
SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

DIPLOMADOLGOZAT

Témavezető:
Dr. Szilágyi László

Végzős hallgató:
Dénes Loránd

2023

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM
MAROSVÁSÁRHELYI KAR
SZÁMÍTÁSTECHNIKA SZAK

Protein osztályozás
Markov Klaszterezéssel

DIPLOMADOLGOZAT

Témavezető:
Dr. Szilágyi László

Végzős hallgató:
Dénes Loránd

2023

UNIVERSITATEA SAPIENTIA DIN CLUJ-NAPOCA
FACULTATEA DE ȘTIINȚE TEHNICE ȘI UMANISTE, TÎRGU-MUREȘ
SPECIALIZAREA CALCULATOARE

Clasificarea secvențelor de proteine prin metoda Markov Clustering

PROIECT DE DIPLOMĂ

Coordonator științific:
Dr. Szilágyi László

Absolvent:
Dénes Loránd

2023

SAPIENTIA HUNGARIAN UNIVERSITY OF TRANSYLVANIA
FACULTY OF TECHNICAL AND HUMAN SCIENCE, TÎRGU-MUREȘ
COMPUTER SCIENCE SPECIALIZATION

Protein classification with Markov Clustering

DIPLOMA THESIS

Advisor:

Dr. Szilágyi László

Student:

Dénes Loránd

2023

UNIVERSITATEA „SAPIENTIA” din CLUJ-NAPOCA

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Specializarea: **Calculatoare**

Viza facultății:



LUCRARE DE DIPLOMĂ

Coordonator științific:

Prof. dr. ing. habil. Szilágyi László

Candidat: **Dénes Loránd**

Anul absolvirii: **2023**

a) Tema lucrării de licență:

Clasificarea secvențelor de proteine prin metoda Markov Clustering

b) Problemele principale tratate:

- Studiu bibliografic privind bazele bioinformaticii
- Metoda de clusterizare Markov clustering și bazele teoretice ale acesteia
- Metode de aliniere ale secvențelor de proteine

c) Desene obligatorii:

- Diagrame UML privind software-ul realizat
- Prezentarea funcționării algoritmului MCL prin reprezentări grafice
- Prezentarea detaliată a interfeței realizate prin imagini

d) Softuri obligatorii:

- Aplicație de clusterizare MCL a secvențelor de proteine
- Interfață grafică de utilizator pentru vizualizarea rezultatelor

e) Bibliografia recomandată:

1. Szilágyi SM, Szilágyi L: A fast hierarchical clustering algorithm for large-scale protein sequence data sets. Computers in Biology and Medicine 48:94–101, 2014
2. Szilágyi L, Szilágyi SM: A modified two-stage Markov clustering algorithm for large and sparse networks. Computer Methods and Programs in Biomedicine 135:15-26, 2016
3. Szilágyi L, Medvés L, Szilágyi SM: A modified Markov clustering approach to unsupervised classification of protein sequences. Neurocomputing 73(13-15):2332-2345, 2010
4. A.Andreeva, et al. Data growth and its impact on the SCOP database: new developments, Nucl. Acids Res.36(2008)D419–D425.
5. S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Biol. 48 (1970) 443–453.

f) Termene obligatorii de consultații: săptămânal

g) Locul și durata practicii: Universitatea „Sapientia” din Cluj-Napoca,

Facultatea de Științe Tehnice și Umaniste din Târgu Mureș

Primit tema la data de: 31.03.2022

Termen

de

predare:

28.06.2023

Semnătura Director Departament

Semnătura coordonatorului

Semnătura responsabilului
programului de studiu

Semnătura candidatului

TARTALOM

1 Bevezetés.....	14
1.1 Informatika felhasználás a biológiában.....	15
1.2 Fehérje és DNS illetve a fehérjeszekvenciák elemzése.....	17
2 Elméleti háttér	19
2.1 Klaszterezés.....	19
2.2 Markov-lánc	20
2.3 Markov-klaszter.....	21
2.4 SCOP és SCOP95 adatbázis.....	22
2.5 Ritka mátrixok és típusaik.....	22
2.6 Felhasznált technológiák	25
C++	25
Felhasználói felület	26
Windows Forms	26
3 Példa Markov klaszterezésre	28
3.1 Inflációs ráta.....	30
4. Program tervezése	37
4.1 Algoritmus.....	37
4.2 Eljárások.....	37
Normalizáció	39
Szimmetrizáció.....	40
Elimináló szimmetrizáció	41
Kiterjesztés.....	41
4.4 Algoritmus.....	42
Változók:	42
Függvények:	42
4.5 Program lépései.....	43

5 Megvalósítás	46
5.1 Szimmetrizáció	46
5.2 Expanzió	47
5.3 Normalizáció	49
5.4 Infláció	50
5.5 Inicializáció	51
5.6 Needleman-Wunsch algoritmus	52
Az illesztés megvalósítása	53
5.7 BLOSUM62	54
6 Felhasználó felület	55
7 Bibliográfia	64

Ábrák jegyzéke

Ábra 1 – Aminósavak neve és jelölései	16
Ábra 2 – Szekvencia ábrázolása	17
Ábra 3 – Diagonális mátrix	23
Ábra 4 – Tridiagonális mátrix	24
Ábra 5 – Sávós mátrixok	24
Ábra 6 – Markov klaszterezés példa MatLab kóddal	28
Ábra 7 – $r=1$ infláció	31
Ábra 8 – $r=2.0972$ infláció	32
Ábra 9 – $r=2.0973$ infláció	33
Ábra 10 – $r=3.11$ infláció	34
Ábra 11 – $r=3.113$ infláció	35
Ábra 12 – $r=7$ infláció	36
Ábra 13 - Esetdiagramm	44
Ábra 14 – Szekvencia diagramm	45
Ábra 15 – Program grafikus felülete	55
Ábra 16 – Inflációs ráta és az epsilon kiválasztása	56
Ábra 17 – Intervallum meghatározása	56
Ábra 18 – Családokban található priteinek számának megjelenítése	57
Ábra 19 – Létrehozott klaszterek ábrázolása	58
Ábra 20 – Megjeleníteni kívánt klaszter kiválasztása	59
Ábra 21 – Klaszterben található proteinek adatai	60
Ábra 22 – Kiválasztott protein	61
Ábra 23 – Összehasonlításra kiválasztott két protein	62
Ábra 24 – Protein illesztés	63

Kivonat

A biológiai jelenségek *jelentős* része visszavezethető a genetikára. A genetika pedig a génszekvenciák alapján működik. A különféle élőlények génszekvenciájának megfejtése a biológiai tudományok nagy kihívása, amely merész becslések szerint is még ötszáz évig munkát ad a tudományos világnak.

Dolgozatom célja egy egyszerű génszekvencia osztályozó algoritmus megvalósítása és beépítése egy grafikus felhasználói felülettel rendelkező szoftverbe, amellyel talán egy apró lépést lehet tenni a génszekvenciák megfejtése felé.

A megvalósított algoritmus génszekvencia párok hasonlósági adatai alapján egy csoportosítást hajt végre egy nagyobb szekvencia adathalmazon. A módszer lehetővé teszi nagy mennyiségű proteinszekvencia egyidejű feldolgozását, a szekvenciák csoportokra való osztását hasonlóság alapján. Mindez azáltal válik lehetővé, hogy a megvalósítás ritka mátrixokkal dolgozik és nem végez el fölösleges műveleteket, mint amilyenek a nullával való összeadás és szorzás. Ugyanakkor a módszernek jelentős memória hatékonysága is van, mert a ritka mátrix nem tárolja a nullás értékeket.

A megvalósított módszer gráfként kezeli a bemeneti szekvencia halmazt. Minden fehérjéhez hozzárendel egy csomópontot a gráfban, és a kezdeti éleket a csomópontok között bemeneti adatként kapja meg a SCOP95 adatbázisból. Ezeket az értékeket a BLAST algoritmussal hozták létre és egy ritka mátrixot alkotnak. A megvalósított algoritmus egy bizonyos módosított Markov klaszterezés, amelyben a két fő művelet, az infláció és a kiterjesztés egymás ellenében dolgozik és a kialakuló egyensúlyi állapotban megmaradó összefüggő részgráfok határozzák meg a csoportosítás végeredményét. Az algoritmus fő paramétere az inflációs ráta, ezzel lehet szabályozni azt, hogy az algoritmus mennyire szakítsa szét a csoportokat egymástól.

A grafikus felhasználói felület lehetőséget ad a 11944 elemű SCOP95 adathalmazon vagy annak részein végrehajtani az osztályozást különféle beállításokkal, illetve meg tudja mutatni a kialakuló fehérje csoportokat. Továbbá bármely csoportból ki tudunk választani két szekvenciát és megnézhetjük azok egymáshoz illesztésének eredményét, melyet a Needleman-Wunsch algoritmussal valósít meg.

Az algoritmus képes lenne akár egymillió protein szekvenciát is feldolgozni néhány óra alatt, de ehhez nem állt rendelkezésemre megfelelő valós adathalmaz.

Kulcsszavak: Markov klaszterezés, gráf, ritka mátrix, hatékony algoritmus, bioinformatika, fehérje szekvencia

Extras

O proporție semnificativă a fenomenelor biologice poate fi urmărită înapoi la genetica, care se bazează pe secvențele de gene. Decodarea acestor secvențe de gene reprezintă o provocare majoră în domeniul științelor biologice. Conform unei estimări îndrăznețe, această sarcină ar putea angaja comunitatea științifică pentru următorii cinci sute de ani.

Scopul tezei mele este să implementez un algoritm simplu de clasificare a secvențelor de gene și să integrez acest algoritm într-un software, echipat cu o interfață grafică. Acest lucru poate reprezenta un pas mic spre soluția decodării secvențelor de gene.

Acest algoritm implementat efectuează clasificarea pe baza datelor de similaritate din perechi de secvențe de gene dintr-un set de date de secvențe mai mare. Această metodologie ne permite să procesăm simultan o cantitate mare de secvențe de proteine și să grupăm aceste secvențe pe baza similarității lor. Acest lucru este posibil prin lucrul cu matrice rare, care elimină operațiile inutile, cum ar fi adăugarea și înmulțirea cu zero. În același timp, această metodă are o eficiență semnificativă a memoriei, deoarece stocăm doar valorile non-zero.

Această metodă tratează setul de date de secvențe de intrare ca un graf. Fiecare proteină este atribuită unui nod în graf, iar marginile inițiale între noduri sunt primite ca date de intrare din baza de date SCOP95. Aceste valori au fost create folosind algoritmul BLAST și formează o matrice rară. Acest algoritm este o formă modificată de clusterizare Markov, în care cele două operații principale, inflația și extinderea, lucrează împotriva celuilalt. Subgrafurile conectate rămase în starea de echilibru, definind rezultatul final al clusterizării. Principalul parametru al algoritmului este rata de inflație, care determină cât de mult algoritmul separă grupurile unul de celălalt.

Interfața grafică oferă oportunitatea de a efectua clasificarea pe setul de date SCOP95 cu 11.944 de elemente sau pe subpărțile sale, folosind diverse setări și poate afișa aceste grupuri de proteine rezultate. În plus, din orice grup, putem selecta două secvențe și putem observa rezultatele alinierii lor, care este implementată folosind algoritmul Needleman-Wunsch.

Algoritmul ar putea procesa până la un milion de secvențe de proteine în câteva ore, dar din păcate nu a fost disponibil un set de date reale adecvat pentru acest scop

Cuvinte cheie: Clusterizare Markov, graf, matrice rară, algoritm eficient, bioinformatică, secvență de proteine

Abstract

A significant proportion of biological phenomena can be traced back to genetics, which is based on gene sequences. Decoding these gene sequences represents a substantial challenge in the field of biological sciences. According to a bold estimate, this task could potentially engage the scientific community for the next five hundred years.

The aim of my thesis is to implement a simple gene sequence classification algorithm and integrate this algorithm into a software, equipped with a graphical user interface. This may represent a small step toward the solution of decoding gene sequences.

This implemented algorithm performs classification based on similarity data from gene sequence pairs within a larger sequence dataset. This methodology allows us to simultaneously process large amount of protein sequences and grouping these sequences based on their similarity. This is made possible by working with sparse matrices which eliminates unnecessary operations, such as addition and multiplication with zero. At the same time, this method has significant memory efficiency, because we only store the none zero values.

This method treats the input sequence dataset as a graph. Each protein is assigned a node within the graph, and the initial edges between the nodes are received as input data from the SCOP95 database. These values were created using the BLAST algorithm and they're form a sparse matrix. This algorithm is a modified form of Markov clustering, in which the two main operations, inflation and expansion, work against each other. Connected subgraphs remaining in the equilibrium state, defining the final result of the clustering. The main parameter of the algorithm is the inflation rate, which determines how much the algorithm separates groups from each other.

The graphical user interface provides the opportunity to perform the classification on the 11,944-element SCOP95 dataset or on its subparts, using various settings and can display these resulting protein groups. Furthermore, from any group, we can select two sequences and observe the results of their alignment, which is implemented using the Needleman-Wunsch algorithm.

The algorithm could process up to one million protein sequences in a few hours, but unfortunately an appropriate real dataset was not available for this purpose.

Keywords: Markov clustering, graph, sparse matrix, efficient algorithm, bioinformatics, protein sequence

Keywords: Markov clustering, graph, sparse matrix, efficient algorithm, bioinformatics, protein sequence

1 Bevezetés

Az életünket szabályozó biokémiai folyamatok összefonódása a DNS és a fehérjék kódolt információján alapul. A DNS a sejtek szerkezeti és funkcionális jellemzőit határozza meg, míg a fehérjék számos élettani funkcióban vesznek részt. A fehérjéket alkotó aminosavak sorrendje, vagyis a fehérje szekvenciája döntő szerepet játszik a fehérje háromdimenziós alakjának és működésének kialakításában.

A fehérje szekvenciák tanulmányozása rendkívül fontos a biológiai funkciók megértésében, és arra is rávilágít, hogyan járulnak hozzá a genetikai mutációk a betegségek kialakulásához. Az ismert fehérje szerkezetekhez hasonló fehérje szekvenciák felfedezése azonban jelentős kihívást jelent. Az információtechnológia területén azonban ez a probléma már könnyebben kezelhető, különösen a Markov klaszterezés algoritmusával.

Ezen algoritmus alkalmazása segíthet a fehérjék közötti rejtett hasonlóságok azonosításában, amelyek alapján megérthetjük a fehérjék szerkezeti és funkcionális összefüggéseit. Az algoritmus optimalizálásának és hatékonyságának növelése érdekében ritka mátrix adatstruktúrát is használtunk.

A szerkezeti összehasonlítások nagyon összetettek lehetnek, ezért valamilyen tapasztalati összefüggésen alapuló közelítést kell alkalmaznunk az összehasonlítások során. Ezt a homológiai és hasonlósági szinteken tehetjük meg. A "homológ" kifejezés azt jelenti, hogy két fehérje közös ősből származik és fejlődéstani kapcsolatban állnak. Ezzel szemben a "hasonlóság" azt jelenti, hogy két fehérje szerkezete vagy szekvenciája valamilyen mértékben egyezik.

Összefoglalva, célunk az, hogy optimalizáljuk az algoritmust, megőrizzük az adatintegritást, kiemeljük az adatok közötti tulajdonságokat, és ezzel segítsünk a biológiai funkciók és kapcsolatok jobb megértésében.

1.1 Informatika felhasználás a biológiában

Az informatika, mint tudományág, a XXI. században áthatja az életünk szinte minden területét, köztük a biológiát is. A bioinformatika, mely a biológia és az informatika metszéspontját képezi, nagymértékben hozzájárult a biológiai kutatások fejlődéséhez és a genomi adatok felfedezéséhez.

Az informatika alkalmazása a biológiában a következő területekre terjed ki: a genomszekvenálás és a genomi adatbázisok kezelése, a proteomika és a fehérjeszekvencia-analízis, a gének és fehérjék szimulációja és modellezése, valamint a számítógépes algoritmusok és adatbányászati technikák alkalmazása a biológiai adatok elemzésében.

Az egyik legjelentősebb terület, ahol az informatika alapvető szerepet játszik, a fehérjeszekvencia-analízis. A fehérjék a sejtek szerkezeti és funkcionális komponensei, melyek számos élettani funkcióban vesznek részt. A fehérjeszekvenciák, vagyis az aminosavak sorrendje, meghatározza a fehérje háromdimenziós szerkezetét és működését.

A fehérjeszekvenciák elemzéséhez számos informatikai eszköz áll rendelkezésre. Az egyik ilyen eszköz a Markov klaszterezés algoritmus (MCL), melyet fehérjeszekvenciák hasonlóságának felmérésére és a fehérjék klaszterezésére használnak. Ez az algoritmus lehetővé teszi a nagy mennyiségű fehérjeszekvencia adat gyors és hatékony feldolgozását, és segít azonosítani a fehérjék közötti összefüggéseket.

A fehérjeszekvenciák analízise mellett az informatikát széles körben használják a gének szekvenciáinak elemzésében is. A genomszekvenálás során nagy mennyiségű genomi adat keletkezik, melynek kezeléséhez szükség van az informatikai eszközök és módszerek alkalmazására. A genomi adatbázisok lehetővé teszik a genomi adatok tárolását, rendezését és elemzését, és hozzájárulnak a gének funkciójának és szerepének jobb megértéséhez.

Az informatikának köszönhetően a biológiai kutatások nagyobb mértékben tudnak támaszkodni a számítógépes szimulációkra és modellezésre. A biológiai folyamatok szimulációja lehetővé teszi a kísérletek előzetes tervezését, az eredmények előrejelzését, és segíti a kutatókat a biológiai rendszerek jobb megértésében.

Amino Acid	Three Letter Code	One Letter Code
Alanine	Ala	A
Arginine	Arg	R
Aspartic Acid	Asp	D
Asparagine	Asn	N
Cysteine	Cys	C
Glutamic Acid	Glu	E
Glutamine	Gln	Q
Glycine	Gly	G
Histidine	His	H
Isoleucine	Ile	I
Leucine	Leu	L
Lysine	Lys	K
Methionine	Met	M
Phenylalanine	Phe	F
Proline	Pro	P
Serine	Ser	S
Threonine	Thr	T
Tryptophan	Trp	W
Tyrosine	Tyr	Y
Valine	Val	V

Ábra 1 – Aminósavak neve és jelölései

1.2 Fehérje és DNS illetve a fehérjeszekvenciák elemzése



Ábra 2 – Szekvencia ábrázolása

A DNS és a fehérjék a biológiai rendszerek alapvető építőkövei. A DNS, a genetikai információ hordozója, felelős a sejtek szerkezeti és funkcionális jellemzőinek meghatározásáért. A fehérjék, melyeket a DNS kódol, kulcsszerepet játszanak a sejt folyamatokban, beleértve a szabályozást, a jelátvitelt, az immunreakciókat és az anyagcserét.

A fehérjék nagy, komplex molekulák, amelyeket aminosavakból épülnek fel. Egy adott fehérje aminosav-sorrendje - azaz a fehérje szekvenciája - döntő jelentőségű a fehérje háromdimenziós struktúrájának és funkciójának kialakításában. A fehérjeszekvenciák ismerete elengedhetetlen a fehérje funkciójának megértéséhez, és segít a betegségekkel összefüggő genetikai mutációk azonosításában.

Az életben létező több tízezer különböző fehérje mind a 20 különböző aminosav kombinációjából áll. Egy adott fehérje aminosav-sorrendje, vagy szekvenciája, döntő jelentőségű a fehérje háromdimenziós struktúrájának és funkciójának kialakításában.

A DNS (dezoxiribonukleinsav) a genetikai információt tárolja, amely meghatározza minden élőlény fizikai jellemzőit. A DNS kettős spirál struktúrája két polinukleotid láncból áll, amelyek összekapcsolódnak az adenin (A) - timin (T) és guanin (G) - citozin (C) bázispárokkal.

A DNS-szekvencia ezen bázispárokból áll, és a fehérjék kódolásához szükséges információt tartalmazza.

A fehérjeszekvenciák elemzése nagy jelentőséggel bír a molekuláris biológiában és a bioinformatikában. Az elemzés célja, hogy felfedje a fehérjeszekvenciákban rejlő információt, amely a fehérje szerkezetét, funkcióját és evolúcióját is magában foglalja.

A fehérjeszekvenciák elemzése során használt módszerek között szerepelnek a páronkénti és többszörös szekvencia összehasonlítások, a motívumkeresés, a szekvencia-profil összehasonlítások és a szekvenciastruktúra előrejelzések. Ezek a módszerek lehetővé teszik a kutatók számára, hogy meghatározzák a fehérjék közötti evolúciós kapcsolatokat, azonosítsák a funkcionálisan fontos régiókat és előrejelzik a fehérje szerkezetét és funkcióját.

A fehérjeszekvenciák elemzésének fontossága aligha túlbecsülhető. A genetikai betegségek kutatásában a fehérjeszekvenciák elemzése lehetővé teszi a kutatók számára, hogy azonosítsák a kóros mutációkat és megértsék, hogyan befolyásolják a fehérje funkcióját. A gyógyszerfejlesztésben a fehérjeszekvenciák elemzése segít a célpontok azonosításában és a gyógyszermolekulák tervezésében. Az evolúciós biológiában a fehérjeszekvenciák elemzése támogatja a fajok közötti kapcsolatok megértését.

2 Elméleti háttér

2.1 Klaszterezés

A klaszterezés az adatbányászat, gépi tanulás és a statisztika területén használt technika, amelynek segítségével az adatokat hasonlóságok alapján különböző csoportokba, úgynevezett klaszterekbe szervezzük. A klaszterezés célja az, hogy az adatokat olyan csoportokba szervezzük, amelyekben a klaszteren belüli elemek hasonlóak, míg a klasztereken kívüli elemek között nagyobb különbségek találhatók.

A klaszterezés számos területen alkalmazható, például ügyfélszegmentációban a marketingben, genomszekvencia adatok elemzésében a biológiában, vagy szociális hálózatok analízisében.

A klaszterezés módszereit két fő csoportra oszthatjuk: hierarchikus és nem hierarchikus módszerek. A hierarchikus módszerek egy fa struktúrájú hierarchiát hoznak létre az adatokból, míg a nem hierarchikus módszerek (mint például a k-means klaszterezés) egyszerűen klaszterekbe szervezik az adatokat anélkül, hogy ezek közötti hierarchiát állítanának fel.

A klaszterezési módszerek különböző klaszterezési feladatokhoz különbözőképpen közelíthetnek, így létrehozva különböző típusú klasztereket.

Néhány fontosabb klaszterezés:

3. **Kizáró klaszterezés:** A kizáró klaszterezés (exclusive clustering) során minden adatpont pontosan egy klaszterbe tartozik, két különböző klaszterben nincsenek közös elemek.
4. **Átfedő klaszterezés:** Az átfedő klaszterezés (overlapping clustering) során egy adatpont egyszerre több klaszterbe is tartozhat. Ez a módszer hasznos lehet olyan helyzetekben, amikor az adatok többféle hasonlósági kritérium szerint is csoportosíthatók.
5. **Teljes klaszterezés:** A teljes klaszterezés (complete clustering) során minden adatpont tartozik valamelyik klaszterbe, nincsenek klaszterezetlen adatpontok.
6. **Részletes klaszterezés:** A részletes klaszterezés (partial clustering) során csak bizonyos adatpontokat klaszterezünk, míg a többit nem. Ez akkor lehet hasznos, ha csak bizonyos adatpontok csoportosítása a cél, például egy specifikus populáció vizsgálata.
7. **Fuzzy klaszterezés:** A fuzzy klaszterezés (fuzzy clustering) esetében egy adatpont nem csak egyértelműen egy klaszterbe tartozik, hanem több klaszterhez is tartozhat bizonyos mértékben. Az adatpontok klaszterekhez tartozását valószínűségi értékekkel jellemezzük.
8. **Nem fuzzy klaszterezés:** A nem fuzzy (non-fuzzy) vagy "éles" klaszterezés esetében minden adatpont egyértelműen csak egy klaszterbe tartozik.

2.2 Markov-lánc

A Markov-láncok nevét a híres orosz matematikusról, Andrej Markovról kapták, aki az elsők között foglalkozott ezzel a területtel.

A Markov-lánc egy véletlen folyamat, amelynek állapotai közötti átmenetek bizonyos tulajdonságokkal rendelkeznek. Az egyik legfontosabb ilyen tulajdonság a Markov-tulajdonság, ami azt jelenti, hogy a folyamat jövőbeli állapotai csak a jelenlegi állapottól függenek, és függetlenek a korábbi állapotoktól.

A Markov-láncokat általában állapotátmeneti mátrix formájában ábrázolják, amelyben a mátrix minden eleme az egyik állapotból a másikba történő átmenet valószínűségét jelenti.

A Markov-láncoknak számos fontos tulajdonsága és fogalma van, mint például az ergodicitás, ami azt jelenti, hogy minden állapot elérhető minden más állapotból, vagy a stacionárius eloszlás, amely a Markov-lánc hosszú távú viselkedését írja le.

A Markov-láncok számos alkalmazási területe van. Például használják a szövegek generálásához gépi tanulásban, ahol a modell megtanulja a szavak közötti átmenetek valószínűségét, és ezt használja fel új szövegek generálásához. A Google PageRank algoritmusában is használnak Markov-láncokat a weboldalak rangsorolására.

A Markov-láncok modelljének két alappillére a dinamika és a populációs eloszlás. Dinamika, mert a modell valamilyen transzformációt vagy változást vizsgál, és az analízis során olyan szabályokat próbál megtalálni, amelyekkel leírható a változás menete. Populációs eloszlás, mert a vizsgálatok során tulajdonképpen ennek változásait figyeljük meg, különböző időpontokban.

A modell mindig több időpillanaton keresztül vizsgálja a változásokat, majd az így kapott adatokat összevetve következtetéseket von le a jövőbeni eseményekre. A modell megállapítja, hogy a vizsgált adatok hogyan oszlanak el egy adott időpontban, figyelembe véve a vizsgálat szempontjait. Az eredményeket általában különböző csoportokba, vagy "állapotokba" soroljuk, melyek megnevezése jellemzően az adott jelenségre utal.

A valószínűségi változót pedig azokra a matematikailag modellezett jelenségekre alkalmazzuk, melyek értékei a véletlen függvényében változnak - ilyen lehet például a dobókockával dobott szám, vagy a pénzérme feldobása során kapott eredmény.

2.3 Markov-klaszter

A Markov klaszterezés (MCL algoritmus) széles körben használt módszer a hálózati struktúrákban, például fehérjékben lévő csoportok azonosítására. Ez az algoritmus feltételez, hogy a hasonlóság nemcsak a szomszédos csomópontok között, hanem az összes csomópont között jelen van. Ezzel a feltételezéssel az MCL algoritmus előnye a párhuzamosság, a skálázhatóság és az alacsony memóriakövetelmény. A Markov klaszterezés algoritmusának alkalmazása a fehérjeszekvenciákban lévő hasonlóságok keresésében számos előnnyel jár. Az ilyen adatok nagy halmazának feldolgozása nagyon időigényes lehet, ezért hatékony módszerekre van szükség. Az MCL algoritmusnak köszönhetően ez a folyamat egyszerűbbé válik, mivel az algoritmus képes nagy adatmennyiséget is feldolgozni.

Az MCL algoritmus a fehérjeszekvenciákban lévő hasonlóságokat úgy kutatja, hogy páronként hasonlítja össze az egyes szekvenciákat, és klaszterekbe rendezve jeleníti meg azokat. Ezt a folyamatot gráfok használatával szemlélteti, ahol a fehérjék a csomópontokat, a közöttük lévő hasonlóságok pedig az éleket reprezentálják.

Ez a módszer lehetővé teszi a fehérjék olyan csoportjainak azonosítását, amelyek azonos vagy hasonló biológiai funkciókat töltenek be. A proteincsaládok, amelyek szekvenciális hasonlóságokat mutatnak, ugyanis valószínűleg közös ősektől származnak, és azonos biológiai szerepük van.

Ez a folyamat a biológiai kutatásban rendkívül hasznos lehet, például a genetikai variációk vagy a fehérjeszerkezetek tanulmányozásában. A klaszterezés segíthet a fehérjék közötti összefüggések feltárásában, és így hozzájárulhat a fehérjék szerepének és funkciójának jobb megértéséhez.

Az MCL algoritmus egyik legnagyobb előnye, hogy képes kezelni a nagy méretű adathalmazokat is. A ritka mátrix adatszerkezet alkalmazásával a rendelkezésre álló erőforrásokat hatékonyan használja fel, és minimalizálja a futási időt. Az algoritmus eredményei ciklusidőben és a kialakult klaszterek tisztaságának elemzésében jelennek meg. A klaszterek tisztaságának elemzése azt jelenti, hogy meg

2.4 SCOP és SCOP95 adatbázis

A SCOP, vagyis a Structural Classification of Proteins (fehérjék strukturális osztályozása), egy nagyon népszerű adatbázis, amelyet széles körben használnak a biológiai kutatásokban és az alkalmazott bioinformatikában. A SCOP célja a fehérjék átfogó strukturális osztályozása azáltal, hogy különböző szintű hierarchiákat alkot a fehérjék szerkezetének, evolúciós rokonságának és a funkciójuknak figyelembe vételével.

A SCOP adatbázis fehérjeszerkezeteket tartalmaz, amelyeket részletesen kategorizáltak és osztályoztak. Az osztályozás több szintű, beleértve a class, fold, superfamily, family, protein és species szinteket. Mindegyik szint különböző strukturális és funkcionális információkat képvisel.


A SCOP95 adatbázist használom ami a SCOP adatbázis egy speciális verziója, amelyben a fehérjeszekvenciák 95%-os vagy alacsonyabb szekvenciaszintű azonossággal rendelkeznek. Ez azt jelenti, hogy a SCOP95 adatbázisban található minden fehérje szekvenciája páronkénti alapon maximum 95%-os hasonlóságot mutat a többi fehérjével. Ez a kritérium lehetővé teszi a redundancia csökkentését, azaz csökkenti azoknak a fehérjéknek a számát, amelyek nagyon hasonló vagy azonos szekvenciával rendelkeznek. Ezáltal a SCOP95 segíti a fehérjék pontosabb strukturális osztályozását és analizését.

2.5 Ritka mátrixok és típusaik

A ritka mátrixok olyan mátrixok, melyekben a nulla elemek dominálnak; vagyis a legtöbb bejegyzés nulla. Ezek a mátrixok gyakran fordulnak elő számos tudományágban és mérnöki alkalmazásban, például a hálózati elemzésekben, a számítógépes grafikában, a genetikai szekvenciális analízisben, a rendszertechnikában és a biológiai számításokban. Azokban az esetekben, ahol a ritka mátrixokat kell kezelni, speciális módszerekre és algoritmusokra van szükség az adatok hatékony tárolása és a számítások gyors végrehajtása érdekében.

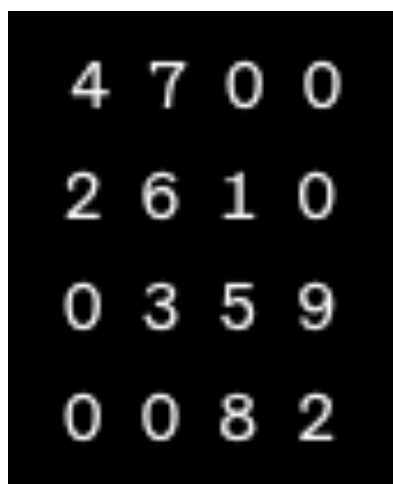
Ritka mátrixok néhány típusa:

1. **Diagonális mátrixok:** Ezek a mátrixok a diagonális vonalukon nem nulla elemeket tartalmaznak, míg minden más elemük nulla. Példa a diagonális mátrixra:


$$\begin{pmatrix} 5 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

Ábra 3 – Diagonális mátrix

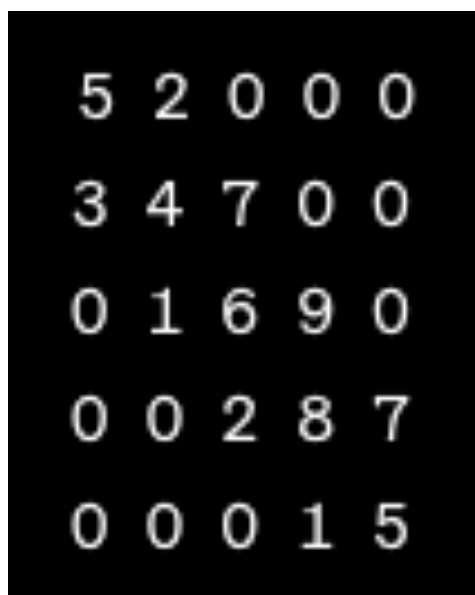
1. **Tridiagonális mátrixok:** Ezek a mátrixok a fő diagonáljukon, valamint a fő diagonál fölötti és alatti sorokban tartalmaznak nem nulla elemeket. Minden más elem nulla. A tridiagonális mátrixok gyakran előfordulnak a numerikus differenciálegyenletek megoldása során. Példa a tridiagonális mátrixra:



4	7	0	0
2	6	1	0
0	3	5	9
0	0	8	2

Ábra 4 – Tridiagonális mátrix

3. **Sávós mátrixok:** Ezek a mátrixok is tartalmazznak olyan sorokat vagy oszlopokat, amelyekben nem nulla elemek vannak, míg minden más elem nulla. A sáv szélessége meghatározza a nem nulla elemek számát az egyes sorokban vagy oszlopokban. Példa a sávós mátrixra:



5	2	0	0	0
3	4	7	0	0
0	1	6	9	0
0	0	2	8	7
0	0	0	1	5

Ábra 5 – Sávós mátrixok

4. **Szimmetrikus mátrixok:** Ezek a mátrixok olyan speciális ritka mátrixok, amelyek fő diagonálja mentén tükröződnek. Ez azt jelenti, hogy a mátrix (i, j) pozíciójú eleme megegyezik a (j, i) pozíciójú elemmel.

2.6 Felhasznált technológiák

C++

C++ egy általános célú programozási nyelv, amelyet Bjarne Stroustrup hozott létre az AT&T Bell Laboratories-ban 1979-ben. A C++ a C programozási nyelven alapul, de több szintű absztrakciót kínál, ami sokféle fejlesztési feladatra alkalmassá teszi.

Néhány jelentős jellemzője és előnye a következők:

1. **Objektumorientált programozás:** A C++ támogatja az objektumorientált programozás paradigmáját, beleértve az osztályokat és objektumokat, az öröklődést, a polimorfizmust és az adatrejtést. Ezek a funkciók segítik a kód szervezését, könnyebbé teszik a bonyolult szoftverrendszerek kezelését, és elősegítik a kód újrafelhasználhatóságát.
2. **Memóriakezelés:** A C++ közvetlenül támogatja a dinamikus és statikus memóriakezelést, amely a fejlesztőnek nagyfokú kontrollt ad a program memóriahasználatának finomhangolásában. Ezt kihasználva a C++ képes nagyon hatékony és gyors alkalmazások létrehozására.
3. **Alacsony szintű hozzáférés:** A C++ lehetővé teszi a fejlesztők számára, hogy közvetlenül hozzáférjenek az alacsony szintű rendszererőforrásokhoz, mint például a memória és a hardvereszközök. Ez lehetővé teszi a nagy teljesítményű és speciális alkalmazások fejlesztését, mint például az operációs rendszerek, a rendszerillesztő programok, és a valós idejű rendszerek.
4. **Standard Template Library (STL):** A C++ rendelkezik egy hatalmas standard könyvtárral, az STL-lel, amely számos előre definiált osztályt és függvényt tartalmaz. Ezek magukban foglalják a konténereket (mint például a vektorok és listák), az algoritmusokat, az iterátorokat és másokat, amelyek megkönnyítik a fejlesztők számára a bonyolult adatszerkezetek és algoritmusok használatát.

A C++ technológia meglehetősen bonyolult, mivel sok szabályt és funkciót tartalmaz, és a memóriakezelés is sok figyelmet igényel. Ennek ellenére a C++ a szoftverfejlesztés egyik alapvető technológiája, és számos területen alkalmazzák.

Felhasználói felület

A felhasználói felület (angolul User Interface, röviden UI) az a felület vagy rendszer, amelyen keresztül a felhasználók kölcsönhatásba lépnek egy szoftverrel, alkalmazással, eszközzel vagy rendszerrel.

A felhasználói felületek célja, hogy az ember és a gép közötti kommunikációt intuitív, könnyen érthető és hatékony módon biztosítsák. A felhasználói felületek lehetnek grafikusak (Graphical User Interface, GUI), szövegalapúak (Command-Line Interface, CLI), érintésalapúak (Touch User Interface) vagy akár hangvezérelt felületek (Voice User Interface, VUI).

A grafikus felhasználói felületeken, mint például a Windows Forms, a felhasználók a vezérlőelemeket (gombok, lenyíló menük, jelölőnégyzetek stb.) használva irányíthatják a programot.

Windows Forms

A Windows Forms technológiát választottam a projektben a grafikus felület (UI) megvalósításához. A döntésem ezen technológia mellett azért esett, mert a Windows Forms segítségével egyszerűen és hatékonyan hozhatók létre egyszerű felületek. Az sem elhanyagolható, hogy a Windows Forms drag-and-drop formatervezési eszközökkel rendelkezik, melyekkel a vezérlőelemek vizuálisan helyezhetők el a felületen. Ez egy intuitív tervezési környezetet biztosít, ami gyorsítja a fejlesztést és csökkenti a hibák esélyét.

A Windows Forms ezenfelül támogatja a Windows rendszer natív vezérlőelemeit, amelyek a felhasználók számára ismerős környezetet biztosítanak. Ennek köszönhetően az alkalmazás könnyen illeszkedik a Windows ökoszisztémába, és követi annak felhasználói felületi irányelveit.

A Windows Forms továbbá lehetővé teszi a fejlesztők számára, hogy eseményekhez kódokat rendeljenek, ami rugalmas, eseményvezérelt programozást tesz lehetővé. Ez a funkció különösen hasznos a bonyolultabb interakciók megvalósításához, mint például a felhasználói beavatkozásokra vagy a hálózati eseményekre adott válaszok.

Továbbá, a Windows Forms alatt elérhető számos előre definiált vezérlő, például gombok, szövegdobozok, listák és menük, amelyekkel gyorsan és egyszerűen készíthető el a felhasználói felület. A Windows Forms segítségével még komplexebb UI elemek, mint például az adattáblázatok és a fájlkezelő párbeszédpanelek is könnyen használhatóak.

Összességében a Windows Forms technológia kiváló választás volt a projekt számára, mivel lehetővé tette az egyszerű és hatékony felhasználói felület kialakítását, miközben a Windows platform széleskörű támogatását és a könnyen használható eszközkészletet biztosította.

3 Példa Markov klaszterezésre

Az egyszerűbb megértés kedvéért az alábbi kódrészlettel bemutatom egy háromszor hármas mairixon, hogy néz ki egy Markov klaszterezés.

Kódrészlet MatLab-kóddal

```
1  S = [1 0.8 0.5; 0.8 1 0.6; 0.5 0.6 1];
2  eps = 0.001;
3  r = 1;
4
5
6  Q = [];
7
8  for ciklus=1:30
9      %inflacio
10     S = S.^r;
11     % normalizalas
12     S = S./ (sum(S,2) * ones(1,3))
13     % szimmetrizalas
14     S = sqrt(S.*S');
15     % normalizalas
16     S = S./ (sum(S,2) * ones(1,3));
17     % szimmetrizalas
18     S = sqrt(S.*S');
19     % normalizalas
20     S = S./ (sum(S,2) * ones(1,3));
21     % expansio
22     S = S*S;
23     Q = [Q; S(1,2) S(1,3) S(2,3)];
24 end
```

Ábra 6 – Markov klaszterezés példa MatLab kóddal

Ez a MATLAB kód egy infláció, normalizáció, szimmetrizáció és expanzió folyamatot hajt végre egy 3x3-as mátrixon (S), összesen 30 iteráción keresztül. Az r értéke az inflációs paraméter.

A lépések a következők:

1. Infláció:

Ez a folyamat szükséges a mátrix elemeinek kiemelésére, hogy jobban hangsúlyozza a nagyobb értékek és kisebb értékek közötti különbségeket. Ezzel szélsőségesebb különbségeket hozunk létre a mátrixban, ami különösen hasznos lehet az adatok klaszterezésekor. Az r paraméter itt a hatványozó faktor, amit a különbözőségek élesítésére használunk.

2. Normalizáció:

A normalizáció célja, hogy a mátrix elemei értékei ne legyenek túlzottan nagyok vagy kicsik, hanem a [0,1] intervallumba essenek. Ez segít stabilizálni a kód futását és egyszerűsíti a további számításokat, mivel a normalizált értékekkel könnyebb dolgozni.

3. Szimmetrizáció:

A szimmetrizálás a mátrixot szimmetrikussá teszi, ami könnyebbé teszi a kezelését és számításokat, mivel a szimmetrikus mátrixokkal végzett műveletek gyakran egyszerűbbek. Továbbá, a szimmetrikus mátrixoknak speciális tulajdonságai vannak, például sajátértékeik mindig valósak, ami szintén előnyös lehet bizonyos matematikai problémákban.

4. Expanzió:

Az expanzió – amikor a mátrixot önmagával szorozva négyzetre emeljük – növeli a mátrix elemeinek értékét. Ez tovább mélyíti a mátrixban lévő mintázatokat és növeli a mátrixban lévő információkon alapuló döntések bizonyosságát.

Az iterációs folyamat során ezek a lépések együttesen működnek annak érdekében, hogy a mátrixot egy stabil állapotba juttassák, ahol az elemei nem változnak tovább. Ez az úgynevezett konvergencia állapot.

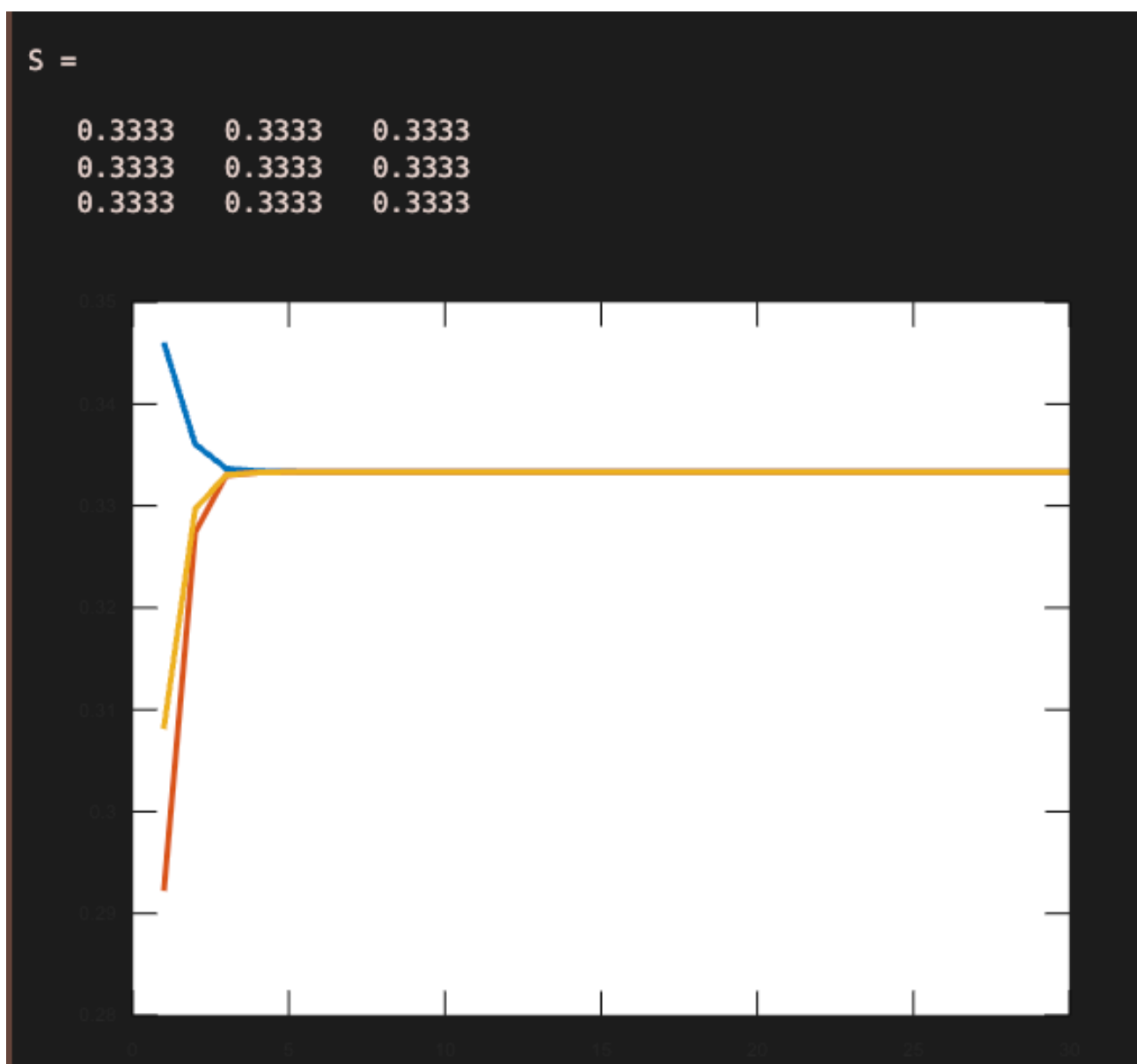
3.1 Inflációs ráta

Az infláció ('S' elemeinek r hatványára emelése) kihangsúlyozza a nagyobb értékeket és csökkenti a kisebbeket. Ezután a normalizáció során minden sor összege 1-re normalizálódik. A szimmetrizáció pedig azt eredményezi, hogy a mátrix diagonálisán kívüli elemei egymásnak megfelelő értékeket vesznek fel.

Az inflációs ráta r változtatása befolyásolja a folyamat konvergenciájának sebességét, ahogy korábban is említettem. Specifikusan, a magasabb r érték gyorsabb konvergenciát eredményez, mint az alacsonyabb r érték.

Ez azt jelenti, hogy a mátrix elemi értékei nagyobb r érték esetén gyorsabban közelítenek az 1-hez (nagyobb értékek) vagy az 0-hoz (kisebb értékek). Az eredményül kapott mátrix ezért gyorsabban válik "élesebbé" vagy binárisabbá, ami azt jelenti, hogy a domináns elemek értékei gyorsabban nőnek, míg a nem domináns elemek értékei gyorsabban csökkennek.

Ugyanakkor a nagyobb r érték esetén a konvergencia túl gyors lehet, ami azt eredményezheti, hogy kevesebb információt kapunk az elemek közötti relatív különbségekről.



Ábra 7 – $r=1$ infláció

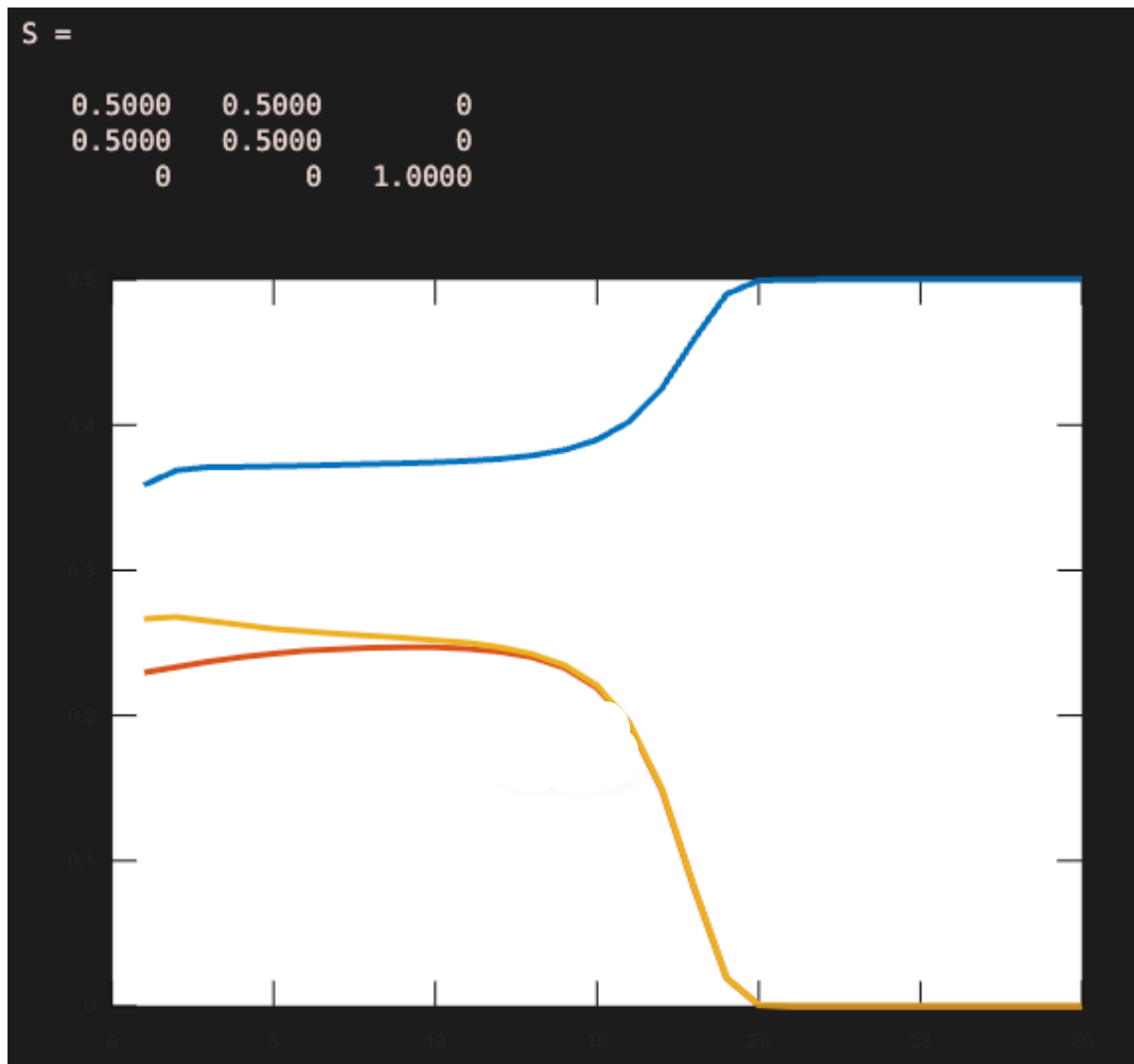
$r = 1$

Az $r = 1$ inflációs rátánál az a jelenség, hogy a mátrix összes eleme 0.33-ra konvergál, arra utal, hogy a mátrix elemei között nincs jelentős különbség a kiindulási értékekben, vagy hogy az iterációs folyamat során ezek a különbségek csökkennek.

Ennek oka az, hogy az infláció és a normalizáció lépései ezzel az r értékkel nem eredményeznek elég nagy különbséget a mátrix nagyobb és kisebb elemei között ahhoz, hogy a végső értékek jelentősen eltérjenek egymástól. Ennek eredményeként, a mátrix minden eleme a

normalizáció miatt az $1/3$ (azaz körülbelül 0.33) érték felé konvergál, mivel a sorok összege minden iteráció után 1 -re normalizálódik, és a 3×3 -as mátrix minden sorában három elem van.

Ha $r = 1$, akkor az infláció lépése nem változtatja meg a mátrixot, mert bármely szám 1 . hatványa önmaga. Ezért, ha $r = 1$, a kód többi része (normalizálás, szimmetrizálás, expansió) határozza meg a mátrix frissítését, és az S mátrix elemei viszonylag gyorsan konvergálhatnak az $1/3$ értékre (0.33).

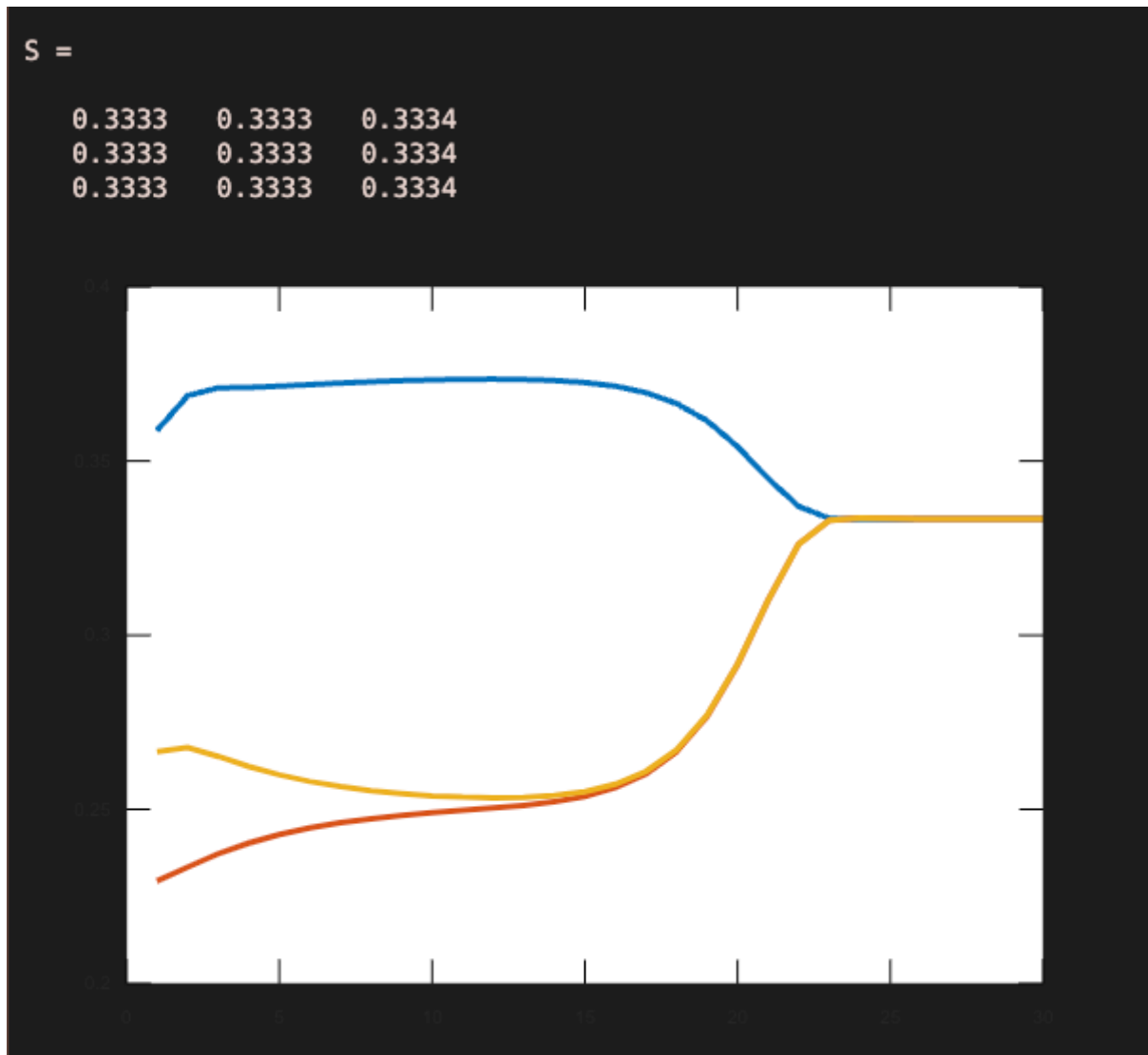


Ábra 8 – $r=2.0972$ infláció

$r = 2.0972$

Az $r = 2.0972$ esetében, bár a végső konvergált érték ugyanaz (0.33), a konvergencia sebessége lassabb, mert az infláció lépése most már jelentősen befolyásolja a mátrixot. Az r hatványozás kihangsúlyozza a mátrix nagyobb elemeit és csökkenti a kisebbeket, mielőtt a sorok normalizálásra

kerülnének. Ez a hatványozási folyamat további iterációkat igényel, hogy a mátrix elemei a 0.33 értékhez konvergáljanak, mert a nagyobb és kisebb értékek közötti különbség kezdetben növekszik az iterációk során, mielőtt végül csökkenne.



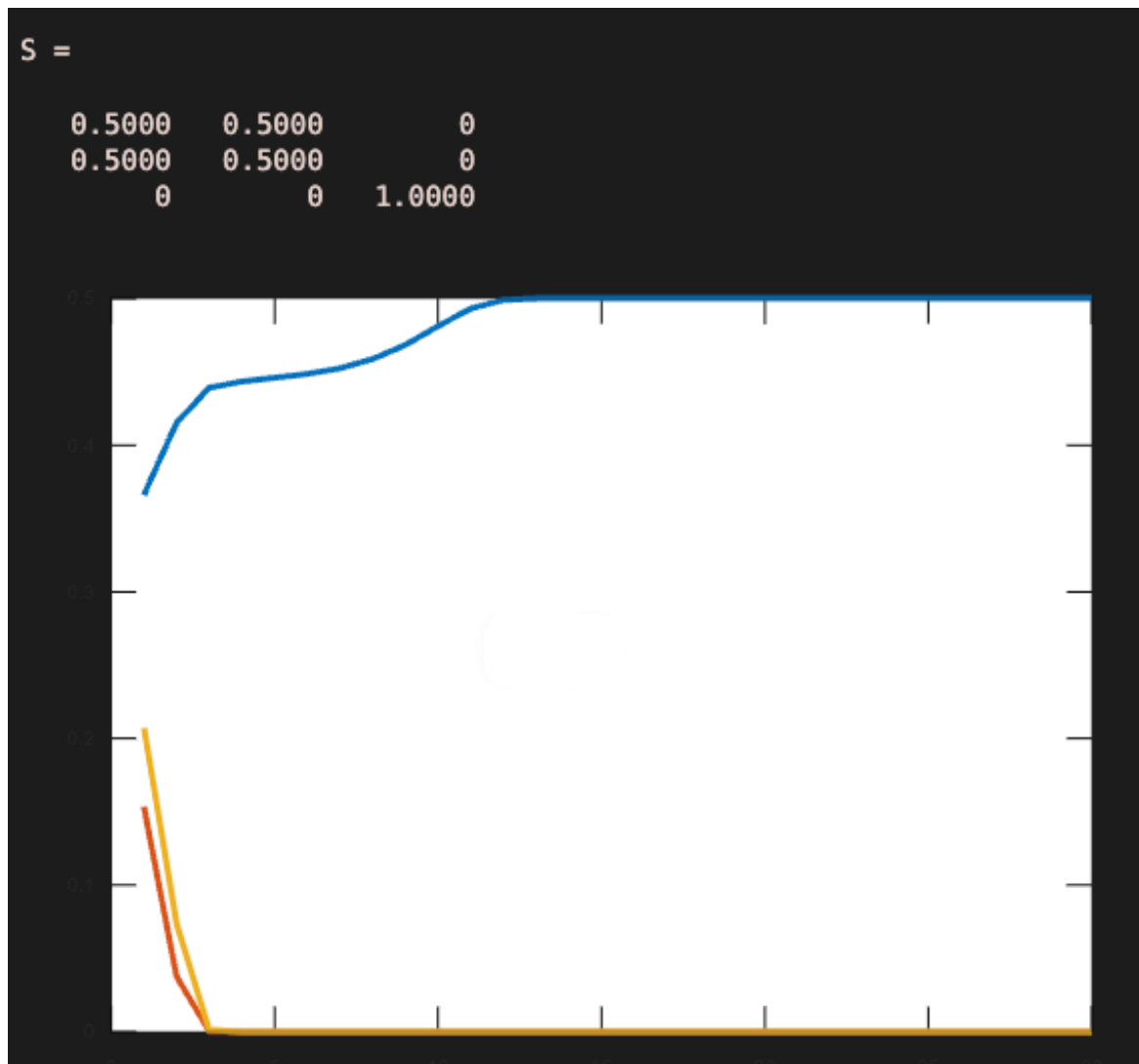
Ábra 9 – $r=2.0973$ infláció

$$r = 2.0973$$

Az $r = 2.0973$ esetben a mátrix első két sora úgy konvergál, hogy a nem-zéró elemei 0.5 értékűek lesznek, míg a harmadik utolsó eleme 1-re konvergál. Ez azt jelzi, hogy az inflációs ráta ezen a szinten képes volt kiemelni és konzerválni bizonyos eredeti különbségeket a mátrix elemei között.

A harmadik sor eredeti értékei (0.5, 0.6, 1) nagyobbak, mint az első két sorban lévő értékek. Az inflációs folyamat során ez a különbség növekszik, a harmadik sor értékei dominánsabbá válnak.

Ez azt jelenti, hogy a harmadik sor értékei - amelyek eredetileg nagyobbak voltak - továbbra is nagyobbak maradtak az iteráció során, és végül 1-re konvergáltak a normalizáció miatt. Az első két sor értékei, amelyek eredetileg kisebbek voltak, az inflációs lépés során még tovább csökkentek, és végül 0.5-re konvergáltak. A mátrix nullái azokban a helyeken jelennek meg, ahol az eredeti értékek már a kezdetben is kisebbek voltak, és az inflációs lépés során ezek az értékek elhanyagolhatóvá váltak.

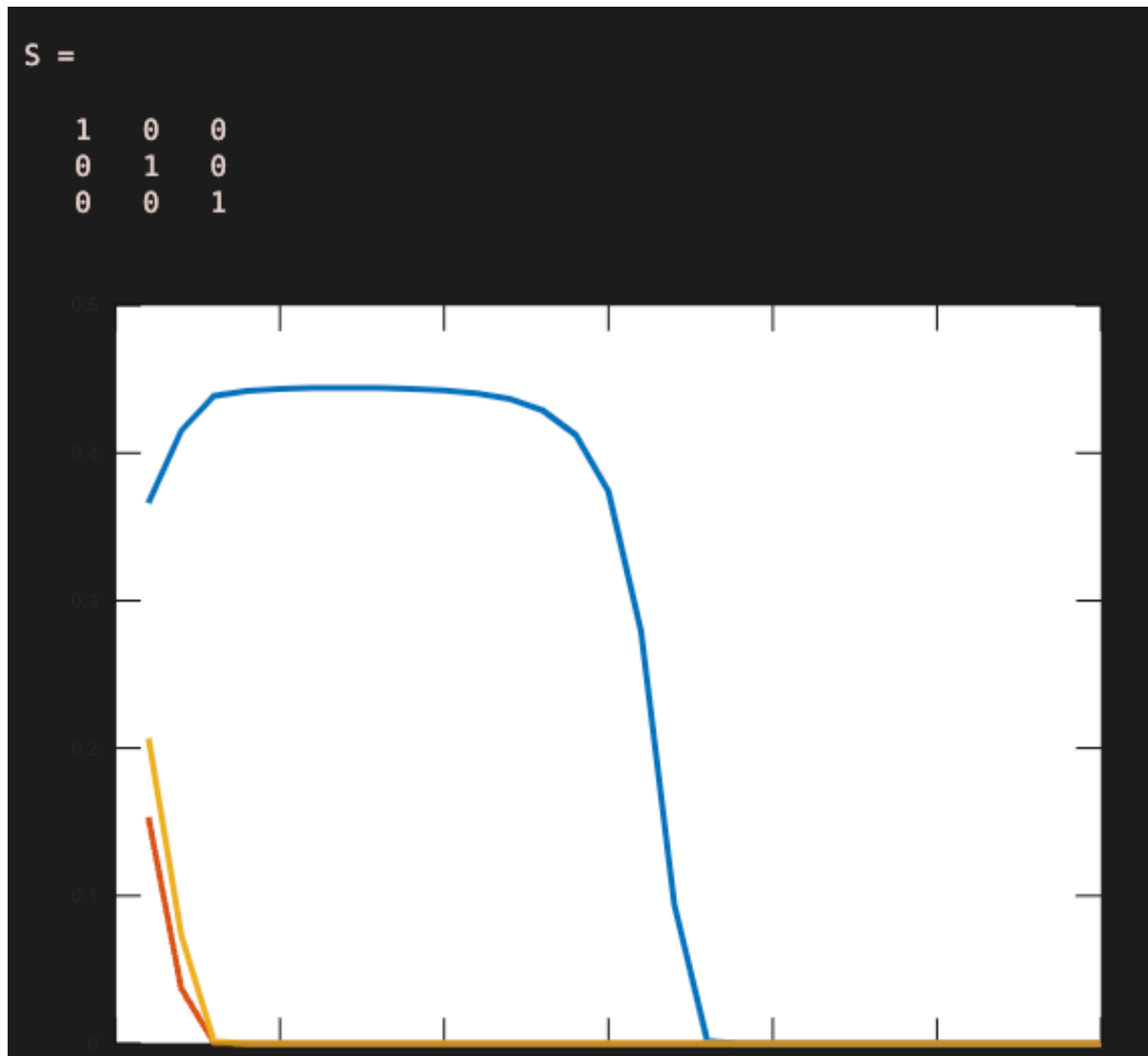


Ábra 10 – $r=3.11$ infláció

$r = 3.11$

Hasonlóan a második ábrához, a nagyobb inflációs ráta ($r=3.11$) nagyobb különbségeket eredményez a mátrix *elemei* között az iterációs folyamat során, ami lassítja a konvergenciát az adott értékekhez képest. A nagyobb értékek továbbra is nagyobbak maradnak a normalizáció után is, de a konvergencia lassabb, mivel az elemek közötti különbségek a hatványozás során

nagyobbak lesznek. Az első két sor 0.5-re, míg a harmadik sor 1-re konvergál, de a konvergencia lassabb, mint kisebb inflációs rátánál.



Ábra 11 – $r=3.113$ infláció

$r = 3.113$

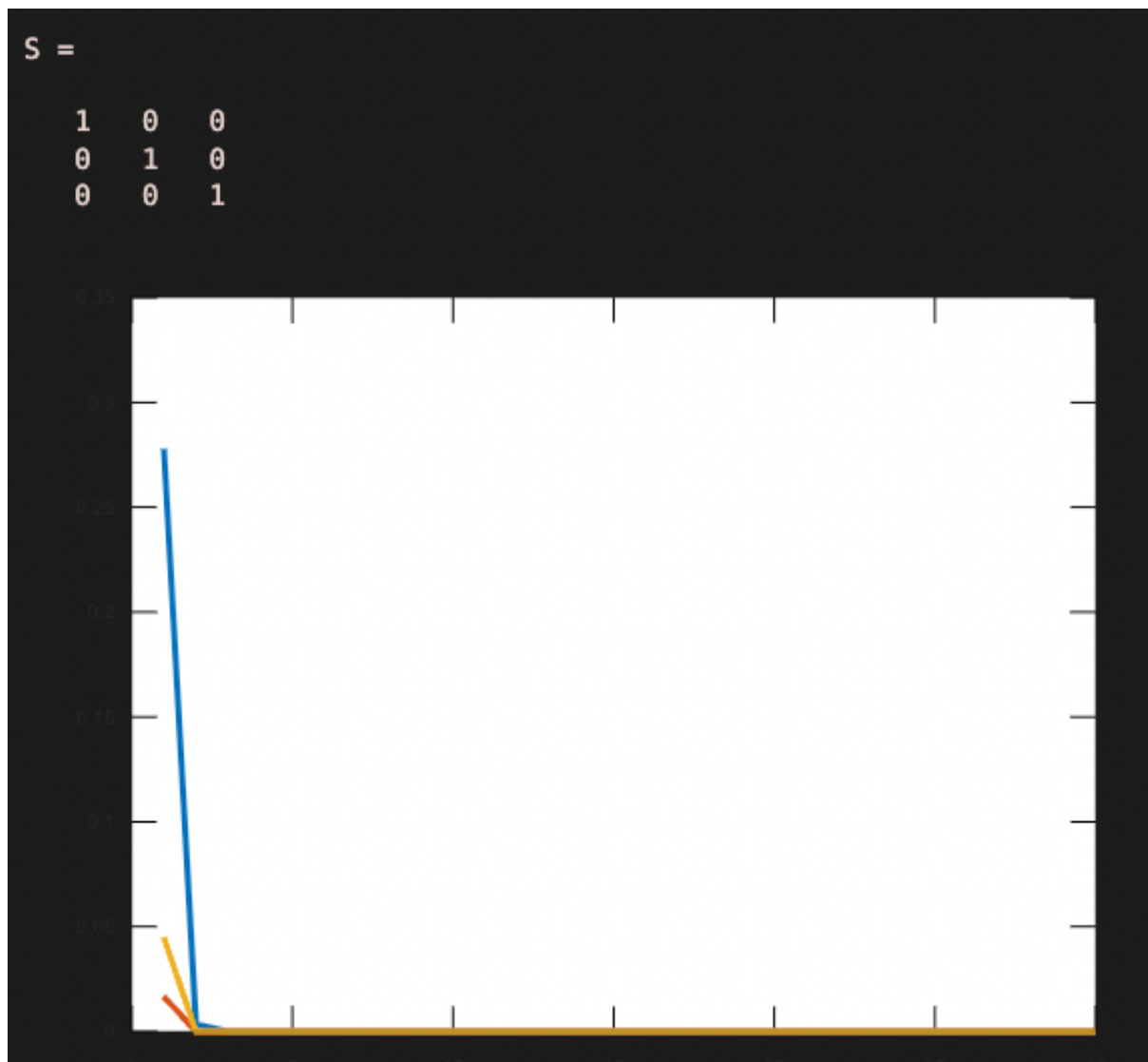
Az ' $r = 3.113$ ' inflációs ráta még erőteljesebben növeli a mátrix elemeinek különbségeit az inflációs lépés során. A mátrix eredeti elemei közül a nagyobb értékek nagyobb mértékben nőnek az infláció során, mint a kisebb értékek. Végül, a mátrix elemei közötti nagy különbségek és az inflációs ráta magas értéke miatt a mátrix az egységmátrixra konvergál.

Ha tovább növeljük az inflációs rátát, a már meglévő tendenciák tovább erősödnek. Az

inflációs ráta nagyobb értéke tovább növeli a mátrix elemei közötti különbségeket az infláció során.

Az inflációs folyamat során a nagyobb értékek tovább nőnek, míg a kisebb értékek tovább csökkennek. Ha a mátrix elemei közötti különbségek a kiindulópontnál már nagyok, a nagyobb inflációs ráta még tovább növeli ezeket a különbségeket.

Más szóval, ha tovább növeljük az inflációs rátát, a mátrix még gyorsabban konvergál az egységmátrixra. Ezzel a módszerrel, a mátrix normalizálása, szimmetrizálása és inflálása során a nagyobb értékek előtérbe kerülnek, míg a kisebb értékek háttérbe szorulnak. A magasabb inflációs ráta még tovább erősíti ezt a folyamatot, ahogy az alábbi $r = 7$ inflációs rátával ezt szemléltetem.



Ábra 12 – $r=7$ infláció

4. Program tervezése

4.1 Algoritmus

A kezdeti program implementációnk során előfordul, hogy két példányra van szükségünk a hasonlósági mátrixokból. Azért fordul elő ez a helyzet, mert az infláció, normalizáció és szimmetrizáció eljárásaink során ugyanazt a mátrixot használjuk, de az expansziós eljárás másik két mátrixot igényel, egyet a bemeneti, és egyet a kimeneti adatok számára. Ezt a kihívást úgy oldjuk meg, hogy alkalmazunk egy ritka mátrixot és egy két dimenziós átlagos mátrixot.

A ritka mátrixban csak a nullától eltérő értékek kerülnek tárolásra. Az ilyen elemeket tároló Mij mátrixban minden nullától eltérő érték párban kerül tárolásra azzal az értékkel, amely a mátrixban szimmetrikus helyzetben található. Ezt a struktúrát az egyszerűsített formájában egydimenziós tömbben tároljuk, amelyben minden sorhoz tartozik egy osztható memóriacím, ezáltal biztosítva a hatékony memória használatot. Továbbá, minden sor kezdőpontját is tároljuk, hogy pontosan tudjuk, honnan indulnak a sorok.

A kifejlesztett adatstruktúra célja, hogy segítséget nyújtson a mátrixon végzendő műveletek egyszerűsítésében. Ebből a négy fő műveletből az expanszió jelentheti a legnagyobb kihívást, mivel ennek során nő meg a mátrixban található nullás elemek száma. A normalizáció és az infláció folyamán a nullától eltérő értékek száma változatlan marad, míg a szimmetrizáció során az adott küszöbérték alatti értékeket figyelmen kívül hagyjuk, ezzel csökkentve azok számát. Az expanszió alkalmazása után a mátrix teljes tartalma átalakul.

4.2 Eljárások

Elsődleges feladatként a fehérje-adatokat, melyeket az adatbázisból nyertünk ki, beillesztjük egy négyzet alakú mátrixba. A következő fázisban meghatározzuk és beállítjuk az alapvető paramétereket, mint az inflációs arány, az epsilon érték és a ciklusszám.

Kulcsfontosságú, hogy ezeket az értékeket óvatosan válasszuk meg, hogy az algoritmus eredményessége ne kerüljön veszélybe.

Ezt követően a meghatározott algoritmusokkal megmunkáljuk a korábban létrehozott mátrixokat. A műveletek végrehajtásának sorrendje éppolyan fontos, mint a tény, hogy bizonyos műveleteket többször meg kell ismételni a mátrixon.

A normalizációs folyamatot például legalább egyszer meg kell hívni a cikluskezelés előtt, hogy a nullától különböző hasonlósági értékeket biztosítsuk.

Miután az algoritmus végigfutott, a végleges eredményeket megjelenítjük a felhasználó számára. Az eredmények között található a ciklusidők és a klaszterek tisztasági mértékei. Ezek alapján vonjuk majd le a végkövetkeztetéseket és tervezzük meg a jövőbeli lépéseket.

Inflációs ráta

Az inflációs ráta gondos megválasztása a fenti kódban kritikus, mivel ez befolyásolja a klaszterezés minőségét és a végleges csoportosítás élességét. Ez az érték meghatározza a mátrix normalizációs lépésében, hogy hogyan emeljük a mátrix elemeit a saját értékük hatványára, ami meghatározza, hogy az algoritmus mennyire élesíti a klaszterhatárokat.

Ez a paraméter tipikusan az intervallum 1,1 és 2,0 között mozog, és általában egyetlen fix értéket vesz fel a futás során. A választott érték jelentősen befolyásolhatja az algoritmus működését. Alacsonyabb értékek esetén az algoritmus hajlamos lehet "elmosódottabb", kevésbé éles klasztereket eredményezni, míg a magasabb értékek esetén az eredmények élesebbek, jól elkülönített klasztereket adnak.

1. Alacsony inflációs ráta esetén (pl. 1,1 közelében): Az algoritmus a normalizáció során kevésbé "élesíti" a klasztereket, ami azt jelenti, hogy a végeredményben található klaszterek határai kevésbé lesznek kijelölve. Ez azt eredményezheti, hogy az algoritmus kevésbé képes jól elkülöníteni a különböző klasztereket, így azok "elmosódottabbak" lehetnek.
2. Magas inflációs ráta esetén (pl. 2,0 közelében): Az algoritmus nagyon erőteljesen élesíti a klasztereket, ami azt eredményezi, hogy a klaszterek közötti határok nagyon jól kijelölhetők. Ezáltal az algoritmus képes lehet jobban elkülöníteni a különböző klasztereket. Viszont túlzottan magas inflációs ráta esetén az eredmények túlságosan szegmentálttá válhatnak, azaz sok kisebb klaszterbe sorolhatja az elemeket.

A megfelelő inflációs ráta kiválasztása érdekében érdemes lehet különböző értékeket tesztelni, hogy megtaláljuk a legjobb egyensúlyt az éles klaszterezés és a túlzottan szegmentált eredmények között. A választás során figyelembe kell venni a kiválasztott adatok jellegét és azt, hogy milyen eredményeket szeretnénk elérni a klaszterezéssel

$$M_{ij(old)} = M_{ij(new)}^r$$

1. M a normalizált valószínűségi mátrix.
2. i és j az M mátrix sorainak és oszlopainak indexei.
3. r az inflációs ráta.

Az infláció tehát minden mátrix elemet a rögzített inflációs rátának (r) megfelelő hatványra emel. Ez az élesítési folyamat segít abban, hogy a mátrixban lévő értékek jobban tükrözzék a különböző klaszterek közötti különbségeket.

Ezután egy újabb normalizációs lépés következik, amely során minden sor összegét 1-re normalizáljuk, hogy ismét valószínűségi mátrixot kapjunk.

Normalizáció

A normalizáció, amely a Markov algoritmusának kulcseleme, arra szolgál, hogy egységes mérőrendszert hozzon létre az összes adatunk számára. Ez annyit tesz, hogy minden adatpontot ugyanazon a skálán értékelünk, ami elengedhetetlen ahhoz, hogy a különböző adatok közötti távolságot pontosan számíthassuk ki. Ez különösen fontos a hasonlóság-alapú klaszterezésnél, mint például a Markov klónok, ahol az adatok közötti hasonlóság az alapja a klaszterezési döntéseknek.

A normálás folyamatában az adott mátrix minden egyes celláját elosztjuk az adott sor összes elemének összegével. Ez a folyamat során normalizálja az összes értéket a sorban, ami azt eredményezi, hogy minden sorban a cellák értékeinek összege egységnyi lesz.

$$\alpha_i = \sum_{j \in \text{row}_i} M_{ij(\text{régi})}$$

Az " i " és " j " indexeket a sorok és oszlopok indexeire használjuk, a " M " pedig a hasonlósági mátrixunkat jelenti. Ez a lépés a relatív súlyokat úgy állítja be, hogy minél nagyobb az adott cella értéke, annál fontosabb a kapcsolat a két fehérje között, így a későbbi klaszterezési lépésekben nagyobb súlyt kap.

A normalizáció tehát segít kiemelni a lényeges kapcsolatokat az adatok között. Ennek eredményeként a hasonlósági klaszterezés hatékonyabb és pontosabb lesz.

Szimmetrizáció

A szimmetrizáció fontos lépés a hasonlósági mátrix előállításában. A fehérjék közötti interakciók általában kétirányúak, tehát ha az A fehérje befolyásolja a B fehérjét, akkor fordítva is igaz ez. Ezért a hasonlósági mátrixnak szimmetrikusnak kell lennie, azaz $M[i][j]$ értékének meg kell egyeznie $M[j][i]$ értékével minden i és j pár esetében.

Ezt a szimmetrizációt azáltal érjük el, hogy minden cella értékét a megfelelő tükrözött cella értékével kombináljuk. A szimmetrizáció legegyszerűbb módja az, ha a két érték egyszerűen összeadódik. Az általunk használt szimmetrizáció a "geometriai középérték" módszerén alapul. Ez a módszer gyakran használatos, amikor két irányú hasonlósági értékeket szeretnénk szimmetrikus formába hozni. A szimmetrizáció során kiszámítjuk a két érték gyökösszorzatát, ami lényegében a geometriai középértékük. Az algoritmusban ez az érték helyettesíti az eredeti, irányfüggő értékeket, így téve a hasonlósági mátrixot szimmetrikussá.

Ez a folyamat biztosítja, hogy a hasonlósági mátrix szimmetrikus lesz, ami elengedhetetlen a további klaszterezési lépésekhez. A szimmetrikus hasonlósági mátrixok megkönnyítik a klaszterezést és javítják az eredmények minőségét.

$$M_{ij(új)} = \begin{cases} \sqrt{M_{ij(régi)} * T_{ij(régi)}} \\ 0 \end{cases}$$

$M(i,j)$ a mátrix i . sorának és j . oszlopának metszéspontjában található érték, és $M(j,i)$ az i . oszlopnak és j . sornak metszéspontjában található érték. Ezt az értéket $T(i,j)$ -vel jelöljük. A T a transzponált rövidítése

A szimmetrizált érték tehát az eredeti értékek geometriai középértéke, melyet az eredeti értékek négyzetszorzatának négyzetgyökével számolunk. Ez a módszer biztosítja a mátrix szimmetrikusságát, és eltávolítja a hasonlósági értékek irányfüggőségét.

Elimináló szimmetrizáció

Az elimináló szimmetrizáció egy speciális típusú szimmetrizációs módszer, amely a szimmetrikus transzformáció után szűri ki azokat az elemeket, amelyeknek az értékei egy bizonyos küszöbérték, jelen esetben az "epsilon" alá esnek. Ezt azért teszi, hogy javítsa a számítási hatékonyságot, és csökkentse a felesleges, alacsony értékű elemek tárolását és kezelését a folyamat későbbi szakaszaiban.

Az elimináló szimmetrizáció tehát nem csak a mátrix szimmetrizálásával foglalkozik, hanem a kezelhető adatmennyiség csökkentésével is. Ez különösen fontos nagy adatmennyiségek esetén, ahol az alacsony értékű elemek száma jelentősen növelheti a számítási költségeket anélkül, hogy lényeges hozzáadott értéket nyújtanának.

Az elimináló szimmetrizációs eljárás az alábbiak szerint működik:

1. A szimmetrizáció során kiszámoljuk az új értéket, amely a két hozzá tartozó érték négyzetgyökének szorzata.

2. Ha az új érték kisebb, mint az epsilon, akkor azt "elimináljuk", vagyis kizárjuk a további számításokból. Ezt a "killed" változó növelésével nyomon követjük: `if (newVal < epsilon) { ++killed; }`

3. Azok az elemek, amelyek megfelelnek az epsilon-nál nagyobb kritériumnak, továbbra is megtartásra kerülnek, azaz "kept", és az indexük is növekszik.

Ez a módszer javítja az algoritmus hatékonyságát, mivel csak a releváns adatokat tárolja és kezeli.

Kiterjesztés

A kiterjesztés (expanzió), vagy néha "széthúzás" egy olyan művelet, amely szélesebbé teszi az adatok disztribúcióját. Ez általában egy olyan műveletet jelent, amely növeli a pontok közötti távolságot, különösen azok között, amelyek már eleve távol vannak egymástól. A cél az, hogy nagyobb kontrasztot és jobb különbségtételt biztosítson az adatpontok között.

Az expanszió tehát egy összetett művelet, amely sorban végzi az értékek növelését és az adatok újraszervezését a hasonlósági mátrixban. Ez a művelet a mátrix széthúzását eredményezi, ami nagyobb kontrasztot teremt az adatpontok között, és így elősegíti a klaszterek elkülönítését.

4.4 Algoritmus

Az algoritmus által felhasznált változók es függvények rövid leírása:

Változók:

epsilon - küszöbérték

infRate - inflációs ráta

nrProt - proteinek száma

NZcount - nem zérós elemek száma

bufferSize - jelenlegi tömb értéke

Függvények:

initialize() - inicializáló, lefoglalja a tárhelyeket

normalizeRows(bool ismatTwo): normalizálja a mátrix sorait.

symmetrize(): szimmetrikussá teszi a mátrixot.

symmetrizeWithElimination(): szimmetrikussá teszi a mátrixot, miközben kiküszöböl néhány elemet.

MCLexpansion(): a Markov lánc klaszterezési eljárásban használt kibővítési fázis.

MCLinflation(): a Markov lánc klaszterezési eljárásban használt inflációs fázis.

cleanUp() - felszabadító művelet

makeOutput() - kimeneti adatok megjelenítése

run() - felhívja a függvényeket

4.5 Program lépései

1. Beállítjuk a paramétereket:

- infláció: A Markov Lánc Klaszterezési (MCL) algoritmus inflációs paramétere, amely befolyásolja a klaszterek sűrűségét és elszigetelését.

- epsilon: Ez a paraméter a ciklus terminálásának feltétele, azaz, ha a mátrix változása az előző iterációhoz képest az epsilon érték alá esik, a program befejezi a futást.

2. Beolvasás: A fájl beolvasása történik, ami tartalmazza a protein adatokat. Lehetőség van a beolvasott adatok szűkítésére, így nem mind a 11944 proteinnel kell dolgozni, hanem csak egy specifikus családdal, aminek a mérete pl 20-40 közé esik.

3. run(): Ez a függvény hajtja végre a többi függvényt, beleértve az inicializációt, a ciklusokat, és a kimeneti adatok előállítását.

4. initialize(): Inicializáló függvény, lefoglalja a szükséges tárhelyeket a futtatás előtt.

5. Ciklus: Az alábbi lépések ismétlődnek addig, amíg a program el nem éri a terminálási feltételt (a mátrix változása az előző iterációhoz képest kisebb, mint az epsilon):

- MCLinflation(): A Markov lánc klaszterezési eljárás inflációs fázisának megvalósítása.

- normalizeRows(bool ismatTwo): A mátrix sorainak normalizálása.

- symmetrizeWithElimination(): A mátrix szimmetrikussá tétele, miközben kiküszöböl néhány elemet.

- normalizeRows(bool ismatTwo): A mátrix sorainak normalizálása.

- symmetrize(): A mátrix szimmetrikussá tétele.

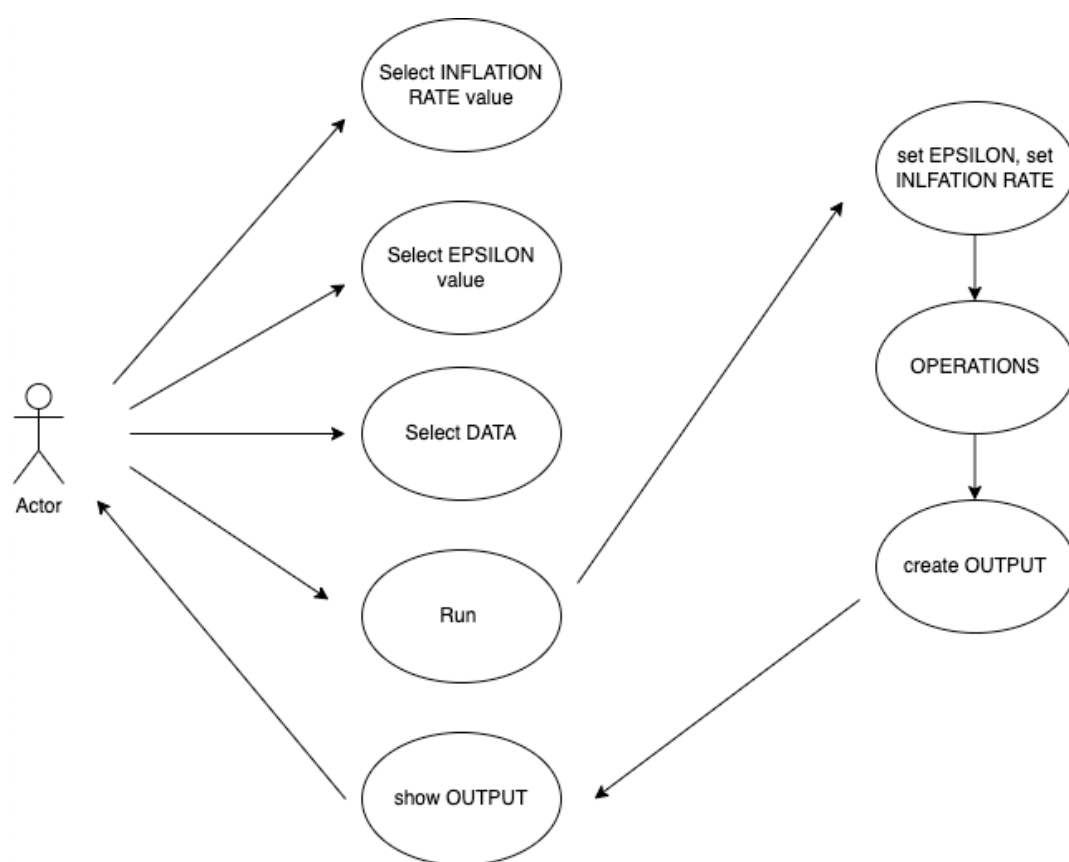
- normalizeRows(bool ismatTwo): A mátrix sorainak normalizálása.

- MCLexpansion(): A Markov lánc klaszterezési eljárás kibővítési fázisa.

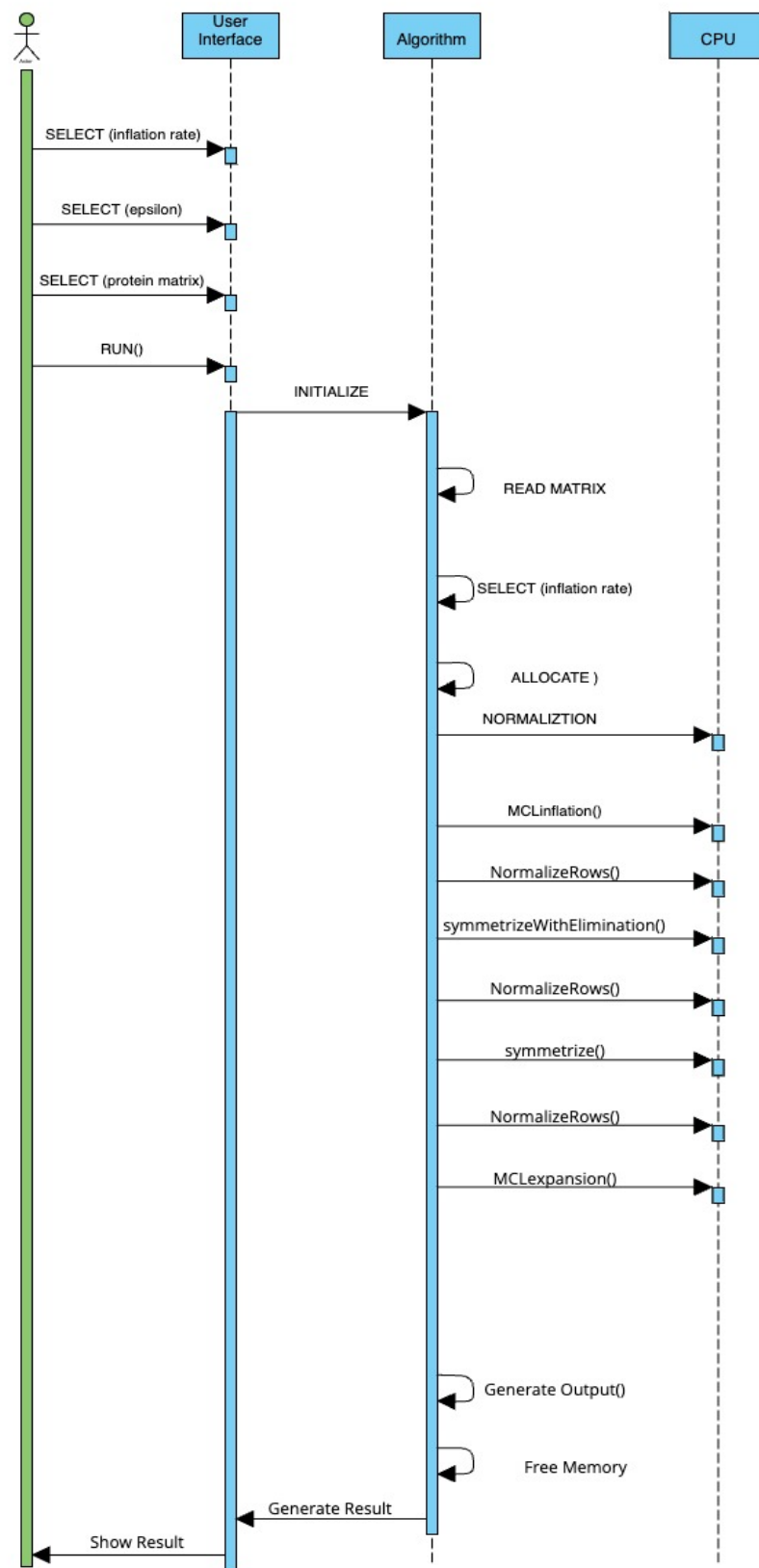
6. makeOutput(): Kimeneti adatok előállítása és megjelenítése.

7. cleanUp(): A futás befejeztével ez a függvény hívódik meg, takarításra használható.

A program futtatását követően meg tudjuk nézni, hogy milyen nagyságú és összetételű csoportokat alakít ki az MCL algoritmus. Emellett megtekinthetjük a csoportokhoz tartozó protein szekvenciákat, és bármely kettőt ezek közül illeszteni tudunk egymáshoz.



Ábra 13 - Esetdiagramm



Ábra 14 – Szekvencia diagramm

5 Megvalósítás

5.1 Szimmetrizáció

A `symmetrize()` függvény feladata, hogy a mátrixot szimmetrikussá tegye. Így működik:

Végigmegy minden nem nulla értéken a mátrixban (matOne-ban). Kiszámítja az új értéket a nem nulla érték és a transzponáltjának gyökét veszi ($\sqrt{\text{matOne.NZbuffer_values}[i] * \text{matOne.NZbuffer_transposed}[i]}$).

Ha az új érték kisebb mint egy előre meghatározott kis szám (epsilon), akkor az új értéket nullává állítja.

Az új értéket beállítja a nem nulla értéknek és a transzponált értéknek.

A függvény hatására a mátrix szimmetrikussá válik, mert a nem nulla értékek és a transzponált értékek azonosak lesznek. Egy mátrix szimmetrikus, ha a mátrix és a transzponáltja megegyezik. Ezt a függvény garantálja azáltal, hogy az összes nem nulla értéket és a transzponált értékeket azonos értékre állítja.

for i = 1 ... n

for j ∈

newVal =

if newVal < eps then newVal = 0

end

end

5.2 Expanzió

Az "expanzió" lényegében egy mátrixszorzás a gráf reprezentáló mátrixon. Ezt a `MCLexpansion()` is valósítja meg: a `matOne` mátrixszorozatát kiszámítja, és az eredményt a `matTwo` mátrixban tárolja.

Az algoritmus lépései:

1. Kezdeként inicializálja a `matTwo` mátrixot, nullázza a nem nulla elemek számát és beállítja az `epsilon2` értékét, ami az `epsilon` felét jelenti.
2. A fő ciklus végigfut a `nrProt` számon.
3. Minden `K` indexű soron elvégzi a mátrixszorzást: a `K` sor minden nem nulla elemére megnézi az elem oszlopában lévő összes nem nulla elemet, és megszorozza azokat az eredeti elemmel. Az eredményeket hozzáadja a `currentRow` tömb megfelelő helyein lévő értékekhez.
4. Ha az így kapott a érték nagyobb mint `epsilon2`, akkor hozzáadja az a értéket a `matTwo` mátrixhoz, mind az értékek, mind a transzponáltak között. Az oszlopindexeket is elmenti.
5. Ezután nullázza a `currentRow` tömböt.
6. Végül beállítja a `matTwo` `rowCount[K]` értékét a `matTwo`-ban lévő nem nulla elemek számának és a `matTwo` `rowStart[K]` értékének különbségére.

Az expanzió képlete nem ritka mátrixok esetén:

```
for i = 1 ... n
  for j = 1 ... n
    sum = 0
    for k = 1 ... n
      sum +=  $M_{ij} \cdot T_{ij}$ 
    end
     $ujM_{ij} = sum$ 
     $ujT_{ij} = sum$ 
  end
end
```

A ritka mátrixokra (sparse matrix) vonatkozó műveletek gyakran hatékonyabbak, mint a sűrű (dense) mátrixokra végzett műveletek, mert a ritka mátrixok csak a nem nulla elemeket tárolják és műveleteket is csak ezeken végeznek. Ezzel szemben a sűrű mátrixok esetében minden elemet tárolni és műveleteket végezni kell, függetlenül attól, hogy az elemek nulla értékűek vagy sem.

A MCL algoritmus expansziós lépése során a mátrix elemeit szorzásokkal és összeadásokkal módosítják. Ha a mátrix ritka, akkor ezeket a műveleteket csak a nem nulla elemekkel kell elvégezni, ami jelentősen csökkentheti a szükséges számítási időt és erőforrást.

Az expanszió művelete során a mátrixot önmagával szorozzuk össze. Ha a mátrix ritka, akkor a szorzás művelete csak a nem nulla elemekkel történik, ami az összes lehetséges művelet számának csökkenését eredményezi.

Ezért a ritka mátrixokon végzett expanszió hatékonyabb, mint a sűrű mátrixokon végzett expanszió.

Az expanszió képlete ritka mátrixok esetén:

```
for  $i = 1 \dots n$ 
     $ujM_{ij} = sum$ 
    for  $k \in row_i$ 
        for  $j \in row_i$ 
             $ujM_{ij} += T_{kj} \cdot M_{kj}$ 
        end
    end
end
```

5.3 Normalizáció

Ez a `normalizeRows()` függvény normalizálja a bemeneti mátrix sorait úgy, hogy minden sor elemeinek összege 1 legyen.

A függvény bemenete egy logikai érték, amely meghatározza, hogy melyik mátrixot kell normalizálni: `matTwo`-t, ha igaz, és `matOne`-t, ha hamis.

A függvény működése a következő:

1. A függvény először meghatározza, hogy melyik mátrixot kell normalizálni (`s` változó).
2. Ezután végigmegy a `nrProt` minden elemén, ami a mátrix sorainak számát jelenti. Minden sorban összeadja az elemeket, és az eredményt a `rowSum` tömb megfelelő helyére menti.
3. Végül ismét végigmegy a `nrProt` minden elemén, és minden sorban normalizálja az elemeket a `rowSum` tömbben tárolt összegével. Itt nem csak a `NZbuffer_values` értékeit normalizálja, hanem a `NZbuffer_transposed` értékeit is, ahol az oszlopnak megfelelő `rowSum` értéket használja.

Tehát összefoglalva, ez a függvény soronként normalizálja a mátrixot, úgy hogy minden sor elemeinek összege 1 legyen, mind a mátrixban, mind a mátrix transzponáltjában.

```
for i = 1 ... n
  for j ∈ rowi
    sum += Mij
  end
  rowSumi = sum
end

for i = 1 ... n
  for j ∈ rowi
     $\tilde{M}_{ij} /= \text{rowSum}_i$ 
```

```

 $\bar{T}_{ij} /= rowSum_j$ 

end

end

```

5.4 Infláció

A MCLinflation() függvény az inflációs rátát (amelyet az infRate változóban tárolnak) használja arra, hogy "felfújja" a matTwo mátrix értékeit.

A függvény működése a következő:

1. Végigmegy a matTwo mátrix összes nemnulla értékén (NZcount az összes nem nulla értékek számát jelzi).
2. Minden iterációban felhasználja a pow() függvényt (amely két argumentumot vesz: az alapot és az kitevőt), hogy az értéket az inflációs rátával megnövelje.
3. A műveletet mind a mátrixban (NZbuffer_values), mind a transzponált mátrixban (NZbuffer_transposed) elvégzi.

Tehát összefoglalva, a MCLinflation() függvény hatására a matTwo mátrix értékei és a transzponáltjának értékei az inflációs rátának megfelelően növekednek, ami segít erősebb és élesebb klasztereket létrehozni a Markov-féle klaszterezésben.

```

for i = 1 ... n
     $\bar{M}_{ij} = M^r_{ij}$ 

     $\bar{T}_{ij} = T^r_{ij}$ 

end

```

5.5 Inicializáció

Az `initialize()` függvény a program előkészítő függvénye, amely beállítja a szükséges változókat és memóriaterületeket a Markov-féle klaszterezéshez.

Az alábbiakban bemutatom, hogy mit csinál a függvény:

1. Beállítja a globális változókat, beleértve a `nrProt`, `infRate`, `epsilon`, `sel_min`, `sel_max`, stb. ezek az adatok, amelyek szükségesek a klaszterezéshez.
2. Beállítja a `matOne` és `matTwo` mátrixokat, beleértve a nem nulla elemek számát (`NZcount`), a pufferméretet (`bufferSize`) és az egyes puffereket (`NZbuffer_values`, `NZbuffer_transposed`, `NZbuffer_colIndex`), valamint a sorkezdő és sorszámot jelző tömböket (`rowCount`, `rowStart`). A pufferméret kiszámítása vagy a `mbSize` bemeneti paraméterből származik, vagy az nem nulla értékek számától függ.
3. Lefoglalja a memóriát a különböző tömbök számára, amelyeket a klaszterezés során használnak, mint például `rowSum`, `currentRow`, `next`, `lxt`, `single`.
4. Előkészíti a `matTwo` mátrixot az adatokkal. Elsőként létrehoz egy `alias` tömböt, amelyet a sorselejtezéshez használ. Ezután beállítja a `matTwo` mátrix nem nulla értékeit és az oszlopindexeit a matrix bemeneti paraméterből, figyelembe véve a `selected` tömböt és az `alias` tömböt.
5. A végén a függvény beállítja a `matTwo` mátrix transzponáltjának értékeit a már beállított nem nulla értékek alapján.

Tehát összefoglalva, az `initialize()` függvény felkészíti a szükséges adatokat és memóriaterületeket a Markov-féle klaszterezés elvégzéséhez, beleértve a `matOne` és `matTwo` mátrixok inicializálását és a szükséges tömbök memóiafoglalását.

5.6 Needleman-Wunsch algoritmus

A Needleman-Wunsch algoritmus egy kulcsfontosságú eszköz a bioinformatikában, amelyet széles körben használnak a genomi információ feldolgozásához és analizálásához. Ennek az algoritmusnak a segítségével összehasonlíthatjuk két szekvencia hosszát, és meghatározhatjuk a leghelyesebb "összehasonlítási utat" a két szekvencia között, ami lehetővé teszi a genomi szekvencia jellemzőinek és a genomi változásoknak a felderítését.

A Needleman-Wunsch algoritmusnak négy fő lépése van:

1. **Mátrix inicializálása:** Először létrehozzák a mátrixot, amelynek mérete $(n+1) \times (m+1)$, ahol n és m a két összehasonlítandó szekvencia hossza. Az inicializálás során minden cellát a lehető legkisebb értékre állítanak. Gyakran ez 0, de negatív szám is lehet, attól függően, hogy milyen büntetési rendszert alkalmaznak a szekvenciaeltérésekért.
2. **A mátrix feltöltése:** A mátrixot feltöltik a szekvenciaértékek alapján. A feltöltés során az algoritmus pontokat ad a szekvencia karaktereinek összehasonlításáért, és büntetést az eltérésekért.
3. **Az optimális összehasonlítási út megtalálása:** Miután a mátrixot feltöltöttük, az algoritmus visszafelé halad a mátrixon, hogy megtalálja az optimális összehasonlítási utat. Ez az út adja a legjobb összehasonlítást a két szekvencia között.
4. **Az optimális összehasonlítás visszaadása:** Az algoritmus visszaadja az optimális összehasonlítást, ami a két szekvencia közötti legjobb összehasonlítás. Ez lehetővé teszi a genomi információk alapos elemzését és az új információk felderítését.

A Needleman-Wunsch algoritmus a globális összehasonlítást támogatja, ami azt jelenti, hogy a teljes szekvenciát használja az összehasonlításhoz, nem csak a részeket.

Az illesztés megvalósítása

1. Először ellenőrzi, hogy a ``visRes->firstSel`` és ``visRes->secondSel`` értékek nagyobbak-e nullánál. Ha igen, ez azt jelenti, hogy a felhasználó kiválasztott két fehérjét a ``listProt`` listából, amiket össze akar hasonlítani.
2. Létrehozza a ``seq1`` és ``seq2`` változókat, amik tartalmazzák a két összehasonlítandó fehérjeszekvenciát, valamint az ``alseq1`` és ``alseq2`` változókat, amik az összehasonlítás eredményét tartalmazzák.
3. Az algoritmus ezután létrehoz egy ``D`` és egy ``trace`` mátrixot, amik a dinamikus programozás során számolt értékeket és a visszakövetést (traceback) tartalmazzák.
4. Ezt követően feltölti a ``D`` és a ``trace`` mátrixokat az algoritmus szerint. Ebben az esetben a ``D`` mátrix értékei azt határozzák meg, hogy mennyire "jó" a fehérjék adott pozícióinak összehasonlítása, míg a ``trace`` mátrix arra szolgál, hogy visszakövessük, melyik lépést választottuk.
5. A mátrixok feltöltése után végrehajtja a visszakövetést (traceback) az algoritmus szerint. Az eredmény az ``alseq1`` és ``alseq2`` szekvenciákban lesz tárolva, amelyek az optimális összehasonlítást tartalmazzák a két fehérjeszekvencia között.
6. A program végül vizuálisan megjeleníti az eredményt: létrehoz egy képet, amelyen a két összehasonlított szekvenciát jeleníti meg, ahol a közös részeket piros színnel, a nem egyező részeket pedig szürke színnel jelöli. A képet "seqalign.png" néven menti le.

A kód a BLOSUM62 mátrixot használja a szekvencia összehasonlítás során.

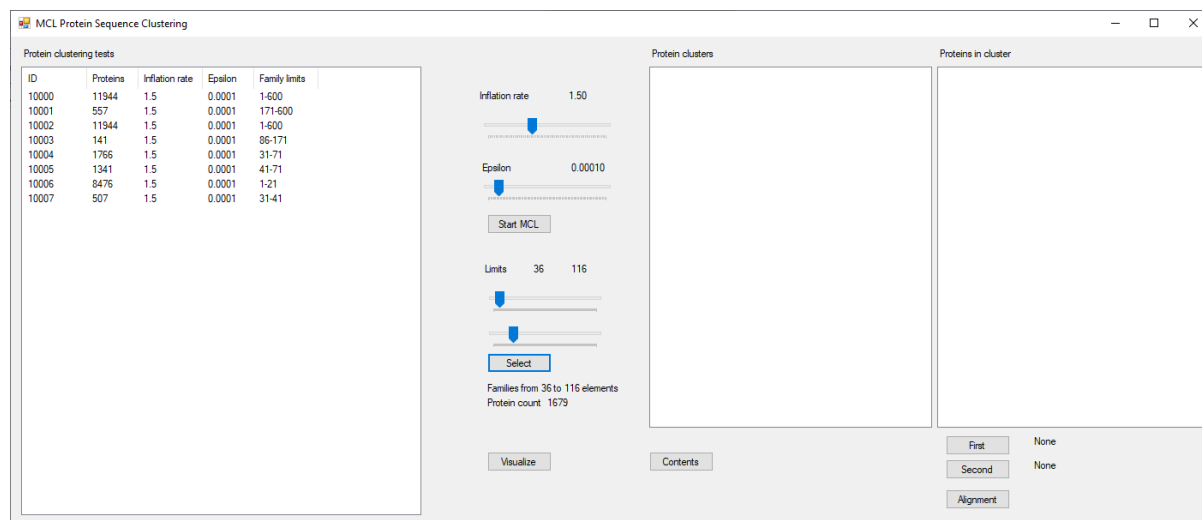
5.7 BLOSUM62

A BLOSUM62 mátrix a bioinformatikában használt eszköz, amelyet a fehérjék aminosav-sorrendjének összehasonlításához használnak. A BLOSUM (BLOCKS SUBstitution Matrix) mátrixok olyan pontozási rendszerek, amelyeket a szekvencia összehasonlító algoritmusokban használnak, beleértve a Needleman-Wunsch algoritmust is. A pontozási rendszerek segítenek az algoritmusoknak abban, hogy pontokat adjanak a szekvencia-karakterek összehasonlításáért, és büntetést a szekvencia-eltérésekért.

A BLOSUM62 mátrixot Steven Henikoff és Jorja Henikoff hozta létre 1992-ben. A "62" a mátrix nevében arra utal, hogy a mátrixot olyan fehérje blokkokból hozták létre, amelyekben a szekvenciák legalább 62%-ban azonosak voltak. Minél nagyobb a szám a BLOSUM mátrix nevében, annál nagyobb a szekvencia-azonosság a blokkok között.

A BLOSUM62 mátrixban minden cella értéke egy adott aminosavpár helyettesítésének valószínűségét jelzi. Pozitív érték jelzi, hogy a helyettesítés valószínűbb, mint véletlenszerű, míg a negatív érték azt jelzi, hogy a helyettesítés kevésbé valószínű. A BLOSUM62 mátrixot gyakran használják a bioinformatikában, mert jól teljesít sokféle fehérje összehasonlításában.

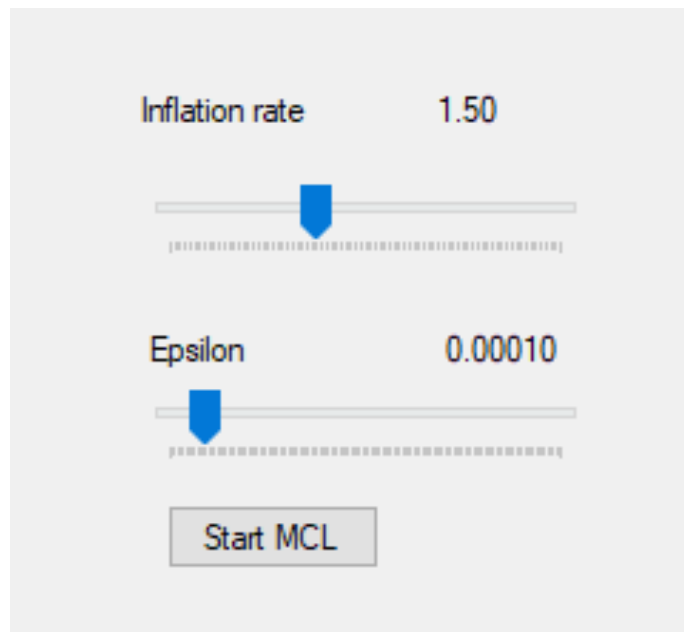
6 Felhasználó felület



Ábra 15 – Program grafikus felülete

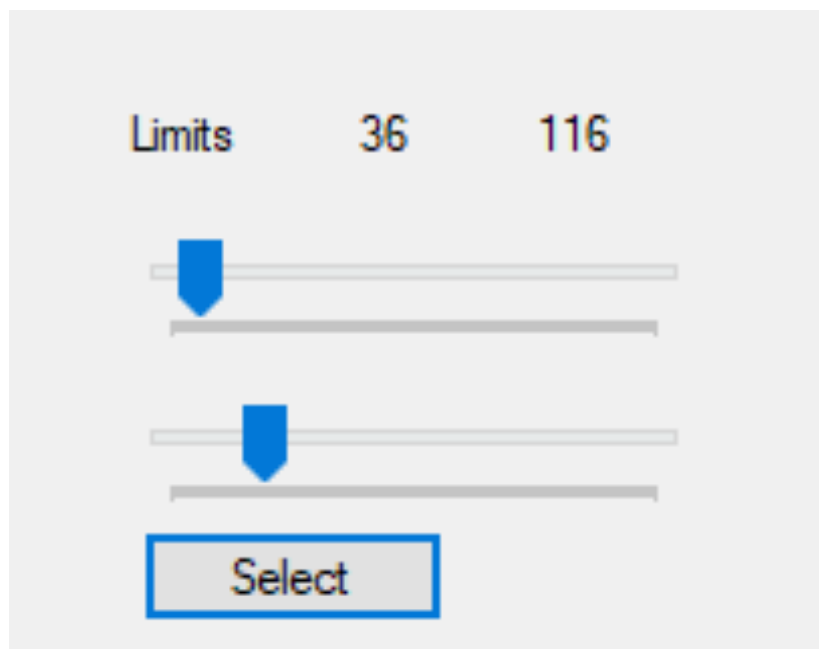
A program grafikus felülete:

Az elemzési folyamat kezdeti lépései között kiemelkedően fontos az inflációs ráta és az epsilon érték meghatározása. Ez a lépés kritikus, hiszen ezek az értékek határozzák meg a későbbi analitikai folyamatok precizitását és megbízhatóságát. A vizsgált esetben az inflációs ráta értéke 1.5, míg az epsilon érték meghatározása 0.00010-re esik.



Ábra 16 – Inflációs ráta és az epsilon kiválasztása

Ezt követően meghatároztuk az intervallumot, amely a vizsgálandó protein családok méretét szabja meg. A jelen esetben alkalmazott intervallum 36-tól 116-ig tartó értékeket tartalmaz.



Ábra 17 – Intervallum meghatározása

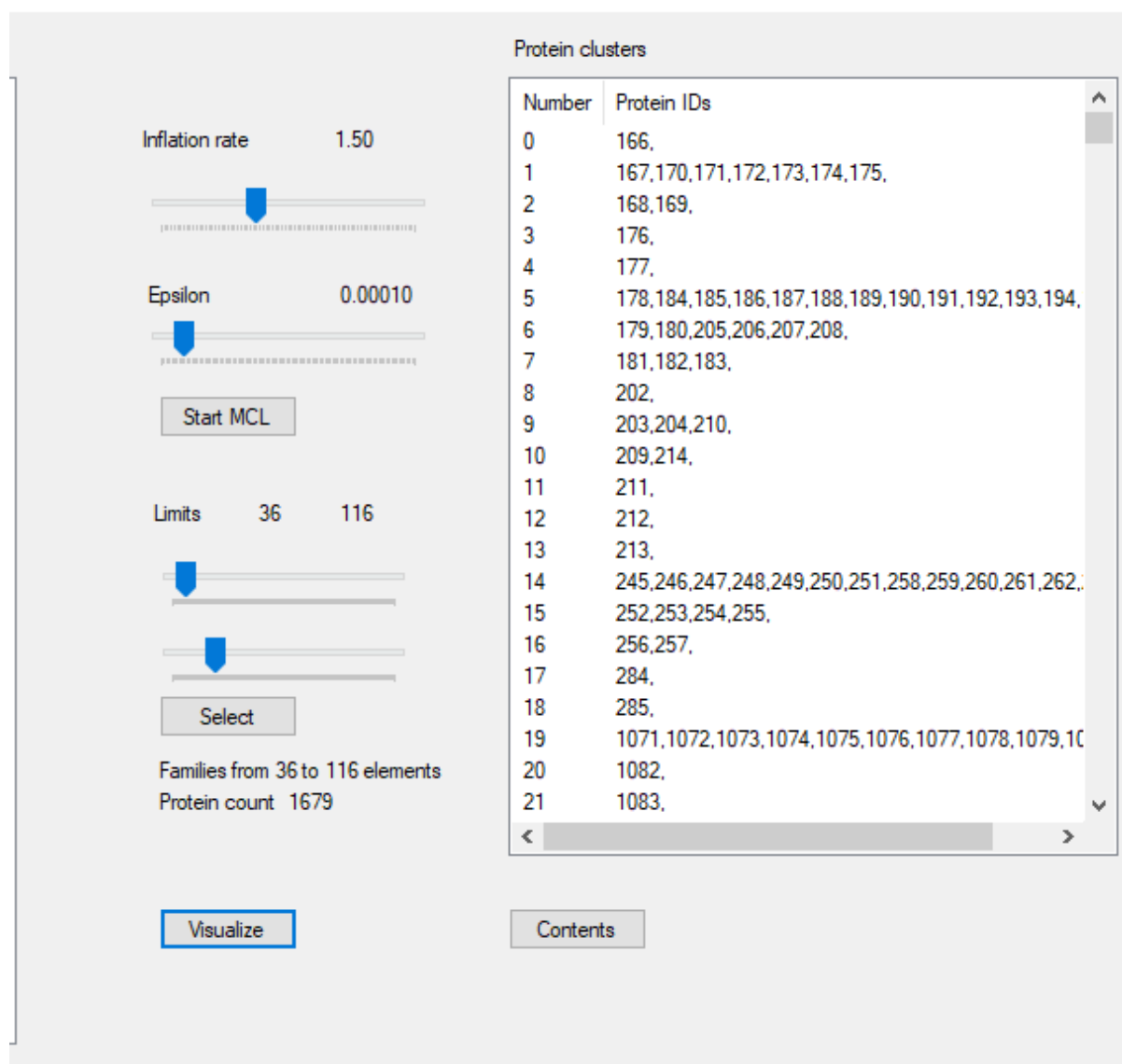
A meghatározott értékek alapján aktiváljuk a 'Start MCL' funkciót, amely képes felbecsülni a protein családokban található proteinek összesített számát. A jelenlegi adataink alapján ez az érték 1679.



Families from 36 to 116 elements
Protein count 1679

Ábra 18 – Családokban található proteinek számának megjelenítése

Végül a vizualizáció érdekében használatba vesszük a 'visualize' gombot, ennek hatására a proteinek kódjai megjelennek a jobb oldali 'protein clusters' ablakban, amelyek közvetlenül hozzáférhetővé és láthatóvá teszik a proteinek azonosítóit. Ez a funkció segít a felhasználónak értelmezni az adatokat.



Ábra 19 – Létrehozott klaszterek ábrázolása

A 'Protein clusters' ablakban interaktív módon lehetőségünk nyílik az adott protein családok kiválasztására, egyszerűen a kívánt családra kattintva. Ez a funkció lehetővé teszi, hogy a rendelkezésükre álló nagy mennyiségű adat között gyorsan és hatékonyan navigáljanak. Jelenlegi példánkban a 19-es családot választottuk ki a vizsgálatra.

Protein clusters	
Number	Protein IDs
0	0,1,2,3,
1	4,
2	5,
3	6,7,8,
4	9,
5	10,
6	11,12,13,
7	14,16,17,18,19,20,21,22,27,28,29,30,31,32,33,34,35
8	15,
9	23,
10	24,25,26,
11	66,
12	67,
13	68,69,
14	70,71,
15	72,73,74,
16	75,
17	76,
18	78,
19	79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95
20	102,103,
21	104,

< >

Contents

Ábra 20 – Megjeleníteni kívánt klaszter kiválasztása

A kiválasztott család részletes adatainak megtekintése érdekében a 'contents' gombra kell kattintani. Ennek hatására a jobb oldalon található 'Protein in cluster' ablakban megjelennek az adott családba tartozó proteinek adatai, beleértve a proteinek egyedi kódjait is.

Protein clusters		Proteins in cluster	
Number	Protein IDs	Name	Sequence
0	0,1,2,3,	d1phna_	mktpteiaiaaadnqgrflsntelqavngryqraasleaarsl
1	4,	d1f99a_	mktpteiaiaaadsqgrflsntelqvngrynratssleaakal
2	5,	d1cpca_	mktpteavaaadsqgrflssteiqtafgrfqasaslaaakalt
3	6,7,8,	d1jboa_	mktpteiaiaaadqgrflsntelqavdgrkravasmaeara
4	9,	d1gh0a_	mktpteavsvadsqgrflssteiqvafgrfqakagleaaka
5	10,	d1phnb_	mldafakvvaqadargeflsntqldalskmvsegnkrldvvi
6	11,12,13,	d1f99b_	mldafakvvaqadargeflsntqldallaivsegnkrldvvnk
7	14,16,17,18,19,20,21,22,27,28,29,30,31,32,33,34,35	d1cpcb_	mldafakvvsqadargeylsgsqldalsalvadgnkmdv
8	15,	d1jbob_	mldafakvvaqadargeflsntqldalsnlvkegnkrldavr
9	23,	d1gh0b_	mldafakvvsqadargemlstaqlalsqmvasesnkrldvvr
10	24,25,26,	d1alla_	sivtksvsnadaearylspgeldrksfvtsgenvriaetmtgai
11	66,	d1b33a_	sivtksvsnadaearylspgeldrksfvtsgenvriaetmtgai
12	67,	d1kn1a_	sivtksvsnadaearylspgeldrksfvtsgenvriaetmtgai
13	68,69,	d1allb_	mldafavinsdsvqgkyldasaiklkayfatgelvraattia
14	70,71,	d1b33b_	mldafavinsdsvqgkyldasaiklkayfatgelvraattia
15	72,73,74,	d1kn1b_	mldafavinsdsvqgkyldasaiklkayfatgelvraattia
16	75,	d1liaa_	mksvittisaadaagrpssdlesiqgniqraaaleaak
17	76,	d1b8da_	mksvittisaadaagrpssdlesiqgniqraaaleaak
18	78,	d1eyxa_	mksvittisaadaagrpssdlesiqgniqraaaleaak
19	79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95	d1liab_	mldafsvvnsdskaaayvggsdlqalktfndgnkrldavn
20	102,103,	d1b8db_	mldafsvvnsdskaaayvggsdlqalktfndgnkrldavn
21	104,	d1eyxb_	mldafsvvnsdskaaayvggsdlqalktfndgnkrldavn

Ábra 21 – Klaszterben található proteinek adatai

A 'Protein in cluster' ablak strukturált formában jeleníti meg az adatokat, amelyeket két oszlopban rendez: az első oszlopban található a protein neve, míg a második oszlopban a protein szekvenciája. Ez a rendezési mód elősegíti az adatok átláthatóságát és a könnyebb adatfeldolgozást, valamint a szekvenciaanalízis későbbi lépéseinek elvégzésében is szerepe van.

Proteins in cluster	
Name	Sequence
d1phna_	mktpteiaiaaadnqgrflsntelqavngryqraaasleaarsl
d1f99a_	mktplteaiaaadsqgrflsntelqvvngrynratssleaakal
d1cpc_a_	mktplteaavaadsqgrflssteiqtafgrfrqasaslaaakalt
d1jboa_	mktpteiaiaaadtggrflsntelqavdgrfkravasmeaara
d1gh0a_	mktplteavsvadsqgrflssteiqvafgrfrqakagleaaka
d1phnb_	mldafakvvaqadargeflsntqldalskmvsegnkrdvvi
d1f99b_	mldafakvvaqadargeflsntqidallaivsegnkrdvvnk
d1cpcb_	mldafakvvsqadargeylsgsqidalsalvadgnkmdvv
d1jbob_	mldafakvvaqadargefltnaqfdalsnlvkegnkrdavn
d1gh0b_	mfdafkvvsqadtrgemlstaqidalsqmvaesnkrdvvr
d1alla_	sivtkshivnadaearylspgeldriksfvtsgemvriaetmtgai
d1b33a_	sivtkshivnadaearylspgeldriksfvssgekrriaqiltdnre
d1kn1a_	sivtkshivnadaearylspgeldriksfvlsgamvriaqilttenre
d1allb_	mqdaitsvinssdvqgkyldasaiqlkayfatgelrvraattis
d1b33b_	mqdaitavinssdvqgkyldtaaleklksyfstgelrvraattia
d1kn1b_	mqdaitsvinssdvqgkyldssaieklkgyfqtgelrvraattia
d1liaa_	mksvittisaadaagrypstsdlqsvqgniqraaarleaaek
d1b8da_	mksvittisaadaagrfpsssdlesiqgniqraaarleaaekls
d1eyxa_	mksvittvisaadsagrfpsssdlesvqgniqrasarleaaek
d1liab_	mldafsrvvnsdskaaayvgsdlqalktfndgnkrdavn
d1b8db_	mldafsrvvtsdskaaayvgsdlqslksfindgnkrdavn
d1eyxb_	mldafsrvisnadakaaayvgsdlqalrtfisdgnkrdavnyi

Ábra 22 – Kiválasztott protein

A következő lépésben, az 'First' és 'Second' gombok alkalmazásával két specifikus proteint választunk ki összehasonlítás céljából. Ez a folyamat lehetővé teszi a proteinek közötti jelentős különbségek és hasonlóságok feltárását, amelyek számos biológiai funkció megértéséhez járulnak hozzá.

A gomb lenyomása után az összehasonlításra kiválasztott proteinre történő kattintással jelezzük a választásunkat.

Ebben a konkrét esetben a 'First' gomb segítségével a '**d1jbob_**' nevű proteint választottuk ki, míg a 'second' gomb használatával a '**d1eyxa_**' nevű proteint. Mindkét protein egyedi és fontos jellemzőkkel bír, így összehasonlításuk értékes információkkal szolgálhat a proteinstruktúrák és funkciók megértésében.

Proteins in cluster

Name	Sequence
d1phna_	mktpteiaiaadnqgrflsntelqavngryqraasleaarsl
d1f99a_	mktpteiaiaadsqgrflsntelqvngrynratssleaakal
d1cpca_	mktpteavaadsqgrflssteiqtafgrfqasaslaaakalt
d1jboa_	mktpteiaiaadtqgrflsntelqavdgrfkraasmeaara
d1gh0a_	mktpteavsvadsqgrflssteiqvafgrfqakagleaaka
d1phnb_	mldafakvvaqadargeflsntqldalskmvsegnkrldvvi
d1f99b_	mldafakvvaqadargeflsntqldallavsegnkrldvvnk
d1cpcb_	mldafakvvsqadargeylsgsqidalsalvadgnkmdvv
d1jbob_	mldafakvvaqadargefltnaqfdalsnlvkegnkrldavn
d1gh0b_	mldafakvvsqadtrgemlstaqidalsqmvaesnkrdvvr
d1alla_	sivtksvvnadaearylspgeldniksfvtsgemvriaetmtgai
d1b33a_	sivtksvvnadaearylspgeldniksfvssgekrriaqiltndre
d1kn1a_	sivtksvvnadaearylspgeldniksfvsgamvriaqiltndre
d1allb_	mldafavvnsdsvqgkyldasaiqlkayfatgelrvraattis
d1b33b_	mldafavvnsdsvqgkyldtaaleklkyfqtgelrvraattia
d1kn1b_	mldafavvnsdsvqgkyldsaieklkyfqtgelrvraattia
d1liaa_	mksvittisaadaagrpssdsqsvqgniqraaarleaakle
d1b8da_	mksvittisaadaagrpssdsqsvqgniqraaarleaakle
d1eyxa_	mksvittisaadaagrpssdsqsvqgniqraaarleaakle
d1liab_	mldafsvvnsdskaaayvgsdlqalktfndgnkrldavn
d1b8db_	mldafsvvnsdskaaayvgsdlqalktfndgnkrldavn
d1eyxb_	mldafsvvnsdskaaayvgsdlqalktfndgnkrldavnyi

First
d1jbob_
Second
d1eyxa_
Alignment

Ábra 23 – Összehasonlításra kiválasztott két protein

A következő lépésben az 'Alignment' gomb használatával összehasonlíthatjuk a kiválasztott két protein szekvenciáját. Ez a művelet lehetővé teszi, hogy a proteinek aminosav-sorrendjét összevetve meghatározzuk a közöttük fennálló hasonlóságokat és eltéréseket. Ez a sorrend meghatározza a protein szerkezetét, ami befolyásolja funkcionális tulajdonságait, beleértve a biológiai folyamatokban betöltött szerepét is. Egy protein szekvenciája általában 20 különböző típusú aminosavat tartalmaz.

```

m d a f a k v v _   a q a d _ a r g e f   l t n a q f d a l _   s n l v k e g n _ _
m k s v i t t v i s   a _ a d s a _ g r f   _ p s s s _ d _ l e   s _ _ v _ q g n i q

g _ _ g _ n _ a y t   n r r m a a c l r d   m e _ i i l r y v t   y a _ i l a g d s s
g e a g e n q e k i   n _ k _ _ _ c y r d   i d h _ y m r l v n   y s l v i g g _ t g

k _ _ _ r l d a v n   r i t s n _ _ a s t   i v a n a a r a l f   a e q p q l i q _ p
r a s a r l e a a e   k l a s n h e a _ _   v v k e a g d a c f   g k y g y l _ k n p

v l d d _ r c l n g   l r e t y q a l g t   p g s s v a v a i q   k n k d a a i a i a
p l d e w g _ i a g   a r e v y r t l n l   p _ t s _ a y _ i _   _ _ _ _ a a f a f t

n d p n _ _ g i t p   g d c s a l m s e i   a g _ _ y _ _ _ f d   r a a a a v a
r d _ r l c g _ _ p   r d m s a _ _ _ q _   a g v e y s t a l d   y i i n s l s

```

Ábra 24 – Protein illesztés

Az összehasonlító vizualizáció során természetesen a felső sorok ábrázolják a 'First' gombbal kiválasztott protein, a 'dljbob_', szekvenciáját, míg a második sorok a 'Second' gombbal kiválasztott, a 'dleyxa_' nevű protein szekvenciáját képviselik. Ez a jelölési rendszer egyszerűsíti a proteinek összehasonlítását, mivel egyszerűen követhető, melyik szekvencia melyik proteinhez tartozik.

A szekvencia-alapú összehasonlítás során a hasonlóságokat piros színnel emeljük ki, mi lehetővé teszi, hogy gyorsan és könnyen azonosítsuk azokat a részeket, ahol a két protein szekvenciája megegyezik.

7 Bibliográfia

- [1] Szilágyi SM, Szilágyi L: A fast hierarchical clustering algorithm for large-scale protein sequence data sets. *COMPUTERS IN BIOLOGY AND MEDICINE* 48:94–101, 2014
- [2] Szilágyi L, Szilágyi SM: A modified two-stage Markov clustering algorithm for large and sparse networks. *COMPUTER METHODS AND PROGRAMS IN BIOMEDICINE* 135:15-26, 2016
- [3] Szilágyi L, Kovács L, Szilágyi SM: Synthetic test data generation for hierarchical graph clustering methods. International Conference on Neural Information Processing (ICONIP 2014, Kuching, Malaysia). In: Loo CK, Yap KS, Wong KW, Teoh A, Huang K (Eds.): *Neural Information Processing*, Springer, LNCS vol. 8835, pp. 303-310, 2014
- [4] Szilágyi L, Medvés L, Szilágyi SM: A modified Markov clustering approach to unsupervised classification of protein sequences. *NEUROCOMPUTING* 73(13-15):2332-2345, 2010
- [5] L. LoConte, B. Ailey, T. J. Hubbard, S. E. Brenner, A. G. Murzin, C. Chothia, SCOP: a structural classification of protein database, *Nucl. Acids Res.* 28(2000) 257–259.
- [6] A. Andreeva, D. Howorth, J. M. Chandonia, S. E. Brenner, T. J. P. Hubbard, C. Chothia, A. G. Murzin, Data growth and its impact on the SCOP database: new developments, *Nucl. Acids Res.* 36(2008) D419–D425.
- [7] S. B. Needleman, C. D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins, *J. Mol. Biol.* 48 (1970) 443–453.
- [8] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, *J. Mol. Biol.* 147 (1981) 195–197.