



37. OTDK Biológia Szekció

Skálázható, MinHash alapú szekvencia-reprezentációk kidolgozása és vizsgálata

Juhász Judit

Molekuláris bionika mérnök BSc - 2020 évfolyam

2024

Témavezető: Dr. Ligeti Balázs

Tartalomjegyzék

1. Bevezetés	2
1.1. Problémafelvetés	2
Pángenom gráfok	3
Szuboptimális megoldás keresése, és a probléma definiálása	3
1.2. Számítástechnikai nehézségek	4
1.3. A szoftver alkalmazási lehetőségei	5
1.4. A megoldás biológiai jelentősége	5
2. <i>Sketching</i> megközelítéssel dolgozó módszerek szövegek hasonlóságá- nak megbecsülésére	5
2.1. Bloom filter	6
2.2. Count-Min Sketch	7
2.3. MinHash	7
A lenyomat generálása	8
A lenyomatok összehasonlítása	8
A MinHash algoritmus alkalmazása szövegek hasonlóságának becslésére . .	9
2.4. HyperLogLog	9
A HyperLogLog algoritmus alkalmazása szekvenciák hasonlóságának becs- lésére	10
3. A problémára alkotott megoldásom	11
3.1. Az input	11
3.2. Az output	11
3.3. Az algoritmus bemutatása	12
Megoldás egy egyszerűbb problémára	12
Megoldás a tényleges problémára	13
4. Az algoritmus elemzése és elméleti korlátai	22
4.1. Az algoritmus lépéseinek összefoglalása	22
4.2. Az inputtal szemben alkotott feltételek	22
4.3. Az algoritmus komplexitásának megbecsülése	24
Időkomplexitás	24

Helykomplexitás	27
5. Gyakorlati eredmények	27
Az input specifikációja	27
5.1. Eredmények	28
Teszt mesterségesen generált adatokkal	29
Jelen lévő szekvenciák keresése valódi genomok alapján generált gráfban .	30
Az eredmények kiértékelése	32
6. Összefoglalás	34

Kivonat

Az újgenerációs szekvenálási eljárások elterjedésével a bioinformatikai információ mennyisége rohamosan nő. A tradicionális megközelítésben a DNS-t lineáris szövegként értelmezzük. Ez a leírás azonban problémákat okozhat, amikor nagyobb csoportok, telepek, fajok genetikai információjáról beszélünk. A genetikai diverzitás leírásához más formátumra van szükségünk. Emiatt a probléma miatt kezd elterjedni a bioinformatikában a pángenom gráfok használata, melyek kompakt formában tartalmazzák az egyedek közötti hasonlóságokat és különbségeket is. Egy pángenom gráf összerakása és értelmezése is komplex, számításelméletileg nehéz feladat lehet. Mivel már a baktériumok genomjának hossza is milliós nagyságrendű lehet, az ilyen problémákra nagyon fontos hatékony, jól skálázódó megoldást találni. A bioinformatikai szoftverek gyakran alkalmaznak szuboptimális, és úgynevezett *sketching* alapú algoritmusokat a hatékonyság érdekében. Dolgozatomban áttekintem a legelterjedtebb algoritmusokat, valamint bemutatom a saját megoldásomat, mely képes megállapítani, hogy van-e egy adott szekvenciához hasonló rész a pángenom gráfban, és ha igen, akkor a gráf mely részén található. Az általam elkészített szoftver várhatóan lehetővé teszi annak ellenőrzését, hogy egy adott gén/ tulajdonság / virális eredetű szakasz jelen van-e egy baktériumtelep legalább néhány egyedében. A kódot a kutatócsoportom fágterápiát kutató része is használni fogja, első körben a saját adataik tesztelésére, majd a pángenom gráfban való tájékozódásra.

1. Bevezetés

1.1. Problémafelvetés

Az újgenerációs szekvenálási eljárások elterjedésével a bioinformatikai információ mennyisége rohamosan nő. A génállomány kizárólag szekvenciális értelmezése azonban problémákat okozhat [1], amikor nagyobb csoportok, telepek, fajok genetikai információjáról beszélünk, mivel a szekvenciális ábrázolás kizárólag egy egyed génállományára koncentrálnak.

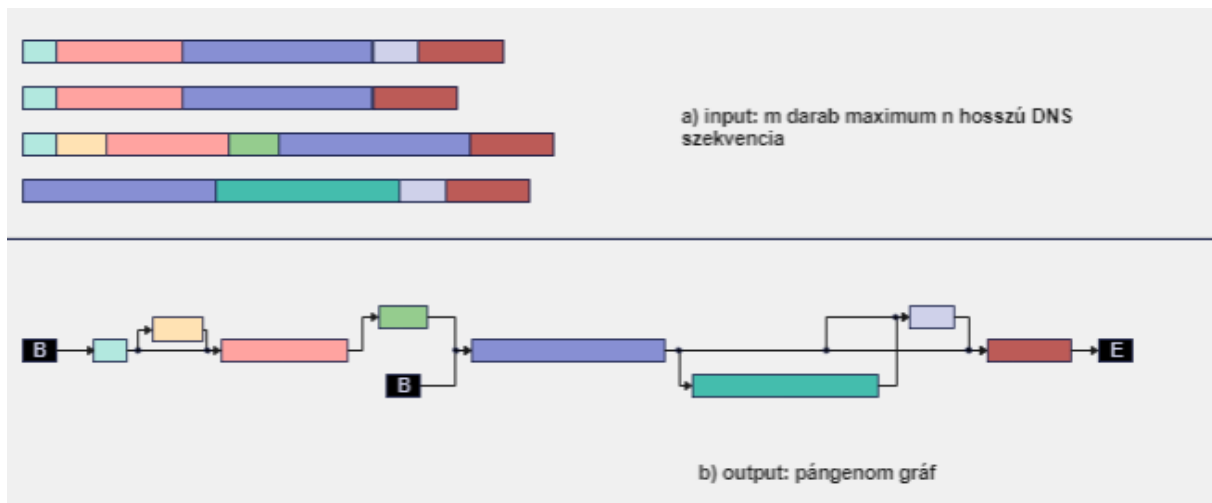
Ezen túl a szekvenciák hasonlóságának elemzésekor felmerülő probléma lehet a sorrendiség figyelembe nem vétele miatt, hogy a referencia genom segítségével rekonstruált genomok közelebbinek tűnhetnek a referenciához, mint amennyire valójában hasonlóak ahhoz. Az ilyen problémák kezelésére kezdett elterjedni a genetikai információ pángenom rendszerként értelmezése. A pángenom rendszer a genetikai információ olyan gyűjteménye, amely tartalmazza a vizsgált csoportból minden egyed génkészletének unióját. A pángenom rendszerek használata csökkentheti a szekvenciális megközelítés esetén megjelenő torzítást azzal, hogy lehetővé teszi egy új genom közvetlen vizsgálatát az összes olyan genommal szemben, amely a pángenom reprezentáció része.

A pángenomika kiemelkedő fontossággal bír a mikrobiológiában, ahol a genomikai plaszticitás és a diverzitás nélkülözhetetlenné teszi. Egyre jellemzőbb, hogy eukarióta genomokra is alkalmazzák. [2] Jelentős további alkalmazása a mikrobiomok vizsgálata. Az elkövetkező időszakban számtalan olyan új kihívással kell szembenézni, amelyben a mikrobiális közösségek segítségünkre lehetnek. Ilyen lehet a klímaválság, és a szuperrezisztencia terjedése. A mikrobiális közösségek jellemzője, hogy hatalmas metabolikus potenciállal rendelkeznek, és számos komplex feladatot képesek megoldani, mint például a szerves anyagok lebontása, a kolonizáció, a patogénekkal szembeni védelmi szerep. Így kulcsfontosságú szerepet játszhatnak ezeknek az új kihívásoknak a leküzdésében. A mikrobiomtervezés nem triviálisan megoldható feladat, pedig alapvető fontosságú lenne az egészségmegőrzés, mezőgazdaság és klímaváltozások kezelése szempontjából. [3] A mikrobiom nem egymástól független baktériumok, archeák, gombák, fágok összessége, hanem egy közösséget alkotó rendszer, amelynek komponensei egymással aktív kölcsönhatásban és interakcióban vannak [4], pl. Quorum sensing révén [5]. Egy ilyen rendszer értelmezé-

séhez rendkívül fontos a mikrobiomon belüli fajok diverzitásának leírása.

Pán genom gráfok

A pán genom rendszerek kezelésére ideális formátum a gráf. A pán genom gráf egy olyan irányított gráf, melynek csúcsai szekvenciákat tartalmaznak (jelen esetben DNS szekvenciákat), élei pedig a csúcsok kapcsolatait definiálják. A pán genom rendszer egyes szekvenciái visszanyerhetők a pán genom gráf összefüggő részgráfjainak Hamilton utaiként. A gráf elágazósága jó mérőszáma lehet a genetikai variabilitásnak. Az ilyen gráfok azonban még mindig nehezen kezelhetők: gyakran hatalmas méretűek, és bejárásuk, vagy a bennük való keresés során könnyen számításelméletileg nehéz problémákba ütközhetünk. A pán genom gráfok és a pán genom rendszerek kapcsolatát az 1. ábra szemlélteti.



1. ábra. Pán genom gráf összeállítása: Az irányított gráfban található utak megfeleltethetők a pán genom rendszer szekvenciáinak. A B-vel jelölt csúcsok egy vagy több szekvencia kezdetét, az E-vel jelölt csúcsok egy vagy több szekvencia végét jelzik. Ezt az információt egy valódi pán genom gráf nem feltétlenül tartalmazza, az ábrán a könnyebb értelmezhetőség miatt szerepel.

Szuboptimális megoldás keresése, és a probléma definiálása

A szakdolgozatomban célja jól skálázódó, szöveg-összehasonlító algoritmusok keresése és megértése, valamint ezek felhasználásával egy olyan szoftver implementálása, mely megkönnyíti a kutatócsoport több tagja számára a pán genom gráfok értelmezését, kezelését.

A félév során főleg arra kerestem megoldást, hogy egy pán genom gráfról és egy szekvenciáról meg tudjam válaszolni a következő kérdést:

A gráfhoz tartozik-e olyan irányított részgráf melyre teljesül, hogy:

- Összefüggő
- Található rajta olyan Hamilton út, mely mentén konkatenálva az érintett csúcsokhoz tartozó szekvenciákat, a bemenetként megadott szekvenciához „megfelelő mértékben” hasonló szekvenciát kapunk.

Intuitívan megfogalmazva a célom a bemeneti szekvenciához hasonló rész keresése a gráfban. A hasonlóság mértéke nehezen definiálható, szükséges elfogadni olyan eseteket is, amikor a megtalált szekvencia a bemeneti szekvenciához képest többletinformációt tartalmaz (inzerció) vagy bemeneti szekvencia részeit nem tartalmazza. Ezek a transzformációk nem mérhetőek Hamming távolsággal. Továbbá az, hogy mekkora mértékben toleráljuk az inzerciót és a deléciót, függ a biológiai kontextustól is. Ebből adódóan a megoldásnak legalább valamilyen mértékben paraméterezhetőnek kell lennie a feladat függvényében.

A Hamming távolság alkalmazása helyett felfoghatjuk a két vizsgált szöveget részszövegek halmazaként, és vizsgálhatjuk ezeknek a halmazoknak a hasonlóságát. Két szöveg részszövegeinek összehasonlítása azonban már önmagában is rosszul, a szövegek hosszának függvényében időben exponenciálisan skálázódik. Ezért a problémára széles körben elterjedt megoldás az, hogy szuboptimális megoldást keresünk. Ez esetben egy olyan megoldás, amely nem pontosan adja meg a két szöveg közötti hasonlóságot, azonban a valósághoz megbízhatóan közeli eredményt ad.

1.2. Számítástechnikai nehézségek

A bemeneti adatok növelésével problémává válhat a programmemória mérete, ezért fontos szempont volt az algoritmus megtervezése során az, hogy a program futásának minden pillanatában elég legyen csak a gráf egy részét memóriában tartani.

A skálázhatóságon túl az algoritmus flexibilitását is érdemes szem előtt tartani. Mind a szekvenciák közötti hasonlóság mértékére, mind a pángenom gráf tulajdonságaira nehezen adható egzakt definíció, ezért belátásom szerint szükséges, hogy az algoritmusnak legyenek szabad, a bemenet függvényében állítható paraméterei.

1.3. A szoftver alkalmazási lehetőségei

Szakdolgozatomat az egyetemen található Neurális Bioinformatika Kutatócsoportban írtam. A kutatócsoport azon része, amellyel dolgoztam, fágterápiával foglalkozik pángenomikai megközelítéssel. A projekt fontos eleme a baktériumközösségek genetikai állományának vizsgálata. A csoportnak már rendelkezésére áll olyan szoftver, amely képes pángenom gráfot építeni, azonban a megépített gráf pontossága nehezen mérhető. Az általam kínált algoritmus már a kezdeti lépésekben is felhasználásra kerül majd a következő kérdéskörök megválaszolására:

- Jelen van-e egy szekvencia a megépített pángenom gráfban,
- A baktériumok plazmidjai bekerülnek-e a pángenom gráfba,
- Bekerül-e egy olyan szekvencia a pángenom gráfba, amely a pángenom rendszernek csak kevés mintájában van jelen,
- A baktériumok genomja tartalmaz-e lizogén fázisú virális genomot.

1.4. A megoldás biológiai jelentősége

A kész szoftver feltehetően lehetővé teszi annak a vizsgálatát, hogy egy adott csoport egyedeinek genetikai állományában megjelenik-e egy kiválasztott genetikai egység. Skálázhatósága miatt reményeim szerint baktériumnál komplexebb élőlények esetében is alkalmazható lesz.

2. *Sketching* megközelítéssel dolgozó módszerek szövegek hasonlóságának megbecsülésére

Halmazok hasonlóságának vagy kardinalitásának a megbecsülésére bevett módszerek a *sketch* alapú algoritmusok [6]. A *sketch* alapú algoritmusok olyan módszerek, amelyek a bemeneti adatokból egy vagy több „sketchet” (vázlatot) hoznak létre. A létrejött *sketch*ek sokkal kisebb méretűek, mint az eredeti adatok, de még mindig megtartják azok lényeges tulajdonságait. A *sketch*ek az általam vizsgált algoritmusok esetében jellemzően véletlen mintavétellel kiválasztott, rögzített hosszú szövegdarabok (kmerek). A továbbiakban szeretnék kitérni néhány a bioinformatikában is széleskörűen alkalmazott *sketch* alapú algoritmusra, és adatszerkezetre.

2.1. Bloom filter

A Bloom filter egy hatékony és helytakarékos adatszerkezet, amely lehetővé teszi az elemek jelenlétének gyors ellenőrzését egy halmazban anélkül, hogy az elemeket ténylegesen tárolná. [7] A Bloom filter előnye, hogy csupán egy bitvektort használ az elemek jelenlétének nyilvántartására. Ez egy valószínűség alapú adatszerkezet. Amikor a felhasználó lekérdezi, hogy egy elem szerepel-e már az adatszerkezetben, kétféle választ kaphat: biztosan nem szerepel, vagy valószínűleg (kiszámítható konfidenciával) szerepel.

A Bloom filter működése során a következő lépéseket hajtja végre:

- **Inicializálás:** A Bloom filterben az összes információt egy biteket tartalmazó tömbben tároljuk. Kezdetben a tömb minden eleme 0.
- **Elem hozzáadása:** Ha egy elemet hozzá szeretnénk adni a Bloom filterhez, akkor azt k darab hashfüggvénnyel egész számokra képezzük. Ha m a biteket tartalmazó tömb hossza, $\mathbf{I}$ pedig a bemeneti értékek halmaza, akkor a hashfüggvényekre a következő megkötések érvényesek.:

Adottak a következő hash függvények:

$$h_1, h_2, \dots, h_k$$

1. $h_l(i) : \mathbf{I} \rightarrow \{0, 1, \dots, m-1\}$, $\forall l \in [1, k] \cap \mathbb{R}$ és $i \in \mathbf{I}$
2. $h_l(i)$ eloszlása egyenletes a $[0, m-1]$ intervallumon, $\forall l \in [1, k] \cap \mathbb{R}$ -re és $i \in \mathbf{I}$.
3. $h_l(i)$ és $h_j(i)$ függetlenek, ha $l, j \in [1, k] \cap \mathbb{R}$ és $j \neq l$ és $i \in \mathbf{I}$.

Miután a fent definiált hash függvényekkel leképeztünk egy elemet, az alábbi számsorozatot kapjuk: $y_1, y_2, \dots, y_k$; ahol: $h_l(i) = y_l, l \in [1, k]$ és $i \in \mathbf{I}$

Az elem eltárolása úgy történik, hogy minden a hashfüggvények által kijelölt y_l pozícióban 1-re állítjuk a bittömb értékét, az eredeti értéktől függetlenül.

- **Elem jelenlétének ellenőrzése:** Amikor ellenőrizni szeretnénk, hogy egy elem benne van-e a Bloom filterben, az előző pontban leírt módszerrel, ugyanazon hash-függvények segítségével leképezzük az elemet a bitvektor pozícióira. Ha minden hash-függvény által mutatott y_l pozíció 1 értékű a bitvektorban, akkor feltételezzük, hogy az elemmel már találkoztunk. Azonban ha bármelyik pozíció 0, akkor az elem biztosan nem szerepelt még.

A Bloom filter megbízhatósága nagyban függ három paramétertől.

- A hash függvények számától (k)

- A bittömb hosszától (m)
- A bemenetek számától (n)

Adott n mellett azonban megfelelő m és k választásával bármilyen $(1 - \alpha)$, $\alpha > 0$ konfidencia elérhető. [8]

2.2. Count-Min Sketch

A Count-Min Sketch (CMS) egy adatszerkezet, amely lehetővé teszi az elemek frekvenciájának gyors és hatékony becslését egy adathalmazon. Lényegében a Bloom filter módosított változata, amely képes az adatok frekvenciáját is tárolni, bittömb helyett egy egész számokat tartalmazó tömbben, melyet szintén 0-kal inicializálunk.

Ahelyett, hogy a Bloom filternél alkalmazott módszer szerint, egy elem hozzáadásánál minden, a hashfüggvények által kijelölt pozíció értékét egyre állítanánk, ebben a verzióban a pozíciókon található értékeket inkrementáljuk.

Egy elem lekérdezése során a bittömbben lévő azon pozíciókon elhelyezkedő értékeknek a minimumát adjuk vissza, melyeket az adott elem képeként kapunk a különböző hashfüggvények által.

Azaz amennyiben az elem képe: $y_1, y_2, \dots, y_k$ és a bittömb y_l pozícióján az x_l érték szerepel, a lekérés eredménye $\min_{l \in [1, k]} x_l$. Mivel az egy adott elemhez tartozó pozíciókat az elem minden előfordulásakor inkrementáljuk, azonban lehetséges, hogy más elem előfordulásakor is, a count min sketch az adatok frekvenciáját felülről becsüli. [9]

2.3. MinHash

A MinHash algoritmus hatékonyan tudja megbecsülni két halmaz hasonlóságának a mértékét. Ebben a kontextusban a halmazok közötti hasonlóságot a Jaccard koefficienssel írjuk le, mely egy A és egy B halmazra a következőképpen számítható:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

Az algoritmus működése azon alapul, hogy minden halmaz részére létrehozunk egy, a halmazt megfelelően reprezentáló, fix hosszúságú lenyomatot (*signature*). A lenyomat konstans hossza biztosítja az algoritmus jó időkomplexitását. A hasonlóság becslésekor a Jaccard koefficiens értékét nem számítjuk ki a teljes halmazokra, csak a lenyomataikra.

A MinHash algoritmusnak többféle gyakorlati megvalósítása is létezik, ebből kettőt szeretnék bemutatni.

A lenyomat generálása

Legyen a lenyomat hossza k . A lenyomat előállításának első lépéseként a vizsgált halmaz minden elemére előállítunk egy hash-értéket egy választott hash-függvény segítségével. A lenyomatot a k darab legkisebb hashérték fogja képezni minden szekvencia esetén. Feltételezhetjük, hogy a lenyomatok metszetének és uniójának számosságával becsülhető a halmazok metszetének és uniójának számossága, valamint ezek felhasználásával a Jaccard koefficiens is.

A minhash algoritmus másik verziójában ahelyett, hogy a lenyomatot egy hash-függvény k darab minimuma adná, inkább k darab hash függvény minimumát használjuk. Fontos kritérium, hogy minden halmazra ugyanazokat a hash függvényeket alkalmazzuk, valamint a különböző hash-függvények által adott minimumokat minden esetben ugyanolyan sorrendben tároljuk. Ekkor az n -edik vizsgált halmaz lenyomata a következőképpen áll elő:

$$h_{n,1}, h_{n,2}, h_{n,2}, \dots, h_{n,k} \quad (2)$$

Ahol $h_{n,l}$ az n -edik halmazon az l -edik hash függvény általi leképezések minimuma.

A lenyomatok összehasonlítása

Mindkét lenyomat generálási módszer esetén a Jaccard koefficiens becsült értéke lesz a halmazok becsült hasonlósága. Amennyiben a lenyomat egy hash függvény az adott halmazon generált k darab legkisebb értékéből áll, a lenyomatok tekinthetők halmazoknak, és ezek metszete és uniója a szokásos módon számítható.

Abban az esetben, ha a lenyomat k darab hash függvényből áll, figyelembe kell azt is venni, hogy a lenyomat melyik eleme melyik hash függvényhez tartozik. Ezért a metszet számosságát az adja, hogy hány pozíción egyezik a két lenyomat.

Mindkét módszer fontos tulajdonsága, hogy amennyiben a halmazok egyenlőek, a Jaccard koefficiens becsült értéke biztosan 1 lesz, ugyanis az elemek hash-értékei meg fognak egyezni, ezért a minimumaik is azonosak lesznek. Dogozatom során a minhash algoritmus azon verzióját használom, amelyben a lenyomatot egy hashfüggvény k darab minimuma képezi.

A MinHash algoritmus alkalmazása szövegek hasonlóságának becslésére

A MinHash algoritmust széles körben alkalmazzák nem csak halmazok, hanem más adatok hasonlóságának megbecsülésére is. Talán a legelterjedtebb alkalmazása a szövegek összehasonlítása, vagy a plágiumkeresés. Elterjedt bioinformatikai felhasználása örökítőanyag szekvenciák klaszterezése. [10]

A szövegek hasonlóságának elemzése megvalósítható a fent leírt módszerekkel amennyiben sikerül halmazként értelmezni a szövegeket. Egy természetes nyelv szövegezetét fel foghatjuk szavak halmazaként. Fontos kiemelni, hogy ezzel a megközelítéssel veszítünk a szöveg információtartalmából: nem vesszük figyelembe a szavak sorrendjét, és azt sem, hogy egy szó hányszor fordul elő. Annak a függvényében, hogy mit szeretnénk megállapítani a szövegekről, ez a tulajdonság lehet hasznos, de okozhat problémát is.

A DNS vagy RNS szekvenciák nem tartalmazzak szavakat, ezért mesterségesen kell felosztani őket. Erre a bevett módszer (*shingling*) a k hosszúságú, minden lehetséges pozíciót lefedő csúszóablakok használata. [10] Az így keletkező szövegrészleteket „kmer”-eknek nevezzük. Például az „AGTAG” szövegből három hosszú csúszó ablak felhasználásával az alábbi kmerek keletkeznek: „AGT”, „GTA”, „TAG”. Az itt használt „k” nem egyezik azzal, melyet a lenyomat hosszának jelölésére használnak az irodalomban. A későbbi fejezetekben külön jelölést fogok ezekre bevezetni.

2.4. HyperLogLog

A HyperLogLog algoritmus egy adathalmaz kardinalitását (az egymástól különböző elemek száma a halmazban) képes becsülni. Célja annak a kikerülése, hogy minden elem vizsgálatakor végig kelljen iterálni a koriábbiakban már előforduló elemeken azt keresve, hogy az aktuális elem előfordult-e már.

A HyperLogLog a kardinalitás megbecsülésére a következő koncepciót használja fel: Amennyiben valaki pénzérmét dobál fel, és számolja, hogy hány egymás után fejként leérkezett pénzérme a rekordja, csak a rekord ismeretében becsülhetjük, hogy hányszor dobta már fel a pénzérmét.

A hyperloglog algoritmus esetén az adathalmaz minden elemét hasheljük. A fenti gondolatmenethez nagyon hasonló elven rögzítjük, hogy a hashértékek bináris reprezentációiban mennyi volt az egymást követő 0-k számának maximuma. Ezután ebből az egy

számból becsüljük a halmaz kardinalitását.

Azonban előfordulhat, hogy szerencsétlen esetben kevés mintavétellel is nagy maximumot kapunk. Az ilyen helyzeteket a HyperLogLog algoritmus úgy kompenzálja, hogy a hashelt értékeket csoportokra osztja az első néhány bit értéke alapján és csoportonként rögzíti a maximumértékeket. Az adathalmaz hashelése után a végső becslés a csoportbeli maximumértékek harmonikus átlaga alapján történik. [11]

A HyperLogLog algoritmus alkalmazása szekvenciák hasonlóságának becslésére

Amennyiben szekvenciákat szeretnénk páronként összehasonlítani és ezek hasonlóságát mérni, használhatjuk a HyperLogLog algoritmust arra, hogy megbecsüljük a szekvenciák kardinalitását külön-külön, valamint az uniójuk kardinalitását. Ezután a metszet becsült kardinalitása megkapható a fent becsült adatokból, továbbá becsülhető a Jaccard koefficiens is. Hasonlóan az eddig bemutatott algoritmusokhoz, a becsült halmazok elemeit itt is a szekvenciákból kinyert kmerek adják. [12]

3. A problémára alkotott megoldásom

A dolgozatom írása során a szakirodalomban talált megoldások felhasználásával implementáltam egy olyan szoftvert, mely képes megállapítani, hogy egy pángenom gráfban van-e olyan részgráf-beli Hamilton út, melynek valamely része hasonlóságot mutat egy bemeneti szekvenciával, valamint azt is megállapítja, hogy a hasonló szakasz a gráf melyik részén található.

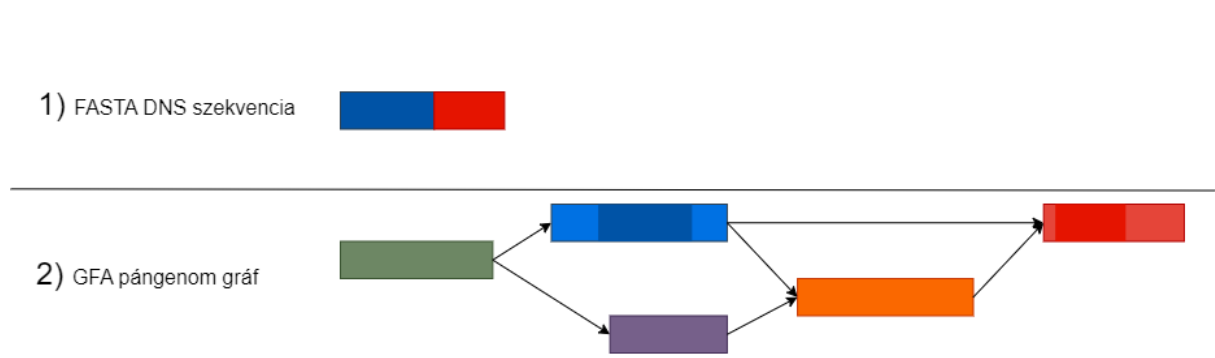
A megoldásom a MinHash algoritmus koncepciójára épül, azonban különböző elméleti, valamint technikai nehézségek miatt az algoritmus több lépését is módosítottam.

3.1. Az input

A program egy szomszédsági listával reprezentált pángenom gráffal, valamint egy vagy több keresett szekvenciával dolgozik. A várt bemenetekre a 2. ábra ad egy nem általánosított példát.

3.2. Az output

A program elvárt viselkedése az, hogy megtalálja minden keresett szekvenciához gráf olyan csúcsait, melyek hasonlítanak az adott szekvenciára, valamint szekvenciánként az egymáshoz közeli csúcsokból a csúcsok összevonásával utakat épít. Az elvárt kimenet a megtalált csúcsok, valamint az épített útvonalak felsorolása szekvenciánként.



2. ábra. Példa a program bemenetére

3.3. Az algoritmus bemutatása

A megoldásom tervezése során a kód hatékonysága nagyon fontos szempont volt. Jelenleg az algoritmus pythonban van implementálva, és nincs optimalizálva. Szem előtt tartottam azonban, hogy ne használjak olyan eszközöket, melyek nem hatékonyak, valamint nem helyettesíthetők más, jobban teljesítő megoldással.

Munkám során igyekeztem azt is szem előtt tartani, hogy olyan megoldást találjak, amely megadott feltételek mellett egyértelműen felismeri a gráfban a szekvenciát, amennyiben a gráf pontosan tartalmazza azt. Az egzakt szekvencia megkeresése a gráfban ugyanis egy sokkal kisebb komplexitású feladat.

A bemeneti pángenom gráfok nagy mérete miatt olyan megoldást kerestem, amely lehetővé teszi, hogy a gráfot ne kelljen egy egységként kezelni. Azzal a megközelítéssel indultam neki a megoldás keresésének, hogy találni szeretnék egy algoritmust, amely konstans méretű információ eltárolásával jellemzi a gráf minden csúcsát olyan módon, hogy ebből az információból később a csúcsok kapcsolatainak ismeretében eldönthető, hogy a gráf mely útvonalai tartalmazzak a keresett szekvenciához hasonló szakaszt. Mivel a GFA formátum először a csúcsokat, majd az éleket írja le, egy ilyen megoldás segítségével a gráf egy olvasással kiértékelhető lenne.

Megoldás egy egyszerűbb problémára

Első lépésben arra az egyszerűsített feladatra kerestem megoldást, amikor a keresett szekvencia csak egy csúcsnak része, azon nem nyúlik túl. Ebben az esetben az élek által leírt információra nincs is szükségem, elég a csúcsokat egymástól függetlenül összehasonlítani a keresett szekvenciával. Az egzakt MinHash algoritmus azonban ebben az esetben sem feltétlenül működik megfelelően. Amennyiben a csúcs, amely tartalmazza a keresett szekvenciát lényegesen hosszabb, mint maga a szekvencia, nem fogunk megbízható becslést kapni.

Tegyük fel, hogy a vizsgált csúcs tartalmazza a pontos szekvenciát. Ilyen esetben az lenne az algoritmus elvárt viselkedése, hogy olyan értékelést ad a csúcsra, amelyből egyértelműen kiderül a hasonlóság. A hagyományos MinHash algoritmus a Jaccard koeficiens értékét becsüli, amely esetünkben $\frac{n}{m}$ lesz, ahol m a csúcs, és n a keresett szekvencia hossza. Ez a hipotetikus eset azonban jelen helyzetben pontosan ugyanolyan jó találat, mintha pontosan egyezne a csúcshoz tartozó, és a keresett szekvencia (ebben az esetben a

Jaccard koefficiens értéke 1, mely jelentősen különbözhet az $\frac{n}{m}$ tört értékétől). A példából következően tehát olyan megoldás lenne szükséges, amelyben a kiértékelés független a két szekvencia hosszától.

A fent említett problémára egy jó megoldás az, ha a lenyomatot csak a keresett szekvenciához számolom ki, majd azt vizsgálom, hogy a vizsgált csúcsból nyert hash-értékek között mennyi jelenik meg a lenyomat elemeiből. Ez a megoldás már ismert módszer a bioinformatikában különböző méretű szekvenciák összehasonlítására Containment Min-Hash néven [13]. A módszer garantálja, hogy amennyiben a csúcs tökéletesen tartalmazza a keresett szekvenciát, a kimeneten ez megjelenjen, mivel a kmerok, amelyek hashértékei bekerültek a keresett szekvencia lenyomatába, biztosan jelen vannak a csúcsához tartozó szekvenciában is. Ezért ebben az esetben, ha a lenyomat hossza k , a csúcsban mind a k darab hash-érték jelen lesz.

A konkrét algoritmus a fent definiált eset kezelésére következő lépésekből áll:

- A keresett szekvenciához lenyomat generálása,
- A gráf csúcsainak beolvasása egyesével,
- Az éppen aktuális csúcson csúszóablakos módszerrel minden kmer hashelése, valamint a hash-érték keresése a keresett szekvencia lenyomatában,
- A megtalált hash-értékek számának összegzése, és a csúcs értékelésének megjegyzése.

Megoldás a tényleges problémára

A fent ismertetett megoldás még nem működik arra az esetre, amikor a keresett szekvencia több csúcs konkatenációjának részszovegeként található meg. Egy ilyen esetet szemléltet a 2. ábra is.

A konkatenáció elemeit adó csúcsok kiértékelésekor várhatóan azt fogjuk látni, hogy tartalmaznak a keresett szekvencia lenyomatával közös hash-értékeket, azonban lehet, hogy keveset ahhoz, hogy biztos döntést lehessen hozni. Amennyiben a gráf nagyon fragmentált, és sok apró csúcsot tartalmaz, az egyre nagyobb valószínűséggel fordul elő, hogy még pontos egyezés esetén is találunk több olyan csúcsot, mely része a keresett szekvenciának, azonban nem tartalmaz a keresett szekvencia lenyomatával közös hash-értéket.

A problémás esetek alapvetően négy részre oszthatóak:

1. A vizsgált csúcs lényegesen kisebb, mint a keresett szekvencia, valamint a keresett szekvencia tartalmazza a csúcsot (vagy egy ahhoz hasonló szekvenciát).

2. A vizsgált csúcs nem lényegesen kisebb, mint a szekvencia, azonban csak a szekvencia egy részét tartalmazza.
3. A vizsgált csúcs hasonló méretű, mint a keresett szekvencia, és hasonlít a keresett szekvenciára
4. A vizsgált csúcs lényegesen nagyobb, mint a keresett szekvencia, és tartalmaz a keresett szekvenciához hasonló szakaszt

A felsorolt négy esetből a 3. és 4. esetre megoldást kínálnak a MinHash valamint a Containment MinHash algoritmusok.

Arra a feltételezésre jutottam, hogy azok az esetek, melyeket a fenti négy eset nem fed le, a megoldás szempontjából elhanyagolhatóak, vagy utólag javíthatóak. Ugyanis a kimaradt eset az, hogy a vizsgált csúcs nem nagyobb, mint a vizsgált szekvencia, és ennek csak egy részéhez hasonló. Ez akkor fordulna elő, ha a szekvencia pont egy rövidebb csúcs (vizsgált csúcs) elején végződik, vagy a végén kezdődik. Ebben az esetben azonban feltételezem, hogy a keresett szekvencia jelentős része korábbi, vagy későbbi csúcsok által le lesz fedve.

Amennyiben minden csúcsot tudnék kategorizálni a fenti csoportokba, vagy egy ötödik csoportba, amely ötödik csoport azokat a csúcsokat tartalmazza, amik nem hasonlítanak a keresett szekvenciához, az alábbi kiértékeléssel rekonstruálható lenne a gráf érintett útszakasza:

A **2. csoportba** tartozó csúcsoknak megvizsgálom az elejét, és a végét. Amennyiben csak a keresett szekvenciához hasonló szakasz egy részét tartalmazzák, az őket megelőző, vagy követő csúcsoknak tartalmazniuk kell azt. Ha ez a feltétel nem teljesül, elvetem a csúcsot.

Az **1. csoportba** tartozó csúcsok önmagukban szintén nem elég hasonlóak a keresett szekvenciához, ezért csak akkor fogadom el őket, ha van olyan szomszédjuk, mely szintén legalább részben hasonlóságot mutat a keresett szekvenciával.

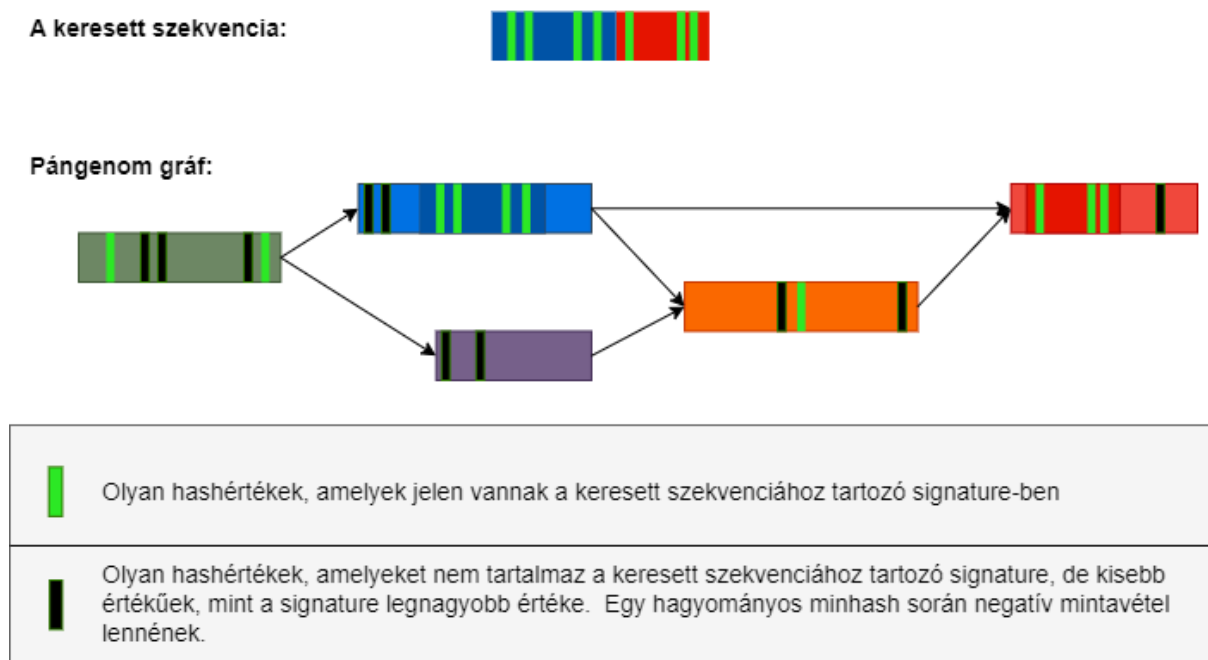
A **3. és 4. csoportba** tartozó szekvenciák akár önmagukban is jó output lehetnek, de az is előfordulhat, hogy egy szomszédjukkal összevonva adják ki a teljes szekvenciát. A **4. csoportba** tartozó szekvenciáknál ezért érdemes megnézni, hogy a keresett szekvenciához hasonló rész közel esik-e a csúcs valamelyik végéhez.

A fenti csoportosítás alapján levonható következtetés, hogy minden csúcsról elég lehet három információt eltárolni:

1. A csúcs bal széle (a keresett szekvenciával arányosan hosszú rész) hasonlít-e annyira a keresett szekvenciára, hogy amennyiben a csúcsnak van olyan megelőző szomszédja, melynek a jobb széle szintén hasonló a keresett szekvenciához, megfelelő konfidenciával lehessen megállapítani a hasonlóságot
2. A csúcs jobb széle (a keresett szekvenciával arányosan hosszú rész) hasonlít-e annyira a keresett szekvenciára, hogy amennyiben a csúcsnak van olyan következő szomszédja, melynek a bal széle szintén hasonló a keresett szekvenciához, megfelelő konfidenciával lehessen megállapítani a hasonlóságot
3. A csúcs egésze, vagy része önmagában megfelelő mértékben hasonlít-e a keresett szekvenciára

A probléma megoldására bevezettem a lenyomatban is jelen levő kmerek számán túl még egy, a csúcsok jellemzésére szolgáló feltételt. Amikor a keresett szekvenciára hasonlító szakasz több csúcs konkatenációjában van jelen, minél több csúcsot kell konkatenálni, annál kevesebb egyezés lesz az egy csúcsban szereplő hash-értékek, és a keresett szekvencia lenyomata között. Ezért fontos lenne olyan esetekben, ahol egy csúcsban csak néhány hashérték van jelen is pozitív döntést hozni. Erre megoldás lehetne, hogy ilyen helyzetben vizsgáljuk, hogy a csúcs lenyomata szerepel-e a szekvenciából nyert hashértékek között, így azonban minden csúcs esetében végig kéne iterálni a keresett szekvencián is, és a dolgozat célja éppen ennek az elkerülése. Azonban vannak olyan értékek a csúcsok hash-értékei között, amelyek biztosan nem jelentek meg a keresett szekvenciában. Ezek azok az értékek, melyek kisebbek, mint a keresett szekvencia lenyomatának legnagyobb értéke, de mégsem szerepelnek a lenyomatban.

Az ilyen mintákra innentől kezdve „*negatív minta*”-ként, a keresett szekvencia lenyomatában szereplőkre pedig „*pozitív minta*”-ként fogok hivatkozni. A 3. ábra szemlélteti a pozitív és negatív minták koncepcióját a 2. gráfon.



3. ábra. Pozitív és negatív minták egy pángenom gráfban

Feltételezésem szerint meghatározható úgy a kmerek és a lenyomat hossza, hogy amennyiben egy szekvenciában akár csak kevés pozitív minta is szerepel, és emellett ezek környezetében nem szerepel negatív minta, a szekvencia adott része megfelelő valószínűséggel ténylegesen hasonló legyen a keresett szekvencia legalább egy részéhez.

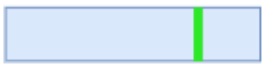
Ehhez azonban elengedhetetlen feltétel az, hogy a pángenom gráf ne legyen túl fragmentált. Az elvárt minimális csúchosszra később feltételt adok.

Az algoritmus következő lépése a kapott minták alapján a csúcsok pontozása. Egy csúcs kiértékelésénél a következő szempontokat szeretném figyelembe venni:

- Amennyiben egy rövidebb szakaszon több „pozitív minta” is van, a csúcs kapjon magas pontszámot
- Amennyiben egy rövidebb szakaszon csak kevés (egy vagy két) „pozitív minta” található, de nincsen „negatív minta”, az adott szakasz kapjon 0 fölötti pontszámot.
- Amennyiben egy szakaszon csak kevés (egy vagy két) „pozitív minta” található, és található benne „negatív minta”, a szakasz kapjon negatív pontszámot.

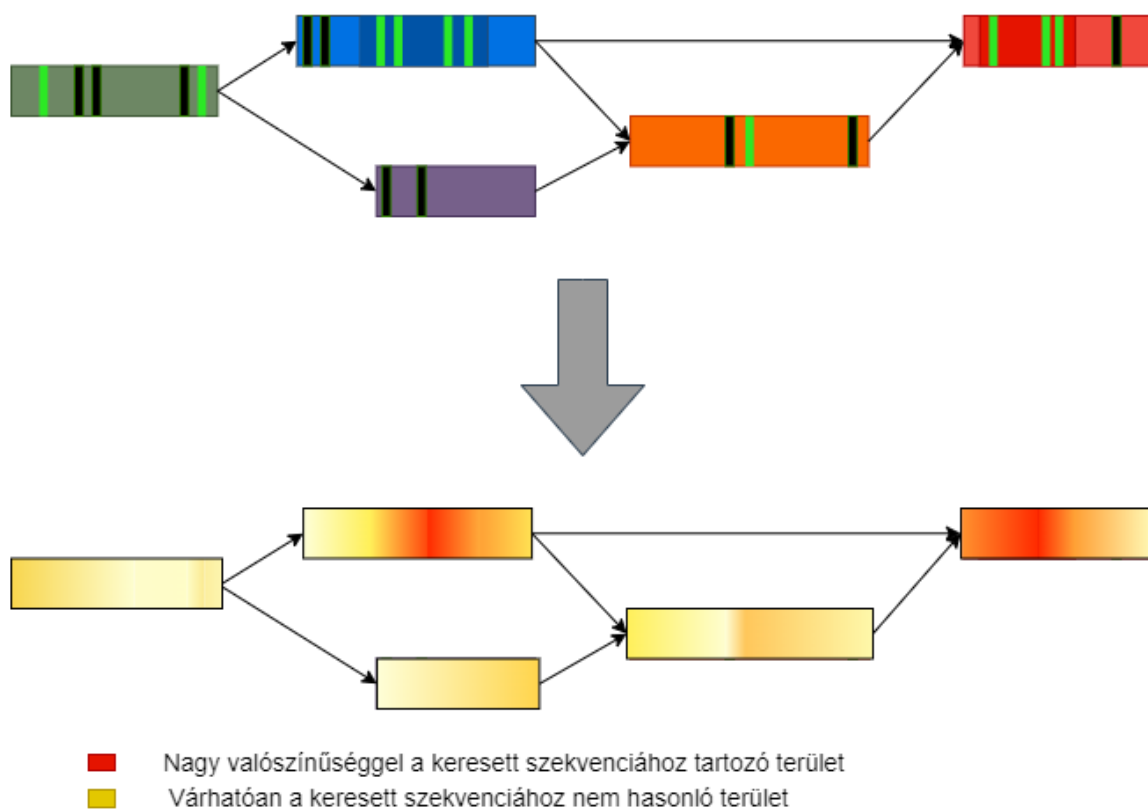
A módszer lényege az, hogy az olyan szakaszokról, melyekben csak kevés hash érték egyezik meg a keresett szekvencia lenyomatában szereplőkkel, nagyobb konfidenciával lehessen eldönteni, hogy az adott minta csak véletlenül fordul elő, vagy ténylegesen a

szekvenciák hasonlósága miatt van jelen. Néhány szekvencia kiértékelését a 4. ábrán szemléltettem.

Feltehetően hasonló szakaszok	Feltehetően nem hasonló szakaszok
	
	
	
	
 Olyan hashértékek, amelyek jelen vannak a keresett szekvenciához tartozó signature-ben	
 Olyan hashértékek, amelyeket nem tartalmaz a keresett szekvenciához tartozó signature, de kisebb értékűek, mint a signature legnagyobb értéke. Egy hagyományos minhash során negatív mintavétel lennének.	

4. ábra. Szekvenciák értékelése a tartalmazott minták alapján

Az egy adott környezetben lévő minták kiértékelésével azt szeretném elérni, hogy a gráf minden pontjára legyen egy lokális információ a pont környezetének a vizsgált szekvenciával való hasonlóságáról. A koncepciót a 2. ábrán szereplő gráf kiértékelésével az 5. ábrán szeretném szemléltetni.



5. ábra. Egy pán genom gráf hasonlóságának kiértékelése egy adott szekvenciával

A gráf csúcsainak kiértékelése A csúcsok értékelésének konkrét algoritmusát a következőképpen terveztem meg:

1. Minden csúcs végigolvasásánál elmentem a talált pozitív és negatív minták pozícióit a csúcson belül.
2. Egy előre definiált méretű csúszóablakot végigtolva a szekvencián, a csúszóablak minden lehetséges pozíciójára adok egy értékelést.
3. Az a két pontszám, amelyet a csúszóablak két szélső pozíciója kapott lesz a visszatérített értékelés a csúcs megfelelő széleire.
4. A legmagasabb pontszámot szintén eltárolom

A csúszóablakba tartozó szakaszok pontszámait egyelőre a minták számának összegzésével veszem, olyan módon, hogy az összegben a pozitív minták pozitív előjellel, a negatív minták pedig negatív előjellel szerepelnek. A későbbiekben érdemes lehet majd más súlyozást

is kipróbálni.

A csúcsok kiértékeléséhez nem kell minden pozíción végigtolni a csúszóablakot, valamint a jelenlegi pontozási módszerrel az sem szükséges, hogy a csúszóablakot végig kelljen olvasni minden pozícióban, amennyiben a pontszámot abból számítom, hogy a csúszóablak arrébtolásával mennyivel változott a csúszóablak tartalma. A mintákat egy listában tárolom, a pozíciójuk szerint sorban. A lista minden helyén egy minta pozíciója áll (a kmer első nukleotidjának pozíciója a csúcsban), valamint az, hogy az adott minta pozitív, vagy negatív minta-e.

Egy n hosszú lista használatával minden pozícióra kiszámítható a hozzátartozó pontszám összesen $O(n)$ lépésben. Ez a viselkedés elérhető úgy, hogy a mintákat tartalmazó listát bejárjuk olyan módon, hogy egy változó méretű intervallumot választunk ki belőle minden iterációban.

A konkrét algoritmust pszeudokódként írtam le. (1. kódrészlet). A pszeudokód egy listát (samples) kap bemenetként, és három értéket térít vissza: a csúcs bal szélének értékelése, a csúcson talált legmagasabb pontszám, a csúcs jobb szélének értékelése. A lista (pozíció, minta típusa) párokat tartalmaz, és elemei a pozíció értékek szerint növekvő sorba rendezettek. A pozíció értékek megfeleltethetők a minták első karakterének pozíciójának a csúcsban, a minta típusa pedig True, amennyiben a minta pozitív, és False amennyiben a minta negatív.

A pszeudokód két esetet külön kezel. Amennyiben a csúcshoz tartozó szekvencia nem hosszabb, mint a megadott csúszóablak (4. sor), mind a három kimenet a listában található minták előleges összegéből adódik (13. sor). Amennyiben a csúcshoz tartozó szekvencia hosszabb, mint a csúszóablak mérete, két változóval iterálok végig a listán, olyan módon, hogy a két változó által mutatott minták pozíciója mindig a lehető legnagyobb legyen, de kisebb, mint a csúszóablak mérete (34. sor). Először azonban megkeresem, hogy hol legyen a csúszóablak jobb széle kezdetben (24. sor). A csúszóablak első pozíciójára adott pontszámot elmentem, mint a csúcs jobb szélének pontszámát (32. sor). Ezután végigiterálok a fent leírt módon a listán, rögzítve a maximális pontszámot minden lépésben (50. sor). Mindig amikor nyílik, vagy csukódik az ablak, ellenőrzöm, hogy milyen irányba változik az aktuális pontszám (35. sor). A ciklus utolsó iterációjában az aktuális pontszámot elmentem a csúcs jobb szélére adott értékelésként (54. sor).

Algorithm 1: Egy csúcs mintáit tartalmazó lista pontszámainak számítása

```
1 Bemenet: Minták: lista, mely a minták pozícióit és típusát (pozitív vagy negatív) tartalmazza,
   a minták pozíciója szerint növekvő sorrendbe rendezve.
2 Kimenet: Bal_pontszám, Jobb_pontszám, Legmagasabb_pontszám: A csúcs bal- illetve jobb
   széléhez számított pontszám, valamint a teljes csúcsra számított legmagasabb pontszám
3 Algoritmus:
4 Ha a csúcs rövidebb mint a csúszóablak akkor
   | // A csúszóablak nem tud csúszni
5   Összeg = 0
   | // A minták előjeles összege
6   Ciklus minden mintára a Minták listában: :
7     | Ha a minta pozitív akkor
8     | | Összeg = Összeg + 1
9     | különben
10    | | Összeg = Összeg - 1
11    | vége
12  vége
13  Visszatérít Összeg, Összeg, Összeg
14 vége
15 Kezdő_pozíció = 0 // A csúszóablak bal széle a csúcson
16 Végződő_pozíció = 0 // A csúszóablak bal széle a csúcson
17 Jó = 0 // A jó minták száma a csúszóablakban az adott iterációban
18 Rossz = 0 // A rossz minták száma a csúszóablakban az adott iterációban
19 Ha az Minták[0] minta pozitív akkor
20 | Jó = Jó + 1
21 különben
22 | Rossz = Rossz + 1
23 vége
24 Ciklus amíg A Végződő_pozíció kisebb mint a minták száma és a Minták[Végződő_pozíció]
   minta pozíció értéke kisebb mint a csúszóablak hossza :
25 | Végződő_pozíció = Végződő_pozíció + 1
26 | Ha a Minták[Végződő_pozíció] minta pozitív akkor
27 | | Jó = Jó + 1
28 | különben
29 | | Rossz = Rossz + 1
30 | vége
31 vége
32 Bal_pontszám = Jó-Rossz
33 Legmagasabb_pontszám=Bal_pontszám
34 Ciklus amíg a Végződő_pozíció kisebb, mint a Minták lista hossza :
35 | Ha a Minták[Végződő_pozíció]-[Kezdő_pozíció] kisebb mint a csúszóablak hossza akkor
36 | | Ha a Minták[Végződő_pozíció+1] minta pozitív akkor
37 | | | Jó = Jó + 1
38 | | különben
39 | | | Rossz = Rossz + 1
40 | | vége
41 | | Végződő_pozíció += 1
42 | különben
43 | | Ha a Minták[Kezdő_pozíció] minta pozitív akkor
44 | | | Jó = Jó - 1
45 | | különben
46 | | | Rossz = Rossz - 1
47 | | vége
48 | | Kezdő_pozíció = Kezdő_pozíció + 1
49 | vége
50 | Ha Jó - Rossz érték nagyobb, mint a Legmagasabb_pontszám akkor
51 | | Legmagasabb_pontszám = Jó - Rossz
52 | vége
53 vége
54 Jobb_pontszám = Jó-Rossz
55 Visszatérít Bal_pontszám, Legmagasabb_pontszám, Jobb_pontszám
56 Algoritmus vége
```

Útvonal létrehozása a kiértékelte csúcsokból

Az útvonalak kiértékelését a következőképpen terveztem meg:

Amennyiben az útvonalhoz tartozó minden csúcsról eldönthető, hogy hasonlít a keresett szekvenciára, az útvonal rekonstruálható egy olyan mohó algoritmussal, amely minden iterációban összevonja egy csúccsá azokat a csúcsokat, amelyek a bennük lévő minták alapján megfelelően jó kiértékelést kaptak. Ilyenkor az új csúcs széleire az a kiértékelés fog vonatkozni, ami a szélén lévő csúcsok megfelelő széleire vonatkozik. Egy iterációban egy csúcsból több új csúcs is keletkezhet, amennyiben szintén ígéretes elődje, vagy utódja van. Nem kizárt, hogy a gráfban előfordulnak körök, ezért mindig amikor összefűzünk két csúcsot ellenőrizni kell, hogy nem szerepel-e egy csúcs kétszer az új csúcsban. Akkor vagyunk készen, amikor egy iterációban már nem keletkezett új csúcs. Kiszűrve azokat a csúcsokat, melyek az értékelés alapján nem hasonlítanak a keresett szekvenciához, vagy csak a szélük kapott jó pontszámot, de a megfelelő oldalon nem volt jól értékelt szomszédjuk, a megmaradó csúcsok már megtalált útvonalakat reprezentálnak.

A fent leírt koncepció több olyan feltételezést is tartalmaz, amelyek nem feltétlenül teljesülnek. Mivel még kevés valódi adaton tudtam tesztelni a kódomat, az útvonalak rekonstrukcióját egyelőre még nem implementáltam.

A bemeneti gráfokról és a kiértékelési módszeremről az alábbi kérdésekre kell még választ kapnom:

- Mi a gráf csúcsainak minimális hossza? Ahhoz, hogy akár tökéletes egyezés esetén magas konfidenciával jusson minta egy csúcsba, szükséges, hogy a csúcsok megfelelően hosszúak legyenek. Amennyiben ez nem teljesül, a helyes útvonalat töredezetten fogom megkapni.
- Mennyi hamis pozitív találatot kapok a csúcsokra? Problémát fog okozni, ha a megtalálása nem jó komplexitású, vagy jellemzően olyan kevés élel kell ellenőriznem, hogy ezek számát tekinthetem konstansnak?
- A pontozási módszerem sikeresen kiválasztja azokat a csúcsokat, amelyek hosszabbak, azonban csak a szekvencia egy (nagyobb) részét tartalmazzák?
- Létezik olyan eset, amikor nem a leghosszabb út a legjobb megoldás?

A jövőbeli feladatok között szerepel a bemeneti gráfok vizsgálata az eddig implementált módszerekkel. A fenti kérdések megválaszolásával felismerhetem, hogy mely kihívások relevánsak az útvonalkereső algoritmus implementálása szempontjából.

4. Az algoritmus elemzése és elméleti korlátai

4.1. Az algoritmus lépéseinek összefoglalása

Összességében az általam tervezett algoritmus a következő lépésekből áll:

1. A keresett szekvenciák lenyomatának kiszámítása. A lenyomat és a szekvencia hosszának elmentése
2. A gráf csúcsainak végigolvasása, minden csúcs kiértékelése egyesével
3. Az élek végigolvasása, és a csúcsok értékelése alapján utak rekonstrukciója

4.2. Az inputtal szemben alkotott feltételek

Amint a korábbi fejezetekben is bemutattam, a problémát nehéz, vagy nem is lehet általánosan megfogalmazni. A megoldásom sem általános, nem fed le minden szélsőséges esetet.

Az algoritmus pontossága erősen függ az alábbi feltételek teljesülésétől:

- A pángenom gráfnak, és a keresett szekvenciának megfelelően nagy méretűnek kell lennie. Hasonlóság keresés közben mintákat veszek a gráfból, és a keresett szekvenciából is. Hamis pozitív esetek akkor állhatnak elő, amikor két mintához tartozó hash-érték megegyezik, vagy amikor a minta olyan csúcsban is előfordul, amely nem hasonlít a keresett szekvenciához. Annak a valószínűsége, hogy egy minta „véletlenül” előfordul egy adott hosszúságú csúcsban exponenciálisan csökken a minta hosszának függvényében. Ezért fontos, hogy a keresett szekvencia legyen olyan hosszú, hogy megfelelő hosszúságú mintákat lehessen választani. Gyakorlati tapasztalataim alapján 25 hosszú kmerek már nagyon ritkán adnak hamis pozitív eredményt, akár több mint 10000 bázispár hosszúságú csúcsok és keresett szekvenciák esetén is. Ez nem meglepő, tekintve, hogy 4^{25} féle 25 hosszú szekvencia lehetséges, és a használt hashfüggvény eloszlása egyenletes.
- Az algoritmus hatékonyságát javítja az is, ha a keresett szekvencia, és a gráfbeli szekvencia között a különbséget okozó pontmutációk eloszlása nem egyenletes a szekvencián. Ha a pontmutációk egymáshoz közel történnek, sokkal több minta marad meg eredeti formájában.

- A gráf csúcsai hosszának el kell érnie egy megadott határértéket ahhoz, hogy megfelelő konfidenciával tartalmazzanak pozitív mintákat a csúcsok.

Amennyiben adott egy n_1 hosszú egy szekvencia, amelynek része egy n_2 hosszúságú rész-szekvencia, a szekvencia lenyomata h darab hashértékből áll, és a mintavételeket $k(k < (n_2))$ hosszúságú kmerek adják, kiszámítható annak a valószínűsége, hogy a rész-szekvencia ne tartalmazzon olyan kmert, amely hash-értékét tartalmazza az n_1 hosszúságú szekvencia lenyomata. Annak a valószínűségét, hogy a szekvencián belül egy kmer többször is előfordulhat, vagy több kmer hash-értéke megegyezhet (hash collision) számításomban elhanyagolom.

A rész-szekvencia elhelyezkedhet a szekvencia közepébe ékelődve, de elhelyezkedhet közvetlenül annak az elején, vagy a végén is. Amennyiben egy mintát választunk csak, a minta a következő féleképpen helyezkedhet el a szekvenciában olyan módon, hogy a rész-szekvencia ne tartalmazza azt teljes egészében:

- Amennyiben a szekvencia a rész-szekvenciával kezdődik, vagy az a közepére ékelődött be, a választott minta $n_2 - (k - 1)$ pozícióban tud kezdődni úgy, hogy benne legyen a rész-szekvenciában. A szekvencia maradék részében $k - 1$ pozíción nem kezdődhet a választott minta, mert ebben az esetben kilógna a szekvenciából. Ebből következően összesen $n_1 - (n_2 - (k - 1)) - (k - 1) = n_1 - n_2$ pozíción kezdődhet a választott minta úgy, hogy azt ne tartalmazza teljesen a rész-szekvencia.
- Ha a szekvencia a rész-szekvenciával végződik, akkor a választott minta szintén $n_2 - (k - 1)$ pozícióban tud kezdődni úgy, hogy benne legyen a rész-szekvenciában. Ebben az esetben azonban a szekvencia bármely kimaradt részéről tudunk mintát választani. Ezért összesen $n_1 - (n_2 - (k - 1)) = n_1 - n_2 + k - 1$ pozíción kezdődhet a választott minta úgy, hogy azt ne tartalmazza teljesen a rész-szekvencia.

Amennyiben úgy veszem, hogy minden esetben $n_1 - n_2 + k - 1$ féle pozíción kezdődhet a választott minta felülről fogom becsülni az esemény valószínűségét, ami optimális, mivel ez az algoritmus szempontjából egy kedvezőtlen esemény.

Összesen n_1 pozíción kezdődhet a kiválasztott minta. Ebből következően annak a valószínűsége, az n_2 hosszúságú rész-szekvencia nem tartalmazza a mintát felülről becsülhető a következőképpen: $\frac{n_1 - n_2 + k - 1}{n_1}$ Ahol k hosszúságú mintát veszünk egy n_1

hosszú szekvenciából, valamint az n_1 hosszú szekvencia teljesen tartalmazza az n_2 hosszúságú szekvenciát.

Továbbá felülről becsülhetjük annak a valószínűségét is, hogy a rész-szekvencia nem tartalmaz h darab mintát a következő képlettel: $(\frac{n_1-n_2+k-1}{n_1})^h$. A képlet nem veszi figyelembe, hogy a valódi lenyomat képzés során egy minta csak egyszer szerepel. Ezzel szintén felülről becsülöm a tényleges valószínűséget, ami számomra megfelelő. A fentiek alapján annak a valószínűsége, hogy h darab mintavétel után a rész-szekvencia tartalmaz legalább egy mintát: $1-(\frac{n_1-n_2+k-1}{n_1})^h$. Amennyiben adott egy α konfidencia, a keresett szekvencia mérete valamint a keresett szekvenciáról készült lenyomat hossza, a megadott konfidenciához tartozó minimális csúchossz a következőképpen számítható:

$$n_2 \geq k + n_1 - 1 - n_1 * \sqrt[h]{1 - \alpha} \quad (3)$$

Természetesen elméleti szempontból lehetséges a lenyomat hosszát a többi paraméterhez igazítani, azonban ez nem javasolt, ugyanis az algoritmus jó idő-, valamint helykomplexitása a lenyomatok konstans hosszán alapul.

4.3. Az algoritmus komplexitásának megbecsülése

Az algoritmus komplexitásának a megbecsülésére az alábbi jelöléseket szeretném rögzíteni:

- A leghosszabb csúcs hossza: $\mathbf{n}$
- A csúcsok száma: $\mathbf{N}$
- A leghosszabb keresett szekvencia hossza: $\mathbf{s}$
- A keresett szekvenciák száma: $\mathbf{S}$
- Egy lenyomat hossza: $\mathbf{h}$
- Egy kmer hossza: $\mathbf{k}$

Időkomplexitás

1. Egy szekvencia lenyomatának kiszámítása:

Egy szekvencia $s-k+1$ kmert tartalmaz. Minden kmer-re ki kell számolni egy hash értéket. Megfelelő hash-függvény választásával ez elvégezhető konstans c_h lépésben. Ezen túl minden hash-értékre ellenőrizni kell, hogy benne van-e az eddigi legkisebb

h darab értékben. Ez egy piros-fekete fát használva elvégezhető kmerenként $O(\log h)$ lépésben olyan módon, hogy az aktuálisan vizsgált hash-értéket beszúrjuk a piros-fekete fába, majd kivesszük a piros-fekete fa maximális elemét. A piros-fekete fa kiegyensúlyozottsága és keresőfa tulajdonsága miatt az elem beszúrása, maximális elem keresése valamint ennek a törlése is megvalósítható egyesével $O(\log h)$ lépésben. A teljes lépés komplexitása tehát: $(s - k + 1) * (c_h + \log h) = O(s)$, mivel k, c_h és h konstansok.

2. Egy csúcs kiértékelésének a kiszámítása:

Egy csúcs kiértékelésénél $n-k+1$ kmeren kell végigiterálni. Minden kmer-re meg kell nézni, hogy szerepel-e valamelyik szekvencia lenyomatában. Ennek a komplexitása becsülhető, mint $S * h = O(S)$. Azt is vizsgálom, hogy amennyiben a kmer nem szerepel egy lenyomatban, kisebb-e, mint a lenyomat legnagyobb eleme. Ez a lépés konstans időben megvalósítható, amennyiben az előző lépésben minden lenyomatnak elmentjük a maximumát. Az eredményeket listában rögzítem, mely $O(n)$ hosszúságú lesz. A lista kiértékelhető $O(n)$ lépésben és S darab lista kiértékelésére van szükség. A teljes lépés komplexitása tehát: $(n-k+1) * S + n * S = O(n * S + n * S) = O(n * S)$

3. A csúcsok összefűzése:

Ennek a lépésnek a legkevésbé becsülhető a komplexitása, mivel nagyban függ az előző lépésektől valamint a bemeneti gráftól. Legrosszabb esetben a bemeneti gráf egy teljes gráf melynek minden csúcsa mindkét szélére megfelelő értékelést kapott. Ekkor a csúcsok minden lehetséges kombinációját végig kéne nézni. Ez a bemenet azonban a valóságban nem áll elő. A gráf sem lehet teljes gráf, várhatóan alacsony fokszámú csúcsokat tartalmaz, valamint a kijelölt csúcsok számára is azt várom, hogy N -hez képest elhanyagolhatóan kicsi, konstans. Amennyiben teljesül a gráfra, hogy a 3. egyenletnek megfelelően hosszú csúcsai vannak, a paraméterek függvényében az egy szekvenciához talált csúcsok száma állandó. Legyen az $\alpha=0.9$, $k=25$, $h=30$. Ezeket az értékeket az egyenletbe helyettesítve a következő egyenlőtlenséget kapjuk a csúcsok minimális hosszára, és a keresett szekvencia méretére:

$$n_2 \geq 25 + n_1 - 1 - 0,93 * n_1 \quad (4)$$

Amennyiben k mérete n_1 méretéhez képest elhanyagolható, a fenti egyenlőtlenséget

közelítem az alábbi módon:

$$n_2 \geq n_1 - 0,93 * n_1 \quad (5)$$

Amiből következik, hogy:

$$13,5 \geq \frac{n_1}{n_2} \quad (6)$$

Ebben az esetben tehát az egy útvonalat kitevő csúcsok méretének konstansnak kell lennie. Ez praktikusán azt jelenti, hogy érdemes a gráf összeállításánál annak a függvényében, hogy mekkora méretű információt szeretnénk utólag kinyerni belőle kell állítani a csúcsok fragmentáltságát. Szerintem ez az elvárás reális bármilyen keresőalgorithmus számára. Fontos szempont lehet pángenom rendszerek összeállításakor, hogy az összeállított rendszer célja genom rendszerek, genomok, vagy rövidebb szekvenciák reprezentációja lesz-e.

Amennyiben teljesül, hogy a keresett útvonal konstans számú csúcs komkatenációjaként áll elő, az útvonal megkeresésének időkomplexitása S , mivel a keresést minden keresett szekvenciára el kell végezni.

Összesítve, amennyiben teljesül, hogy az utolsó lépésben az összevonandó csúcsok száma független a bemeneti adatok méretétől, az algoritmus teljes időkomplexitása a következőképpen alakul:

$$O(S * s + N * S * n + S) = O(S * (s + N * n + 1)) \quad (7)$$

A programot kezdetben kevés keresett szekvencia feldolgozására kezdtem megtervezni, mivel a kutatócsoport jelenlegi igényeinek ez megfelelő.

Abban az esetben, ha a keresett szekvenciák száma is konstans, az algoritmus komplexitása a következő:

$$O(s + N * n) \quad (8)$$

Azaz az algoritmus által végzett lépésszám a keresett szekvencia, valamint a gráf teljes hosszának függvényében is lineáris, ami a probléma komplexitását tekintve nagyon jó eredmény.

Helykomplexitás

Az algoritmus nem tárolja egyszerre az egész gráfot, egyszerre egy csúcs beolvasására van szüksége. A csúcsokról tárolt információ konstans, ezért a helykomplexitás csak a csúcsok, valamint a keresett szekvenciák számától függ, valamint abban a lépésben, amikor csúcsokat értékelem ki, az aktuális csúcs méretétől lineárisan.

5. Gyakorlati eredmények

A kód a [14] gitlab repository-ban található.

Az input specifikációja

A szoftver egy DNS pángenom gráffal, valamint egy vagy több, a gráfban megkeresendő DNS szekvenciával dolgozik.

A GFA (Graphical Fragment Assembly) formátum

A gráfot GFA[15] fájlként várom. A gfa formátumú fájlban először fel van sorolva az összes csúcs. A csúcsokat leíró egységek három mezőből állnak, az 1 számú táblázatnak megfelelően.

Mező sorszáma	Mező neve	Mező értéke
1	RecordType	Karakter, csúcs esetén "S"
2	Name	A szekvenciához tartozó ID
3	Sequence	A DNS szekvencia

1. táblázat. A GFA formátum csúcsokat leíró sorainak mezői

A csúcsok leírása után következnek a „linkek” (élek). Egy él leírása a 2 számú táblázatban leírt mezőket tartalmazza.

A FASTA formátum A szoftver a keresendő szekvenciákat FASTA formátumban várja. Egy futás alatt több szekvenciát is össze tud hasonlítani a gráffal. A következő fejezetekben azonban a keresendő szekvenciára egyes számban fogok hivatkozni a tisztább, és érthetőbb leírás érdekében, majd a dolgozat végén kitérek arra is, hogy hogyan változik az algoritmus, amennyiben a program több keresendő szekvenciát is kap bemenetként. A FASTA formátum szekvenciák leírásával szolgál. (<https://www.ebi.ac.uk/seqdb/>

Mező sorszáma	Mező neve	Mező értéke
1	RecordType	Karakter, él esetén „L”
2	From	A kezdődő csúcs neve
3	FromOrient	A kezdődő csúcs orientációja
4	To	A végződő csúcs neve
5	ToOrient	A végződő csúcs orientációja
6	Overlap	A kezdődő, és a végződő csúcs közötti átfedő szakasz hossza

2. táblázat. A GFA formátum éleket leíró sorainak mezői. Egy csúcs orientációja a következőt jelenti: A csúcshoz tartozó szekvencia értéke a leírásban szereplő szekvencia: („+”karakter jelöli), vagy ennek a reverz komplemente („-”karakter jelöli)

confluence/display/JDSAT/FASTA+Help+and+Documentation) Minden bejegyzés egy „>” karakterrel kezdődik. Ezzel egy sorban a szekvencia leírása szerepel, majd a következő sorokban egészen a következő „>” karakterig maga a szekvencia.

5.1. Eredmények

Az algoritmus működését a SPAdes [16] szoftver által generált gráfokon teszteltem, és a Bandage [17] szoftver segítségével vizualizáltam. Hashfüggvényként a MurmurHash3 hash-függvényt [18] használok, amelyet 3-as seed-el inicializáltam.

Az eredmények kiértékelése során két algoritmust is futtattam minden gráfon. Az **első számú algoritmus** a 3. fejezetben bemutatott komplexebb algoritmus. Minden csúcstra három értékelést ad, egyet a csúcs elején található szekvencia részletre, egyet a végén található, valamint egy csúcson belüli globális legjobbat. Kiestáz minden olyan csúcsot, amelyre bármelyik visszatérített érték nagyobb, mint 0. A pontszámokat kiértékelő csúszóablak hosszát mindig a keresett szekvencia hosszával megegyező méretűre állítottam.

A **második számú algoritmus** a Containment MinHash algoritmus. Csúcsonként visszatéríti a keresett szekvencia lenyomatában megtalálható hash-értékek számát. Futása során kiestáz minden olyan csúcsot, melyről az aktuálisan vizsgált keresett szekvencia lenyomatával legalább egy közös hash-érték képezhető. Ez a módszer várhatóan nagyobb konfidenciával adja vissza a hasonló csúcsokat, azonban nem használható pozicionális információ kinyerésére, valamint nagyobb eséllyel térít vissza olyan csúcsokat is, amelyeknek csak nagyon rövid részlete hasonlít a szekvenciához, vagy olyan kmert tartalmaznak, melyet a keresett szekvencia nem tartalmaz, vagy nem is tartalmaznak hasonló szakaszt,

csak hash ütközés lépett fel.

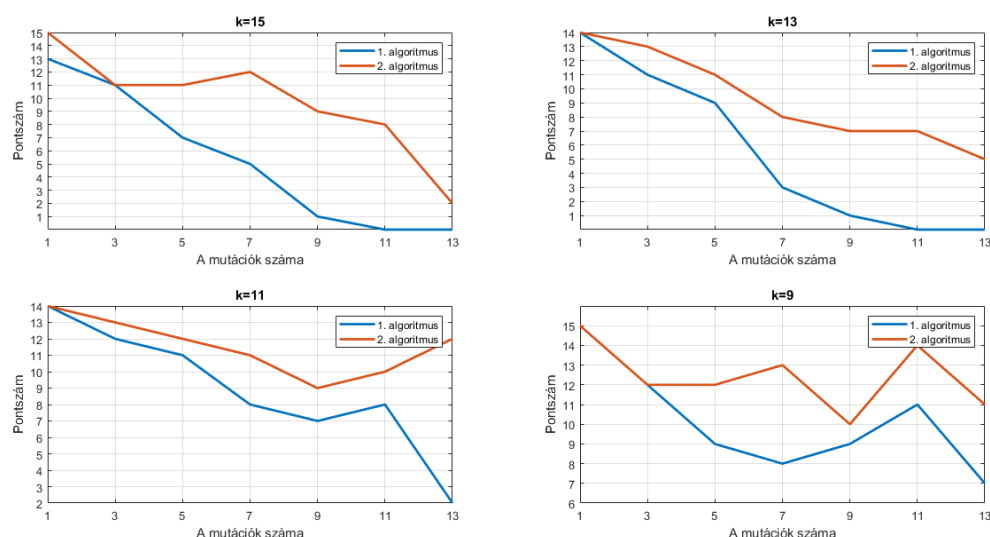
Teszt mesterségesen generált adatokkal

Az első tesztekhez olyan gráfokat használtam, melyeket egy szintén a kutatócsoportban dolgozó társam generált. A SPAdes algoritmus bemenetét egy random nukleotidokból generált szekvenciáról képezett readok alkották.

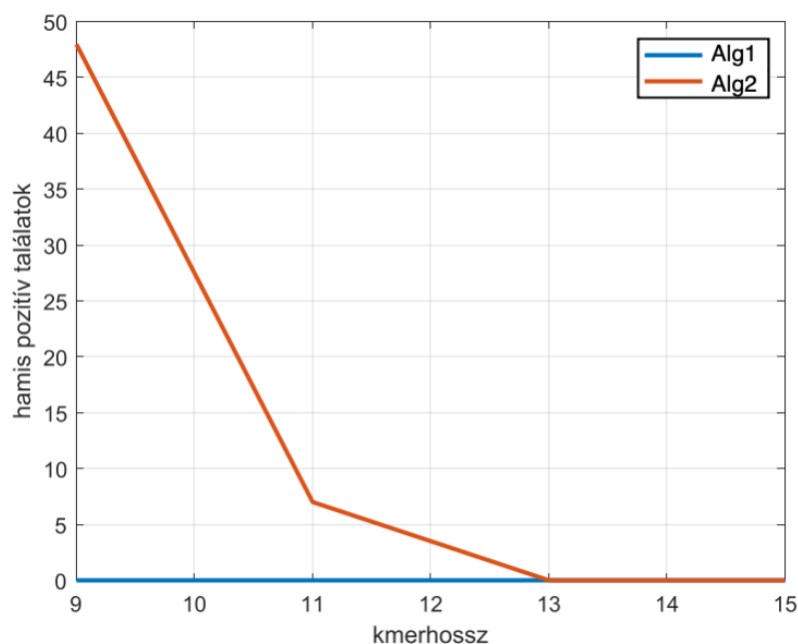
A keresett szekvenciákat úgy generáltam, hogy egy hosszabb csúcsból kivágtam egy rész-szekvenciát, melyről 7 újabb szekvenciát generáltam, szekvenciánként különböző számú mutációt végezve ezeken.

A gráf csúcsainak együttes hossza körülbelül 27000 bázispár, a kiválasztott csúcs hossza nagyságrendileg 1000 bázispár, az ebből kivágott szekvencia hossza pedig 250 bázispár volt.

A generált szekvenciák megkereséséhez 15 hosszúságú lenyomatot használtam. A két algoritmus teljesítményét vizsgáltam 15,13,11 és 9 hosszú kmerek használatával is. Az eredményeket a 6. ábra szemlélteti. Látható, hogy a második algoritmus minden kmer hosszra jobban teljesít, azonban amint az a 7. ábra mutatja, rövidebb kmerek esetén a hamis pozitív eredmények száma megugrik, míg az első algoritmus egyik esetben sem adott hamis pozitív eredményt.



6. ábra. Az algoritmusok által adott pontszámok a mutációk számának függvényében. A második algoritmus esetében a maximális pontszámot ábrázoltam.



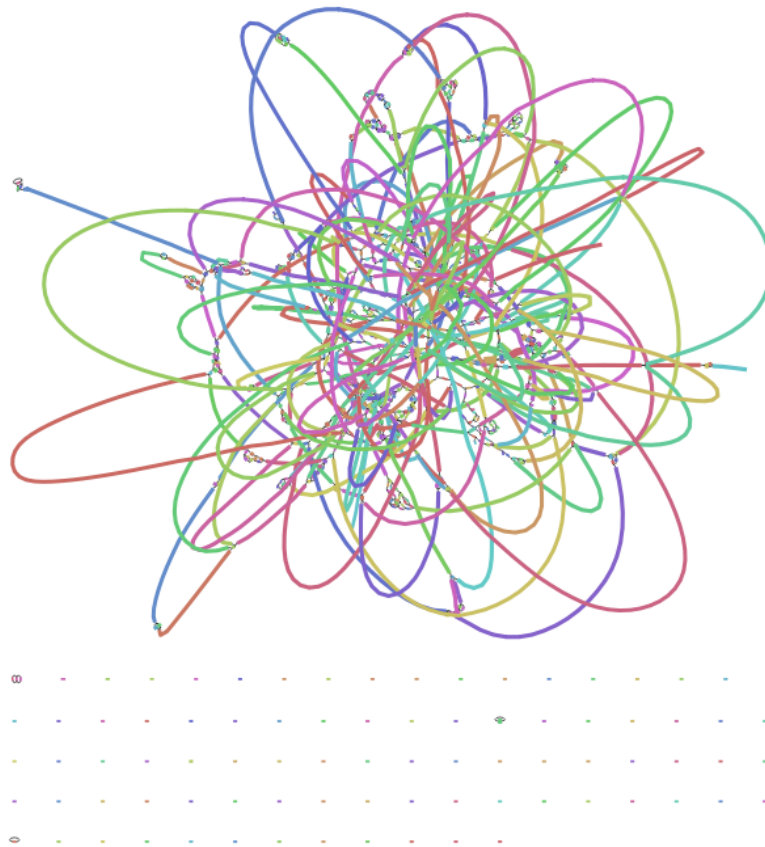
7. ábra. Az algoritmusok által adott hamis pozitív eredmények száma a kmerhossz függvényében.

Jelen lévő szekvenciák keresése valódi genomok alapján generált gráfban

A második teszthez olyan gráfokat kaptam, melyekhez a SPAdes bemenetét valós baktériumok DNS-éről képzett readok alkották. A keresett szekvenciákat két, a baktériumok genomja által tartalmazott virális DNS alkotta.

Az gráf csúcsainak a hosszának az összege 4 710 000 bázispár körüli. Az egyik (2-es szekvencia) keresett szekvencia nagyságrendileg 40 000, a másik (1-es szekvencia) pedig 20 000 bázispár hosszúságú. A gráfot Bandage szoftver által generált vizualizációja a 8. ábrán látható.

A kereséshez az alábbi paramétereket használtam: a kmerek hossza 25 bázispár, a lenyomat pedig 30 hash-értékből áll.



8. ábra. A második teszthez használt gráf megjelenítése a Bandage szoftver segítségével. A különböző színnel jelölt darabok a csúcsok, melyek hossza valódi hosszukkal arányos. A gráf tartalmaz a többi csúccsal nem összefüggő részeket, melyek az ábra alján láthatóak. A csúcsok színezése véletlenszerű.

A két algoritmus által az első és második szekvenciához talált csúcsokat a 3. és a 4. táblázatok foglalják össze.

Csúcs azonosítója	1. algoritmus	2. algoritmus
897738	18	18
837320	2	2
141876	1	1
760234	2	2
887932	6	6
155168	1	1
910366	-	1

3. táblázat. Az 1. szekvenciához talált csúcsok valamint az ezekhez tartozó pontszámok algoritmusonként. A talált csúcsok rövidebbek voltak, mint a szekvencia, ezért az első algoritmus mindhárom kimenete azonos értékű.

Csúcs azonosítója	1. algoritmus	2. algoritmus
909464	8	8
909764	22	22

4. táblázat. Az 2. szekvenciához talált csúcsok valamint az ezekhez tartozó pontszámok algoritmusonként. A talált csúcsok rövidebbek voltak, mint a szekvencia, ezért az első algoritmus mindhárom kimenete azonos értékű.

A 3. táblázatban megfigyelhető, hogy egy olyan csúcs (910366) van a találatok között, melyet a 2. algoritmus megtalált, azonban az első nem. Ezt okozhatja az, hogy a csúcsnak csak rövid szakasza tartozik a keresett szekvenciához, de az is előfordulhat, hogy véletlen az egyezés, és a hasonló rész nem lényegesen hosszabb, mint a kmer hossza, vagy nem is azonos a két kmer, csak a hash-értékük egyezik meg. Mindkét szekvenciához találtam 30 egyező hashértéket a gráf csúcsaiban, ezért valószínűleg tényleg módosítás nélkül tartalmazza a gráf a szekvenciát. A 10 ábrán látható a talált csúcsok környezete. A talált csúcsok nagyon közel vannak egymáshoz, ami megerősíti, hogy valódi találatok. Megfigyelhető, hogy több rövidebb csúcs a talált csúcsok mégsem teljesen szomszédosak, több rövid csúcs is közéjük ékelődött.

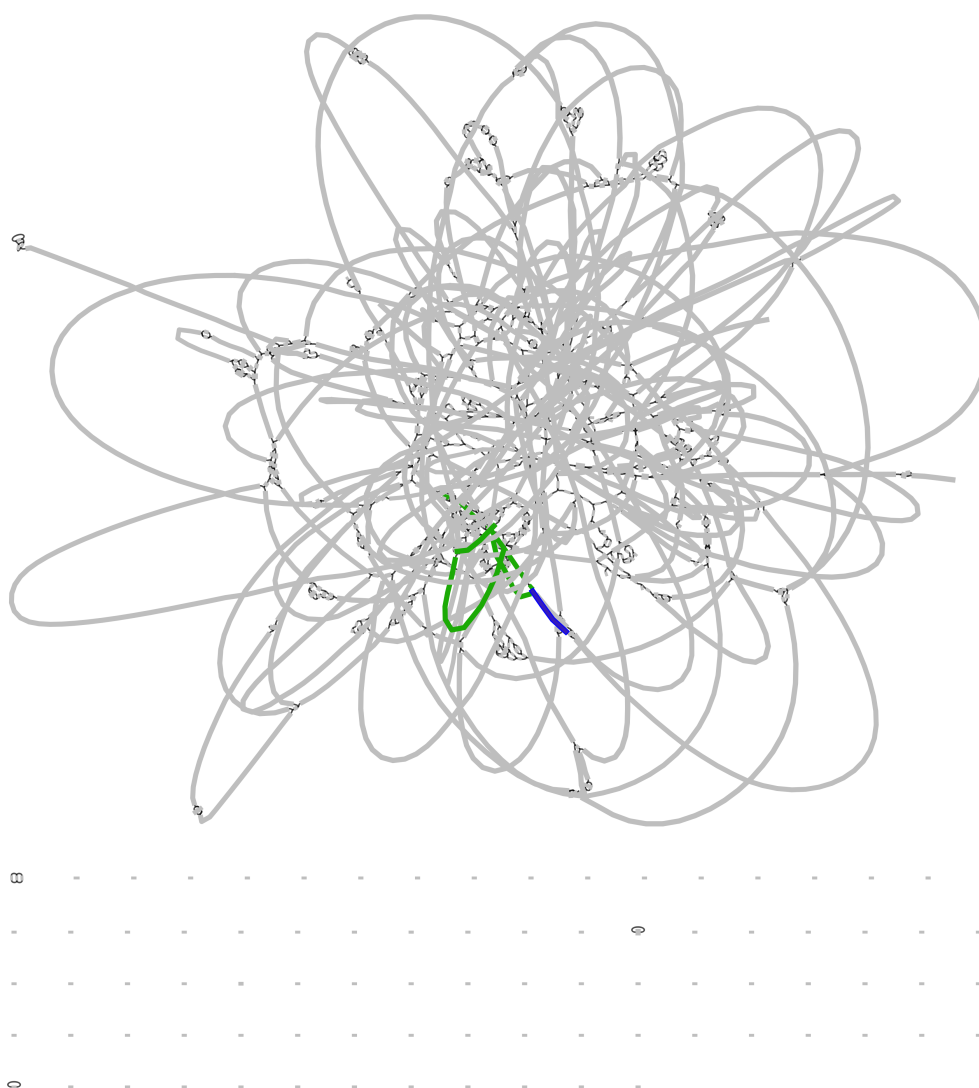
Amint a 9 ábrán látszik, a találatok száma és mérete elenyészően kicsi a teljes gráfhoz képest.

Az eredmények kiértékelése

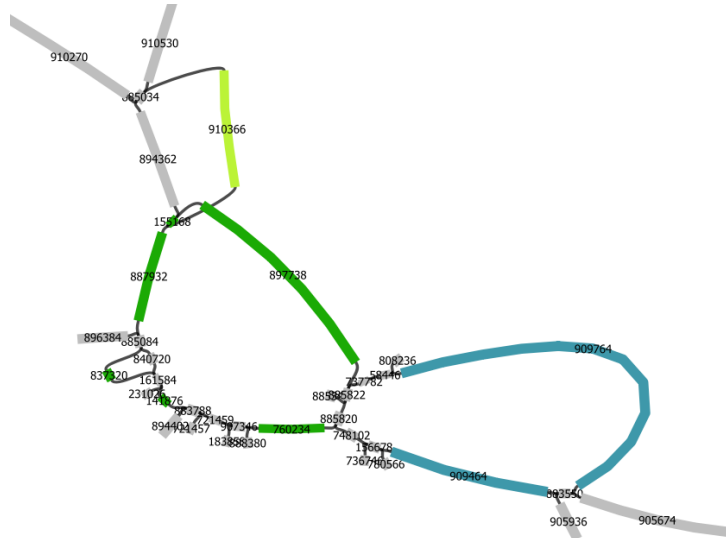
A kirajzolt gráfokból egyértelműen látszik, hogy az algoritmus képes pangenom gráfban szekvencia keresésére. Az első algoritmus a legtöbb esetben kevésbé teljesített jól, mint az egyszerűbb koncepción alapuló második algoritmus, azonban azt gondolom, hogy komplexebb helyzetekben, amikor fontos a hasonló szakasz helyzete a csúcson belül, vagy kevésbé hasonlít a keresett szekvencia a gráfban található-ra, szükséges lehet az elsőnek a használata is.

A második algoritmus nagyobb adaton már meglepően kevés hamis pozitív eredményt adott. Azt gondolom, hogy a két algoritmus jól használható egymás kiértékelésére, mivel az első kiszűri a hamis pozitív, a második pedig a hamis negatív eredményeket. Az első algoritmus által nem talált, de a második által pedig jól pontozott szekvenciák pedig gyakran azonosítják az út kezdetét vagy végét, ahol egy hosszabb szekvenciába belelóg a keresett szekvenciának egy rövidebb szakasza, melyet ellensúlyoznak a negatív minták.

Ezen túl azzal is sokat lehet majd javítani szerintem az első algoritmus pontosságán,



9. ábra. A találatok vizualizációja. A színes csúcsok valamelyik algoritmus alapján hasonlóak az egyik keresett szekvenciához. A szürkék nem mutattak hasonlóságot.



10. ábra. A megtalált csúcsok környezete gráfban, valamint a csúcsok azonosítói. A világoszöld csúcs a 2. algoritmus által talált, 1. szekvenciához tartozó csúcs. A sötétzöld csúcsok szintén az 1. szekvenciához tartoznak, ezeket mindkét algoritmus megtalálta. A kék csúcsok a 2. szekvenciához talált csúcsok. Ezeket mindkét algoritmus felismerte.

ha ideálisabb méretű csúszóablakot használok a minták kiértékelésére. Érdekes azt is megfontolni, hogy csak a csúcsok szélein használjak rövidebb csúszóablakot, és így megkapjam azokat a szekvenciákat is amelyek a rekonstruálandó út szélén helyezkednek el.

A valódi SPAdes által generált gráfok gyakran tartalmaznak nagyon rövid csúcsokat. Amennyiben ilyen bemenete lesz az algoritmusomnak, érdemes lehet azt majd úgy módosítani, hogy a egy küszöbnél rövidebb csúcsokat az utak kiértékelésénél vonja össze szomszédaikkal, és így ne szakadjon meg az útvonal minden rövidebb csúcsnál.

6. Összefoglalás

Dolgozatom célja hasonlósági alapon szekvenciák keresése egy pángenom gráfban. Definiáltam a megoldandó problémát, valamint bemutattam a pángenom gráfokat, biológiai jelentésüket és jelentőségüket. Rámutattam továbbá a nehézségre, miszerint a szekvenciák közötti hasonlóság nehezen, vagy nem is definiálható.

Megmutattam, hogy számítástechnikailag miért nehéz a probléma megoldása. Feldolgoztam a hasonló problémákra használt algoritmusokról szóló szakirodalmat. Bemutattam a sketching alapú algoritmusok működési alapelveit és megismerkedtem olyan korszerű és hatékony algoritmusokkal mint például a HyperLogLog vagy a MinHash, melyek

bevett eszközök halmazok hasonlóságának vagy kardinalitásának becslése esetén. Bemutattam, hogy ezek az algoritmusok hogyan alkalmazhatóak halmazok helyett szövegekre és milyen változtatásokra van szükség amennyiben nem természetes nyelvi szövegeken, hanem biológiai szekvenciákon szeretnénk használni őket.

Elemeztem a MinHash algoritmus korlátait és megmutattam, hogy csak hasonló méretű szekvenciák esetén működik megbízhatóan. Bemutattam a különböző méretű szekvenciák összehasonlítására szolgáló Containment MinHash algoritmust és definiáltam, hogy milyen további problémák megoldása szükséges még a gráfbeli keresés megvalósításához.

Létrehoztam egy olyan algoritmust, amely képes szekvenciák összehasonlítására olyan módon, hogy a gráf csúcsainak kisebb szakaszait is külön vizsgálja. Beláttam, hogy az algoritmus kevés keresett szekvencia esetén jó időkomplexitással rendelkezik a gráf nagyságának és a keresett szekvenciák hosszának függvényében is, valamint a pontossága is viszonylag jól ellenőrizhető. Körbejártam az algoritmus pontosságához szükséges bemeneti feltételeket. Definiáltam, hogy milyen feltételek kell, hogy teljesüljenek a vizsgált pángenom gráfra.

Megvizsgáltam az algoritmusom és a Containment MinHash algoritmus szenzitivitását egymáshoz képest.

Mindkét algoritmus futtattam példa gráfokon, valamint néhány példát be is mutattam. A gyakorlati eredményekből következtetéseket vontam le az általam implementált funkciók felhasználhatóságára.

Levontam a következtetést, miszerint az általam alkotott algoritmus és a Containment MinHash jól használhatóak egymás ellenőrzésére, mivel az egyik hamis pozitív, a másik pedig hamis negatív döntések meghozására hajlamos. Megállapítottam, hogy az implementált szoftver jól használható a definiált feladatra, azonban kisebb módosításokkal növelhető a szenzitivitása.

Hivatkozások

- [1] Z. Yang, A. Guarraçino, P. J. Biggs és tsai., „Pangenome graphs in infectious disease: a comprehensive genetic variation analysis of *Neisseria meningitidis* leveraging Oxford Nanopore long reads“, *Frontiers in Genetics*, 14. évf., 2023, ISSN: 1664-8021. DOI: [10.3389/fgene.2023.1225248](https://doi.org/10.3389/fgene.2023.1225248). cím: <https://www.frontiersin.org/articles/10.3389/fgene.2023.1225248>.
- [2] J. M. Eizenga, Novak és tsai., „Pangenome Graphs“, *Annual Review of Genomics and Human Genetics*, 21. évf., 139–162. old., 2020. aug. DOI: [10.1146/annurev-genom-120219-080406](https://doi.org/10.1146/annurev-genom-120219-080406).
- [3] L. H. Lin DM Koskella B, „Phage therapy: An alternative to antibiotics in the age of multi-drug resistance“, *World J Gastrointest Pharmacol Ther.*, 2017. DOI: [10.4292/wjgpt.v8.i3.162](https://doi.org/10.4292/wjgpt.v8.i3.162).
- [4] M. B. Albright, S. Louca, D. E. Winkler és et al., „Solutions in Microbiome Engineering: Prioritizing Barriers to Organism Establishment“, *ISME Journal*, 16. évf., 331–338. old., 2022.
- [5] N. Eisenhauer, P. Bonfante, F. Buscot és tsai., „Biotic Interactions as Mediators of Context-Dependent Biodiversity-Ecosystem Functioning Relationships“, *Research Ideas and Outcomes*, 8. évf., e85873., 2022.
- [6] M. Das A. Schatz, „Sketching and sampling approaches for fast and accurate long read classification“, *BMC Bioinformatics*, 23. évf., 2022. DOI: <https://doi.org/10.1186/s12859-022-05014-0>.
- [7] P. J. Melsted P., „Efficient counting of k-mers in DNA sequences using a bloom filter“, *BMC Bioinformatics*, 12. évf., DOI: <https://doi.org/10.1186/1471-2105-12-333>.
- [8] S. A. Alsuhibany, „A space-and-time efficient technique for big data security analytics“, *2016 4th Saudi International Conference on Information Technology (Big Data Analysis) (KACSTIT)*, 2016, 1–6. old. DOI: [10.1109/KACSTIT.2016.7756065](https://doi.org/10.1109/KACSTIT.2016.7756065).
- [9] Y. Shibuya, D. Belazzougui és G. Kucherov, „Set-Min sketch: a probabilistic map for power-law distributions with application to k-mer annotation“, *bioRxiv*, 2021. DOI: [10.1101/2020.11.14.382713](https://doi.org/10.1101/2020.11.14.382713). eprint: <https://www.biorxiv.org/content/early/2021/02/25/2020.11.14.382713.full.pdf>. cím: <https://www.biorxiv.org/content/early/2021/02/25/2020.11.14.382713>.
- [10] M. Steinegger és J. Söding, „Clustering Huge Protein Sequence Sets in Linear Time“, *Nature Communications*, 9. évf., 2542. old., 2018. DOI: [10.1038/s41467-018-04964-5](https://doi.org/10.1038/s41467-018-04964-5).
- [11] S. Heule, M. Nunkesser és A. Hall, „Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm“, *Proceedings of the 16th International Conference on Extending Database Technology*, 2013, 683–692. old.
- [12] L. B. D. Baker D.N., „Fast and accurate genomic distances with HyperLogLog“, *Genome Biol*, 20. évf., 2019. DOI: <https://doi.org/10.1186/s13059-019-1875-0>.

- [13] D. Koslicki és H. Zabeti, „Improving MinHash via the containment index with applications to metagenomic analysis“, *Applied Mathematics and Computation*, 354. évf., 206–215. old., 2019, ISSN: 0096-3003. DOI: <https://doi.org/10.1016/j.amc.2019.02.018>. cím: <https://www.sciencedirect.com/science/article/pii/S009630031930116X>.
- [14] „Git repository“, cím: https://dev.itk.ppke.hu/juhasz.judit/python_graph_minhash.
- [15] F. Almodaresi, H. Sarkar, A. Srivastava és R. Patro, „A space and time-efficient index for the compacted colored de Bruijn graph“, *Bioinformatics*, 34. évf., 13. sz., i169–i177. old., 2018. jún., ISSN: 1367-4803. DOI: [10.1093/bioinformatics/bty292](https://doi.org/10.1093/bioinformatics/bty292). eprint: https://academic.oup.com/bioinformatics/article-pdf/34/13/i169/50316242/bioinformatics_34_13_i169.pdf. cím: <https://doi.org/10.1093/bioinformatics/bty292>.
- [16] A. Bankevich, S. Nurk, D. Antipov és tsai., „SPAdes: a new genome assembly algorithm and its applications to single-cell sequencing“, *Journal of computational biology*, 19. évf., 5. sz., 455–477. old., 2012.
- [17] R. R. Wick, M. B. Schultz, J. Zobel és K. E. Holt, „Bandage: interactive visualization of de novo genome assemblies“, *Bioinformatics*, 31. évf., 20. sz., 3350–3352. old., 2015. jún., ISSN: 1367-4803. DOI: [10.1093/bioinformatics/btv383](https://doi.org/10.1093/bioinformatics/btv383). eprint: https://academic.oup.com/bioinformatics/article-pdf/31/20/3350/49036207/bioinformatics_31_20_3350.pdf. cím: <https://doi.org/10.1093/bioinformatics/btv383>.
- [18] S. Dahlgaard, M. Knudsen és M. Thorup, „Practical hash functions for similarity estimation and dimensionality reduction“, *Advances in Neural Information Processing Systems*, 30. évf., 2017.