthoz, így klasztereket képez a halmazon. Ezek után, általános lépésben, meghatározzuk a létrejött klaszterek középpontjait, és a következő lépésben az összes pontot a legközelebbi új középponthoz soroljuk. Az algoritmus akkor áll meg, amikor a klaszterek középpontjától vett négyzetes hibaösszeg nem, vagy csak minimálisan, változik.

Természetesen, ilyen módon, a négyzetes hibafüggvény lokális minimumába (is) érkezhetünk. Az eljárást célszerű több kezdeti véletlen választással indítani.

Az algoritmus futási ideje t iterációs lépés esetén $O(nkt)$.

Az eljárás leginkább egymástól jól elkülönülő halmazokból álló klaszterszerkezet felderítésére használható. A k paraméter helyes megadása ekkor is komplikált lehet.

Ezen dolgozat alkalmazásában a k-means eljárás az adatbázis egészére kezdetben, és a más módszerekkel kialakított klaszterszerkezet finomítására egyaránt alkalmazható. Hiszen, ha már van fogalmunk a közelítő klaszterszerkezetről, akkor a k paraméter és a kezdeti kiindulási pontok sokkal célravezetőbben adhatók meg.

0.4.3. Sűrűség-alapú klaszterezés

A sűrűség-alapú módszerek a tér sűrűbb, összefüggő területeit tekintik klaszteréknek, a kisebb sűrűségű, köztes területekbe eső objektumokat zajként definiáljuk. A sűrűség-alapú megközelítéssel lehetségessé válik tetszőleges alakú és nagyságú klaszterhalmazok felderítése.

A továbbiakban feltesszük, hogy adott a klaszterezendő, véges számú objektumot tartalmazó **D** adatbázis (halmaz), az objektumok közötti távolságfüggvény(metrika), továbbá ε valós és $MinPts$ természetes számok, ahol ε-t elérési, $MinPts$-t sűrűségi paraméternek nevezzük.

A **D** halmaz egy **p** objektumának *ε-környezetén* azon objektumok **D**-beli részhalmazát értjük, amelyek távolsága az adott objektumtól legfeljebb ε.

Most következik néhány fogalom definíciója, amelyek a sűrűség-alapú klaszterező módszerek kifejtéséhez szükségesek.

0.4.1. Definíció. *A **D** halmaz egy objektumát **belső objektum**nak nevezzük, ε és $MinPts$ feltételekkel, ha ε-környezete legalább $MinPts$ objektumot tartalmaz.*

Belső objektumok definiálása azt a szemléletet formalizálja, amely szerint a klasztertag objektumok környezete – ennek a környezetnek az alakja a távolságfüggvény megválasztásától függ – megfelelő számú objektumot tartalmaz, azaz kellőképpen "sűrű". Emellett a klaszter határoló objektumait is célszerű klasztertagoknak tekinteni. A határoló objektumok esetében viszont nem követelhetjük meg ugyanazt a sűrűség-feltételt a környezetükre vonatkozóan.

A következő definíciók meghatározzák a határoló objektumokat is, tehát azokat, amelyek környezete nem annyira sűrű, mint a klaszter belsejében lévő objektumoké, de a klaszter belső objektumainak kellő közelségében vannak. A további definíciók is ε és $MinPts$ paraméterekkel értendők.

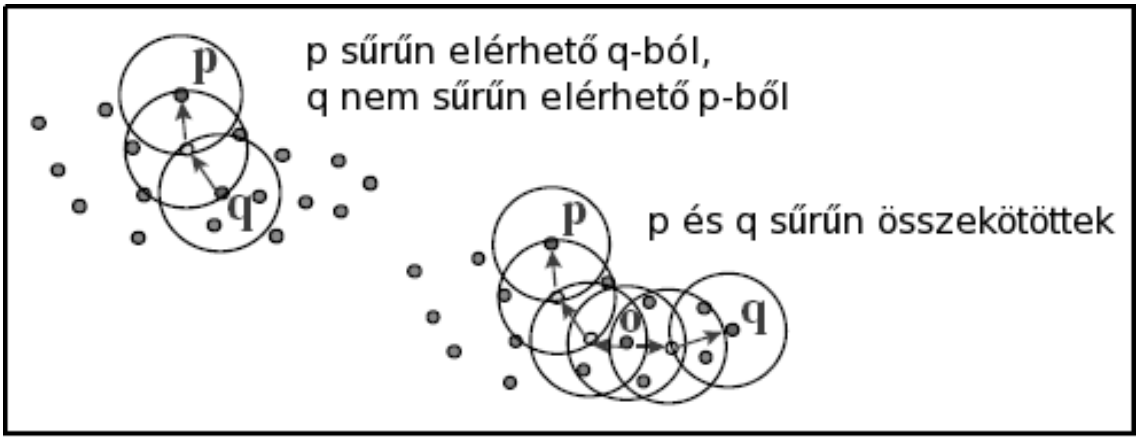
0.4.2. Definíció. *A halmaz **p** objektuma **közvetlenül sűrűn elérhető** a halmaz **q** objektumából, ha xtextbfq belső objektum és **p** a **q** ε-környezetében van.*

Egy tetszőleges objektum, tehát belső objektum vagy nem, csak belső objektumból lehet közvetlenül sűrűn elérhető.
Eddig egy objektum saját (lokális) tulajdonságai lettek megfogalmazva (valamelyik klaszterbe tartozás vagy nem tartozás, vagy valamelyik klaszter belső pontja stb) A következő definíció már a klaszterek összefüggőségének meghatározásához hasznos.

0.4.3. Definíció. ***p** objektum **sűrűn elérhető q**-ból, ha létezik $p_1, \ldots, p_n$ objektumok sorozata, ahol $p_1 = q$, $p_n = p$, továbbá p_i belső objektum, és p_{i+1} közvetlenül sűrűn elérhető p_i-ből $i = 1, \ldots, n-1$ esetén.*

A sűrűn elérhetőség tranzitív reláció, ugyanakkor asszimetrikus, hiszen **q**-nak belső objektumnak kell lennie, míg **p**-re ez nem feltétlenül igaz. Tehát klasztertag lehet minden, a klaszter egy belső objektumából sűrűn elérhető objektum. Ezek alapján a klaszterhalmaz két – beleértve a határolókat is, vagyis nem belső objektumokat – objektumának viszonyát a következő szimmetrikus relációval határozhatjuk meg.

0.4.4. Definíció. ***p* és *q* sűrűn összekötött**, *ha létezik olyan* ***o*** *belső objektum, amiből* ***p*** *is és* ***q*** *is sűrűn elérhető.*



6. ábra. Sűrűn elérhetőség és sűrűn összekötöttség MinPts=4 paraméterrel

A következő algoritmus a fenti definíciókat felhasználva találja meg a sűrűségi klasztereket.

DBSCAN

A DBSCAN (Density Based Spatial Clustering of Applications with Noise — zajos alkalmazások sűrűség-alapú térbeli klaszterezése) [3] algoritmus a – páronként – sűrűn összekötött objektumok maximális halmazait tekinti klaszteresnek. Azokat az objektumokat, amelyek egyetlen klaszterhez sem tartoznak, a speciális, *ZAJ* nevű halmazhoz rendeljük.

0.4.5. Definíció. *Legyen D objektumok halmaza. Ekkor C egy klaszter, ε és MinPts paraméterekkel, ha nemüres részhalmaza D-nek és teljesíti az alábbi feltételeket:*

1. ***Maximalitás.*** *Ha $p \in C$ és q sűrűn elérhető p-ből, ε és MinPts feltételekkel, akkor $q \in C$ is teljesül.*
2. ***Összefüggőség.*** *Ha $p, q \in C$, akkor p és q sűrűn összekötöttek ε és MinPts feltételekkel.*

0.4.6. Definíció. *A D halmaz egy eleme a ZAJ halmazhoz tartozik, ha egyetlen klaszterben sem szerepel.*

Ezek alapján egy klaszter legalább MinPts elemből áll.

A következő két egyszerű lemma kimondja az intuitív elképzelések helyességét, és egyben útmutatást ad a klaszterek megtalálásához használandó algoritmus felépítéséhez.

1. Lemma. *Legyen $p \in D$ belső objektum. Ekkor $O = \{o |\ o \in D$ és o sűrűn elérhető p-ből, ε és MinPts feltételekkel$\}$ halmaz klasztert alkot D-ben, ε és MinPts paraméterekkel.*

2. Lemma. *Legyen C egy klaszter ε és MinPts paraméterekkel, $p \in C$ egy belső objektum. Ekkor C pontosan azokból az objektumokból áll, amelyek sűrűn elérhetőek p-ből.*

A fenti lemmákból következik, hogy az algoritmus úgy állapíthatja meg a bemeneti halmaz klaszterszerkezetét, hogy annak egy tetszőleges belső objektumából kiindulva, megtalálja az abból sűrűn elérhető objektumok halmazát, és ezzel meghatározza azt a legbővebb klasztert, amelyiknek ez az objektum tagja. Ennek az eljárásnak alávetve a bemeneti halmaz összes (belső) objektumát, megtaláljuk az összes klasztert.

A DBSCAN algoritmus pszeudokódja tehát a következőképpen néz ki.

Bemenet: $D, \varepsilon, MinPts$;
  // *Kezdetben minden objektum feldolgozatlan*
$Klaszter\_Index := 1$;
for $i = 1, \ldots, |D|$ **do**
  if $D(i).Klaszter\_ID == FELADOLGOZATLAN$ **then**
    if **KLASZTER_BOVITESE**$(D(i), Klaszter\_Index, \varepsilon, MinPts)$ **then**
      $Klaszter\_Index = Klaszter\_Index + 1$;
    end if
  end if
end for
  // *a főciklus vége*

KLASZTER_BOVITESE($Objektum, Klaszter\_ID, \varepsilon, MinPts$**);**
  // *függvény leírása*
TAR:= ε_környezet($Objektum, \varepsilon$);
if $TAR.Meret < MinPts$ **then**
  Klasztert_hozzarendel($Objektum$,ZAJ);
  RETURN $False$;
else
  Klasztert_hozzárendel(TAR,$Klaszter\_ID$);
  TAR.Töröl($Objektum$);
  while TAR <> Üres **do**
    Akt_Objektum:=TAR.Kivesz();
    TAR_AKT:= ε_környezet($Akt\_Objektum, \varepsilon$);
    if TAR_AKT.Méret >= $MinPts$ **then**
      for i =1... TAR_AKT.Méret **do**
        $Kov\_Objektum$:=TAR_AKT(i);
        if $Kov\_Objektum.Klaszter\_ID \in \{FELDOLGOZATLAN, ZAJ\}$ **then**
          if Kov_Objektum.Klaszter_ID = $FELDOLGOZATLAN$ **then**
            TAR.Betesz($Kov\_Objektum$);
          end if
          Klasztert_hozzárendel($Kov\_Objektum$,$Klaszter\_ID$);
        end if
      end for

```
    end if
    TAR.Töröl(Akt_Objektum);
  end while
  RETURN True;
end if
```

Kezdetben minden objektum a FELDOLGOZATLAN klaszterhez tartozik. A $Klaszter\_Index$ változóban az aktuális klaszter sorszámát tároljuk, kezdetben ez 1 (első létrejövő klaszter). A főciklusban a bemeneti halmaz, eddig feldolgozatlan, objektumaira egyenként alkalmazzuk az KLASZTER_BOVITESE függvényt és inkrementáljuk az aktuális klaszter sorszámát, amennyiben igaz visszatérési értéket kaptunk a függvénytől (vagyis, ha az adott belső objektumhoz tartozó új klaszter már nem bővíthető tovább). Miután a bemeneti D halmaz végére értünk, a főciklus véget ér.

A KLASZTER_BOVITESE függvénnyel növeljük az eddig feldolgozatlan belső objektumnál létrejövő új sűrűségi klaszter méretét, elérve annak maximumát.

A függvény a főciklustól egy feldolgozatlan *Objektum* objektumot, a következő klaszter $Klaszter\_ID$ indexét és az ε elérési, illetve $MinPts$ sűrűségi paramétereket kapja meg bemenetként.

A függvény legelején meghatározzuk az *Objektum* ε-környezetét és annak objektumait a TAR tárolóba tesszük. A következő elágazásnál ellenőrizzük, hogy az *Objektum* belső objektum-e, vagyis tartalmaz-e legalább $MinPts$ elemet ε-környezete, és ha nem, akkor a ZAJ halmazhoz rendeljük, végül a függvény hamis visszatérési értékkel tér vissza a főciklus if-elágazásába. Megjegyzendő, hogy ez a ZAJ-hoz rendelés nem végleges, az *Objektum* később még lehet közvetlenül sűrűn elérhető a halmaz egy belső objektumából, de ezt egy másik, belső objektum feldolgozásánál állapítjuk majd meg.

Az elágazás *else* ágán, tehát, az *Objektum* egy feldolgozatlan belső objektum. Ekkor megvan minden információnk, hogy megtaláljuk az *Objektum*-hoz tartozó maximális klasztert, ami a lemma alapján az *Objektum*-ból sűrűn elérhető objektumhalmazzal azonos.

Kezdetben az *Objektum* ε-környezetéhez tartozó objektumokhoz hozzárendeljük a $Klaszter\_ID$ klaszterindexet, mivel az *Objektum*-ból sűrűn elérhetőek. Az *Objektum* ε-környezetét már besoroltuk klaszterbe, így azt töröljük a TAR tárolóból.

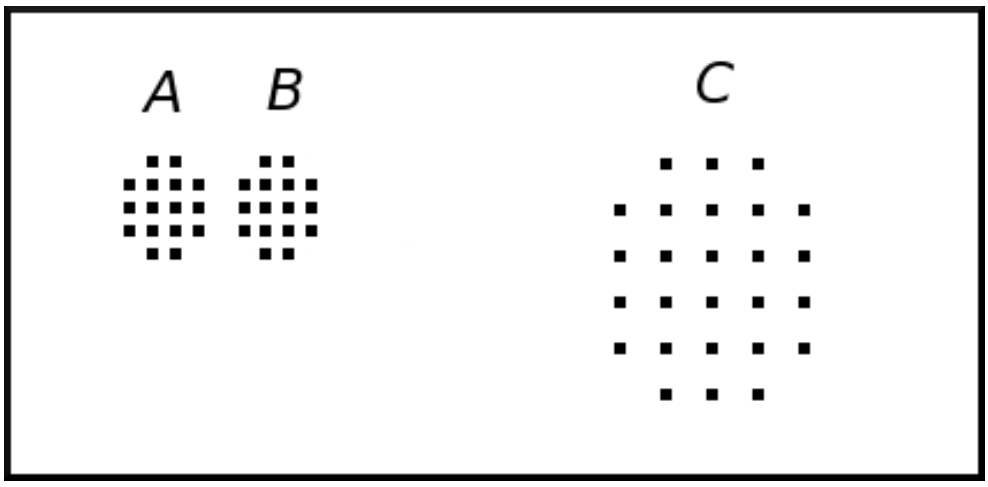
A következő *while*-ciklussal a TAR tároló elemeit az *Akt_Objektum* változónak adjuk értékül - törölve azokat a tárolóból - amíg el nem fogynak. Az *Akt_Objektum* ε-környezetét a TAR_AKT tárolóba tesszük, és ellenőrizzük, hogy mérete eléri-e a $MinPts$ sűrűségi korlátot. Amennyiben igen, vagyis, ha *Akt_Objektum* belső objektum, akkor TAR_AKT elemeit is meg kell vizsgálnunk, hiszen az eredeti *Objektum* objektumból sűrűn elérhetőek lesznek, valamint ε-környezetük is stb., amennyiben belső objektumok lennének.

Tehát, ha *Akt_Objektum* belső objektum, akkor környezetét – TAR_AKT tároló elemeit – megvizsgáljuk a következő for-ciklusban, egyenként a $Kov\_Objektum$ változóhoz rendelve az objektumokat. A következő elágazásoknál, amennyiben a $Kov\_Objektum$ feldolgozatlan, úgy betesszük a kiindulási TAR tárolóba későbbi feldolgozásra (hiszen a sűrűn elérhetőség tranzitivitása miatt az eredeti objektumból nem csak $Kov\_Objektum$, hanem annak ε-környezete is sűrűn elérhetőek lesznek, belső objektum esetén, lsd. fenti lemmák) és hozzárendeljük a Klaszter_ID számú klaszterhez.

Ha a *Kov_Objektum* a ZAJ halmazba tartozik, úgy oda csak abban az esetben kerülhetett, ha a KLASZTER_BOVITESE függvény legelső elágazásában már kiderült róla, hogy nem belső objektum. Így kikerül a ZAJ halmazból és hozzárendeljük a Klaszter_ID sorszámú klaszterhez, de későbbi feldolgozása fölösleges, mivel belőle további sűrűn elérhető objektumok nem származhatnak.

A DBSCAN algoritmus futásideje alapesetben $O(n^2)$. Ez csökkenthető speciális adatszerkezetek használatával $O(n * log(n))$-re.

Az algoritmus nagyon érzékeny bemeneti paramétereire. Ugyanakkor az ε elérési és $MinPts$ sűrűségi paraméter megfelelő kiválasztása bonyolult feladat, és nem mindig lehetséges a halmaz klaszterszerkezetének meghatározása globálisan érvényes paraméterekkel, a klaszterek különböző sűrűsége miatt.



7. ábra. A DBSCAN algotitmus számára bonyolult eset 2 dimenzióban.

Ezen a példán jól látható, hogy $MinPts$ paramétert, például, 5-nek választva az algoritmus vagy zajnak definiálja a C halmazt, vagy egy klaszternek látja az A és B halmazokat és a másik klaszter C lesz (az A és B közötti távolság épp a C pontjainak rácstávolságával egyenlő, és az A és B pontjainak rácstávolságának a duplája).

OPTICS

A DBSCAN algoritmusnak, lényegében, általánosítása az OPTICS (Ordering Points To Identify the Clustering Structure - a pontok rendezése a klaszterszerkezet meghatározásához) [5] sűrűség-alapú módszer. Legalábbis, ami a közelítő, sűrűségi klaszterszerkezet meghatározását illeti. Ugyanis, tulajdonképpen, az optimális bemeneti ε elérési paraméter meghatározását is lehetővé teszi, sőt, különböző sűrűségű klaszterek megtalálása, észlelése is lehetséges a segítségével.

Objektumainkat ebben a szakaszban *pontok*nak hívjuk, igazodva az algoritmus elnevezéséhez.

Az OPTICS algoritmus célja a sűrűségi klaszterszerkezet (sűrűségi klaszterszerhalmazok) meghatározása a legtöbb lehetséges ε elérési paraméterre egyszerre, továbbá az

adatbázis különböző helyein csoportosuló(sűrűsödő), egymástól eltérő sűrűségű klaszterek megtalálása. Az algoritmus, szerkezetileg, kiterjesztett DBASCAN-ként működik, amely felderíti a klaszterszerkezetet az összes, a generáló ε-nál kisebb, paraméterre.

Az OPTICS azon a megfigyelésen alapul, hogy fixálva a $MinPts$ sűrűségi paramétert, a kevésbé sűrű sűrűségi klaszterek (vagyis a nagyobb ε paraméterrel meghatározottak) teljes egészében tartalmazzák a sűrűbb klasztereket (tehát az előbbinél kisebb ε-nal meghatározottakat). Ez abból az állításból következik, hogy a belső pont tulajdonság és a sűrűn elérhetőség is érvényben marad ε növelésekor a pontokra.

A sűrűségi klasztereknek ezt a tulajdonságát felhasználva, a bővülő (egyre kevésbé sűrű) klasztereket megtalálhatjuk az adathalmaz pontjainak, egy speciális sorrendben történő, feldolgozásával. Először, előbbiek szerint, a sűrűbb klaszterekbe tartozó, tehát kisebb ε paraméter mellett belső pontokat, illetve az ezekből sűrűn elérhető pontokat dolgozzuk fel. A feldolgozott pontok sorrendjét, és a hozzájuk rendelt két értéket tároljuk, a belső_távolságot és az elérési_távolságot. Klasztertagságot nem rendelünk pontokhoz. Ennek a két jellemzőnek a definíciója a kövekező:

0.4.7. Definíció. *Legyen adott* ***p*** *pont a halmazból, és legyenek adottak ε pozitív valós és $MinPts$ pozitív egész paraméterek. Jelentse* ***MinPts_távolság****(p)* ***p****-től a* ***p*** *MinPts. legközelebbi szomszédjáig vett távolságot. Ekkor* ***belső_távolság***$_{\varepsilon,MinPts}(p) :=$

$$\begin{cases} UNDEF, & \textit{ha } \boldsymbol{p} \textit{ nem belső pont } \varepsilon \textit{ és } MinPts \textit{ feltételekkel;} \\ \boldsymbol{MinPts\_távolság}(p), & \textit{különben.} \end{cases}$$

Egy **p** pont belső_távolsága az a legkisebb $\varepsilon' \leq \varepsilon$ valós szám, amire **p** belső pont, ε' és $MinPts$ paraméterekkel. Ha ilyen ε' nem létezik, vagyis ha **p** még a definiáló ε-nal sem belső pont, akkor **p** belső_távolsága UNDEF.

0.4.8. Definíció. *Legyenek adottak* ***p*** *és* ***o****, két pont a halmazból, illetve ε távolság és $MinPts$ pozitív egész paraméterek. Ekkor* ***p*** *pont* ***o****-ra vonatkozó elérési_távolsága:*

elérési_távolság$(p, o)_{\varepsilon,MinPts} :=$

$$\begin{cases} UNDEF, & \textit{ha } \boldsymbol{o} \textit{ nem belső pont } (\varepsilon \textit{ és } MinPts); \\ max\left(\textit{belső\_távolság(o), távolság(o,p)}\right), & \textit{különben.} \end{cases}$$

Tehát **p** pont **o**-ra vonatkozó elérési_távolsága az a legkisebb ε' távolság, amire **p** közvetlenül sűrűn elérhető **o**-ból, ε' és $MinPts$ paraméterek mellett. Emellett megköveteljük, hogy **o** belső pont legyen legfeljebb ε paraméternél. Ha ez nem teljesül, akkor az elérési_távolság UNDEF. p elérési_távolsága függ attól, hogy melyik o pontra viszonyítva számoljuk. A belső távolság, definíciója szerint, lehet nagyobb ε-nál, ugyanakkor meg kell jegyezni, hogy az algoritmus folyamán csak olyan pontoknál állapítjuk meg, amelyekre ez nem állhat fenn.

Az OPTICS algoritmus további, részletesebb leírása három lényeges részre osztható. A főciklus, a *KITERJESZTÉS()* és a *OrderSeeds::Feltölt()* függvények vázolására. Az algoritmus a következőképpen készíti el a bemeneti halmaz sorbarendezését , minden pontnak tárolva a belső_távolságát és a ponthoz tartozó elérési távolságot, a fenti definícióknak megfelelően.

```
Bemenet: D, ε, MinPts, SorrendFile;
    // kezdetben minden pont Feldolgozatlan.
  SorrendFile.megnyit();
  for i = 1, ..., |D| do
    Objektum:= D(i);
    if Objektum.Feldolgozatlan then
      KITERJESZTÉS(Objektum, D, ε, MinPts, SorrendFile);
    end if;
  end for;
  SorrendFile.zár();    // a főciklus vége

  KITERJESZTÉS(Objektum, D, ε, MinPts, SorrendFile);
    // függvény leírása
  SZOMSZÉDOK := D.Szomszédok(Objektum, ε);
  Objektum.Feldolgozatlan := HAMIS;
  Objektum.Elérési_távolság := UNDEFINED;
  Objektum.Belső_távolság(SZOMSZÉDOK, ε, MinPts);
  SorrendFile.Kiír(Objektum);
  if Objektum.Belső_távolság <> UNDEFINED then
    OrderSeeds.Feltölt(SZOMSZÉDOK, Objektum);
    while NOT OrderSeeds.Üres() do
      AktObjektum := OrderSeeds.Következő();
      SZOMSZÉDOK:=D.Szomszédok(AktObjektum, ε);
      AktObjektum.Feldolgozatlan := HAMIS;
      AktObjektum.Belső_távolság(SZOMSZÉDOK, ε, MinPts);
      SorrendFile.Kiír(AktObjektum);
      if AktObjektum.Belső_távolság <>UNDEFINED then
        OrderSeeds.Feltölt(SZOMSZÉDOK, AktObjektum);
      end if;
    end while;
  end if;

  OrderSeeds::Feltölt(SZOMSZÉDOK, KözépObjektum);
    // függvény leírása
  k_távolság := KözépObjektum.Belső_távolság;
  for ALL Objektum FROM SZOMSZÉDOK do
    if Objektum.Feldolgozatlan then
      Új_elérési_táv:=
      max(k_távolság,KözépObjektum.Távolság(Objektum));
      if Objektum.Elérési_távolság=UNDEFINED then
        Objektum.Elérési_távolság := Új_elérési_táv;
        Beszúr(Objektum, Új_elérési_táv);
      else
        if Új_elérési_táv<Objektum.Elérési_távolság then
```

```
            Objektum.Elérési_távolság := Új_elérési_táv;
            Léptet(Objektum, Új_elérési_táv);
          end if;
        end if;
      end if;
    end for;
```

Az algoritmus bemenete a D adathalmaz, ε és $MinPts$ paraméterek, illetve a $SorrendFile$, ahová a kimenet kerül, vagyis az adatbázispontok indexei és a hozzájuk tartozó belső távolság és elérési távolság. Kezdetben az adatbázis minden pontja feldolgozatlan.

A ***főciklus***hoz tartozó programrészben megnyitjuk a $SorrendFile$-t írásra, és forciklussal a D bemeneti adathalmaz összes elemére, amennyiben az addig feldolgozatlan maradt, lefuttatjuk a *KITERJESZT()* metódust. Miután ezt megtettük, a $SorrendFile$-t lezárjuk és az algoritmus véget ér.

A ***KITERJESZT()*** függvény a főciklustól egy feldolgozatlan *Objektum*-ot, a bemeneti D halmazt, ε és $MinPts$ paramétereket, továbbá a $SorrendFile$ címét kapja paraméterként.

Első lépésként a *D.Szomszédok()* metódusával a *SZOMSZÉDOK* tárolóba kerül az *Objektum* ε-környezete. A bemeneti *Objektum Feldolgozatlan* attribútumát $HAMIS$-ra állítjuk, hiszen őt most feldolgozzuk. Az *Objektum Elérési_távolság*-át $UNDEFINED$-ra állítjuk, mivel ez előtt a pontok feldolgozása megszakadt, ezeknél az elérési bemeneti paramétereknél, vagy épp most kezdődött el az algoritmus. Felderítjük *Belső_távolság*át az *Objektum* osztályának *Belső_távolság()* függvényével, amely a *SZOMSZÉDOK* halmazt (tehát az *Objektum* ε-környezetét) valamint ε és $MinPts$ értékeket kap meg paraméterként, a definíciónak megfelelően. Végül az *Objektum*ot kiírjuk a $SorrendFile$-ba a *Belső_távolság*ával és az *Elérési_távolság*ával együtt (a függvény meghívásánál átadott paraméterpont esetén az *Elérési_távolság* mindig $UNDEFINED$ lesz).

Amennyiben az *Objektum Belső_távolság*a nem $UNDEFINED$, tehát, ha belső pont a generáló ε és $MinPts$ paraméterekkel, akkor a következő if elágazásnál továbblépünk, különben a program irányítása visszatér a főciklushoz. Ha tehát az *Objektum Belső_távolság*a $\neq UNDEFINED$, akkor az *OrderSeeds* tárolóba – amelyik *Elérési_távolság* szerint növekvő sorrendben tárolja a belehelyezett pontokat, lsd. lentebb – belehelyezzük az *Objektum SZOMSZÉDOK*-ban lévő ε-környezetét az *OrderSeeds.Feltölt()* metódussal (itt fontos, hogy ezek a pontok az *Objektum* környezetéhez tartoznak, az OrderSeeds::Feltölt() metódusának leírása lentebb következik).

A soron következő while-ciklus addig fut, amíg az *OrderSeeds* tárolóban van pont. Az *OrderSeeds.Következő()* metódus a legkisebb *Elérési_távolság*gal rendelkező soronkövetkező pontot veszi ki az *OrderSeeds* tárolóból, és rendeli az az *AktObjektum* változóhoz. A *SZOMSZÉDOK* halmazba most az *AktObjektum* ε-környezete kerül, amelyet ismét a *D.Szomszédok()* metódussal állítunk elő. Továbbá az *AktObjektum Feldolgozatlan* attribútumát $HAMIS$-ra állítjuk, mivel a következőkben feldolgozzuk (kiírjuk) és meghatározzuk annak belső_távolságát. Ezek után kiírjuk az *AktObjektum*-ot is a $SorrendFile$-ba paramétereivel együtt.

Amennyiben az *AktObjektum* is belső pont volt, legfeljebb ε elérési paraméterrel,

akkor annak – már a *SZOMSZÉDOK* halmazban lévő – ε-környezetét is feltöltjük az OrderSeeds tárolóba, a már említett *OrderSeeds::Feltölt()* metódussal. Ekkor a program irányítása visszatér a while-ciklushoz, amely addig folytatja a folyamatot, mindig a legkisebb elérési távolsággal rendelkező pontot véve, amíg találunk további, legfeljebb a generáló ε paraméternél, közvetlenül sűrűn elérhető pontokat az, eredetileg a függvény inicializálásánál szereplő, *Objektum*-ból. Ha tehát a sorozat megszakad, akkor egy újabb klaszterhalmaz építése következik egyetlen belső objektumból kiindulva, a sűrűbb klasztertől (vagyis a kisebb ε'-paraméterrel sűrűségi klaszterből) kiindulva, egészen a generáló ε paraméternél sűrűségi klaszterekig bezárólag.

Az ***OrderSeeds::Feltölt()*** metódus a *KITERJESZT()* függvénytől megkapja a *KözépObjektum* pontot, és annak *SZOMSZÉDOK* halmazban lévő ε-környezetét. Feladata a halmazban szereplő pontok beillesztése az *OrderSeeds* tárolóba, amely a pontokat, elérési távolságuk szerinti, növekvő sorrendben tárolja, mégpedig ezt a minimális elérési távolságot mindig a - az addig látótérbe került - legközelebbi belső ponthoz viszonyítva határozzuk meg, amelyből a beillesztendő pont közvetlenül sűrűn elérhető.

Ez a művelet a következőképpen zajlik: *k_távolság* változónak adjuk értékül a bemeneti *KözépObjektum* belső távolságát (amely biztosan nem $UNDEFINED$, mivel csak a, legalább ε-nál, belső pontokra hívtuk meg a *Feltölt()* függvényt). A következő for-ciklussal a *SZOMSZÉDOK* halmaz összes pontjára, az Objektum változónak adva értékül azokat, leellenőrizzük, hogy az feldolgozatlan-e, és csak a feldolgozatlan pontokkal foglalkozunk a továbbiakban (a feldolgozott pontok környezetét már felderítettük és azokat kiírtuk a $SorrendFile$-ba). Az *Új_elérési_táv* változó értéke lesz a *k_távolság* és a *KözépObjektum*-tól az *Objektum*-ig vett távolságok maximuma (ahogyan az elérési távolság definíciójában szerepel). Ha most az *Objektum* elérési távolsága UNDEFINED, akkor az *Objektum*-hoz hozzárendeljük az *Új_elérési_táv* elérési távolságot és betesszük az *OrderSeeds* tárolóba a *Beszúr()* metódussal. Eddig ott biztosan nem szerepelt, ugyanis csak definiált elérési távolsággal kerülhetett be, és az is igaz, hogy mindenképpen bekerül egy pont az *OrderSeeds*-be ha definiáltuk az elérési távolságát, ugyanis a programnak csak ezen a szakaszán fordul elő. Különben, ha már van az *Objektum*-nak elérési távolsága (természetesen a *KözépObjektum*-tól különböző objektumhoz viszonyítva), akkor leellenőrizzük, hogy a mostani *Új_elérési_táv* elérési távolság kisebb-e az előzőnél, és ha kisebb, előrébb léptetjük az *Objektum*ot az *OrderSeeds* elérési távolságok szerint rendezett tárolóban a *Léptet()* metódussal.

Az OPTICS tehát, egyetlen belső objektumból kiindulva, egy sűrűbb *virtuális* klasztert bővítve, az eredeti objektumból sűrűn elérhető további objektumokkal rekurzívan bővítve, építi fel a nagyobb, kevésbé sűrű *virtuális* klasztert, egészen a generáló ε-nál definiálható sűrűségi klaszterig bezárólag. Eközben klasztertagságokat nem feleltetünk meg a pontoknak. Az klaszterező információt a pontok sorrendje, és a hozzájuk tartozó két eltárolt paraméter tartalmazza. Ha például egy új klaszter indulásánál egy, a *KITERJESZT()* függvény meghívásánál paraméterként átadott objektum után nem sokkal, a sorrendben következő objektumokat vizsgáljuk, ezek mind egy sűrűbb klaszter tagjai lesznek. Ennek a sűrűségnek a mértékét a pontokhoz rendelt elérési távolság mutatja, ami egy klaszteren belül, a klaszterező sorrendben haladva, növekszik, vagy legalábbis

nem lépi át a megkövetelt elérési paramétert. Ez biztosítja a klasztertagok egy bizonyos, korábban szereplő, belső pontból való közvetlenül sűrűn elérhetőségét a megfelelő elérési paramétereken belül.

Strukturális ekvivalenciája miatt, az OPTICS futásideje közel azonos a DBSCAN algoritmusáéval, tehát $O(n^2)$. Ami indexelési technikák használatával, amik a környezetek felderítését gyorsíthatják, javítható $O(n * log(n))$-re. Mindkét esetben, ugyanis, a pontok ε-környezetének felderítése domináns. [5]-ben említett, különböző kísérletek sorozatából következtetve, átlagosan, 1.6-szorosa az OPTICS futásideje a DBSCAN-énak azonos adathalmazokon.

A DBSCAN algoritmus által megtalált klaszterek akár közvetlenül is kinyerhetők a klaszterező sorrendből (némely határoló objektum, a feldolgozás sorrendjétől függően, lemaradhat az őt megillető klasztertagságból). Ezt a következő pszeudokód részletben vázolt módon tehetjük meg.

DBSCAN_Klaszterek_Kinyerése ($Sorbarendezett\_D, \varepsilon', MinPts$)
// Előfeltétel, hogy a ε' távolság kisebb legyen a generáló ε-nál
Klaszter_ID := ZAJ; *// ZAJ++=1;*
for $i = 1, \ldots, |Sorbarendezett\_D|$ **do**
Objektum:= Sorbarendezett_D(i);
if Objektum.Elérési_távolság> ε' **then**
if Objektum.Belső_távolság $\leq \varepsilon'$ **then**
Klaszter_ID := Klaszter_ID+1;
Objektum.klaszter_ID := Klaszter_ID;
else
Objektum.klaszter_ID := ZAJ;
end if;
else
Objektum.klaszter_ID := Klaszter_ID;
end if;
end for;

A sűrűségi klaszterszerkezetet, tehát (néhány ponttól eltekintve), az objektumok számától függő, lineáris időben határozhatjuk meg, egyszerűen végigmenve a pontokon és klasztertagságokat megfeleltetve.

Ezen dolgozat szempontjából azonban, a fentebb már nagyjából vázolt, a klaszterező sorbarendezésben rejlő, klaszterszerkezet feltárási, és az ezt lehetővé tevő vizualizációs lehetőség az érdekesebb. A magas dimenziójú adatoknál, ugyanis, nagyon nehéz objektív képet kapni a klaszterszerkezetről.

Az adathalmaz klaszter-sorbarendezésének vizualizációjából a következőképpen láthatjuk a klaszterstruktúrát, ha minden ponthoz megjelenítjük annak elérési távolságát, egy 1 pixel széles, elérési távolság magas téglalapot ábrázolva, a sorrendnek megfelelően. Egy tetszőleges p pont elérési távolságát egy, a sorbarendezésben korábban előforduló, o

pontra vonatkozóan határoztuk meg, és az objektumok közül mindig a minimális elérési távolságút dolgoztuk fel, továbbá, ennek, az o belső pontnak a belső távolsága, a definíció szerint, kisebb p elérési távolságánál. Ennek következtében, a sorbarendezett elérési távolságokat ábrázoló ábrán egy vízszintes vonalat húzva ε' magasságban, az ez alatt lévő, kisebb elérési távolságú, folytonos szakaszok legfeljebb ε' paraméterű sűrűségi klasztert alkotnak.

Meg kell jegyezni, hogy ez az ábrázolási mód független a bemeneti adatok dimenziószámától, így igen magas dimenziós vektorokból származó elérési távolságok megjelenítésére is alkalmas. Továbbá, ε' távolságnak csak "elég nagynak" kell lennie ahhoz, hogy az összes, kisebb paraméterű sűrűségi klaszterre vonatkozó információt tartalmazza az ábrázolás.

rész II

Az algoritmusok és szoftverek alkalmazása, eredmények

0.4.4. A kísérlet menete és a bemeneti adatok

Mint azt a bevezetőben már említettem, a fehérje-kötőhelyek bizonyos jellemzőjének – ligandok egy 1838 elemű csoportjával való merev dokkolásából származó, valós komponensű, 1838 dimenziós vektorok – klaszterezése segítségével próbálom meghatározni a – dokkolás szempontjából – hasonló fehérje-kötőhelyek csoportjait. Ez a ligandhalmaz a NCI Diversity Set [6]. A fehérje-kötőhelyek halmaza, a PDB adatbázis 4222 elemű részhalmazának [7] kötőhelyei.

A fehérje és ligand fájlok előzetes szűrése után, az aktív tartományok kijelölése következik: a ligandmolekulák és a kötőhelyek középpontjainak meghatározása. Ezek után, a molekulák (fehérjék és ligandok) páronkénti dokkolását végzem AutoDock szoftverrel. Mindegyik kötőhelynek egy energiavektor felel meg: a ligandokkal végzett dokkolás energiáinak vektora. A keletkező energiavektorok halmazán adatbányászati algoritmusokkal keresek hasonló csoportokat, és ezeket elemzem.

0.4.5. A dokkolási paraméterfájlok, a megkapott vektorok

Az adatfájlok konzisztenciájának ellenőrzése után, a ligandok atomjainak résztöltését és a fehérje atomok résztöltését és szolvencia paramétereit meg kell állapítani az AutoDock szoftver alkalmazásához. (mol2-ből pdbq és pdb-ből pdbqs fájlformátumba való konvertálás). Ezekhez az átalakításokhoz az AutoDock standard alkalmazásait használtam. Továbbá, mivel merev dokkolást hajtok végre, szükség van a ligandok tartalmának ROOT, ENDROOT kulcsszavak közé fogására.

Az AutoGrid program .gpf (grid parameter file) paraméterfájlt igényel az energiapotenciálok számolásához. A GPF fájlt automatikusan generálja egy AWK alapú program, ezt a lépést kiváltottam a dokkolást végző programommal. Ennek segítségével az AutoGrid a ligand és a fehérje fájlt megkapva paraméterként, generálja a ligandban előforduló atomokhoz tartozó energiapotenciálokat. Mivel az energiapotenciálok számolásának folyamata rendkívül időigényes, több percet vesz igénybe egy PC-n végrehajtva egy ligand-fehérje párra (mérettől és összetételtől függően), és a ligandatomok halmaza nem nagy: így több nagyságrenddel gyorsítható, ha minden fehérjéhez, az összes liganddal való dokkolásához csupán egyszer számoljuk az energiapotenciálokat, viszont a ligandhalmazban előforduló összes atomtípusra.
A vizsgált atomhalmaz szén (C), aromás szén (A), nitrogén (N), oxigén (O), kén (S), fluor (F), klór (c), bróm (b) és hidrogén (H) atomokból áll, néhány kivételtől eltekintve. Ennek megfelelően a paraméterfájl, amelyet a dockings.pl program generál, a következőképpen néz ki minden fehérje esetén:

```
1  receptor $prot_name.pdbqs              #macromolecule
2  gridfld  $prot_name.maps.fld            #grid_data_file
3  npts     $grid_points               #num.grid points in xyz
4  spacing  $grid_spacing                  #spacing (Angstroms)
5  gridcenter $grid_center  #xyz-coordinates or "auto"
6  types CANOSHFcbX                       #atom type names
7  smooth 0.500 #store minimum energy within radius (Angstroms)
```

```
map $prot_name.C.map     #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6 #C-C lj
nbp_r_eps  3.75 0.0230026 12  6 #C-N lj
nbp_r_eps  3.60 0.0257202 12  6 #C-O lj
nbp_r_eps  4.00 0.0257202 12  6 #C-S lj
nbp_r_eps  3.00 0.0081378 12  6 #C-H lj
nbp_r_eps  3.00 0.0081378 12  6 #C-X lj
nbp_r_eps  2.65 0.0538015 12  6 #C-M lj
sol_par 12.77 0.6844 #C atomic fragmental volume, solvation param.
constant 0.000      #C grid map constant energy
map $prot_name.A.map #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6 #A-C lj
nbp_r_eps  3.75 0.0230026 12  6 #A-N lj
nbp_r_eps  3.60 0.0257202 12  6 #A-O lj
nbp_r_eps  4.00 0.0257202 12  6 #A-S lj
nbp_r_eps  3.00 0.0081378 12  6 #A-H lj
nbp_r_eps  3.00 0.0081378 12  6 #A-X lj
nbp_r_eps  2.65 0.0538015 12  6 #A-M lj
sol_par 10.80 0.1027 #A atomic fragmental volume, solvation param.
constant 0.000 #A grid map constant energy
map $prot_name.N.map #filename of grid map
nbp_r_eps  3.75 0.0230026 12  6 #N-C lj
nbp_r_eps  3.50 0.0237600 12  6 #N-N lj
nbp_r_eps  3.35 0.0265667 12  6 #N-O lj
nbp_r_eps  3.75 0.0265667 12  6 #N-S lj
nbp_r_eps  2.75 0.0084051 12  6 #N-H lj
nbp_r_eps  2.75 0.0084051 12  6 #N-X lj
nbp_r_eps  2.40 0.0555687 12  6 #N-M lj
sol_par  0.00 0.0000#N atomic fragmental volume, solvation param.
constant 0.000 #N grid map constant energy
map $prot_name.O.map #filename of grid map
nbp_r_eps  3.60 0.0257202 12  6 #O-C lj
nbp_r_eps  3.35 0.0265667 12  6 #O-N lj
nbp_r_eps  3.20 0.0297000 12  6 #O-O lj
nbp_r_eps  3.60 0.0297000 12  6 #O-S lj
nbp_r_eps  1.90 0.3280000 12 10 #O-H hb
nbp_r_eps  2.60 0.0093852 12  6 #O-X lj
nbp_r_eps  2.25 0.0621176 12  6 #O-M lj
sol_par  0.00 0.0000 #O atomic fragmental volume, solvation param.
constant 0.236  #O grid map constant energy
map $prot_name.S.map  #filename of grid map
nbp_r_eps  4.00 0.0257202 12  6 #S-C lj
nbp_r_eps  3.75 0.0265667 12  6 #S-N lj
nbp_r_eps  3.60 0.0297000 12  6 #S-O lj
nbp_r_eps  4.00 0.0297000 12  6 #S-S lj
```

```
nbp_r_eps  2.50 0.0656000 12 10 #S-H hb
nbp_r_eps  3.00 0.0093852 12  6 #S-X lj
nbp_r_eps  2.65 0.0621176 12  6 #S-M lj
sol_par  0.00 0.0000 #S atomic fragmental volume, solvation param.
constant 0.000  #S grid map constant energy
#map $prot_name.P.map  #filename of grid map
#nbp_r_eps  4.10 0.0257202 12  6        #P-C lj
#nbp_r_eps  3.85 0.0265667 12  6        #P-N lj
#nbp_r_eps  3.70 0.0297000 12  6        #P-O lj
#nbp_r_eps  4.10 0.0297000 12  6        #P-S lj
#nbp_r_eps  3.10 0.0093852 12  6        #P-H lj
#nbp_r_eps  3.10 0.0093852 12  6        #P-X lj
#nbp_r_eps  2.75 0.0621176 12  6        #P-M lj
#sol_par  0.00 0.0000 #P atomic fragmental volume, solvation param.
#constant 0.000 #P grid map constant energy
map $prot_name.H.map  #filename of grid map
nbp_r_eps  3.00 0.0081378 12  6 #H-C lj
nbp_r_eps  2.75 0.0084051 12  6 #H-N lj
nbp_r_eps  1.90 0.3280000 12 10 #H-O hb
nbp_r_eps  2.50 0.0656000 12 10 #H-S hb
nbp_r_eps  2.00 0.0029700 12  6 #H-H lj
nbp_r_eps  2.00 0.0029700 12  6 #H-X lj
nbp_r_eps  1.65 0.0196465 12  6 #H-M lj
sol_par  0.00 0.0000 #H atomic fragmental volume, solvation param.
constant 0.118  #H grid map constant energy
map $prot_name.F.map #filename of grid map
nbp_r_eps  3.54 0.0162608 12  6 #F-C lj
nbp_r_eps  3.29 0.0167954 12  6 #F-N lj
nbp_r_eps  3.15 0.0187852 12  6 #F-O lj
nbp_r_eps  3.54 0.0187852 12  6 #F-S lj
nbp_r_eps  2.54 0.0059400 12  6 #F-H lj
nbp_r_eps  2.54 0.0059400 12  6 #F-X lj
nbp_r_eps  2.19 0.0392931 12  6 #F-M lj
sol_par  0.00 0.0000 #F atomic fragmental volume, solvation param.
constant 0.000  #F grid map constant energy
map $prot_name.c.map   #filename of grid map
nbp_r_eps  4.04 0.0302197 12  6 #c-C lj
nbp_r_eps  3.79 0.0311999 12  6 #c-N lj
nbp_r_eps  3.65 0.0348827 12  6 #c-O lj
nbp_r_eps  4.04 0.0348827 12  6 #c-S lj
nbp_r_eps  3.04 0.0110335 12  6 #c-H lj
nbp_r_eps  3.04 0.0110335 12  6 #c-X lj
nbp_r_eps  2.69 0.0729729 12  6 #c-M lj
sol_par  0.00 0.0000 #c atomic fragmental volume, solvation param.
constant 0.000  #c grid map constant energy
```

```
 98  map $prot_name.b.map #filename of grid map
 99  nbp_r_eps  4.17 0.0358776 12  6 #b-C lj
100  nbp_r_eps  3.92 0.0370508 12  6 #b-N lj
101  nbp_r_eps  3.77 0.0414166 12  6 #b-O lj
102  nbp_r_eps  4.17 0.0414166 12  6 #b-S lj
103  nbp_r_eps  3.17 0.0130977 12  6 #b-H lj
104  nbp_r_eps  3.17 0.0130977 12  6 #b-X lj
105  nbp_r_eps  2.81 0.0866349 12  6 #b-M lj
106  sol_par  0.00 0.0000 #b atomic fragmental volume, solvation param.
107  constant 0.000  #b grid map constant energy
108  #map $prot_name.I.map  #filename of grid map
109  #nbp_r_eps  4.36 0.0427235 12  6        #I-C lj
110  #nbp_r_eps  4.11 0.0441342 12  6        #I-N lj
111  #nbp_r_eps  3.96 0.0493465 12  6        #I-O lj
112  #nbp_r_eps  4.36 0.0493465 12  6        #I-S lj
113  #nbp_r_eps  3.36 0.0156073 12  6        #I-H lj
114  #nbp_r_eps  3.36 0.0156073 12  6        #I-X lj
115  #nbp_r_eps  3.01 0.1032075 12  6        #I-M lj
116  #sol_par  0.00 0.0000 #I atomic fragmental volume, solvation param.
117  #constant 0.000    #I grid map constant energy
118  map $prot_name.X.map     #filename of grid map
119  nbp_r_eps  3.00 0.0081378 12  6 #X-C lj
120  nbp_r_eps  2.75 0.0084051 12  6 #X-N lj
121  nbp_r_eps  2.60 0.0093852 12  6 #X-O lj
122  nbp_r_eps  3.00 0.0093852 12  6 #X-S lj
123  nbp_r_eps  2.00 0.0029700 12  6 #X-H lj
124  nbp_r_eps  2.00 0.0029700 12  6 #X-X lj
125  nbp_r_eps  1.65 0.0196465 12  6 #X-M lj
126  sol_par  0.00 0.0000 #X atomic fragmental volume, solvation param.
127  constant 0.000   #X grid map constant energy
128  elecmap $prot_name.e.map #electrostatic potential map
129  dielectric -0.1146 #<0,distance-dep.diel; >0,constant
```

Ahol a $prot_name, $grid_points, $grid_spacing, $grid_center paraméterek megfelelően a fehérjefájl nevét, dimenziónkénti rácspontok számát, rácstávolságot és a rács középpontját jelentik.
A fehérjefájl neve megegyezik az épp következőével. A rácspontok számát minden dimenzióban 60-nak választottam, a rácstávolságot 0.375 Angstromnek. Ezek az alapértelmezett értékek és a kísérletnek tökéletesen megfeleltek. A rács középpontjainak a fehérjék aktív központjainak középpontjai lettek kiválasztva.

Az AutoDock program bemenete a fehérjefájl, a ligandfájl és a .dpf paraméterfájl (docking parameter file). Ekkor egy nehezen értelmezhető nagy kimenetfájlba írja a futás eredményeit.
A programot úgy módosítottam, hogy minden fehérjéhez egyetlen fájlba írja az összes dokkolás kimenetét, egy 1838 dimenziós valós vektort készítve. A .dpf fájlokat generáló

AWK alkalmazást úgy módosítottam, hogy egyetlen lefutást hajtson végre minden ligandhoz (run kulcsszó).

Összefoglalva: a dockings.pl program a fehérjéket és a ligandokat tartalmazó könyvtárt megkapva paraméterként, az összes fehérjéhez ciklusban elkészíti a GPF fájlt, a fentebb vázolt paraméterezéssel, ezek után alkalmazza a mkdpf3 AWK alkalmazást a DPF fájlok elkészítéséhez, módosításokkal; végül ciklusban az összes ligandmolekulát dokkolja a fehérjével, és az energiavektorokat adja vissza kimenetként.

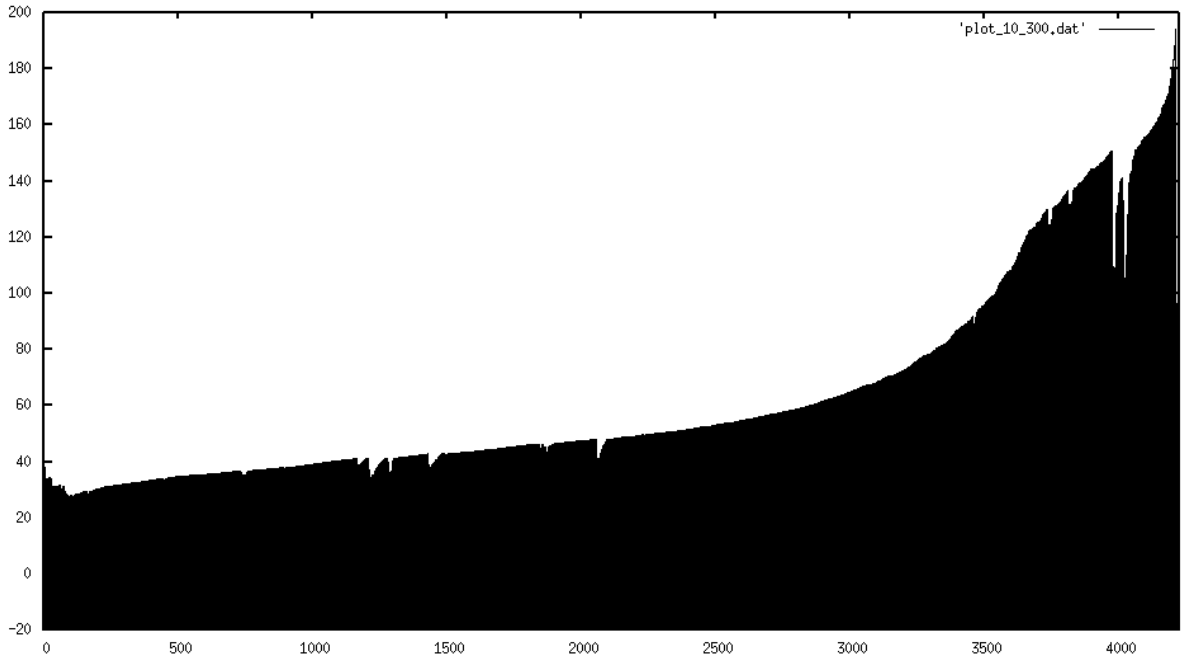
Az eljárás körülbelül két hetet vett igénybe 4 PC-n futtatva.

0.4.6. Klaszterezési eredmények és értékelés

A dokkolt vektorhalmazon lefuttatva az implementált OPTICS algoritmust, adott esetben, $MinPts = 10$ és $\varepsilon = 300$ paraméterekkel, négyzetes távolságfüggvénnyel, a következő típusú kimenet keletkezik (ebben '-1'-el az UNDEFINED értéket jelölöm).

```
# optics_10_300.dat file
10gs 37.8323 -1
1uoq 33.8842 37.8323
1e8n 33.605 33.8842
1ee1 35.0233 33.605
1blh 34.3813 33.605
2dld 34.0944 33.605
1dc6 35.6177 33.605
1mzj 35.5358 33.8842
1est 34.3093 33.8842
1inc 35.0298 33.9308
```

Az első oszlopban a fehérje-komplexum pdb kódja, a második és harmadik oszlopokban a belső és az elérési távolság található. Ennek a sorbarendezésének a pontoknak, az elérési távolság ábrázolása a következő.



8. ábra. $MinPts = 10$ és $\varepsilon = 300$ paraméterű elérési távolság diagramm.

Ezt az ábrát elemezve látható, hogy ennél a távolságfüggvénynél, egy nagyobb és sűrűbb klaszter mellett, több kisebb, valamivel kevésbé sűrű klaszterhalmaz is felfedezhető.

Általában, ε paramétert elég nagynak kell választani, tehát úgy, hogy a klaszterezendő halmaz majdnem minden pontja egy klaszterbe tartozzon ε maximális paraméternél. Ekkor az elérési távolságok diagrammja tartalmazni fogja az összes, ε-nál kisebb paraméterű (tehát sűrűbb) klaszterekre vonatkozó információt. Erre a bemeneti halmazra, ez is látható az ábrán, $\varepsilon = 300$ bőven elegendő generáló elérési távolságnak.

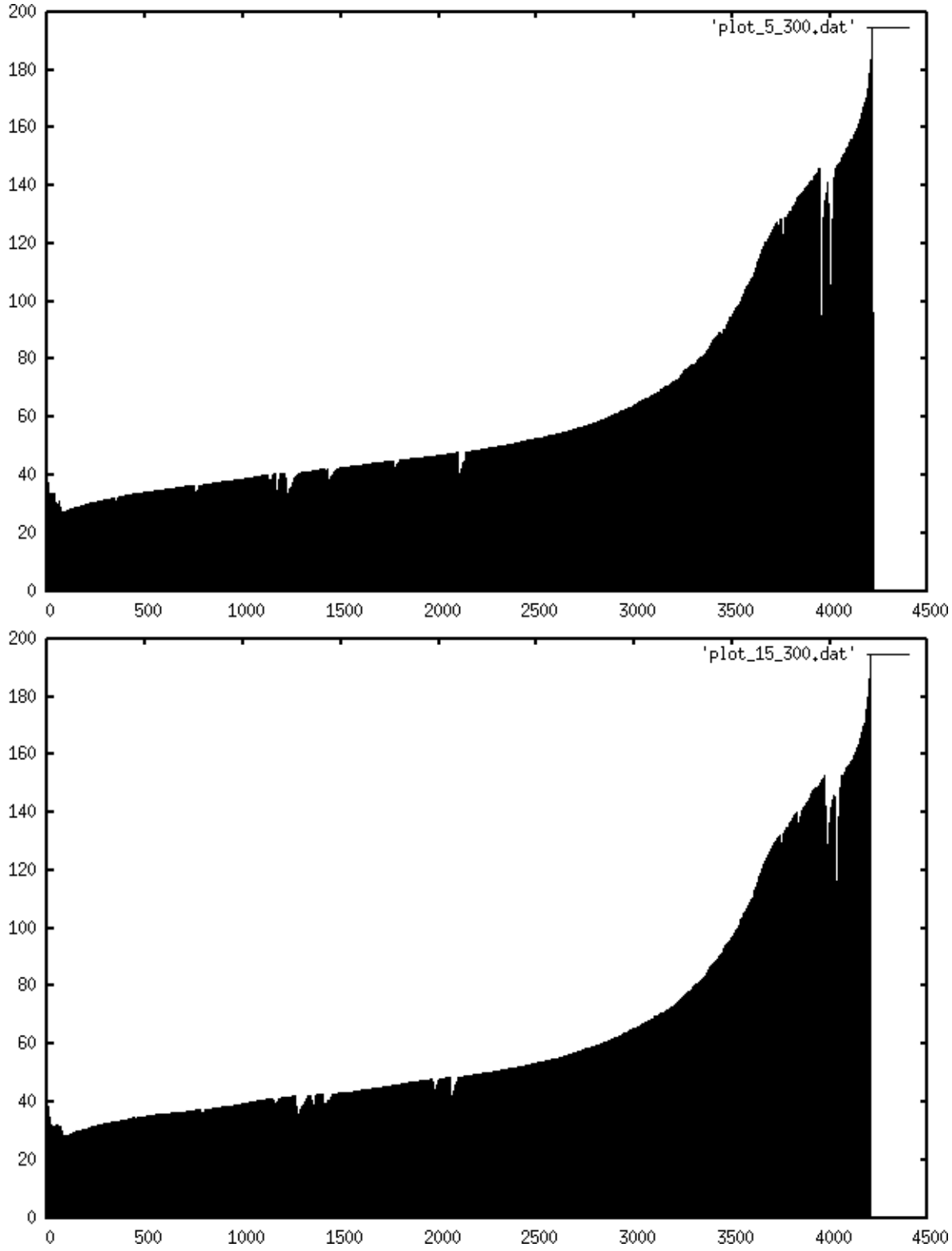
A programot futtattam sok különböző $MinPts$ paraméterre, 4-től 20-ig. Az eredmények, lényegében, strukturálisan, ekvivalensek lettek. [5] tanulsága szerint általános érvényű jelenség, hogy az elérési távolság diagramm alakja nem igazán függ $MinPts$ megválasztásától. Ami megfigyelhető, $MinPts$ növelésekor az ábra "kisimul", míg kisebb értékekre "tüskéssé" válik.

A végső eredményklaszterezést, négyzetes távolságfüggvényre, az OPTICS algoritmus esetén, a $MinPts = 10$ és $\varepsilon = 300$ paraméterekre keletkező klaszterezési sorrendből állapítottam meg.

Alternatív klaszterezési lehetőségek

A k-means algoritmus a teljes bemeneti vektorhalmaz klaszterezését adja meg, ennek összes hátrányával együtt: a klaszterekhalmazok optimális számát nehéz becsülni, a klaszterekhalmazok nagy átmérőjűek, és az elszigetelt vektorok (a többi vektortól nagy távolságra lévők) is feltétlenül klaszterhalmazba kerülnek. Így az eljárás nem igazán alkalmas a kutatás céljának elérésére ezen adatok esetében.

Dimenziócsökkentési technikákkal is növelhető a klaszterhalmazok nagysága. Így még mindig maradhatnak zajhoz tartozó elemek, viszont romlik a klaszterek minősége. Az eredeti cél a fehérjekötőhelyek csoportosítása későbbi dokkolások előrejelzéséhez, csak igazán szigorú klaszterezési feltételekkel érhető el.



9. ábra. $MinPts = 5$ és $MinPts = 15$, $\varepsilon = 300$ paraméterű elérési távolság diagrammok.

rész III

Mellékelt kódok

0.4.7. Dokkolás megvalósításához írt programok

dockings.pl

```
1  #!/usr/bin/perl -w
2
3  ###############################################################
4  #       Dockola‘s el_mkdpf3, autogrid3 segitsegevel           #
5  # Energiavektor keszitese: ciklusban lefut minden feherjere #
6  ###############################################################
7
8  #       Parameterek
9          $input_dir='.';
10         $awk_dir='/user/dock/awk';
11 #
12
13 my ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) =
14    localtime(time);
15
16 open(DB,"/data/PDB/sc-PDB/sc-PDB/db.txt");
17   @db=<DB>;
18   chomp(@db);
19 close(DB);
20
21 sub dpf3gen {# atirja a dpf3gen.awk parametereit :
22              # ga_pop_size, ga_num_evals, ga_num_generations
23
24   open(DPF3GEN,"<$awk_dir/dpf3gen.awk");
25     my @dpf3gen=<DPF3GEN>;
26   close(DPF3GEN);
27
28   foreach my $sor (@dpf3gen){
29
30   if($sor=~/print "ga_pop_size/){
31   $sor="    print \"ga_pop_size ".$_[0].
32     "  # number of individuals in population\"\n";}
33
34   if($sor=~/print "ga_num_evals/){
35   $sor="    print \"ga_num_evals ".$_[1].
36      "  # maximum number of energy evaluations\"\n"}
37
38   if($sor=~/print "ga_num_generations/){
39   $sor="    print \"ga_num_generations ".$_[2].
40      "  # maximum number of generations\"\n";}
41   }
42
```

```
43    open(DPF3GEN,">$awk_dir/dpf3gen.awk");
44      print DPF3GEN @dpf3gen;
45    close(DPF3GEN);
46
47  }
48
49
50  sub center { #a bemenet sc-PDB 4 karakterrbol allo kodja.
51               #kimenet: a ligand kozeppontjanak 3 koordinataja.
52      my $sc;
53      foreach $sc (@db) {
54       if ($sc=~/^$_[0]/){
55      my ($p_name,$p_x,$p_y,$p_z,$p_tx,$p_ty,$p_tz,$p_anum,
56           $l_x,$l_y,$l_z,$l_tx,$l_ty,$l_tz,$l_anum)= split(/ +/,$sc);
57        return ($l_x,$l_y,$l_z);
58       }
59      }
60  }
61
62
63  sub grid { #a bemenet sc-PDB 4 karakterrbol allo kodja.
64             #kimenet: a ligand terjedelem-teglatestenek meretei.
65    my $sc;
66    foreach $sc (@db) {
67     if ($sc=~/^$_[0]/){
68      my ($p_name,$p_x,$p_y,$p_z,$p_tx,$p_ty,$p_tz, $p_anum,
69      $l_x,$l_y,$l_z,$l_tx,$l_ty,$l_tz,$l_anum)= split(/ +/,$sc);
70     return ($l_tx,$l_ty,$l_tz);
71     }
72    }
73  }
74
75  sub gpf { # legyart egy .gpf-et a pdbq@NCI szamara
76    #Parameterek: pl ('1tmp_prot','16 16 16','.75','1.417 1.994 -0.001')
77    # =(protein neve kiterjeszes nelkul,racspontok#,racstav.,racskozep)
78  my $prot_name=$_[0];
79  my $grid_points=$_[1];
80  my $grid_spacing=$_[2];
81  my $grid_center=$_[3];
82
83  open(GPF_FILE,">$prot_name.gpf");
84  select(GPF_FILE);
85
86  print <<END_GPF;
87  receptor $prot_name.pdbqs             #macromolecule
```

```
gridfld  $prot_name.maps.fld           #grid_data_file
npts     $grid_points              #num.grid points in xyz
spacing  $grid_spacing                 #spacing (Angstroms)
gridcenter $grid_center  #xyz-coordinates or "auto"
types CAOSNH       #atom type names + H elott PI kene, N elott F
smooth 0.500      #store minimum energy within radius (Angstroms)
map $prot_name.C.map               #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6  #C-C lj
nbp_r_eps  3.75 0.0230026 12  6  #C-N lj
nbp_r_eps  3.60 0.0257202 12  6  #C-O lj
nbp_r_eps  4.00 0.0257202 12  6  #C-S lj
nbp_r_eps  3.00 0.0081378 12  6  #C-H lj
nbp_r_eps  3.00 0.0081378 12  6  #C-X lj
nbp_r_eps  2.65 0.0538015 12  6  #C-M lj
sol_par 12.77 0.6844 #C atomic fragmental volume, solvation param.
constant 0.000                         #C grid map constant energy
map $prot_name.A.map                         #filename of grid map
nbp_r_eps  4.00 0.0222750 12  6  #A-C lj
nbp_r_eps  3.75 0.0230026 12  6  #A-N lj
nbp_r_eps  3.60 0.0257202 12  6  #A-O lj
nbp_r_eps  4.00 0.0257202 12  6  #A-S lj
nbp_r_eps  3.00 0.0081378 12  6  #A-H lj
nbp_r_eps  3.00 0.0081378 12  6  #A-X lj
nbp_r_eps  2.65 0.0538015 12  6  #A-M lj
sol_par 10.80 0.1027 #A atomic fragmental volume, solvation param.
constant 0.000                         #A grid map constant energy
map $prot_name.O.map                         #filename of grid map
nbp_r_eps  3.60 0.0257202 12  6  #O-C lj
nbp_r_eps  3.35 0.0265667 12  6  #O-N lj
nbp_r_eps  3.20 0.0297000 12  6  #O-O lj
nbp_r_eps  3.60 0.0297000 12  6  #O-S lj
nbp_r_eps  1.90 0.3280000 12 10  #O-H hb
nbp_r_eps  2.60 0.0093852 12  6  #O-X lj
nbp_r_eps  2.25 0.0621176 12  6  #O-M lj
sol_par  0.00 0.0000 #O atomic fragmental volume, solvation param.
constant 0.236                         #O grid map constant energy
map $prot_name.S.map                         #filename of grid map
nbp_r_eps  4.00 0.0257202 12  6  #S-C lj
nbp_r_eps  3.75 0.0265667 12  6  #S-N lj
nbp_r_eps  3.60 0.0297000 12  6  #S-O lj
nbp_r_eps  4.00 0.0297000 12  6  #S-S lj
nbp_r_eps  2.50 0.0656000 12 10  #S-H hb
nbp_r_eps  3.00 0.0093852 12  6  #S-X lj
nbp_r_eps  2.65 0.0621176 12  6  #S-M lj
sol_par  0.00 0.0000 #S atomic fragmental volume, solvation param.
```

```
133 constant 0.000                          #S grid map constant energy
134 #map $prot_name.F.map                          #filename of grid map
135 #nbp_r_eps  3.54 0.0162608 12  6  #F-C lj
136 #nbp_r_eps  3.29 0.0167954 12  6  #F-N lj
137 #nbp_r_eps  3.15 0.0187852 12  6  #F-O lj
138 #nbp_r_eps  3.54 0.0187852 12  6  #F-S lj
139 #nbp_r_eps  2.54 0.0059400 12  6  #F-H lj
140 #nbp_r_eps  2.54 0.0059400 12  6  #F-X lj
141 #nbp_r_eps  2.19 0.0392931 12  6  #F-M lj
142 #sol_par  0.00 0.0000  #F atomic fragmental volume, solvation param.
143 #constant 0.000                 #F grid map constant energy
144 map $prot_name.N.map                   #filename of grid map
145 nbp_r_eps  3.75 0.0230026 12  6  #N-C lj
146 nbp_r_eps  3.50 0.0237600 12  6  #N-N lj
147 nbp_r_eps  3.35 0.0265667 12  6  #N-O lj
148 nbp_r_eps  3.75 0.0265667 12  6  #N-S lj
149 nbp_r_eps  2.75 0.0084051 12  6  #N-H lj
150 nbp_r_eps  2.75 0.0084051 12  6  #N-X lj
151 nbp_r_eps  2.40 0.0555687 12  6  #N-M lj
152 sol_par  0.00 0.0000    #N atomic fragmental volume, solvation param
153 constant 0.000                          #N grid map constant energy
154 map $prot_name.H.map                          #filename of grid map
155 nbp_r_eps  3.00 0.0081378 12  6  #H-C lj
156 nbp_r_eps  2.75 0.0084051 12  6  #H-N lj
157 nbp_r_eps  1.90 0.3280000 12 10  #H-O hb
158 nbp_r_eps  2.50 0.0656000 12 10  #H-S hb
159 nbp_r_eps  2.00 0.0029700 12  6  #H-H lj
160 nbp_r_eps  2.00 0.0029700 12  6  #H-X lj
161 nbp_r_eps  1.65 0.0196465 12  6  #H-M lj
162 sol_par  0.00 0.0000   #H atomic fragmental volume, solvation param.
163 constant 0.118                   #H grid map constant energy
164 elecmap $prot_name.e.map               #electrostatic potential map
165 dielectric -0.1146                #<0,distance-dep.diel; >0,constant
166 END_GPF
167
168 close(GPF_FILE);
169 return 1;
170 }
171
172 sub strpad {#pl:  &strpad($vmi,'0',20,'e');
173 my $in_strpad=$_[0];
174
175         if($_[3] eq 'e') {
176                 while(length($in_strpad)<$_[2]){
177                         $in_strpad=$_[1].$in_strpad;
```

```
178                 }
179         } elsif($_[3] eq 'u') {
180                 while(length($in_strpad)<$_[2]){
181                         $in_strpad.=$_[1];
182                 }
183         } else { return $in_strpad; }
184
185         return $in_strpad;
186 }
187
188
189  opendir(DIR,$input_dir) || die "can't open $input_dir";
190  @dir=readdir(DIR);
191  chomp(@dir);
192  @dir=sort(@dir);
193  @dir_=@dir;
194  closedir DIR;
195  #print @dir;
196
197  &dpf3gen(50,250000,130000);
198
199
200
201  foreach $sor (@dir) {
202
203
204     if ($sor=~/.pdbqs$/) {
205
206   $macro=substr($sor,0,-6);
207
208
209   $center=join(' ',&center(substr($macro,0,4)));
210   &gpf($macro,'60 60 60','0.375',$center);
211
212   select(STDERR); print "PROTEIN $sor :\n\n";
213   select(STDOUT); print "PROTEIN $sor :\n\n";
214
215   `autogrid3 -p $macro.gpf -l $macro.glg`; # /mnt/hda8/autogrid3/
216
217
218     #print "\n\n:: $ii. dokkola`s ::\n\n";
219     foreach $file (@dir_){
220       if ($file=~/.pdbq$/) {
221         $lig=substr($file,0,-4);
222         `el_mkdpf3 $file $sor`;
```

```
223           #print "\n  -->\t$lig,$macro:\n";
224           for($dk=0;$dk<5;$dk++){
225            `/user/dock/autodock/autodock3 -p $lig$macro.dpf -l
226                    $lig$macro$dk.dlg`;
227           }#/RAM1/temp.dlg
228           unlink("$lig$macro.dpf");
229           #unlink("$lig$macro.dlg");
230
231         }
232       }
233
234
235
236
237     unlink("$macro.A.map");unlink("$macro.C.map");
238     unlink("$macro.F.map");unlink("$macro.H.map");
239     unlink("$macro.N.map");unlink("$macro.O.map");
240     unlink("$macro.S.map");unlink("$macro.e.map");
241     unlink("$macro.glg");
242     unlink("$macro.gpf");
243     unlink("$macro.maps.xyz");
244     unlink("$macro.maps.fld");
245
246     #mkdir("$jj");
247     #unlink("$lig$macro.dlg");
248     #rename("$lig$macro.dlg","$jj/$lig$macro.dlg");
249     #unlink("$macro.d");
250
251     #unlink("$sor");
252
253       }
254
255    }
256
257
258
259
260
```

manalysis.pl

```
1   #!/usr/bin/perl -w
2
3   #   20050924
4   #
```

```
#   Megnezi, hogy hol nem stimmel a szerkezet
#    (a konyvtarban talalhato *.mol2 omlesztett mol2-fileokra)
#

$input_dir='.';

 opendir(DIR,$input_dir) || die "can't open $input_dir\n";
 @dir=readdir(DIR);
 chomp(@dir);
 @dir=sort(@dir);
 closedir DIR;


 @tmp_file=();
 $tmp_id=0;
 $tmp_bonds=0;
 $tmp_atoms=0;

 $area_flag='0';
 $molecule_cnt=0;
 $atom_cnt=0;
 $bond_cnt=0;

 $error_flag=0;
 $line_cnt=0;

foreach $file (@dir){
 if($file=~/\.mol2$/){
    print "\n FILE: $file \n";
    open(MOL2,$file) || die "can't open $file\n";
   $line_cnt=0;
    while(<MOL2>){
       $line_cnt++;
      if($_=~/^\@<TRIPOS>MOLECULE/){
         if($bond_cnt!=$tmp_bonds){
            print "Error: in bond number of $tmp_id (at $line_cnt)\n";
            $error_flag=1;
         }
         if($area_flag ne '0' && @tmp_file!=($atom_cnt+$bond_cnt+8)){
            print "Error: in molecule size (at $line_cnt)\n";
            $error_flag=1;
         }
         if($error_flag){
            print "\n\n";
            print @tmp_file;
```

```
50              print "\n\n";
51              $error_flag=0;
52           }
53            $area_flag='MOLECULE';
54             $molecule_cnt=0;
55            @tmp_file=$_;
56            next;
57        }
58
59        push(@tmp_file,$_);
60        chomp($_);
61
62        if($_=~/^\@<TRIPOS>ATOM/){
63           if($molecule_cnt!=5){
64              $error_flag=1;
65              print "Error:in $area_flag section of $tmp_id (at $line_cnt)\n";
66           }
67            $area_flag='ATOM';
68             $atom_cnt=0;
69            next;
70        }
71
72        if($_=~/^\@<TRIPOS>BOND/){
73           if($atom_cnt!=$tmp_atoms){
74              $error_flag=1;
75              print "Error: in atom number of $tmp_id (at $line_cnt)\n";
76           }
77            $area_flag='BOND';
78             $bond_cnt=0;
79            next;
80        }
81
82
83        if($area_flag eq 'MOLECULE'){
84         $molecule_cnt++;
85           if($molecule_cnt==1){
86              if(!($_=~/ZINC\d+/)){
87              $error_flag=1;
88              print "Error: in molecule number $tmp_id (at $line_cnt)\n";
89              }
90              $tmp_id=$_;
91              $tmp_id=~s/\s+//g;
92           }
93           if($molecule_cnt==2){
94              if($_=~/^\s+(\d+)\s+(\d+)\s+/){
```

```
                $tmp_atoms=$1;
                $tmp_bonds=$2;
             } else {
                $error_flag=1;
                print "Error: in atom and bond number (at $line_cnt)\n$_\n";
             }
          }
       }

       if($area_flag eq 'ATOM'){
        $atom_cnt++;
        if(!($_=~/^\s+$atom_cnt\s+/)){
           $error_flag=1;
           print "WARNING: ".($atom_cnt+7).
  ". line of $tmp_id doesn't match atom number (at $line_cnt):\n$_\n";
        }
       }

       if($area_flag eq 'BOND'){
        $bond_cnt++;
        if(!($_=~/^\s+$bond_cnt\s+/)){
          $error_flag=1;
          print "WARNING: ".($atom_cnt+$bond_cnt+8).
  ". line of $tmp_id doesn't match bond number (at $line_cnt):\n$_\n";
        }
       }

     }
       if($bond_cnt!=$tmp_bonds){
          print "Error: in bond number of $tmp_id (at $line_cnt)\n";
          $error_flag=1;
       }
       if($error_flag){
          print "\n\n";
          print @tmp_file;
          print "\n\n";
          $error_flag=0;
       }
    close(MOL2);
  }
}

```

```
1   #!/usr/bin/perl -w
2
3   #  20050924
4   #
5   #   Konyvtarszerkezetbe menti az ellenorzott molekulakat, egyenkent
6   #
7
8   $input_dir='';
9   $file_per_dir=10000;
10
11  opendir(DIR,$input_dir) || die "can't open $input_dir\n";
12   @dir=readdir(DIR);
13   chomp(@dir);
14   @dir=sort(@dir);
15   closedir DIR;
16
17
18   @tmp_file=();
19   $tmp_id=0;
20   $tmp_bonds=0;
21   $tmp_atoms=0;
22   $tmp_file_name='';
23
24   $tmp_dir='';
25   $tmp_file_dir='';
26
27   $area_flag='0';
28   $molecule_cnt=0;
29   $atom_cnt=0;
30   $bond_cnt=0;
31
32   $error_flag=0;
33   $line_cnt=0;
34   $file_cnt=0;
35
36  foreach $file (@dir){
37   chdir($input_dir);
38   if($file=~/\.mol2$/){
39     print "\n FILE: $file \n";
40     open(MOL2,$file) || die "can't open $file\n";
41
42    $tmp_file_dir=substr($file,0,4);
43
```

```
44      mkdir($input_dir.$tmp_file_dir);
45      chdir($input_dir.$tmp_file_dir);
46
47        $line_cnt=0;
48        $file_cnt=0;
49       while(<MOL2>){
50         $line_cnt++;
51        if($_=~/\@<TRIPOS>MOLECULE/){
52           if($file_cnt%$file_per_dir==0){
53             chdir($input_dir.$tmp_file_dir);
54             $tmp_dir='/'.(1+int($file_cnt/$file_per_dir));
55            mkdir($input_dir.$tmp_file_dir.$tmp_dir);
56            chdir($input_dir.$tmp_file_dir.$tmp_dir);
57           }
58          #writing the file
59          if($area_flag ne '0'){
60          $file_cnt++;
61           if($error_flag){
62             $tmp_file_name=$tmp_id.".mol2_error";
63           } else {
64             $tmp_file_name=$tmp_id.".mol2";
65           }
66
67           open(MOL2FILE,">$tmp_file_name");
68            print MOL2FILE @tmp_file;
69           close(MOL2FILE);
70          }
71          #
72
73          if($bond_cnt!=$tmp_bonds){
74            print "Error: in bond number of $tmp_id (at $line_cnt)\n";
75            $error_flag=1;
76          }
77          if($area_flag ne '0' && @tmp_file!=($atom_cnt+$bond_cnt+8)){
78            print "Error: in molecule size (at $line_cnt)\n";
79            $error_flag=1;
80          }
81          if($error_flag){
82            print "\n\n";
83            print @tmp_file;
84            print "\n\n";
85            $error_flag=0;
86          }
87
88           $area_flag='MOLECULE';
```

```
89            $molecule_cnt=0;
90           @tmp_file='@<TRIPOS>MOLECULE'."\n";
91           next;
92        }
93
94        push(@tmp_file,$_);
95        chomp($_);
96
97        if($_=~/\@<TRIPOS>ATOM/){
98          if($molecule_cnt!=5){
99            $error_flag=1;
100           print "Error:in $area_flag section of $tmp_id (at $line_cnt)\n";
101         }
102          $area_flag='ATOM';
103           $atom_cnt=0;
104          next;
105       }
106
107       if($_=~/\@<TRIPOS>BOND/){
108         if($atom_cnt!=$tmp_atoms){
109           $error_flag=1;
110           print "Error: in atom number of $tmp_id (at $line_cnt)\n";
111         }
112          $area_flag='BOND';
113           $bond_cnt=0;
114          next;
115       }
116
117
118       if($area_flag eq 'MOLECULE'){
119        $molecule_cnt++;
120         if($molecule_cnt==1){
121           if(!($_=~/ZINC\d+/)){
122           $error_flag=1;
123           print "Error: in molecule number $tmp_id (at $line_cnt)\n";
124           }
125           $tmp_id=$_;
126           $tmp_id=~s/\s+//g;
127         }
128         if($molecule_cnt==2){
129           if($_=~/^\s+(\d+)\s+(\d+)\s+/){
130             $tmp_atoms=$1;
131             $tmp_bonds=$2;
132           } else {
133             $error_flag=1;
```

```
134             print "Error: in atom and bond number (at $line_cnt)\n$_\n";
135           }
136         }
137       }
138
139       if($area_flag eq 'ATOM'){
140        $atom_cnt++;
141        if(!($_=~/^\s+$atom_cnt\s+/)){
142          $error_flag=1;
143          print "WARNING: ".($atom_cnt+7).
144    ". line of $tmp_id doesn't match atom number (at $line_cnt):\n$_\n";
145        }
146       }
147
148       if($area_flag eq 'BOND'){
149        $bond_cnt++;
150        if(!($_=~/^\s+$bond_cnt\s+/)){
151         $error_flag=1;
152         print "WARNING: ".($atom_cnt+$bond_cnt+8).
153    ". line of $tmp_id doesn't match bond number (at $line_cnt):\n$_\n";
154        }
155       }
156
157      }
158       #writing the file
159       if($area_flag ne '0'){
160        if($error_flag){
161          $tmp_file_name=$tmp_id.".mol2_error";
162        } else {
163          $tmp_file_name=$tmp_id.".mol2";
164        }
165
166        open(MOL2FILE,">$tmp_file_name");
167         print MOL2FILE @tmp_file;
168        close(MOL2FILE);
169       }
170       #
171
172       if($bond_cnt!=$tmp_bonds){
173         print "Error: in bond number of $tmp_id (at $line_cnt)\n";
174         $error_flag=1;
175       }
176       if($error_flag){
177         print "\n\n";
178         print @tmp_file;
```

```
179         print "\n\n";
180         $error_flag=0;
181       }
182
183     close(MOL2);
184    }
185   }
186
```

0.4.8. OPTICS algoritmus

optics_main.cpp

```
1   #include "db_object.cpp"
2   #include "File_ptr.cpp"
3   #include "seed_list.cpp"
4
5   double dst[MAX_PTS][MAX_PTS];
6     DB_Object object[MAX_PTS];
7     int object_num=0;
8
9     int MinPts;
10    int eps;
11
12  int main(int argc,char** argv){//MinPts, eps
13    sscanf(argv[1],"%d",&MinPts);
14    sscanf(argv[2],"%d",&eps);
15
16    int cluster[MAX_PTS];
17    int cluster_id=0; //noise
18     for(int i=0;i<MAX_PTS;i++){
19      cluster[i]=cluster_id;
20     }
21
22     SeedList seedlist;
23
24
25   {//beolvassuk az adatokat: object_num db  object[i] (0-tol object_num-1 ig)
26    File_ptr data_file(INPUT_FILE,"r");
27    char line[1000000];
28    char tmp_line[100];
29    int line_char;
30    int tmp_line_ptr;
31    int coord_num;
32
33    double coords[DIM];
34
35    while(fgets(line,1000000,data_file)!=NULL){
36       line_char=0;
37       tmp_line_ptr=0;
38       coord_num=-1;
39       while(line[line_char]!='\n'){ // ! '' a char, "" a string!
40        if(line[line_char]==','){
41          if(coord_num==-1){
42            tmp_line[tmp_line_ptr]='\0';
```

```
43            object[object_num].put_name(tmp_line);
44            tmp_line_ptr=0;
45            coord_num++;
46          } else {
47            tmp_line[tmp_line_ptr]='\0';
48            sscanf(tmp_line,"%lf",&coords[coord_num]);
49            tmp_line_ptr=0;
50            coord_num++;
51          }
52        } else {
53          tmp_line[tmp_line_ptr]=line[line_char];
54          tmp_line_ptr++;
55        }
56        line_char++;
57       }
58        tmp_line[tmp_line_ptr]='\0';
59        sscanf(tmp_line,"%lf",&coords[coord_num]);
60
61      object[object_num].put_vector(coords);
62      object[object_num].num=object_num;
63      object_num++;
64    }
65   }
66
67
68   {
69    for(int pt=0;pt<object_num;pt++){
70       dst[pt][pt]=0.0;
71      for(int pt2=0;pt2<pt;pt2++){
72        dst[pt][pt2]=dist(&object[pt],&object[pt2]);
73        dst[pt2][pt]=dst[pt][pt2];
74      }
75    }
76
77
78    for(int pt=0;pt<object_num;pt++){
79     if(!object[pt].processed){
80       object[pt].processed=true;
81       object[pt].get_neighbors();
82       object[pt].reachability_distance=-1;
83       object[pt].write();
84
85       if(object[pt].core_point){
86         seedlist.update_neighbors(pt);
87
```

```
88
89            int current_pt=seedlist.pop();
90           while(current_pt!=-1){
91             object[current_pt].get_neighbors();
92             object[current_pt].processed=true;
93             object[current_pt].write();
94
95             if(object[current_pt].core_point){
96                seedlist.update_neighbors(current_pt);
97             }
98
99             current_pt=seedlist.pop();
100          }
101       }
102     }
103    }
104
105
106   }
107
108
109  }
```

constants.h

```
1   //adatok dimenzioja
2   #define DIM 1838
3
4   //adatpontok max. szama
5   #define MAX_PTS 4222
6
7   #define INPUT_FILE "/home/elemer/szakdoli/cpp/data/dim_1838_head.dat"
```

File_ptr.cpp

```
1
2
3   class File_ptr
4   {
5     FILE* p;
6    public:
7     File_ptr() { p=NULL; }
8     File_ptr(const char* n, const char* a) { open(n,a); }
9     File_ptr(FILE* pp) { p=pp; }
10   ~File_ptr() { close();}
11    operator FILE*() { return p;}
```

```
12
13    void open(const char* n, const char* a) {
14      if ((p=fopen(n,a))==NULL) {
15        fprintf(stderr,"nincs ilyen file\n");
16      }
17    }
18
19    void close() { if (p) fclose(p); }
20  };
21
```

db_object.h

```
1   #include "constants.h"
2   #include "iostream.h"
3   #include "stdio.h"
4   #include "stdlib.h"
5
6   class DB_Object{
7      protected:
8       double vector[DIM];
9       char name[20];
10
11      public:
12      int num;
13
14      bool processed;
15      bool core_point;
16
17      double core_distance;          // <0  <=> UNDEF.
18      double reachability_distance;  // <0 <=> UNDEF.
19
20      int b_eps[MAX_PTS];
21      int b_eps_max;
22      int b_eps_act;
23
24      DB_Object(double* i);
25      DB_Object();
26      ~DB_Object();
27
28      void put_vector(double*);
29      double abs();
30      double* get_ptr();
31
32      void put_name(char*);
```

```
33      void get_name();
34
35      void get_neighbors();
36      void write();
37
38  };
39
```

db_object.cpp

```
1   #include "db_object.h"
2
3   extern DB_Object object[MAX_PTS];
4   extern int object_num;
5   extern double dst[MAX_PTS][MAX_PTS];
6   extern int MinPts;
7   extern int eps;
8
9
10  double dist(DB_Object* a,DB_Object* b){
11          double dist=0.0;
12          double* ia;
13          double* ib;
14
15          ia=(*a).get_ptr();
16          ib=(*b).get_ptr();
17
18          for(int j=0;j<DIM;j++){
19                  dist+=(ia[j]-ib[j])*(ia[j]-ib[j]);
20          }
21          return sqrt(dist);
22  }
23
24
25  DB_Object::DB_Object(double* i){
26          for(int j=0;j<DIM;j++){
27                  vector[j]=*i;
28                  i++;
29          }
30          processed=false;
31          core_point=false;
32          b_eps_max=0;
33          b_eps_act=0;
34
35          core_distance=-1;
```

```
36          reachability_distance=-1;
37  }
38
39  DB_Object::DB_Object(){
40          processed=false;
41          core_point=false;
42          b_eps_max=0;
43          b_eps_act=0;
44
45          core_distance=-1;
46          reachability_distance=-1;
47  }
48
49  DB_Object::~DB_Object(){
50  }
51
52  void DB_Object::put_vector(double* i){
53          for(int j=0;j<DIM;j++){
54                  vector[j]=*i;
55                  i++;
56          }
57  }
58
59  double DB_Object::abs(){
60          double abs=0.0;
61          for(int j=0;j<DIM;j++){
62                  abs+=vector[j]*vector[j];
63          }
64          return sqrt(abs);
65  }
66
67  double* DB_Object::get_ptr(){
68          return vector;
69  }
70
71  void DB_Object::put_name(char* inndex){
72          for(int i=0;i<20;i++){
73                  name[i]=inndex[i];
74          }
75  }
76
77  void DB_Object::get_name(){
78          cout<<name;
79  }
80
```

```
81  void DB_Object::get_neighbors(){
82          bool inserted;
83          for(int pt2=0;pt2<object_num;pt2++){
84                  if(num!=pt2){
85                          if(dst[num][pt2]<=eps){
86                                   inserted=false;
87                                  for(int i=0;i<b_eps_max;i++){
88                                    if(dst[num][b_eps[i]]>dst[num][pt2]){
89                                      for(int j=b_eps_max;j>i;j--){
90                                          b_eps[j]=b_eps[j-1];
91                                      }
92                                      b_eps[i]=pt2;
93                                      b_eps_max++;
94                                      inserted=true;
95                                      break;
96                                    }
97                                  }
98                                  if(!inserted){
99                                          b_eps[b_eps_max]=pt2;
100                                         b_eps_max++;
101                                 }
102                         }
103                 }
104         }
105         if(b_eps_max>=MinPts){
106                 core_point=true;
107                 core_distance=dst[b_eps[MinPts-1]][num];
108         }
109 }
110
111 void DB_Object::write(){
112   get_name(); cout<<" "<<core_distance<<" "<<reachability_distance<<endl;
113 }
114
115
116
```

seed_list.h

```
1  #include "constants.h"
2
3  extern DB_Object object[MAX_PTS];
4  extern double dst[MAX_PTS][MAX_PTS];
5
6  class SeedList{
```

```
7           protected:
8                   int list[MAX_PTS];
9
10           public:
11                  int max;
12
13                  SeedList();
14                  ~SeedList();
15
16                  void insert(int, double);
17                  void update(int, double);
18                  void update_neighbors(int);
19                  int pop();
20
21  };
22
```

seed_list.cpp

```
1   #include "seed_list.h"
2
3   SeedList::SeedList(){
4     max=0;
5   }
6
7   SeedList::~SeedList(){
8   }
9
10  void SeedList::insert(int pto, double r_d){
11    if(max==0 || object[list[max-1]].reachability_distance>r_d){
12      list[max]=pto;
13      max++;
14    } else {
15      for(int i=0;i<max;i++){
16        if(object[list[i]].reachability_distance<r_d){
17          for(int mi=max;mi>i;mi--){
18            list[mi]=list[mi-1];
19          }
20          list[i]=pto;
21          max++;
22          break;
23        }
24      }
25    }
26  }
```

```
27
28  void SeedList::update(int pto, double r_d){
29     bool move=false;
30    for(int i=0;i<max;i++){
31      if(list[i]==pto){
32        move=true;
33      }
34      if(move){
35        if( i==(max-1) || object[list[i+1]].reachability_distance<=r_d){
36          break;
37        } else {
38          list[i]=list[i+1];
39          list[i+1]=pto;
40        }
41      }
42    }
43  }
44
45  void SeedList::update_neighbors(int pto){
46    double  new_r_dist;
47    for(int i=0;i<object[pto].b_eps_max;i++){
48      if(!object[object[pto].b_eps[i]].processed){
49        if(object[pto].core_distance>dst[object[pto].b_eps[i]][pto]){
50          new_r_dist=object[pto].core_distance;
51        } else {
52          new_r_dist=dst[object[pto].b_eps[i]][pto];
53        }
54
55        if(object[object[pto].b_eps[i]].reachability_distance<0){
56           object[object[pto].b_eps[i]].reachability_distance=new_r_dist;
57           insert(object[pto].b_eps[i],new_r_dist);
58        } else {
59          if(new_r_dist<object[object[pto].b_eps[i]].reachability_distance){
60            update(object[pto].b_eps[i],new_r_dist);
61            object[object[pto].b_eps[i]].reachability_distance=new_r_dist;
62          }
63        }
64      }
65
66     }
67  }
68
69  int SeedList::pop(){
70    if(max>0){
71      max--;
```

```
72      return list[max];
73    } else {
74      return -1;
75    }
76  }
77
```

get_clusters.pl **- DBSCAN megvalósítása**

```
1   #!/usr/bin/perl -w
2   #Extract DBSCAN-clustering
3
4   $eps=$ARGV[0];
5   $in_file=$ARGV[1];
6
7   $cluster_id=0;
8   open(I,$in_file) || die "can't open $in_file!\n";
9           while(<I>){
10                  chomp;
11                  if($_=~/^(.+)\s+(.+)\s+(.+)$/){
12                   $prot_id=$1;
13                   $core_dist=$2;
14                   $reachability_dist=$3;
15
16                   if($reachability_dist>$eps || $reachability_dist<0){
17                          if($core_dist<$eps && $core_dist>0) {
18                                  $cluster_id++;
19                                  print "$prot_id $cluster_id\n";
20                          } else {
21                                  print "$prot_id 0\n";
22                          }
23                   } else {
24                          print "$prot_id $cluster_id\n";
25                   }
26                  } else {
27                          print STDERR " Error in line $_\n"
28                  }
29          }
30  close(I);
```

Irodalomjegyzék

[1] Grolmusz Vince, Hubenkó Elemér: Clustering Binding Sites on Protein Surfaces, European Life Science Organization Meeting (ELSO 2005), 2005. szeptember 1- 4, Drezda, Németország.

[2] E. W. Forgy: Cluster analysis of multivariate data: Efficiency versus interpretability of classifications. Biometric Soc. Meetings, Riverside, California (1965).

[3] Ester M., Kriegel H.-P., Sander J., Xu X.: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining, Portland, OR, 1996, pp. 226-231.

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Michael Wimmer, Xiaowei Xu: "Incremental Clustering for Mining in a Data Warehousing Environment".

[5] Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander: "OPTICS: Ordering Points To Identify the Clustering Structure", Proc. ACM SIGMOD'99 Int. Conf. on Management of Data, Philadelphia PA, 1999.

[6] W.H. Lindstrom, G.M. Morris, R.H. Huey, M.F. Sanner and A.J. Olson. The NCI diversity set for AutoDock, http://www.scripps.edu/pub/olson-web/doc/autodock/ .

[7] Nicodeme Paul, Esther Kellenberger, Guillaume Bret, Pascal Muller, Didier Rognan. Recovering the true targets of specific ligands by virtual screening of the Protein Data Bank. Proteins: Structure, Function and Bioinformatics, 54(4):671 - 680, 2004. http://bioinfo-pharma.u-strasbg.fr/scpdb/scpdb_form.html