

Glava 1

Gde u genomu počinje replikacija genoma?

1.1 Uvod

Na samom početku, želimo da definišemo pojam bioinformatike i da pokušamo da shvatimo koji je njen osnovni cilj. Da bismo to postigli, pogledajmo tri definicije, iz različitih izvora:

- "Bioinformatika je nauka koja se bavi prikupljanjem i analizom kompleksnih bioloških podataka poput genetskih kodova." - Oksfordski rečnik (engl. *Oxford Dictionary*)
- "Bioinformatika predstavlja prikupljanje, klasifikaciju, čuvanje i analizu biohemiskih i bioloških informacija korишћenjem računara, a posebno se primenjuje u molekularnoj genetici i genomici." - Rečnik Meriam-Webster (engl. *Merriam-Webster Dictionary*)
- "Bioinformatika je interdisciplinarno polje koje radi na razvoju metoda i softverskih alata za razumevanje bioloških podataka." - Wikipedija (engl. *Wikipedia*)

Na osnovu ove tri definicije možemo zaključiti da: *Bioinformatika predstavlja primenu računarских tehnologija u istraživanjima u oblasti biologije i srodnih nauka.*

Bioinformatika ima široku primenu i njene primene rastu zajedno sa razvojem discipline. Kao što možemo videti na slici ispod, primena bioinformatike se može sagledati kroz personalizovanu medicinu. Naime, na osnovu prikupljene veće količine podataka i njihove analize, uz pomoć različitih računarskih metoda, na primer metoda veštačke inteligencije, možemo doći do informacija potrebnih da na najbolji način lečimo pacijenta ili mu odredimo terapiju koja će mu na najbolji, najbrži i najbezbolniji način pomoći da prevaziđe određene zdravstvene probleme.

Bioinformatika je spoj više različitih disciplina, kao što su: statistika, istraživanje podataka, računarstvo, računarska biologija, biologija, biostatistika.

1.2 Replikacija genoma

1.2.1 DNK

Dezoksiribonukleinska kiselina (akronimi DNK ili DNA, od engl. *deoxyribonucleic acid*) je nukleinska kiselina koja sadrži uputstva za razvoj i pravilno funkcionisanje svih živih organizama. Zajedno sa RNK i proteinima, DNK je jedan od tri glavna tipa makromolekula koji su esencijalni za sve poznate forme života.

Sva živa bića svoj genetički materijal nose u obliku DNK, sa izuzetkom nekih virusa koji imaju ribonukleinsku kiselinu (RNK). DNK ima veoma važnu ulogu ne samo u prenosu genetičkih informacija sa jedne na drugu generaciju, već sadrži i uputstva za građenje neophodnih célijskih organela, proteina i RNK molekula. DNK segment koji sadrži ova važna uputstva se naziva gen.

DNK se sastoji iz dva polimerna lanca koji imaju antiparalelnu orijentaciju, i svaki od njih je sastavljen od azotnih baza:

- adenin (A)
- timin (T)
- guanin (G)
- citozin (C)

Lanci DNK su međusobno spojeni i to tako da se veze uspostavljaju isključivo između adenina i timina ili između guanina i citozina. Na osnovu toga, ako nam je poznat sastav jednog lanca, lako možemo zaključiti i sastav drugog lanca, zbog čega se kaže da su DNK lanci **međusobno komplementarni**.

Da bismo lakše manipulisali sa informacijama koje DNK nosi i približili sadržaj računarskoj struci, DNK ćemo posmatrati kao nisku nad azbukom {A, C, G, T}.

1.2.2 Replikacija genoma u céliji

Replikacija genoma je jedan od najvažnijih zadataka célije. Pre nego što se podeli, célija mora da najpre replicira svoj genom, tako da svaka od cérki célija dobije svoju kopiju.

Džejms Votson (engl. *James Watson*) i Fransis Krik (engl. *Fransis Crick*) su 1953. godine napisali rad u kome su primetili da postoji mehanizam za kopiranje genetskog materijala. Oni su uočili da se lanci roditeljskog DNK molekula odvijaju tokom replikacije i da se, potom, svaki lanac ponaša kao uzorak za sintezu novog lanca (na osnovu toga što se uvek spajaju iste amionokiseline A-T i G-C, rekreiranje lanca je moguće). Kao rezultat ovakvog ponašanja, proces replikacije počinje parom komplementarnih lanaca i završava se sa dva para komplementarnih lanaca, kao što se može videti na slici ispod.

Replikacija počinje u regionu genoma koji se naziva **početni region replikacije** (skraćeno *oriC*), izvode je enzimi koje se nazivaju DNK polimeraze, koje predstavljaju mašine za kopiranje na molekularnom nivou.

Nalaženje početnog regiona replikacije predstavlja veoma važan problem, ne samo za razumevanje funkcionalisanja kako se célije repliciraju, već je koristan i u raznim biomedicinskim problemima. Na primer, neki metodi genskih terapija uključuju genetski izmenjene mini genome, koji se zovu virusni vektori, zbog svoje sposobnosti da prodrnu kroz célijski zid (poput pravih virusa). Virusni vektori u sebi nose veštačke gene koji unapređuju postojeći genom. Genska terapija je prvi put uspešno izvršena 1990. godine na devojčici koja je bila toliko otporna na infekcije da je bila primorana da živi isključivo u sterilnom okruženju.

Osnovna ideja genske terapije je da se pacijent, koji pati od nedostatka nekog bitnog gena, zarazi virusnim vektorom koji sadrži veštački gen koji enkodira terapeutski protein. Jednom kad je unutar célije, vektor se replicira, što dovodi do lečenja bolesti pacijenta. Da bi moglo da dođe do ovoga, biologima je neophodno da znaju gde je *oriC*.

Pitamo se kako ćelija prepoznaće oriC? Sigurno je da postoji neka niska aminokiselina koja označava oriC, ali kako ga prepoznati? Ograničimo se na bakterijski genom, koji se sastoji od jednog kružnog hromozoma. Istraživanje je pokazalo da je region, koji predstavlja oriC kod bakterija, dug svega nekoliko stotina nukleotida.

Poznato je da DNKA utiče na početak replikacije. DNKA je protein koji se vezuje na kratki segment unutar oriC, poznatiji kao **DNKA boks**. Ona predstavlja poruku unutar sekvene DNK koja govori proteinu DNKA da se veže baš tu. Postavlja se pitanje: kada pronaći taj region bez prethodnog poznavanja izgleda DNKA boks?

Da bismo bolje razumeli *problem skrivene poruke* uzmimo za primer priču Edgara Alana Poa - "Zlatni jelenak" (engl. "The Gold-Bug"). Naime, u toj priči jedan od likova, Vilijam Legrand (engl. William Legrand), treba da dešifruje poruku :

```
53++!305))6*;4826)4+. )4+);806*;48!8`6 0))85;)8*:+*8!83(88)5*!;46(;88*96*?;8
)*+(;485);5*!2:+*(;4956*2(5*4)8`8*;40 69285);)6!8)4++;1(+9;48081;8:8+1;48!8 5;4)
485!528806*81(+9;48;(88;4(+?34;48 )4+;161;:188;+?;
```

On uočava da se ";48" pojavljuje veoma često, i da verovatno predstavlja "THE", najčešću reč u engleskom jeziku. Znajući to, zamenjuje karaktere odgovarajućim slovima i postepeno dešifruje celu poruku.

```
53++!305))6*THE26)H+. )H+)806*THE !E`60))E5;)E*:+*E!E3(EE)5*!TH6(T EE*96*?;E)**+
(THE5)T5*!2:+*(TH956 *2(5*H)E`E*TH0692E5)T)6!E)H++T1( +9THE0E1TE:E+1
THE!E5T4)HE5!52880 6*E1(+9THET(EETH(+?34THE)H+T161T :1EET+?T
```

Želeli bismo da ovaj princip primenimo na naš problem nalaska *oriC-a*. Ideja je da uvidimo da li postoje reči koje se neuobičajeno često pojavljuju. Uvedimo termin k-gram da označimo string dužine k i COUNT(Text, Pattern) da označimo broj puta kojih se k-gram *Pattern* pojavio u tekstu *Text*. Osnovna ideja je da pomeramo prozor, iste dužine kao k-gram *Pattern*, niz teksta, usput proveravajući da li se pojavljuje *Pattern* u nekome od njih.

```
1  PATTERNCOUNT(Text, Pattern)
2      count = 0
3      for i = 0 to |Text| - |Pattern|
4          if Text(i,|Pattern|) = Pattern
5              count = count + 1
6      return count
```

Za neki *Pattern* kažemo da je on *najčešći k-gram* u tekstu *Text*, ako je njegov COUNT najveći među svim k-gramima. Na primer, **ACTAT** je najčešći 5-gram u tekstu *Text* = ACAACTATGCAACTATCGGGACAACTATCCT, a **ATA** je najčešći 3-gram u *Text* = CGATATATCCATAG.

Sada, problem pronađivanja čestih reči možemo posmatrati kao računarski problem:

Problem čestih reči: Pronaći najčešće k-grame u niski karaktera.

Ulaz: Niska Text i ceo broj k.

Izlaz: Svi najčešći k-grami u niski Text.

Osnovni algoritam za pronađivanje čestih k-grama u stringu *Text* proverava sve k-grame koji se

pojavljuju u tom stringu (takvih k-grama ima $|Text| - k + 1$) i potom izračunava koliko puta se svaki k-gram pojavljuje. Da bismo implementirali ovaj algoritam, moramo da izgenerišemo niz $COUNT$, gde je $COUNT(i) = COUNT(Text, Pattern)$ za $Pattern = Text(i, k)$.

```

1 FrequentWords(Text, k)
2   FrequentPatterns <- an empty set
3   for i = 0 to |Text| - k
4     Pattern <- the k-mer Text(i,k)
5     COUNT(i) <- PatternCount(Text, Pattern)
6   maxCount <- max value in array COUNT
7   for i = 0 to |Text| - k
8     if COUNT(i) = maxCount
9       add Text(i,k) to FrequentPatterns
10  remove duplicates from FrequentPatterns
11  return FrequentPatterns

```

Pitamo se, sada, kolika je složenost ovakvog pristupa? Ovaj algoritam, iako uspešno nalazi ono što se od njega traži, nije najefikasniji. S obzirom na to da svaki k-gram zahteva $|Text| - k + 1$ provera, svaki od njih zahteva i do k poređenja, pa je broj koraka izvršavanja funkcije $PatternCount(Text, Pattern)$ zapravo $(|Text| - k + 1) * k$. Osim toga, *FrequentWords* mora pozvati *PatternCount* $|Text| - k + 1$ puta (po jednom za svaki k-gram teksta), tako da je ukupan broj koraka $(|Text| - k + 1) * (|Text| - k + 1) * k$. Iz navedenog, možemo zaključiti da je ukupna cena izvršavanja algoritma *FrequentWords* $O(|Text|^2 * k)$.

Primer: Pronalazak čestih reči kod bakterije *Vibrio cholerae*

Posmatrajmo, najpre, tablicu najčešćih k-grama u *oriC* regionu bakterije *Vibrio cholerae*. Da li nam se čini da se neki k-grami pojavljuju neuobičajeno često?

k	3	4	5	6	7	8	9
count	25	12	8	8	5	4	3
k -mers	tga	atga	gatca	tgatca	atgatca	atgatcaa	atgatcaag
			tgatc				cttgatcat
							tcttgatca
							ctcttgatc

Slika 1.1: Tablica najčešćih k-grama u *oriC* regionu bakterije *Vibrio cholerae*

Na primer, 9-gram **ATGATCAAG** se pojavljuje tri puta u *oriC* regionu, da li nas to izne- nađuje?

Označili smo najčešće 9-grame, umesto nekih drugih k-grama, jer je eksperimentima pokazano da su DNKA boksovi kod bakterija dugi 9 nukleotida. Uočimo da postoje četiri različita 9-grama koji se ponavljaju tri ili više puta u ovom regionu, to su: ATGATCAAG, CTTGATCAT, TCTTGATCA i CTCTTGATC.

Mala verovatnoća da se neki 9-gram toliko puta pojavi u *oriC*-u kolere, govori nam da neki od četiri 9-grama koje smo pronašli može biti potencijalni DNKA boks, koji započinje replikaciju. Ali, koji?

atcaatgatcaacgtaaagcttctaagc**ATGATCAAG**gtgctcacacagtttatccacaacacctgagtgg
atgacatcaagataggcgttgtatctccttcgtactctcatgaccacggaaag**ATGATCAAG**
agaggatgatttcttgccatatcgaaatacttgtgacttgtgctccaattgacatctcagc
gccatattgcgtggccaagggtacggagcgggattacgaaagcatgatcatggctgttctgttt
atcttggtttgcgtgagacttgttaggatagacgggtttcatcactgactagccaaagccttactct
gcctgacatcgaccgtaaattgataatgaatttacatgcttcgcgacgattac**CTTGATCAT**cg
atccgattgaagatcttcaattgttaattcttgcctcgactcatagccatgatgagct**CTTGATCA**
Tgtttcctaacccttattttacgaaag**ATGATCAAG**ctgctgct**CTTGATCAT**cgtttc

Slika 1.2: Prikaz 9-grama ATGATCAAG i njegovog komplementa u *oriC* regionu *Vibrio cholerae*

Podsetimo se da nukleotidi A i T, kao i C i G, su komplementarni. Ako imamo jednu stranu lanca DNK i neke slobodne nukleotide, možemo lako zamisliti sintezu komplementarnog lanca, kao što se vidi na slici ispod.

Posmatrajmo ponovo sliku 1.7. Na njoj možemo uočiti 6 pojavljivanja niski ATGATCAAG i CTTGATCAT, koji su zapravo komplementarni. Naći 9-gram koji se pojavljuje 6 puta u DNK nisci dužine 500 nukleotida, je još više iznenađujuće, nego pronaći 9-gram koji se pojavljuje tri puta. Ovo posmatranje nas dovodi do toga da je ATGATCAAG (zajedno sa svojim komplementom) zaista DNKA boks *Vibrio cholerae*. Ovaj zaključak ima smisla i biološki, jer DNKA proteinu, koji se vezuje i započinje replikaciju, nije bitno za koji od dva lanca se vezuje.

Primer: Pronalazak čestih reči kod bakterije *Thermotoga petrophila*

Nakon što smo pronašli skrivenu poruku za *Vibrio cholerae*, ne bi trebalo da odmah zaključimo da je ta poruka ista kod svih bakterija. Najpre bi trebalo da proverimo da li se ona nalazi u *oriC* regionu drugih bakterija, možda različite bakterije, imaju drugačije DNKA boksove. Uzmimo, za primer, *oriC* region bakterije *Thermotoga petrophila*. Ona predstavlja bakteriju koja obitava u izrazito toplim regionima, na primer u vodi ispod rezervi nafte, gde temperature prelaze 80 stepeni Celzijusa. Pogledajmo kako izgleda *oriC* region ove bakterije.

aactctatacccttttgtcaatttgtgatttatagagaaaatcttattaactgaaactaa
aatggtaggttggtaggtttgtacatttgttagtatctgatTTtaattacataccgt
tattgtattaaattgacgaacaattgcatggaattgaatatatgcaaaacaaacctaccacaaac
tctgtattgaccatTTtaggacaactcagggtggtaggttctgaagctctcatcaatagactat
tttagtcttacaaacaatattaccgttcagattcaagattctacaacgcgtttatggcgtt
gcagaaaacttaccacctaAAatccagtatccaagccgatttcagagaaacctaccacttacctac
cacttacacctaccacccgggtggtaaggcagacattattaaaacctcatcagaagcttggtaaa
aaatttcaataactcgaaacacctaccacccgtcccattatttactactactaataatagcagta
taattqatctqaaaqaqqtggtaaaaaa

Slika 1.3: Prikaz *oriC* regiona *Thermotoga petrophila*

Možemo lako uočiti da se u ovom regionu nigde ne javljaju niske ATGATCAAG ili CTT-GATCAT, iz čega zaključujemo da različite bakterije mogu koristiti različite DNKA boksove, kako bi pružile skrivenu poruku DNKA proteinu. Odnosno, za različite genome imamo različite DNKA boksove.

Najčešće reči u ovom *oriC* su: AACCTACCA, ACCTACCAC, GGTAGGTTT, TGGTAGGTT, AACACCTACC, CCTACCACC. Pomoću alata koji se zove Ori-Finder, nalazimo CCTACCACC i njegov komplement GGTGGTAGG kao potencijalne DNKA boksove naše bakterije. Ove dve niske se pojavljaju ukupno 5 puta.

Naučili smo da pronađemo skrivene poruke ako je *oriC* dat, ali ne znamo da pronađemo *oriC* u genomu.

1.2.3 Pronalaženje početnog regiona replikacije

Zamislimo da pokušavamo da nađemo *oriC* u novom sekvenciranom genomu bakterije. Ako bismo tražili niske poput ATGATCAAG/CTTGATCAT ili CCTACCACC/GGTGGTAGG to nam verovatno ne bi mnogo pomoglo, jer novi genom može koristiti potpuno drugačiju skrivenu poruku. Posmatrajmo, zato, drugačiji problem: umesto da tražimo grupe određenog k-grama, pokušajmo da nađemo svaki k-gram koji formira grupu u genomu. Nadajmo se da će nam lokacije ovih grupa u genomu pomoći da odredimo lokaciju *oriC*-a. Ideja je da pomeramo prozor fiksirane dužine L kroz genom, tražeći region u kome se k-gram pojavljuje više puta uzastopno. Za L ćemo uzeti vrednost 500, koja predstavlja najčešću dužinu *oriC*-a kod bakterija.

Definisali smo k-gram kao *grupu*, ako se pojavljuje više puta unutar kratkog intervala u genomu. Formalno, k-gram *Pattern* formira (L, t) grupu unutar niske *Genome* ako postoji interval genome, dužine L , u kome se k-gram pojavljuje barem t puta.

Problem pronalaženja grupa. Naći k-gramme koji formiraju grupe unutar niske karaktera.

Uzorak: Niska Genome i celi brojevi k (dužina podniske), L (dužina prozora) i t (broj podniski u grupi).

Izlaz: Svi k-grami koji formiraju (L, t) -grupe u niski Genome.

U genomu bakterije E.coli postoji 1904 različitih 9- grama koji formiraju $(500, 3)$ -grupe. Koji od njih ukazuje na početni region replikacije?

Iskrivljeni dijagrami

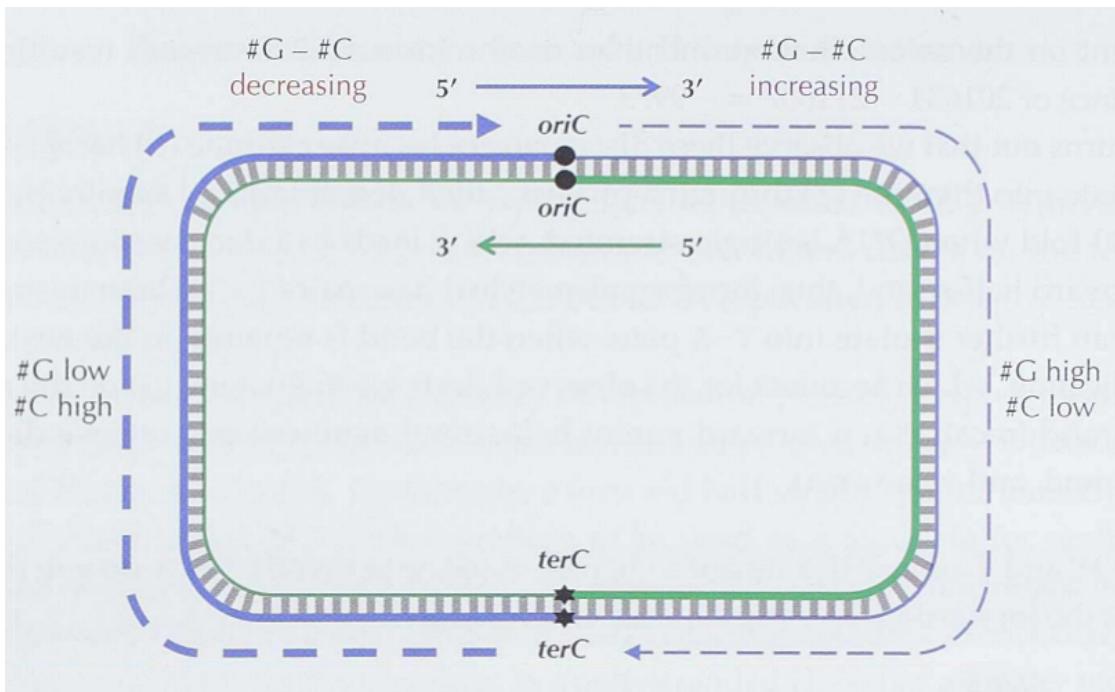
S obzirom na to da imamo veliku količinu statističkih podataka, pitamo se kako ih možemo upotrebiti da bismo došli do lokacije *oriC*-a? U tome nam mogu pomoći **iskrivljeni dijagrami** (engl. *skew diagram*). Osnovna ideja je da prođemo kroz genom i da računamo razliku između količine guanina(G) i citozina(C). Ako ova razlika raste, onda možemo pretpostaviti da se krećemo niz polulanac koji ide na desno (u nastavku samo polulanac, smer $5' \rightarrow 3'$), a ako razlika počne da se smanjuje, onda pretpostavljamo da smo na obrnutom polulancu ($3' \rightarrow 5'$). Zbog procesa koji se naziva deaminacija (gubljenje aminokiselina), svaki polulanac ima manjak citozina u poređenju sa guaninom, a svaki obrnuti polulanac ima manjak guanina u odnosu na citozin.

Posmatrajmo iskrivljeni dijagram bakterije Ešerihija Koli. Lako uočavamo minimalnu vrednost skew dijagrama.

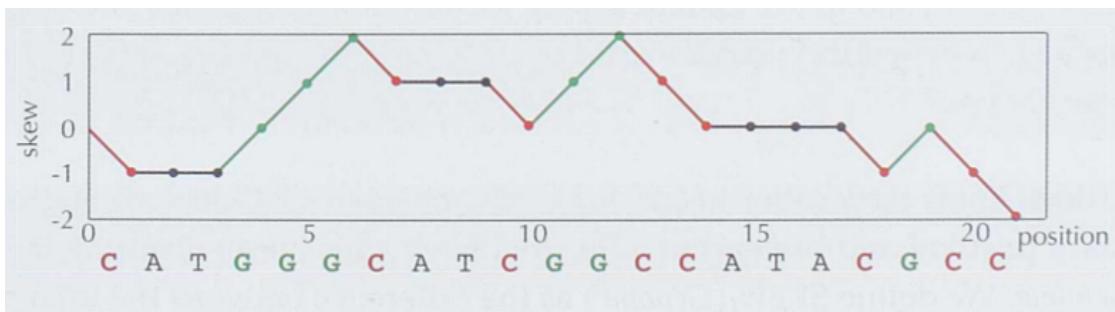
Minimalna vrednost iz iskrivljenog dijagrama ukazuje baš na ovaj region:

```
aatgatgatgacgtcaaaaggatccggataaaaacatggtgattgcctcgcataacgcggta
tggaaatggattgaagccccggccgtggattctactcaactttgtcggcttgagaaagacc
tgggatcctgggtattaaaaagaagatctatttatttagagatctgttctattgtatctc
ttatttagatgcactgccctgtggataacaaggatccggctttaaagatcaacaacctgg
aaaggatcattaactgtaatgtcggtatcctggaccgtataagctggatcagaatga
ggggttatacacaactcaaaaactgaacaacagttttttggataactaccgggtgatc
caagcttcctgacagagtatccacagtagatcgcacgatctgtataacttattgagtaaa
ttaaccacgatcccagccattttctgcccggatctccggatgtcgatcaagaatgt
tgatttcagtg
```

Slika 1.6: Region na koji pokazuje minimalna vrednost iskrivljenog dijagrama Ešerihije koli.



Slika 1.4: Prikaz kretanja.



Slika 1.5: Iskrivljeni dijagram genoma Genome = CATGGGCATCGGCCATACGCC.

Uočimo da u ovom regionu nema čestih 9-grama (koji se pojavljuju 3 ili više puta). Iz toga zaključujemo da, iako smo uspeli da nađemo *oriC* bakterije Ešerihija koli, nismo uspeli da nađemo DNKA boksove. Međutim, pre nego što odustanemo od potrage, osmotrimo još jednom *oriC* Vibrio colerae, kako bismo pokušali da nađemo način da izmenimo naš algoritam i uspemo da lociramo DNKA boksove u Ešerihiji koli. Veoma brzo, može se uvideti da osim tri pojavljivanja ATGATCAAG i tri pojavljivanja CTTGATCAT, *oriC* Vibrio cholerae sadrži i dodatna pojavljivanja ATGATCAAC i CATGATCAT koji se razlikuju samo u jednom nukleotidu od gornjih niski. Ovo još više povećava šanse da smo naišli na prave DNKA boksove, a ima i biološkog smisla. Naime, DNKA se može vezati i za nesavršene DNKA boksove, one koji se razlikuju u nekoliko nukleotida.

Cilj nam je da sada izmenimo algoritam čestih reči (*FrequentWords*) tako da možemo da pronađemo DNKA boksove koji su predstavljeni čestim k-gramima, sa mogućim izmenama na pojedinim nukleotidima. Ovaj problem nazvaćemo problem čestih reči sa propustima.

Problem čestih reči sa propustima. Pronaći najčešće k-grame sa propustima u niski karaktera.

Ulaz: Niska Text i celi brojevi k i d.

Izlaz: Svi najčešći k-grami sa najviše d propusta u niski Text.

Pokušajmo, još jednom, sa pronalaskom DNKA boksova kod Ešerihije koli, tako što ćemo naći najčešće 9-grame sa propustima i komplementima u regionu *oriC* koji nam je predložen minimalnom vrednošću iskrivljenog dijagrama. Pokušaćemo sa malim prozorom koji ili počinje ili se završava ili je centriran na poziciji najmanje iskrivljenosti. Ovakvim izvođenjem pronalazimo TTATCCACA/TGTGGATAA kao najčešći 9-gram. Međutim, ovo nije jedini 9-gram. Za ostale 9-grame još uvek ne znamo čemu služe, ali znamo da nose skrivene informacije, da se grupišu unutar genoma i da većina njih nema veze sa replikacijom.

Slika 1.7: Prikaz pronađenih niski sa propustima i komplementima u *oriC* regionu Ešerihije koli.

```
aatgatgatgacgtcaaaaggatccggataaaacatggtgattgcctcgataacgcgg  
tataaaaatggattgaagcccgccgtggattctactcaactttgtcggctttagaaaa  
gacctggatcctgggtattaaaaagaagatctattttagatctgttctattgt  
gatctcttatttaggatcgcactgcccTGTGGATAAcaaggatccggctttaaagatcaa  
caacctggaaaggatcattaactgtgaatgatcggtgatcctggaccgtataagctggg  
atcagaatgagggTTATACACAactcaaaaactgaacaacagtgttcTTTGGATAAC  
taccgggttcatccaagcttcctgacagagTTATCCACAgtagatcgcacgatctgtata  
cttatttgagtaattaaacccacgatcccagccattctctgccggatctccggatg  
tcgtgatcaagaatgttcatcttcagtg
```

Glava 2

Koji DNK šabloni igraju ulogu molekularnog sata?

2.1 Biološki problem

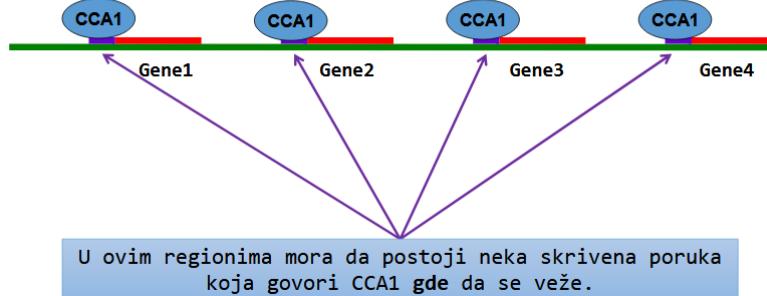
Bioritam svih živih bića kotroliše „unutrašnji časovnik” koji još zovemo i cirkadijalni. Ljudi koji često putuju avionom na drugi kraj sveta mogu to da osećate kada pokušaju da zaspete nakon promene nekoliko vremenskih zona. Kao i svaki sat, i ovaj može da se pokvari, što rezultuje genetskom bolešću pod nazivom sindrom odložene faze spavanja. Njegova osnova je na molekularnom nivou. Mnogi procesi su kontrolisani ovim časovnikom, što ilustruje slika 2.1. Kao što vidimo, postoji tačno određeno vreme u toku dana kada telo ima najnižu temperaturu, kada kreće i prestaje lučenje hormona poput melatoninina (neophodnog za kvalitetan san) i slično.

Naučnici su se pitali kako ćelije znaju kada treba da uspore ili ubrzaju proizvodnju određenih proteina. Ranih sedamdesetih, Ron Konopka i Sejmor Benzer su napravili prve korake ka rešavanju ove misterije. Do danas je otkriveno mnogo cirkadijalnih gena koji koordiniraju ponašanje stotine drugih gena.

Kod čoveka, cirkadijalni ritam je promenljiv, tj. varira od osobe do osobe. Mi ćemo se u daljem tekstu fokusirati na biljke, jer je kod njih cirkadijalni ritam pitanje života i smrti, stoga ne sme biti promenljiv. Geni biljaka moraju znati kada sunce izlazi i zalazi kako bi znali kada treba vršiti fotosintezu, jer je ona od krucijalne važnosti za život biljke, a usko povezana sa količinom sunčeve svetlosti. Primeri specifičnih ponašanja nekih biljaka u zavisnosti od cirkadijalnog ritma dati su na slici 2.2.

Ispostavlja se da svaka ćelija biljke čuva podatak o tome da li je dan ili noć nezavisno od drugih ćelija, kao i da su samo tri gena odgovorna za upravljanje satom. Oni kodiraju regulatorne proteine (transkripcione faktore)- to su LCY, CCA1 i TOC1. Spoljašnji faktori, kao što je količina sunčeve svetlosti, kontrolišu regulatorne gene i regulatorne proteine kako bi organizmi prilagodili svoju gensku ekspresiju, odnosno da li će se protein sintetisati u to vreme ili ne. Dakle, svaki metabolički proces je regulisan cirkadijalnim časovnikom kojim upravljaju regulatorni proteini, odlučujući kada će se koji protein u ćeliji biljke sintetisati. Regulatorni proteini upravljaju genskom ekspresijom tako što se vežu za DNK u trenucima kada je potrebno sintetisati neki protein važan za cirkadijalni ritam, kao što je prikazano na slici 2.3. Naravno, to vezivanje se ne može ostvariti bilo gde u okviru DNK, već postoje posebna mesta, kao što je prikazano na slici 2.4. Ta posebnost je neophodna iz više razloga. Regulatorni protein mora znati gde treba da se veže, a isto tako DNK mora znati kako da to protumači, odnosno, to za nju mora biti signal da je vreme započeti sintezu, kao i pružiti informaciju o kom se proteinu radi. Ta posebna mesta predstavljaju manje skupove nukleotida u DNK koji nazivamo **regulatorni motivi**. Naš zadatak je da ih pronađemo.

Slika 2.1: Mesta vezivanja regulatornih proteina



2.2 Informatički problem

Kako bismo informatički rešili problem pronalaženja regulatornih motiva u DNK, moramo predstaviti sve gorepomenute biološke pojmove na način koji bi informatičar i njegov računar mogli razumeti i obraditi.

- Iz ove tačke gledišta, DNK predstavlja nisku karaktera nad azbukom: A, G, T, C.
- Kodirajuće sekcije DNK (one koje se prepisuju i prevode u proteine) za nas će biti podnische niske DNK.
- Regulatorni motiv je šablon koji se pojavljuje tačno jednom u svakoj kodirajućoj sekciji DNK.

Hajde da damo u računarskom smislu korektnu definiciju problema:

Informatička definicija problema nalaženja regulatornih motiva u DNK

Ulaz: N niski koje predstavljaju kodirajuće sekvence DNK.

Izlaz: Podniska dužine k koja predstavlja regulatorni motiv (skrivenu poruku, mesto vezivanja regulatornih proteina).

Ovaj problem nas podseća na problem pronalaženja *OriC-a*, koji smo rešavali svodeći ga na pronalaženje čestih reči u tekstu koristeći algoritam *FrequentWords*, detaljno opisan u prethodnom poglavljju. Da bismo taj algoritam primenili ovde, neophodno je da od niza niski dobijemo konkatenacijom jednu veliku nisku, na koju ćemo primeniti ovaj algoritam.

Međutim, ovaj algoritam nije primenljiv ako motivi mutiraju. Možemo pokušati da prime-nimo algoritam *FrequentWordsWithMissmatches*. To bi nas dovelo do tačnog rešenja, ali nakon previše vremena. Naime, kada smo nalazili *OriC*, tražili smo uzorke dužine 9 karaktera (DnaA box je bio te dužine). U ovom slučaju, tražimo motive koji su najčešće dužine 15 karaktera, pa nam ovaj algoritam ne radi dovoljno brzo. Dakle, potrebna nam je nova ideja.

2.3 Problem ubačenog motiva

I dalje pokušavamo da pronađemo skrivenu poruku u DNK koja kodira mesto vezivanja regulatornih proteina za DNK. Dali smo korektnu definiciju problema i, kao pravi matematičari, pokušali najpre da svedemo problem na neki drugi, koji smo već rešili. Međutim, to nas je

odvelo u veliku vremensku složenost, pa samim tim i neupotrebljivost pronađenog algoritma za rešavanje. Zaključili smo da nam je potreban novi pristup, tj. da moramo probati da problem rešimo direktno - bez pomoći algoritama za *OriC*. Za ovako nešto, neophodno je najpre definisati još nekoliko pojmove vezanih za šablove koje pokušavamo da pronađemo:

- Mutirani šablon predstavlja šablon u kome se na nekim mestima može pojaviti mutacija, odnosno odstupanje od početne niske.
- (k, d) motiv je k -gram koji se pojavljuje u svakoj sekvenci sa najviše d razlika.
- Kanonski motiv predstavlja motiv koji tražimo (bez uticaja mutacija).
- Instance su mutirani motivi - oni koji se pojavljuju u niskama sa najviše d grešaka, odnosno razlika u odnosu na kanonski motiv.

Sada, kada smo tačno i precizno definisali pojmove koji će nam igrati uloge regulatornih motiva, izmenićemo i samu definiciju problema, tj. malo promeniti scenario tako da se sve uloge lepo uklope, pazeći naravno da ne izgubimo na tačnosti definicije.

Problem ubačenog motiva: Pronalaženje (k, d) motiva u skupu niski

Ulaz: Skup niski Dna i celi brojevi k (dužina motiva) i d (maksimalni broj razlika).

Izlaz: Svi (k, d) motivi u skupu Dna .

Nova definicija problema izrodiće nekoliko novih ideja za njegovo rešavanje, a te ideje izrodiće nove algoritme. U daljem tekstu prikazaćemo nekoliko rešenja, uz osvrt na ideju koja nas je do njega dovela.

2.3.1 Enumeracija motiva

Početna ideja je zasnovana na gruboj sili - za svaki k -gram ćemo ispitati da li je (k, d) motiv za dati skup niski. Dakle, trebalo bi generisati 4^k kombinacija i za svaku ispitati da li je (k, d) motiv. Postavlja se pitanje da li je potrebno ispitati svih 4^k kandidata.

Pogledajmo na primer sledeći skup niski:

AAATTTAAATTTAAATTT

TTTAAATTTAAATTTAAA

Da li ima smisla proveravati 8-gram GGAAGGAA? Naravno da nema! Karakter G se ne pojavljuje ni u jednoj niski iz našeg skupa. Zato sigurno ne može biti traženi (k, d) motiv. Dakle, prvo možemo izbaciti kandidate koji se ne pojavljuju ni u jednoj niski iz skupa Dna . Takođe, možemo primetiti da je skup instanci jednog kandidata koje se pojavljuju u ostalim niskama (ceo skup Dna , bez niske u kojoj je dati kandidat podniska) prilično ograničen. Ograničenje nameće broj d . Dakle, jasno je da treba proveravati samo one instance koje se od kandidata razlikuju na najviše d pozicija. Ovakvo redukovana pravila potrage iznedrīće algoritam koji se naziva *MotifEnumeration*, koji bismo u obliku pseudokoda predstavili na sledeći način:

```

1 MotifEnumeration(Dna, k, d)
2   for each k-mer a in Dna
3     generate all possible k-mers a1 differing from a
4       by at most d mutations
5
6       for each such k-mer a1
7         if a1 is a (k, d)-mer in each sequence in Dna
8           output a1

```

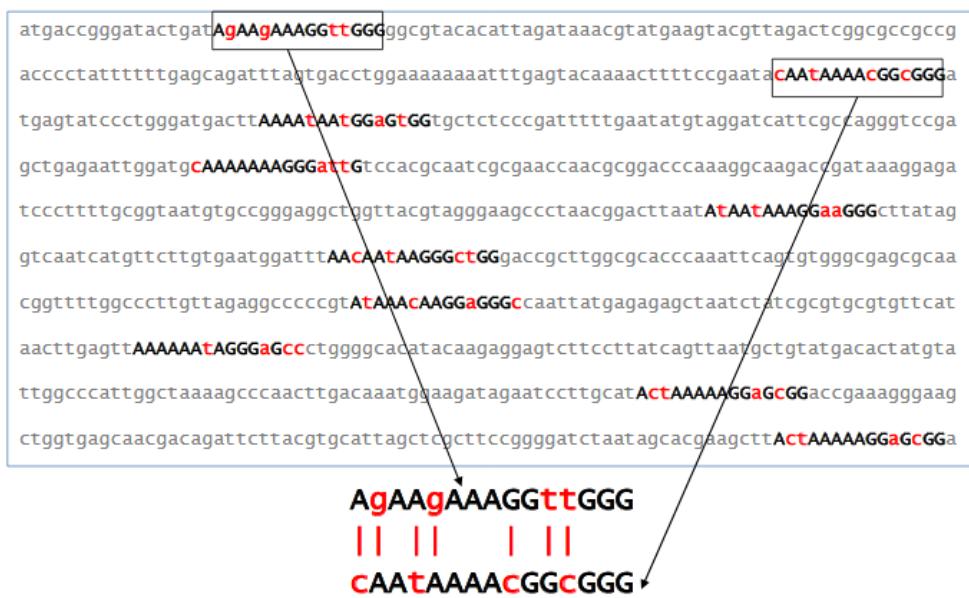
Ovaj algoritam je suviše spor kada su k i/ili d veliki brojevi. Hajde da malo analiziramo složenost ovog algoritma. Možemo videti da najveću (i jedinu) komponentu složenosti čini dvostruka forpetlja, koja zavisi od broja kandidata koje pretražujemo i broja njihovih instanci. Kada su k i/ili d veliki brojevi, instanci ima mnogo, i pretraga ide sporo. Zbog toga, u praksi, ovaj algoritam ne pokazuje dobre rezultate za velike vrednosti k i/ili d . Dakle, potrebna nam je nova ideja.

2.3.2 Najsličniji k-grami u parovima niski

Videli smo da pretraga grubom silom ima veliku složenost, usled (iako redukovanih ali i dalje ogromnog) broja kandidata i njihovih instanci. Pokušali smo da redukujemo broj provera i nismo postigli mnogo. To nas dovodi do razmišljanja da ključ rešenja optimalnog algoritma ne leži u broju, već možda u redosledu. Hajde da konstruišemo ovakav primer:

```
atcgtagAAATTAAAGGGgtcaactg
ggatcaagctAAATCTAAAGGGcttcag
gatctacccaAAACTTAAAGGGgtaaac
```

Velikim slovima predstavljen je (12,1)-motiv koji tražimo. Grubom silom pretraga bi počela na samom početku prve niske. Kako naš motiv počinje na 9. poziciji prve niske, vidimo da bi 8 puta naleteli na čorsokak. To znači da bi se 8 velikih iteracija izvršilo uzalud. Ako bismo nekako mogli početi od devete pozicije, uštedeli bismo vreme. Ali, kako da znamo koja je tajna pozicija od koje treba krenuti? Primećujemo da se podnische atcgtag i ggatcaag prilično razlikuju (7/8 pozicija razlike). Dakle, to sigurno ne može biti sastavni deo neke instance. Međutim, podnische AAATTAAAGGG i AAATCTAAAGGG su prilično slične (1/12 pozicija razlike). To je svakako dobar kandidat. Ako bismo u prve dve niske pronašli kandidate sa dosta sličnosti, šanse da smo odmah pogodili (k, d) motiv bi bile velike. Sve i da nismo, ovim putem ćemo obilaziti prvo verovatnije kandidate, pa bi ukupan broj pretraga sa velikom verovatnoćom bio dosta manji. Dakle, možemo probati da poredimo parove niski iz Dna , da uočimo dva najsličnija k-grama u dve niske iz Dna , od njih napravimo kanonski, i za njega proveravamo da li je (k, d) motiv. Malo veći primer možete videti na slici 2.2 i probati sami da nađete ubačeni motiv.



Slika 2.2: Najsličniji k-grami u parovima niski - primer

Zvuči predobro da bi bilo istinito? Pa možda ste i u pravu. Sličnost para niski je takođe određena brojem d . One se od kanonskog motiva mogu razlikovati na d pozicija, što znači da se među sobom, mogu razlikovati na $2d$ pozicija. Na malom izolovanom primeru, gde je bilo $d = 1$, zaista su slične niske isplivale kao dobri kandidati. Međutim, u praksi, sa $(15,4)$ -motivima, broj dozvoljenih razlika je malo veći, čak 8. Kako izgledaju takvi parovi, prikazano je na slici 2.3, gde se jasno vidi koliko je to zaista mnogo i kako ove dve niske više i nisu tako slične.



Slika 2.3: Najsličniji k-grami u parovima niski - problem

Primer koji pokazuje koliko je ovo zaista loše rešenje je eksperiment rađen nad 10 slučajno generisanih niski iz *Dna* dužine 600, sa ubačenim $(15, 4)$ motivom. Pristupom pronalaženja parova niski iz *Dna*, pronađeno je nekoliko hiljada parova k-grama koji su se razlikovali na manje od 8 pozicija. Ovo sigurno nisu skrivene poruke, jer ih je previše.

Da bismo ih pretražili u nekom smislenom redosledu, potrebno nam je da ih nekako rangiramo. Dakle, potreban nam je novi pristup.

2.3.3 Matrice motiva

Dosadašnje pretrage skupa *Dna*, nisu dale dobre rezultate. Pokušavali smo da ubrzamo pretragu na razne načine, ali bez puno uspeha. Možda je vreme da probamo da potpuno promenimo pristup, da mislimo izvan kutije. Pokušajmo da analiziramo jedan skup rešenja, i da malim koracima unazad dođemo do početnih uslova.

Pretpostavićemo da imamo neku kolekciju motiva i stavimo ih u matricu. Recimo da smo uradili jednom grubu pretragu i dobili neke rezultate. Da bismo olakšali komunikaciju, uvećemo par pojmove koje ćemo koristiti u daljem tekstu.

- Najpopularniji nukleotid u nekoj koloni matrice je onaj koji se pojavljuje najveći broj puta.
- Konsenzus niska predstavlja nisku koja se dobija nadovezivanjem najpopularnijih nukleotida iz svake kolone.
- Skor predstavlja broj nepopularnih simbola (onih koji nisu najpopularniji u svojoj koloni) u matrici.

Za lakše usvajanje, ilustrovani primer ovih pojmove prikazan je na slici 2.4. Najpopularniji nukleotidi u svakoj koloni matrice motiva su napisani velikim slovima i boldovani. Oni su nadovezani u konsenzus nisku. Nepopularni simboli su napisani malim slovima, i njihov ukupan broj predstavlja skor.

Analizirajmo malo našu matricu rešenja. Ako bismo prepostavili da su ovo instance nekog (k, d) motiva, jasno, želeli bismo da one budu što sličnije, tj. da se razlikuju na što manje pozicija. Dakle, želimo da nam broj nepopularnih simbola bude što manji, odnosno želimo mali skor. Dakle, ako ovako postavimo problem, skor je zapravo "mera neuspela", pa je ključ dobrog

	T	C	G	G	G	G	g	T	T	T	T	t	t
	c	C	G	G	t	G	A	c	T	T	T	a	C
	a	C	G	G	G	G	A	T	T	T	t	t	C
Motifs	T	t	G	G	G	G	A	c	T	T	t	t	t
	a	a	G	G	G	G	A	c	T	T	C	C	C
	T	t	G	G	G	G	A	c	T	T	C	C	C
	T	C	G	G	G	G	A	T	T	c	a	t	
	T	C	G	G	G	G	A	T	T	c	C	t	
	T	a	G	G	G	G	A	a	c	T	a	C	
	T	C	G	G	G	t	A	T	a	a	C	C	

Consensus(Motifs) T C G G G G A T T T C C

Score(Motifs) 3+ 4+ 0+ 0+ 1+ 1+ 1+ 5+ 2+ 3+ 6+ 4= 30

Slika 2.4: Matrica motiva, konsenzus niska i skor

rešenja u minimizovanju ove veličine. Novi pojmovi, novi pristup i nova ideja, zahtevaju i malo drugačiju definiciju problema:

Problem pronalaženja motiva: Za dati skup niski iz *Dna* naći skup k-grama (po jedan iz svake niske) sa minimalnim skorom među svim mogućim k-gramima iz datog skupa niski.

Ulaz: skup niski *Dna* i ceo broj *k*.

Izlaz: skup k-grama *Motifs*, po jedan iz svake niske *Dna*, tako da je vrednost skora matrice *Motifs* minimalna.

Kada smo definisali problem, vreme je da krenemo da ga rešavamo. Prvo što nam svakako pada na pamet je gruba sila. To je obično najjednostavniji algoritam, sa ne tako dobrom brzinom. Ali, ako on da zadovoljavajuću brzinu, nema potrebe komplikovati dalje, zar ne?

Dakle, krećemo redom. Uzimamo po jedan k-gram iz svake niske skupa *Dna* i računamo skor. Tako redom obidemo sve kombinacije k-grama, pamteći minimalni skor, i na kraju, proglašimo rešenjem onaj skup motiva sa najmanjim skorom.

Rešenje je dobro, pa hajde da pogledamo složenost. Neka je *t* broj niski i *n* dužina svake niske. Zaključujemo da je $(n - k + 1)$ broj k-grama jedne niske, odnosno, $(n - k + 1)^t$ broj kombinacija koje treba da proverimo. Vreme provere je konstantno (treba samo izračunati skor i ažurirati neku vrednost). Dakle, ukupna složenost je: $(n - k + 1)^t$. Eksponencijalna složenost svakako nije dobra, tako da zaključujemo da je ovaj algoritam prespor. Moramo ga nekako unaprediti.

Vratimo se na našu matricu motiva i pokušajmo da primetimo nešto što bi nam pomoglo da malo ubrzamo stvari. Pogledajmo primer dat na slici 2.5.

Izračunali smo skor po vrstama i primetili da je isti kao skor po kolonama. Da li je to slučajnost? Ispostavlja se da nije. Suština veličine skor leži u tome da broji nepopularne simbole. Nije važno kako ih prebrajamo, ako su svi na broju. Pojam brojanja definisaćemo na malo formalniji način. Definisaćemo veličinu koja se zove Hamingonovo rastojanje.

Hamingovo rastojanje između dve niske istih dužina predstavlja broj pozicija na kojima se one razlikuju. Npr. Hamingonovo rastojanje niski AACCTTGG i ATCCATGG je 2. Kada imamo Hamingovo rastojanje, možemo reći da je skor po kolonama suma Hamingovih rastojanja k-grama iz matrice od konsenzus niske.

Motifs	T	C	G	G	G	G	g	T	T	T	T	t	t	3
	C	C	G	G	t	G	A	C	T	T	T	a	C	4
	a	C	G	G	G	G	A	T	T	T	T	t	C	2
	T	t	G	G	G	G	A	c	T	T	t	t	t	4
	a	a	G	G	G	G	A	c	T	T	T	C	C	3
	T	t	G	G	G	G	A	c	T	T	T	C	C	2
	T	C	G	G	G	G	A	T	T	T	c	a	t	3
	T	C	G	G	G	G	A	T	T	T	c	C	t	2
	T	a	G	G	G	G	A	a	c	T	a	C	C	4
	T	C	G	G	G	t	A	T	a	a	C	C	3	
														=30
Consensus(Motifs)	T	C	G	G	G	G	A	T	T	T	T	C	C	
Score(Motifs)	3+	4+	0+	0+	1+	1+	1+	5+	2+	3+	6+	4=30		

Slika 2.5: Matrica motiva, konsenzus niska i skor - detaljnije

Vreme je da pokušamo da unapredimo naš algoritam. Želimo nekako da nađemo način da uradimo suprotno - da od konsenzusa niske i niza *Dna* dobijemo matricu *Motifs*. Koristićemo sledeće osobine koje smo primetili na matrici *Motifs*:

1. Kako skor predstavlja sumu rastojanja od k-grama iz matrice do konsenzusa niske, minimalni skor predstavlja minimizovanu sumu tih rastojanja.
2. Kako je skor po kolonama i vrstama isti, možemo rastojanje računati i po vrstama. Tako dolazimo do reforomulacije našeg problema, tj. njegovog ekvivalenta koji koristi naša nova zapažanja.

Iskoristićemo ove osobine kako bismo još jednom predefinisali naš problem tako da odražava sliku iz novog pristupa.

Problem pronalaženja motiva - reformulacija: Naći k-gram *Pattern* i skup k-grama *Motifs* iz skupa niski *Dna* koji minimizuju rastojanje između svih mogućim k-grama *Pattern* i svih mogućih skupova k-grama *Motifs*.

Ulaz: skup niski iz *Dna* i ceo broj *k*.

Izlaz: k-gram *Pattern* i skup k-grama *Motifs* iz skupa niski *Dna* koji minimizuju $d(\text{Pattern}, \text{Motifs})$.

Ovo je ekvivalentno našem prethodnom problemu jer smo primetili da $d(\text{Pattern}, \text{Motifs})$ predstavlja skor ukoliko je *Pattern* konsenzus niska. Postavlja se pitanje da li smo ovim otežali naš problem. Ispostaviće se da nismo, jer ne moramo ispitivati sve skupove k-grama *Motifs*, već je dovoljno da iskoristimo problem niske medijane koji ispituje sve k-grame *Pattern*.

2.3.4 Problem niske medijane

Prethodnom definicijom problema, nameće se zaključak da je sledeći prirodni korak proći kroz sve kombinacije skupa k-grama *Motifs* i k-grama *Pattern*, kako bismo našli kombinaciju sa najmanjim rastojanjem. Međutim, videli smo i da algoritmi zasnovani na gruboj sili, tj. iscrpnoj pretrazi kombinacija u potrazi za pravom, vode ka ogromnoj složenosti i lošoj praktičnoj primeni.

Zato ćemo ovaj put preskočiti taj korak i odmah krenuti da modifikujemo algoritam tako da dobijemo manju složenost. Primetićemo da imamo dve dimenzije pretrage (računarski bi to bile

dve ugnježdene for petlje). Jedna je po svim k-gramima *Pattern* a druga po skupovima *Motifs*. Jasno je da bi se složenost prepolovila (tj. korenovala) ako bismo imali samo jednu dimenziju. Zvuči neverovatno, ali je moguće, ako primetimo da nam unutrašnja for petlja nije potrebna, jer možemo uzeti cele niske umesto motiva. Tačnije, želimo da računamo rastojanje jednog k-grama od celog skupa *Dna*. Računanjem rastojanja od samih niski iz *Dna* umesto od njihovih podniski dužine k , štedimo dosta vremena, a postižemo isti efekat. Međutim, da bismo to učinili, najpre moramo definisati nekoliko pojmoveva, na prvom mestu jer još uvek nismo rekli kako se računa rastojanje sem ako imamo dve niske i to iste dužine. Dakle, definisacemo najpre sledeće pojmove:

- Hamingovo rastojanje između dve niske različitih dužina predstavlja minimum Hamingovih rastojanja između kraće niske i svih podniski duže niske odgovarajuće dužine.
- Definišemo rastojanje između k-grama i skupa (dužih) niski kao sumu rastojanja između tog k-grama i svih niski iz skupa.
- Niska medijana za skup niski *Dna* predstavlja onaj k-gram koji minimizuje rastojanje između tog k-grama i skupa *Dna* - $d(k\text{-}gram, Dna)$.

Novi pojmovi i nova ideja, prirodno vode i do nove definicije problema. Moramo dobro definisati problem koji rešavamo ako želimo da nas to dovede do dobrog algoritma za njegovo rešavanje.

Problem niske medijane: pronaći nisku medijanu.

Ulaz: skup niski *Dna*.

Izlaz: k-gram $k\text{-mer}$ koji minimizuje rastojanje $d(k\text{-mer}, Dna)$.

Sada imamo sav potreban alat da rešimo problem. Eliminišući jednu dimenziju, pretragućemo vršiti samo po svim mogućim kombinacijama k-grama *Pattern* (njih 4^k) i kao rezultat vratiti onaj k-gram koji ima najmanje rastojanje od skupa *Dna*. Ovaj algoritam se naziva *MedianString* i prikazan je sledećim pseudokodom:

```

1 MedianString(Dna, k)
2     best-k-mer ← AAA...AA
3     for each k-mer from AAA...AA to TTT...TT
4         if d(k-mer, Dna) < d(best-k-mer, Dna)
5             best-k-mer ← k-mer
6     return best-k-mer

```

Hajde da analiziramo složenost i vidimo da li smo zaista napredovali. Neka je t dužina niza *Dna* i n dužina niske iz *Dna*. Tada nam je za izračunavanje rastojanja između jednog k-grama i jedne niske iz skupa *Dna* potrebno $k \cdot (n - k + 1)$ poređenja. Kako imamo t niski, za izračunavanje rastojanja k-grama od skupa potrebno nam je $t \cdot k \cdot (n - k + 1)$, odnosno približno $t \cdot k \cdot n$, jer je n daleko veće od k pa čini primarnu komponentu složenosti u $(n - k + 1)$. Ceo taj postupak moramo izvršiti za svaki k-gram, odnosno 4^k puta, pa imamo, dakle, da ukupna složenost algoritma iznosi: $4^k \cdot n \cdot t \cdot k$. Primećujemo da je ovo dobar napredak u odnosu na $n^t \cdot k \cdot t$, ali i dalje imamo eksponencijalnu složenost. Dakle, iako je *MedianString* algoritam mnogo brži od grube sile, za velike k je i dalje prespor.

2.3.5 Probabilistički pristup

Uopšteno o pristupu

Analizirali smo nekoliko determinističkih algoritama i videli da, iako uvek daju tačno rešenje, troše preveliku količinu vremena. Čak i najbolji među njima, *MedianString*, imao je eksponencijalnu složenost. To nas navodi na razmišljanje da je možda ponovo došlo vreme da uradimo

veliki zaokret u pristupu samom problemu i pokušamo da nađemo kompromis između tačnosti i utroška vremena. Algoritmi koji ovo omogućavaju se nazivaju probabilistički algoritmi. Oni daju tačno rešenje sa određenom verovatnoćom, ali za dosta manje vremena, i oni su naša sledeća stanica u potrazi za ubaćenim motivima. Kao i svaki put kada menjamo pristup, upoznaćemo se najpre sa nekoliko osnovnih pojmova:

- *Count* matrica neke matrice motiva prikazuje koliko se puta koji nukleotid ponavlja u svakoj koloni matrice motiva.
- Profilna matrica neke matrice motiva prikazuje učestalost pojavljivanja nukleotida u svakoj koloni matrice motiva.

Ilustrovani primer ovih pojmova prikazan je na slici 2.9.

Slika 2.6: *Count* i profilna matrica

	T	C	G	G	G	G	g	T	T	T	t	t
Motifs	c	C	G	G	t	G	A	c	T	T	a	C
	a	C	G	G	G	G	A	T	T	T	t	C
	T	t	G	G	G	G	A	c	T	T	t	t
	a	a	G	G	G	G	A	c	T	T	c	c
	T	t	G	G	G	G	A	c	T	T	c	c
	T	C	G	G	G	G	A	T	T	c	a	t
	T	C	G	G	G	G	A	T	T	c	c	t
	T	a	G	G	G	G	A	a	c	T	a	c
	T	C	G	G	G	T	A	T	a	a	C	C

<i>Count(Motifs)</i>	A:	2	2	0	0	0	0	9	1	1	1	3	0
	C:	1	6	0	0	0	0	0	4	1	2	4	6
	G:	0	0	10	10	9	9	1	0	0	0	0	0
	T:	7	2	0	0	1	1	0	5	8	7	3	4

<i>Profile(Motifs)</i>	A:	.2	.2	0	0	0	0	.9	.1	.1	.1	.3	0
	C:	.1	.6	0	0	0	0	0	.4	.1	.2	.4	.6
	G:	0	0	1	1	.9	.9	.1	0	0	0	0	0
	T:	.7	.2	0	0	.1	.1	0	.5	.8	.7	.3	.4

Osnovna ideja ovog pristupa može se ilustrovati bacanjem k četverostranih kockica sa otežanim stranama. Svaka kockica pretstavlja jednu poziciju u motivu dužine k , i na stranama ima simbole A, C, T i G. Strane kockica su otežane, tako da svaka strana pada sa onom verovatnoćom koja je zapisana u profilnoj matrici, u preseku kolone rednog broja pozicije u motivu i reda karaktera koji je na toj strani kockice. Na primer, ukoliko imamo profilnu matricu kao na slici 2.10, bacali bismo 6 četverostranih kockica, gde bi recimo druga kockica bila otežana tako da simbol A pada sa verovatnoćom $7/8$, simbol C sa verovatnoćom 0, simbol T sa verovatnoćom $1/8$ i simbol G sa verovatnoćom 0. Naravno da u praksi ne možemo imati verovatnoću nula, ali sa našim zamišljenim kockicama, to je legitimna situacija. Zašto ovo nije dobro i kako se to popravlja, diskutovaćemo malo kasnije.

Neka su redom pali karakteri: A, T, A, C, A i G. Verovatnoću podniske koju oni čine možemo izračunati na osnovu profilne matrice, pretpostavljajući da su bacanja nezavisni događaji. Dakle, pročitaćemo iz profilne matrice verovatnoću karaktera A na prvoj poziciji ($1/2$), verovatnoću

karaktera T na drugoj ($1/8$), ..., i pomnožićemo sve te vrednosti. Proizvod verovatnoća iznosi 0.001602 .

Slika 2.7: Računanje verovatnoće k-grama

Neka je data profilna matrica <i>Profile</i> :						
A	1/2	7/8	3/8	0	1/8	0
C	1/8	0	1/2	5/8	3/8	0
T	1/8	1/8	0	0	1/4	7/8
G	1/4	0	1/8	3/8	1/4	1/8

$$\Pr(\text{ATACAG} \mid \text{Profile}) =$$

$$\frac{1}{2} \times \frac{1}{8} \times \frac{3}{8} \times \frac{0}{1/8} \times \frac{5/8}{1/4} \times \frac{7/8}{1/8} = 0.001602$$



Ako se setimo kako smo definisali matricu motiva, konsenzus nisku i profilnu matricu, lako možemo zaključiti da je ova verovatnoća zapravo verovatnoća da je niska ATACAG konsenzus niska. Sada nam same kockice više nisu potrebne, već možemo uzeti realne k-grame iz niski skupa *Dna* i za njih računati. Na primer, za nisku CTATAAACCTTACAT iz nekog skupa *Dna*, možemo izračunati verovatnoće svih njenih podniski dužine 6 (rezultate možete videti na slici 2.11) i potražiti onaj sa najvećom verovatnoćom (u primeru sa slike 2.11 to je AACCT). On je svakako dobar kandidat za ubačeni motiv. U ovome leži sama suština svih probabilističkih algoritama za traženje ubačenih motiva.

Slika 2.8: Primer: najverovatniji 6-gram

6-mer	Pr (6-mer Profile)	
CTATAAAACCTTACAT	$1/8 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
CTATAAAACCTTACAT	$1/2 \times 7/8 \times 0 \times 0 \times 1/8 \times 0$	0
CTATAAAACCTTACAT	$1/2 \times 1/8 \times 3/8 \times 0 \times 1/8 \times 0$	0
CTATAAAACCTTACAT	$1/8 \times 7/8 \times 3/8 \times 0 \times 3/8 \times 0$	0
CTATAAAACCTTACAT	$1/2 \times 7/8 \times 3/8 \times 5/8 \times 3/8 \times 7/8$.0336
CTATAAACCTTACAT	$1/2 \times 7/8 \times 1/2 \times 5/8 \times 1/4 \times 7/8$.0299
CTATAAAACCTTACAT	$1/2 \times 0 \times 1/2 \times 0 \times 1/4 \times 0$	0
CTATAAAACCTTACAT	$1/8 \times 0 \times 0 \times 0 \times 0 \times 1/8 \times 0$	0
CTATAAAACCTTACAT	$1/8 \times 1/8 \times 0 \times 0 \times 3/8 \times 0$	0
CTATAAAACCTTACAT	$1/8 \times 1/8 \times 3/8 \times 5/8 \times 1/8 \times 7/8$.0004

U daljem tekstu, predstavićemo nekoliko probabilističkih algoritama za rešavanje problema ubačenih motiva. Opisaćemo same algoritme, ali i diskutovati o njihovim dobrim i lošim stranama.

Važna napomena: Kao što smo već napomenuli, cenu malog vremena ovi algoritmi plaćaju tačnošću. Dakle, rešenje može malo odstupati od traženog. Zbog toga, za postizanje najbolje tačnosti, probabilističke algoritme treba pokretati više puta i uzeti najbolji rezultat.

Algoritam *Greedy motif search*

Prvi algoritam koji ćemo predstaviti je pohlepni algoritam. Kao ulaz, on prima skup niski *Dna*, broj njegovih elemenata t i dužinu ubačenog šablonu k , a kao izlaz daje matricu ubačenih

motiva *BestMotifs*. Suština algoritma leži u njegovom imenu: grabi k-mere iz jedne niske redom, na osnovu svakog od njih nađi najbolju matricu u koju se on uklapa, i uzmi onu koja ima najmanji skor. Na početku *BestMotifs* se inicijalizuje na t podniski dužine k , od kojih svaka predstavlja prvih k karaktera jedne niske iz skupa *Dna*. Iterira se kroz prvu nisku skupa *Dna*, i iteracija ima onoliko koliko ima podniski dužine k u njoj. U svakoj iteraciji, računa se po jedna matrica motiva.

Gradjenje te matrice je jedan vid pohlepe ovog algoritma. Grabimo već izračunate vrednosti, kako bismo što bolje odredili sledeće. Prvi red čini tekuća podniska iteracije (podniska prve niske skupa *Dna* koja počinje od karaktera na poziciji rednog broja iteracije). Za svaki sledeći red se pravi profilna matrica prethodno izračunatih redova i na osnovu nje računa najverovatnija podniska iz one niske skupa *Dna*, koja ima redni broj isti kao redni broj reda matrice koji računamo. Dakle, svaki sledeći motiv u matrici se sve lepše poklapa sa ostatkom matrice. Sledеći korak u iteraciji je da izračunamo skor i ako je manji, da ažuriramo vrednost *BestMotifs*. Dakle, ona matrica koja ima najmanji skor je tražena matrica *BestMotifs*. Sam algoritam ilustrovan je sledećim pseudokodom:

```

1 GreedyMotifSearch(Dna, k, t)
2   BestMotifs ← motif matrix formed by first k-mers in each string from Dna
3   for each k-mer Motif in the first string from Dna
4     Motif1 ← Motif
5     for i = 2 to t
6       form Profile from motifs Motif1, ..., Motifi-1
7       Motifi ← Profile - most probable k-mer in the i-th string in Dna
8       Motifs ← (Motif1, ..., Motift)
9       if Score(Motifs) < Score(BestMotifs)
10      BestMotifs ← Motifs
11    return BestMotifs

```

GreedyMotifSearch je brži od *Median string* algoritma. Povoljan je i za veće k , ali cena toga je manja tačnost. Tačnost se može poboljšati primenom Laplasovog pravila, o kojem će biti nešto više reči malo kasnije.

Randomized motif search

Posmatrajući algoritam *GreedyMotifSearch*, možemo primetiti da nismo postigli pun potencijal po pitanju prelaska na probabilistički pristup. Broj iteracija je i dalje fiksan (zavisi isključivo od dužine niski iz skupa *Dna* i dužine ubačenog motiva), a i sam njihov redosled je nametnut deterministički. Kako uvek krećemo od istog skupa motiva, i pratimo iste utabane korake, pokretanjem ovog algoritma više puta nećemo poboljšati ni tačnost ni vreme konvergencije.

Hajde da pokušamo da potpuno izbacimo determinizam iz priče i da pustimo da izgled same niske *Dna* i zakon verovatnoće igraju glavne uloge u konvergenciji algoritma. Prva ideja o poboljšanju probabilizma bi mogla biti vezana za sam početak algoritma. Hajde da umesto kretanja od početnih pozicija, krećemo od nekih nasumično odabranih. To sasvim sigurno donosi malo probabilizma u našu priču. Međutim, i dalje imamo iteriranje po jednoj od niski, što fiksira i broj iteracija i njihov redosled. To je ono što nam najviše i smeta.

Na početku sekcije o probabilističkim algoritmima, naučili smo da pravimo profilnu matricu od matrice motiva, a u algoritmu *GreedyMotifSearch* smo videli da je moguće i obrnuto. To je dobra ideja za osnovu iteracije. Od matrice motiva bismo mogli praviti profilnu matricu, a od profilne matrice ponovo matricu motiva. Ponavljanjem tog postupka, mogli bismo iterirati bez potrebe da se vezujemo za bilo kakav redosled podniski u nekoj od niski iz *Dna*. Ostalo je još da razmislimo kako bismo znali kada da stanemo.

Pretpostavljajući da krećemo od neke nasumične matrice motiva sa lošim, tj. velikim skorom, pretpostavljajući da se krećemo ka dobrom rešenju koje ima mali skor, kao i da to činimo postepeno gradeći sve bolja, tj. verovatnija rešenja pomoću profilnih matrica, prirodno je da

pomislimo da će se skor smanjivati iz iteracije u iteraciju. Dakle, iteriraćemo dokle god nam se skor smanjuje.

Ova ideja je zapravo osnova algoritma *RandomizedMotifSearch*. Njegova suština leži u tome da u velikom broju iteracija ažuriramo matricu motiva i profilnu matricu tako da dobijamo sve verovatniji skup motiva, zaustavljajući se kada postignemo najniži skor. Ovaj algoritam je ilustrovan sledećim pseudokodom:

```

1 RandomizedMotifSearch(Dna, k, t)
2   randomly select k-mers Motifs = (Motif1, ..., Motift) in each string from Dna
3   bestMotifs ← Motifs
4   while forever
5     Profile ← Profile(Motifs)
6     Motifs ← Motifs(Profile, Dna)
7     if Score(Motifs) < Score(bestMotifs)
8       bestMotifs ← Motifs
9     else
10    return bestMotifs

```

Radi lakšeg razumevanja, hajde da ilustrujemo rad ovog algoritma na jednom primeru. Jedan takav, dosta uprošćen primer možemo videti na slici 2.9. Prikazan je algoritam *RandomizedMotifSearch* koji konvergira u jednoj iteraciji.

Neka je dat skup *Dna* od 5 niski dužine 10 karaktera, sa ubaćenim (4,1) motivom. Na slici 2.9 u gornjem levom uglu, prikazan je skup *Dna*, tako što su velikim slovima ispisani karakteri instanci ubacenog motiva: ACCT, ATGT, GCGT, ACGA i AGGT, crnom bojom označene mutacije na motivima, a plavom nemutirani karakteri motiva.

Prvi korak jeste određivanje početne vrednosti matrice *Motifs*. Odabratemo slučajno 5 pozicija (u opsegu od 1 do 7) na kojima će početi 4-grami kojima ćemo inicijalizovati matricu *Motifs*, i neka su to pozicije 7, 5, 1, 2 i 7. 4-grami koji počinju na ovim pozicijama: taac, GTct, ccgG, acta i AGGT, čine početnu matricu *Motifs*. Sledeci korak jeste izgradnja profilne matrice na osnovu matrice *Motifs*. Prebrojavanjem svakog karaktera u svakoj koloni matrice *Motifs* možemo odrediti *count* matricu, a zatim deljenjem svakog broja iz *count* matrice sa brojem niski, tj. 5 u našem primeru, dobijamo profilnu matricu.

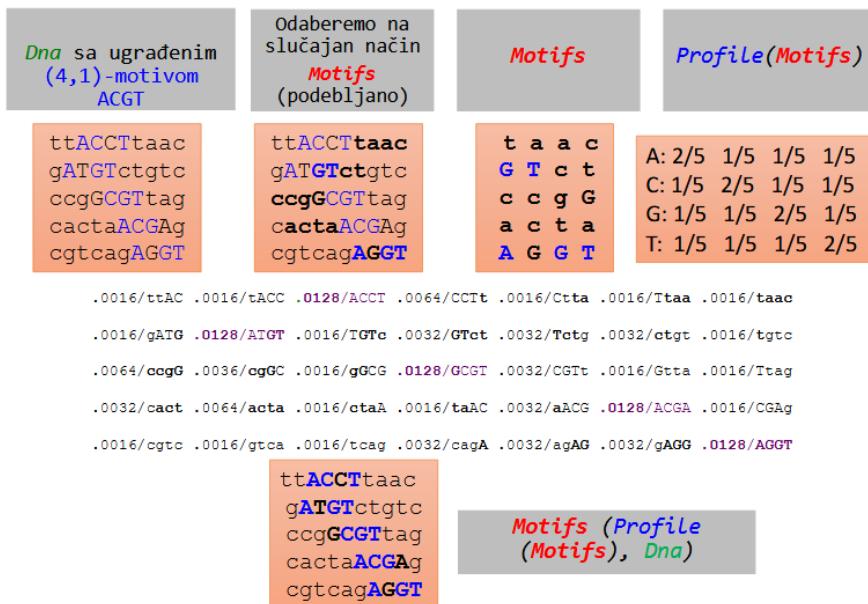
Sada kada imamo profilnu matricu, možemo izračunati nove najverovatnije motive u niskama iz *Dna*, i ažurirati matricu *Motifs*. Postupak računanja matrice motiva prikazan je na slici 2.12 u središnjem delu. Za svaku nisku iz skupa *Dna*, odredićemo verovatnoće svih njenih podniski, na osnovu profilne matrice, i iz svake niske odabrati po jednu koja ima najveću verovatnoću (na slici su prikazane ljubičastom bojom).

Sledeci korak bi bio da ponovo napravimo profilnu matricu itd. Međutim, u našem primeru, vidimo da je to kraj. Dobili smo traženo rešenje. Treba napomenuti da je ovo veštački kreiran primer i da je u praksi broj iteracija dosta veći.

Velika prednost ovog algoritma je u nasumičnom izboru početnog skupa motiva, što nam omogućava da ga pokrećemo iznova i izaberemo najbolje rešenje.

Gibsovo sempliranje

Algoritam *RandomizedMotifSearch* je dobar i efikasan, ali ima jednu malu manu. Matrica motiva se potpuno menja iz iteracije u iteraciju, što može dovesti do toga da se izmene i neki dobro izračunati redovi. To svakako ne želimo. Ako bismo u svakoj iteraciji menjali samo po jedan motiv iz matrice, tj. posle manjih izmena pravili novu profilnu matricu i proveravali skor, mogli bismo bolje da kontrolišemo to kvarenje već nameštenih motiva. Ova modifikacija algoritma *RandomizedMotifSearch* se naziva Gibbsovo sempliranje. Na prvi pogled deluje kao da povećava broj iteracija i usporava konvergenciju, ali to što su promene manje, nikako ne znači da su manje i efikasne. Ako idemo pravo sitnjim koracima, stižemo brže i lakše na cilj, nego ako krupnim

Slika 2.9: Primer rada algoritma *RandomizedMotifSearch*

koracima idemo levo-desno. Osnovna razlika Gibsovog sempliranja u odnosu na algoritam *RandomizedMotifSearch* se nalazi u sadržaju iterativnog koraka. Umesto da od cele matrice motiva pravimo profilnu matricu i na osnovu nje ažuriramo celu matricu motiva, sada ćemo nasumično odabrati jedan motiv koji ćemo izbaciti, profilnu matricu praviti od ostatka matrice, a ažuriranje matrice motiva vršiti samo nad izbačenim redom u matrici. Postupak kreiranja same profilne matrice i ažuriranja motiva je potpuno isti kao u algoritmu *RandomizedMotifSearch*. Umesto pseudokodom, ovaj put ćemo algoritam prikazati u koracima:

1. Formirati *Motifs* izborom jednog k-grama iz svake sekvene **na slučajan način**
2. **Na slučajan način** odabrati jedan od k-grama i ukloniti ga iz *Motifs*; označimo sekvencu kojoj taj k-gram pripada sa *RemovedSequence*
3. Kreirati profilnu matricu *Profile* od preostalih k-grama u *Motifs*
4. Za svaki k-gram iz *RemovedSequence*, izračunati $Pr(k-mer | Profile)$; na taj način dobijamo $n - k + 1$ verovatnoća: $p_1, p_2, \dots, p_{n-k+1}$
5. **Bacimo kockicu** sa $n - k + 1$ strana kod koje je verovatnoća da će pasti na i-tu stranu proporcionalna verovatnoći p_i
6. Odredimo k-gram iz sekvene *RemovedSequence* kao onaj koji ima najveću verovatnoću i dodamo ga u *Motifs*
7. Ponavljamo korake 2-6

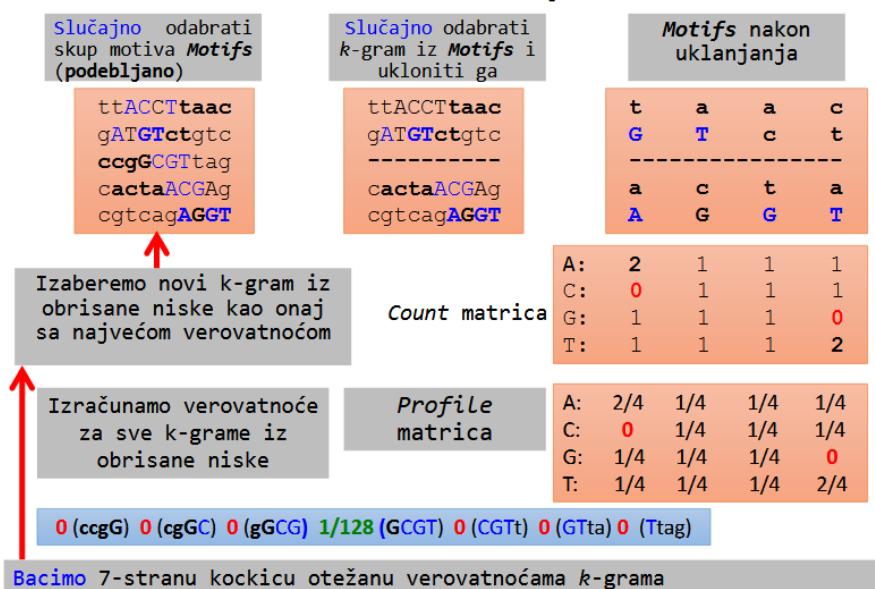
Radi lakšeg razumevanja, prikazaćemo i ovaj algoritam na istom primeru kao i *RandomizedMotifSearch*. Skica rada algoritma je prikazana na slici 2.13.

Neka je ponovo dat isti skup *Dna* od 5 niski dužine 10 karaktera, sa ubačenim (4,1) motivom. Na slici 2.10 u gornjem levom uglu, prikazan je skup *Dna*, tako što su velikim slovima ispisani karakteri instanci ubačenog motiva: ACCT, ATGT, GCGT, ACGA i AGGT, crnom bojom označene mutacije na motivima, a plavom nemutirani karakteri motiva. Prvi korak je isti kao i u algoritmu *RandomizedMotifSearch*: određivanje početne vrednosti matrice *Motifs*. Odabraćemo

slučajno 5 pozicija (u opsegu od 1 do 7) na kojima će početi 4-grami kojima ćemo inicijalizovati matricu *Motifs*, i neka su to pozicije 7, 5, 1, 2 i 7. 4-grami koji počinju na ovim pozicijama: taac, GTct, ccgG, acta i AGGT, čine početnu matricu *Motifs*.

Sledeći korak jeste izbacivanje jednog motiva. Nasumičnim izborom jednog broja iz opsega (1,5), odredili smo motiv iz reda 3 da bude izbačen. Matrica *Motifs* sada izgleda kao na slici 2.10 u gornjem desnom uglu. Naredni korak je izgradnja profilne matrice na osnovu matrice *Motifs*. Ovaj postupak je takođe isti kao u algoritmu *RandomizedMotifSearch*, samo je skup nad kojim radimo za jedan 4-gram manji. Prebrojavanjem svakog karaktera u svakoj koloni matrice *Motifs* možemo odrediti *count* matricu (slika 2.10, centar-desno), a zatim deljenjem svakog broja iz *count* matrice sa brojem motiva, tj. 4 u našem primeru, dobijamo profilnu matricu.

Sada kada imamo profilnu matricu, možemo izračunati novi najverovatniji motiv u trećoj niski iz *Dna*, i ponovo popuniti treći red u matrici *Motifs*. Vidimo da je podniska GCGT jedina sa pozitivnom verovatnoćom, pa samim tim i najvećom, tako da njome popunjavamo treći red matrice *Motifs*. Sledeći korak bi bio da ponovo izbacimo nasumično odabrani motiv, napravimo profilnu matricu itd. Iterirali bismo sve dok skor ne prestane da se smanjuje.



Slika 2.10: Primer rada algoritma Gibsovo sempliranje

Međutim, sada ćemo se ovde zaustaviti kako bismo primetili nešto mnogo važno. Kada smo računali verovatnoće podniski kandidata za zamenu trećeg motiva, dobili smo samo jednu nenulu vrednost od 7. To nije dobro. Jedna nula u profilnoj matrici može anulirati verovatnoće koje su sve do množenja sa njom bile prilično velike. Zbog toga one nisu baš poželjne. Ako zastanemo malo i razmislimo, shvatićemo da nisu previše ni realne, tj. da ne odražavaju realnu verovatnoću, već da su u velikom broju slučajeva uzrokovane malim obimom skupa nad kojim računamo. Zbog toga, modifikacija nula u profilnoj matrici predstavlja vrlo dobru i poželjnu modifikaciju ovog algoritma, a ta modifikacija se sprovodi korišćenjem Laplasovog pravila.

Mala napomena: Setimo se da smo već pominjali Laplasovo pravilo kao dobro poboljšanje i *GreedyMotifSearch* i *RandomizedMotifSearch* algoritama. Zapravo, ovo je korisna modifikacija svakog algoritma koji se zasniva na računanju profilne matrice i korišćenju njenih vrednosti.

Pre nego pređemo na modifikaciju, naglašićemo još da se prednost Gibsovog algoritma ogleda u najmanjem skoru, kao i to da je mana ovog algoritma zaglavljivanje u lokalnom minimumu usled pretrage ograničenog skupa rešenja.

Laplasovo pravilo kao poboljšanje Gibsovog semplera

Davne 1650. godine, Oliver Kromvel je napisao: "Molim Vas, tako Vam Hrista, mislite da je moguće da grešite". Ovu izjavu, je uobličio Denis Lindli i nazvao Kromvelovo pravilo: "Treba da ostavimo malu mogućnost da Sunce sutra neće izaći". Nešto novija replika bi se mogla naći u seriji Smolvil i rečima Leksa Lutora: "Uvek ostavljam malu mogućnost da nisam u pravu. Tako me ništa ne može iznenaditi. Šuština Kromvelog pravila leži u tome da, ukoliko potpuno odbacimo mogućnost nekog događaja, sasvim sigurno ostavljamo mogućnost pogreške. Zbog toga, treba izbegavati verovatnoće 0 i 1.

Za poboljšanje Gibsovog semplera (zapravo se može primeniti na bilo koji algoritam koji računa profilnu matricu), izvršićemo malu modifikaciju algoritma primenom Laplasovog pravila. Primenjujemo Laplasovo a ne Kromvolevo pravilo, samo iz razloga notacije i matematičkog zapisa. Laplasovo pravilo je samo preciznija (matematička) formulacija Kromvelovog pravila, sa istom suštinom.

Laplasovo pravilo: U malim skupovima podataka uvek postoji šansa da se događaj koji je moguć ne desi. Slučajni algoritmi uvode pseudovrednosti koje povećavaju verovatnoće retkih događaja i eliminisu frekvencije jednake nuli zabeležene na osnovu iskustva.

Suština Laplasovog pravila leži u tome da, ukoliko znamo da se događaj nekada u prošlosti desio, on ne može imati verovatnoću nula. Stoga u računanje, pored vrednosti dobijenih u našem eksperimentu, ubacujemo i dve pseudovrednosti: događaj se desio i događaj se nije desio.

Računanje uslovne verovatnoće bez primene Laplasovog pravila bi bilo:

Ako su X_1, \dots, X_{n+1} uslovno nezavisne slučajne logičke promenljive (neuspeh 0, uspeh 1), tada: $\Pr(X_{n+1} = 1 | X_1 + \dots + X_n = s) = s/n$

Računanje uslovne verovatnoće sa primenom Laplasovog pravila bi bilo:

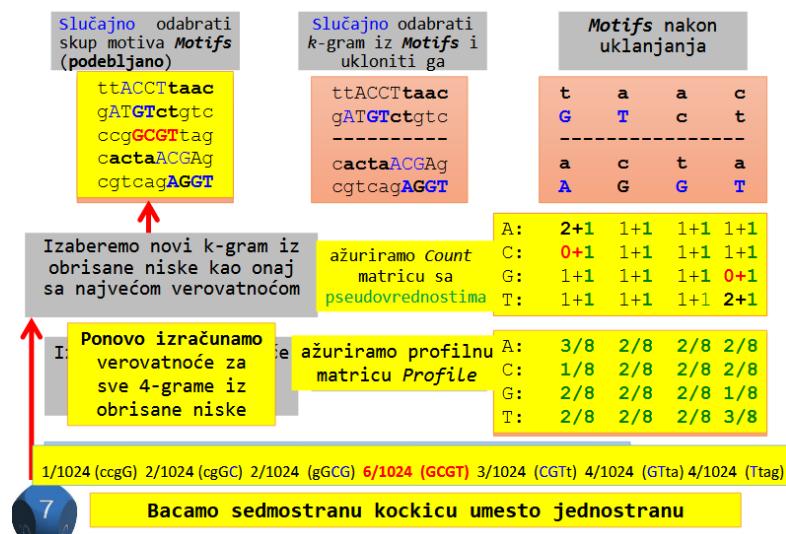
Ako su X_1, \dots, X_{n+1} uslovno nezavisne slučajne logičke promenljive (neuspeh 0, uspeh 1), tada: $\Pr(X_{n+1} = 1 | X_1 + \dots + X_n = s) = (s+1)/(n+2)$

Modifikacija Gibsovog algoritma primenom Laplasovog pravila je sadržana u dva mala koraka. Prva izmena je u okviru računanja *count* matrice, gde se vrednost svakog polja uvećava za 1. To je zapravo pseudovrednost: događaj se desio. Pod događaj, podrazumeva se pojava određenog karaktera na određenoj poziciji u matrici motiva. Druga izmena je u okviru računanja profilne matrice, zato što *count* matrica izgleda nešto drugčije.

Modifikaciju Gibsovog algoritma ćemo prikazati samo na primeru, jer je skup koraka u suštini isti i ne prikazuje najbolje samu modifikaciju. Koristićemo isti primer na kome smo već demonstrirali rad Gibsovog semplera. Na slici 2.11 prkazan je rad modifikovanog Gibsovog semplera. Možemo primetiti da *count* i profilna matrica izgledaju nešto drugčije (dodata su pseudovrednosti), kao i da je rezultat 7 ne-nula verovatnoća.

2.3.6 Koji princip odabrat?

Odgovor na ovo pitanje je - nema pravila. Nekada će se bolje pokazati jedan, a nekada drugi. Najbolje je da isprobamo više algoritama i vidimo koji se najbolje ponaša u našem slučaju. Možemo koristiti već zapažene prednosti i mane u izboru, kao na primer biranje *Median string* algoritma pri malim vrednostima k , ali isprobavanje je ipak najpouzdaniji pristup.



Slika 2.11: Primer rada Gibsovog sempliranja sa primenom Laplasovog pravila

Glava 3

Kako složiti genomske slagalice od milion delova?

U ovom poglavlju predstavićemo dati problem i prikazati kako možemo primeniti grafovske algoritme nad problemom slaganja genomske slagalice. Poglavlje će započeti pričom o sekvencirajućem genoma. Do sada smo videli šta za nas znači pojam DNK – da se podsetimo, to je niska karaktera nad abecedom $\Sigma = \{A, C, G, T\}$. Biološki posmatrano, DNK je molekul koji se nalazi u svakoj ćeliji svakog organizma i da je u njemu zapisan način pravilnog razvoja i funkcionalnosti svakog organizma.

3.1 Šta je sekvenciranje genoma?

Sa biološke strane, genom jednog organizma predstavlja njegov genetski materijal. Pod genetskim materijalom podrazumevamo delove genoma koji se nasleđuju koji se nasleđuju i koji svi predstavnici jedne vrste dele u velikoj meri. Kod većine organizama, genetski materijal je sadržan u DNK, odnosno, genom je ekvivalentan sa DNK molekulima. Kod nekih drugih organizama koji su u manjini, kao što su, na primer, virusi, važi da oni ne sadrže DNK i njihov genetski materijal se nalazi u ribonukleinskoj kiselini – RNK – o kojoj će biti reči u nastavku.

Kod čoveka, genom sadrži oko tri milijarde nukleotida. Dakle, sa računarske strane posmatrano genom je niska karaktera nad abecedom $\Sigma = \{A, C, G, T\}$. Kompleksnost organizma nije u relaciji sa veličinom genoma. Genomi nekih organizama su i stotinu puta veći od humanog genoma. Na primer, jedna vrsta amebe ima 670 milijardi nukleotida ili jedna vrsta kaktusa koja raste u Japanu ima 150 milijardi nukleotida.

Sekvenciranje genoma podrazumeva otkrivanje sastava genoma. U pitanju je eksperimentalni proces – da bismo saznali šta se nalazi u sastavu jednog genoma, potreban nam je uzorak tkiva odgovarajuće vrste. U nastavku dajemo kratak pregled razvoja sekvenciranja genoma.

3.1.1 Kratka istorija sekvenciranja genoma

Kao što smo videli, sekvenciranje genoma je eksperimentalni proces, za koji je neophodna veoma sofisticirana tehnologija. Pre svega možemo govoriti o razvoju fizičko-hemijskih tehnika koje bi dovele do mogućnosti saznavanja sastava genoma. Walter Gilbert i Frederick Sanger su 1977. godine razvili nezavisne metode sa sekvenciranje genoma, za koje su, 1980. godine, podelili Nobelovu nagradu. Međutim, iako su njihovi metodi bili pionirski u ovoj oblasti, njihove metode za sekvenciranje su bile veoma skupe – za sekvenciranje humanog genoma je bilo potrebno 3 milijarde dolara.

Krajem 2000-ih Sanger metodom je sekvencioniran veliki broj genoma. Visoka cena je bila ograničavajući faktor i za dalji napredak je bila neophodna nova tehnologija sekvencioniranja.

NGS (skr. *Next Generation Sequencing*) predstavlja metode nove generacije sekvencioniranja, odnosno, novu generaciju mašina sekvencera koji vrše sekvencioniranje. *Illumina*, jedan od proizvođača sekvencera, smanjuje trošak sekvencioniranja humanog genoma sa 3 milijarde na 10 hiljada dolara. Kompanija *Complete Genomics* otvara genomsku fabriku u Silikonskoj dolini koja sekvencionira stotine genoma mesečno. Pekinški genomski institut (*Beijing Genome Institute*, skr. *BGI*) preuzima *Complete Genomics* 2013. godine i postaje najveći svetski centar za sekvencioniranje genoma. Na slici ?? prikazano je kako se cena sekvencioniranja menjala godinama.

3.1.2 Sekvenciranje ličnih genoma

Prirodno je zapitati se o značaju poznavanja sastava genoma. Što je tiče biljaka, neke od primena sekvenciranja su: razvoj novih biljnih vrsta u poljoprivredi, određivanje pogodnog podneblja za neku biljnu vrstu, u farmaciji, i dr. Međutim, najznačajnija primena je u sekvenciranju ličnih genoma.

Genomi se kod različitih ljudi razlikuju na malom broju pozicija (u proseku sadrže jednu mutaciju na hiljadu nukleotida). Ova razlika je odgovorna za, na primer, različite visine kod ljudi, da li će imati sklonost ka visokom holesterolu ili ne, za veliki broj genetskih bolesti, itd.

Godine 2010. Nicholas Volker je postao prvo ljudsko biće čiji je život spašen zahvaljujući genomskom sekvencioniranju. Lekari nisu mogli da postave tačnu dijagnozu i morali su da ga podvrgnu velikom broju operacija pokušavajući da je utvrde. Sekvenciranje je otkrilo retku mutaciju na jednom genu (XIAP) koja je bila povezana sa oštećenjem njegovog imunog sistema. Ovo otkriće je navelo lekare na adekvatnu terapiju koja je rešila problem.

3.2 Eksplozija u štampariji

Razmotrimo sledeći primer. Neka imamo hiljadu kopija istog izdanja novina na jednoj gomili, a ispod njih postavljen je dinamit. Upalimo fitilj i zamislimo da nije sve samo izgorelo već da se raspršilo u milione delića papira. Kako možemo da iskoristimo te delice da bismo saznali koje su bile vesti iz tog izdanja? Ovaj problem nazvaćemo *Problem novina* (videti sliku ??).

Problem novina je mnogo teži nego što izgleda. Kako smo imali više kopija istog izdanja, i kako smo izgubili neki deo informacija prilikom eksplozije, ne možemo samo da prilepimo delice novina kao da su slagalica. Umesto toga, potrebno je da preklopimo delove različitih novina kako bismo rekonstruisali jedan primerak.

Određivanje redosleda nukleotida u genomu, odnosno sekvenciranje genoma, predstavlja bitan problem u bioinformatici. Već smo pomenuli da dužine genoma variraju – humani genom je dugačak oko 3 milijarde nukleotida, dok je genom jednoćelijskog organizma *Amoeba dubia* čak 200 puta duži.

Razmotrimo sada povezanost problema novina i sekvenciranja genoma. Kopije izdanja u problemu novina odgovaraju ono predstavlja ulaz u sekvencerima – uzorak tkiva. Moderne mašine za sekvenciranje ne mogu da procitaju ceo genom nukleotid po nukleotid od početka do kraja (kao što bismo pročitali knjigu). Mogu samo da iseckaju genom i generišu njegova kratka očitavanja (engl. *reads*). Kako to zapravo funkcioniše?

Na slici ?? ilustrovan je proces sekvenciranja. Sekvencer dobija milione kopija istog genoma. Zatim vrši očitavanja čime dobijamo delice odnosno kratke podnische. Neki delovi odnosno očitavanja biće izgubljena (kao delići novina u eksploziji, dakle gubimo deo informacija). Očitavanja su izmešana i ono što nam sekvencer daje je zapravo kolekcija podniski koje treba spojiti u jednu. Sastavljanje genoma nije isto kao i slaganje slagalice – moramo da koristimo preklapajuća očitavanja da bismo rekonstruisali genom.

3.3 Problem sekvenciranja genoma

Do sada smo videli šta predstavlja sekvenciranje genoma, koja je njegova biološka podloga i kako se on definiše kao biološki problem. Pređimo sada na formulisanje računarskog problema sekvenciranja genoma kao problem rekonstrukcije niske.

Problem sekvencioniranja genoma: Rekonstruisati genom na osnovu očitavanja.

Ulaz: Kolekcija niski *Reads*.

Izlaz: Niska *Genome* rekonstruisana na osnovu *Reads*.

Ovo nije dobro definisan problem. Potrebno je uvesti dodatne pojmove kako bismo uspeli da problem sekvencioniranja genoma predstavimo kao problem rekonstrukcije niske.

Definišemo pojam *k-gramski sastav niske* na sledeći način. *k*-gramske sastave niske *Text*, u označi $\text{Composition}_k(\text{Text})$, predstavlja kolekciju podniski dužine *k* niske *Text*, pri čemu su u kolekciju uključeni duplikati. Na primer, neka je *Text* = TAATGCCATGGGATGTT. Njen 3-gramske sastave niske *Text* izgleda:

```

1 Composition_3(TAATGCCATGGGATGTT) =
2           TAA
3           AAT
4           ATG
5           TGC
6           GCC
7           CCA
8           CAT
9           ATG
10          TGG
11          GGG
12          GGA
13          GAT
14          ATG
15          TGT
16          GTT

```

odnosno, ako kolekciju uredimo po leksikografskom poređenju:

```

1 Composition_3(TAATGCCATGGGATGTT) =
2   AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT

```

Sada možemo malo bolje da definišemo problem.

Problem rekonstrukcije niske: Rekonstruisati nisku na osnovu njenog *k*-gramskog sastava.

Ulaz: Kolekcija *k*-grama.

Izlaz: Niska *Genome* takva da je $\text{Composition}_k(\text{Genome})$ ekvivalentno kolekciji *k*-grama.

Započnimo prvo sa naivnim pristupom rešavanju ovog problema. Odaberimo jedan *k*-gram za početni. Zatim nižemo ostale tako da se sufiks poslednjeg odabranog poklopi sa prefiksom nekog od preostalih *k*-grama. Pri tome, ako ima više takvih *k*-grama, biramo proizvoljan jedan. Na ovaj način možemo doći do rešenja, ali je veoma skupo. Pri tome, velika je šansa da ćemo se negde zaglaviti (tj. nijedan od preostalih *k*-grama neće biti kandidat za nadovezivanje na tekuću nisku) ili zbog izbora početnog *k*-grama ili zbog izbora nekog od preostalih *k*-grama kada je postojalo više odgovarajućih. Sledeći primer ilustruje ovaj problem.

Neka nam je dat sledeći 3-gramske sastav: AAT ATG ATG ATG CAT CCA GAT GCC GGA GGG GTT TAA TGC TGG TGT. Treba rekonstruisati nisku koja ima takav sastav. Biramo početni 3-gram, neka to bude na primer TAA. Zatim na njega treba nadovezati 3-gram koji počinje njegovim sufiksom dužine 2, odnosno onaj 3-gram koji ima prefiks AA. U našem slučaju, postoji jedan takav 3-gram i njega nadovezujemo na tekuću nisku, tako da sada imamo TAAT. Zatim biramo 3-gram čiji je prefiks AT. Ovog puta imamo 3 kandidata, ali, na našu sreću, sva tri su isti 3-grami, ATG. U takvom slučaju nije bitno koji smo odabrali, jer su svi jednakci. Nadovezujemo ga na tekuću nisku i dobijamo TAATG. Tražimo 3-grame sa prefiksom TG, koji do sad nisu upotrebljeni. Ponovo pronalazimo 3 kandidata. Međutim, u ovom slučaju, svi kandidati predstavljaju različite 3-grame, a to su TGC, TGG i TGT. Naivni pristup kaže da biramo jedan od njih, i recimo da smo odabrali TGT i dobili nisku TAATGT. Sada nam je potreban 3-gram sa prefiksom GT i tu dolazi do zaglavljivanja. Imamo još 3-grama koji nisu iskorišćeni za rekonstrukciju niske, ali nijedan ne možemo da iskoristimo u ovom trenutku. U takvim situacijama treba se vratiti u nazad do koraka u kom je bilo više kandidata.

3.4 Rekonstrukcija niske kao problem Hamiltonove putanje

Videli smo da nam naivni pristup ne odgovara i moramo smisliti bolje rešenje. Mogli bismo da iskoristimo znanja iz teorije grafova za rešavanje ovakvog problema. U tom slučaju, prvi zadatak je da našu nisku predstavimo u vidu grafa.

3.4.1 Genom kao putanja

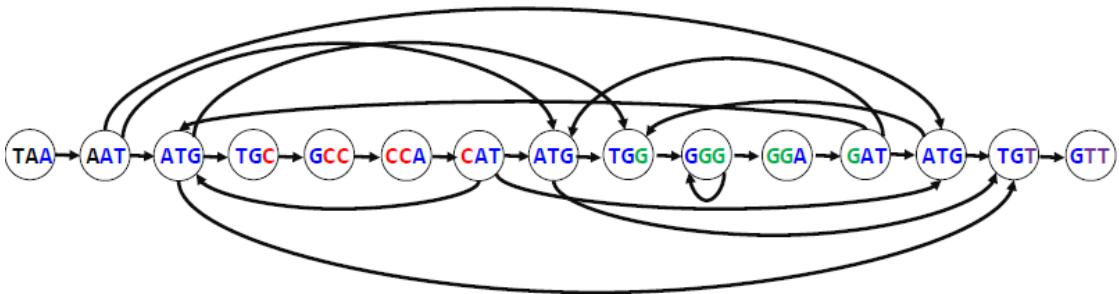
Vratimo se na prethodni primer. Dat nam je naredni 3-gramske sastav:

```

1 Composition_3(TAATGCCATGGGATGTT) =
2           TAA
3           AAT
4           ATG
5           TGC
6           GCC
7           CCA
8           CAT
9           ATG
10          TGG
11          GGG
12          GGA
13          GAT
14          ATG
15          TGT
16          GTT

```

Ovakav 3-gramske sastav možemo predstaviti kao graf na sledeći način. Svakom čvoru u grafu odgovara jedan od k -grama. Zatim, potrebne su nam grane koje će povezati te čvorove. Dva čvora su povezana usmerenom granom ako izlazni čvor ima sufiks koji je jednak prefiksu ulaznog čvora te grane, kao što je prikazano na slici 3.1.



Slika 3.1: Grafo koji odgovara 3-gramskom sastavu niske *TAATGCCATGGGATGTT*.

Jasno je da postoji više puteva u ovom grafu. Postavlja se pitanje – da li možemo da pronađemo genomsku putanju u ovom grafu, od svih koje postoje?

Podsetimo se šta je Hamiltonova putanja. Hamiltonova putanja je putanja koja posećuje svaki čvor u grafu tačno jednom. To je upravo ono što nam je potrebno za rešavanje problema. Svaki čvor predstavlja jedan k -gram i potrebno nam je da svi k -grami budu uključeni u rekonstruisanu nisku tačno jednom.

Problem Hamiltonove putanje: Naći Hamiltonovu putanju u grafu.

Ulas: Grafo.

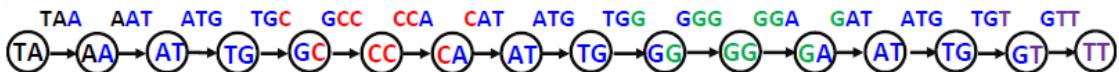
Izlaz: Putanja koja posećuje svaki čvor u grafu tačno jednom.

Iako deluje kao da smo rešili sve probleme, zapravo smo naišli na još jednu veliku prepreku. Naime, pronalaženje Hamiltonovog puta u grafu je NP-kompletan problem, što znači da ne postoji algoritam koji ga rešava u polinomijalnom vremenu. U tom slučaju, moramo da se vratimo na početak, a to je predstavljanje k -gramskega sastava grafovom.

3.5 Rekonstrukcija niske kao Ojlerove putanje

U prethodnoj sekciji, k -grame smo predstavili čvorovima u grafu i u njemu tražili Hamiltonov put, odnosno, put koji obilazi svaki čvor tačno jednom. Videli smo da za taj problem još uvek nije poznat efikasan algoritam pa se sada pitamo kako možemo izmeniti graf tako da ne zahteva traženje Hamiltonove putanje.

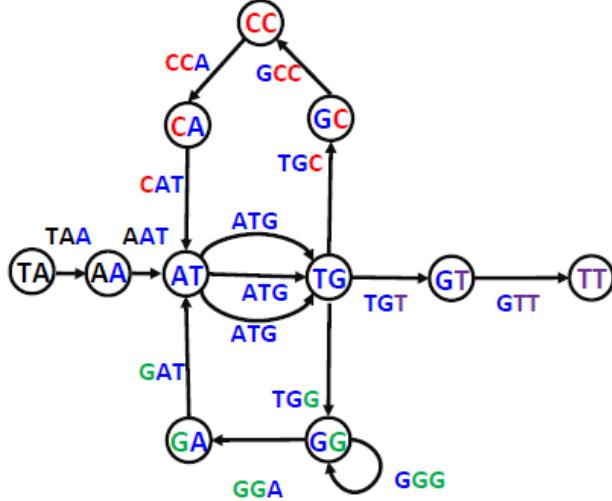
Ono što se javlja kao ideja jeste obeležavanje grana umesto čvorova. Dakle, svaka grana biće obeležena jednim k -gramom. Izlazni čvor biće obeležen prefiksom k -grama te grane, dok će ulazni čvor biti obeležen sufiksom istog tog k -grama. Slika 3.2 ilustruje ovaj postupak za nisku *TAATGCCATGGGATGTT*.



Slika 3.2: Grafo koji odgovara 3-gramskom sastavu niske *TAATGCCATGGGATGTT*. Grane su obeležene 3-gramima, a čvorovi 2-gramima koji predstavljaju prefiks i sufiks.

Primećujemo da su neki čvorovi obeleženi identično (na primer, imamo tri čvora sa oznakom *AT*). Sve čvorove koji imaju istu oznaku treba spojiti u jedan, pri čemu zadržavamo sve grane koje su ulazile u taj čvor ili su izlazile iz njega. Ponavljamo postupak dokle god imamo čvorove

koji imaju istu oznaku i na kraju dobijamo graf koji nazivamo *De Brojnov graf*. Slika 3.3 ilustruje De Brojnov graf dobijen ovom procedurom od polaznog grafa sa slike 3.2.



Slika 3.3: De Brojnov graf koji odgovara niski *TAATGCCATGGGATGTT*.

Ovime smo dobili novu reprezentaciju niske pomoću grafa. Prirodno se postavlja naredno pitanje – gde se nalazi niska *Genome* u ovoj reprezentaciji grafa? Kako nam se 3-grami sada nalaze na granama, a ne u čvorovima, potrebno je da pronađemo putanju u grafu koja prolazi sve grane tačno jednom. Takav put nazivamo *Ojlerova putanja*. Srećom, algoritam za pronalaženje Ojlerove putanje u grafu nije NP-kompletan i možemo efikasno da je pronađemo.

Problem Ojlerove putanje: Pronaći Ojlerovu putanju u grafu.

Ulaz: Graf.

Izlaz: Putanja koja poseće svaku granu u grafu tačno jednom.

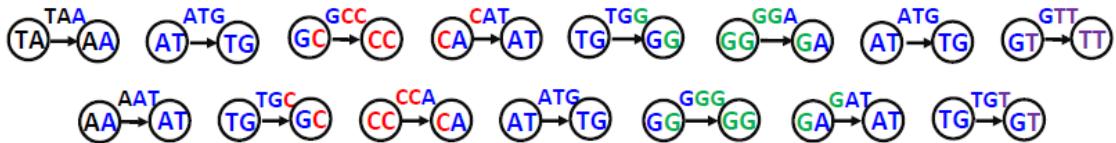
Sada znamo kako možemo da dobijemo nisku kada znamo De Brojnov graf koji odgovara njenom k -gramskom sastavu. Međutim, konstruisali smo De Brojnov graf na osnovu genoma, ali u realnim primenama, genom je nepoznat.

3.6 De Brojnovi grafovi na osnovu kolekcije k -grama

Videli smo kako možemo od zadate niske pronaći De Brojnov graf. Nažalost, u primenama nije nam poznata niska, ali znamo njen k -gramski sastav. Postavlja se pitanje kako možemo konstruisati De Brojnov graf od k -gramskog sastava niske.

Za svaki k -gram pravimo dva čvora i jednu granu – oznaka grane je upravo taj k -gram, oznaka izlaznog čvora je prefiks, a ulaznog čvora je sufiks datog k -grama. Time dobijamo nepovezani graf kao na slici 3.4. Zatim lepimo identične čvorove sve dok ne dobijemo graf čiji svi čvorovi imaju različite oznake. Na slici 3.5 dat je jedan korak ovog postupka, međutim, tu nije kraj jer i dalje postoje čvorovi sa istim oznakama.

Po završetku postupka dobijamo de Brojnov graf koji je isti kao onaj koji smo dobili kada smo znali nisku, odnosno, graf na slici 3.3. Svaka grana je označena jednim k -gramom, a svaki čvor je označen prefiksom, odnosno, sufiksom odgovarajuće izlazne, odnosno, ulazne grane, redom. Naravno, čvorovi koji imaju identične oznake su zaledjeni.



Slika 3.4: Svaki k-gram prestavljen je pomoću dva čvora i jedne grane.



Slika 3.5: Prvi korak u postupku lepljenja čvorova. Napomenimo da postupak nije završen jer treba zlepiti čvorove sa identičnim oznakama (na primer, AT).

3.7 Ojlerova teorema

Za rešavanje problema Ojlerove putanje koji smo predstavili u prethodnoj sekciji možemo iskoristiti rešenje narednog problema. Ovaj problem je značajan jer postoji teorema koja ga prati, a koja određuje uslove za njegovo rešavanje.

Problem Ojlerovog ciklusa: Pronaći Ojlerov ciklus u grafu.

Ulaz: Graf.

Izlaz: Ciklus koji posećuje svaku granu u grafu tačno jednom.

Kažemo da je graf Ojlerov ako sadrži Ojlerov ciklus. Ispostavlja se da postoje određene karakteristike koje određuju da li je graf Ojlerov. Uvedimo pojmove povezan graf i balansiran graf. Kažemo da je graf *povezan* ako za svaku dva čvora postoji putanja koja ih povezuje. Graf je *balansiran* ako za svaki čvor važi da mu je izlazni stepen jednak ulaznom. Naredna teorema govori o potrebnim i dovoljnim uslovima da graf bude Ojlerov.

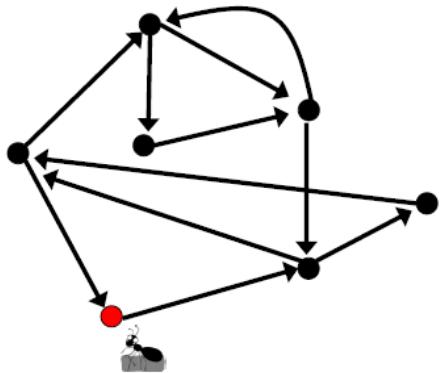
Teorema 3.1 (Ojlerova teorema). *Svaki Ojlerov graf je balansiran. Svaki povezan graf i balansiran graf je Ojlerov.*

3.7.1 Dokaz Ojlerove teoreme

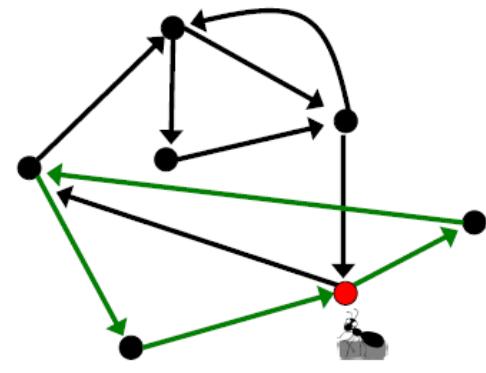
Neka nam je dat povezan balansiran graf. Da bismo pokazali da graf sadrži Ojlerov ciklus, postavićemo mrava na bilo koji od čvorova tog grafa, kao na slici 3.6. Zašto baš mrav? Poznato je da mravi nikada ne idu istim putem dva puta pa smo sigurni da će naš mrav proći svaku granu tačno jednom.

Puštamo mrava da slučajno odabira grane kojima će se kretati. Ako je veoma pametan, običi će svaku granu jednom i vratiće se u početni čvor. Međutim, velike su šanse da nije veoma pametan i da će se u nekom čvoru zaglaviti, odnosno, neće imati granu koju već nije obišao.

Da li mrav može da se zaglavi u bilo kom čvoru? Ispostavlja se da može da se zaglavi samo u početnom čvoru (jer je graf balansiran). U trenutku kada se zaglavio on je napravio ciklus. Samo, taj ciklus nije Ojlerov jer još uvek nije obišao sve grane. Ideja je da odabere drugačiji početni čvor iz kog će krenuti obilazak. Koji čvor će izabrati? Treba da izabere čvor iz ciklusa koji ima izlaznih grana koje još uvek nije obišao (slika 3.7).



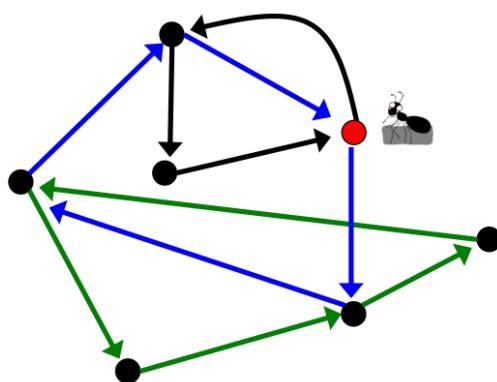
Slika 3.6: Mrav je postavljen u crveni čvor i odatle kreće obilazak povezanog balansiranog grafa.



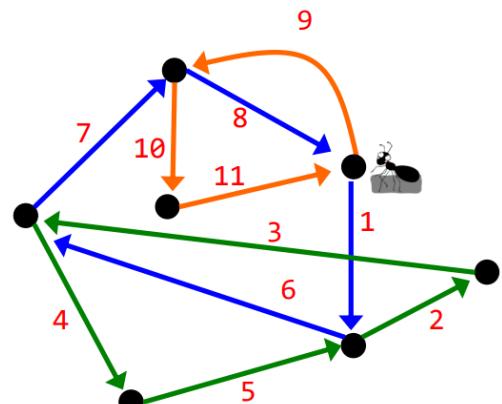
Slika 3.7: Mrav se zaglavio i pokušava ponovo iz drugog čvora koji pripada ciklusu i ima izlazne grane koje nisu posećene.

Sada, mrav pokušava ispočetka iz novog čvora. Prvo obilazi ciklus koji je već pronašao u prethodnom pokušaju, a zatim nastavlja obilazak preko neposećenih grana. Na taj način, ciklus se uvećava dok se ne dođe do Ojlerovog. Ukoliko se ponovo zaglavi (slika 3.8) ponovo bira novi početni čvor, obilazi pronađeni ciklus (koji i dalje nije Ojlerov) i tako sve dok ne uspe da obide sve grane (slika 3.9).

Dokaz Ojlerove teoreme daje primer konstruktivnog dokaza, koji ne dokazuje samo željeni rezultat, već pruža metod za konstrukciju onoga što nam je potrebno. Ukratko, pratili smo kretanje mrava dok nije pronašao Ojlerov ciklus u povezanim balansiranim grafom, što je sumirano u algoritmu `EulerianCycle`.



Slika 3.8: Mrav se ponovo zaglavio i pokušava od novog čvora.



Slika 3.9: Mrav je konačno uspeo da pronađe dobar početni čvor i Ojlerov ciklus. Grane su obeležene redosledom kojim su posećene.

```

1 EulerianCycle(BalancedGraph)
2 begin
3     form a Cycle by randomly walking in BalancedGraph (avoiding already visited
        ↳ edges)
4     while Cycle is not Eulerian
    
```

```

5      select a node newStart in Cycle with still unexplored outgoing edges
6      form a Cycle' by traversing Cycle from newStart and randomly walking
7      ↘ afterwards
8      Cycle ← Cycle'
9      return Cycle
9 end

```

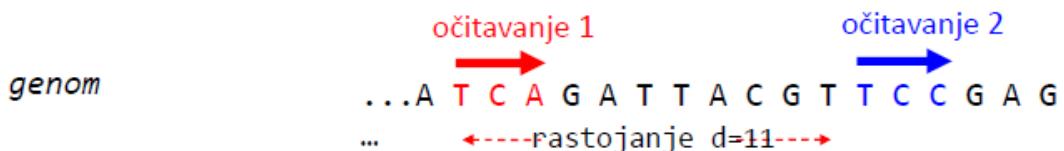
Ovaj algoritam radi u linearном vremenu. Da bi se zaista postigla ta efikasnost, potrebne su efikasne struktutre podataka za održavanje ciklusa koje mrav pronalazi kao i za liste neiskorišćenih grana za svaki čvor i lista čvorova u trenutnom ciklusu koji imaju neiskorišćene grane.

3.8 Sastavljanje parova očitavanja

Predstavimo kao da su svi naši problemi rešeni. Međutim, može se javiti više Ojlerovih putanja u grafu. Srećom, i za ovo imamo jednostavno rešenje.

3.8.1 DNK sekvenciranje sa parovima očitavanja

Imamo više identičnih kopija genoma i na slučajnim pozicijama sećemo genom na fragmente iste dužine *InsertLength*. Zatim generišemo *parove očitavanja* – dva očitavanja sa krajeva svakog fragmenta na jednakoj, fiksiranoj udaljenosti. Pod *uparenim k-gramom* podrazumevamo par *k*-grama na fiksiranom rastojanju *d* u genomu. *Upareni k-gramske sastav*, u oznaci *PairedComposition_k(Text)*, sastoji se od svih *k*-grama niske *Text* i njihovih parova.



Slika 3.10: TCA i TCC na rastojanju $d = 11$ čine jedan upareni 3-gram.

Dajmo jedan primer. Neka imamo nisku TAATGCCATGGGATGTT, i upareni 3-gram TAA i GCC. Upareni *k*-gramske sastav date niske prikazan je na slici 3.11.

TAA	AAT	ATG	TGC	GCC	CCA	CAT	ATG	TGG	GGG	GGA
GCC	CCA	CAT	ATG	TGG	GGG	GGA	GAT	ATG	TGT	GTT
AAT	ATG	ATG	CAT	CCA	GCC	GGA	GGG	TAA	TGC	TGG
CCA	CAT	GAT	GGA	GGG	TGG	GTT	TGT	GCC	ATG	ATG

Slika 3.11: *PairedComposition* niske TAATGCCATGGGATGTT i njegov leksikografski poredak.

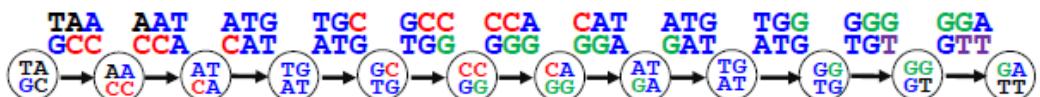
Sada možemo formulisati naredni problem.

Problem rekonstrukcije niske na osnovu parova očitavanja: Rekonstruisati nisku na osnovu njenih uparenih k -grama.

Ulaz: Kolekcija uparenih k -grama.

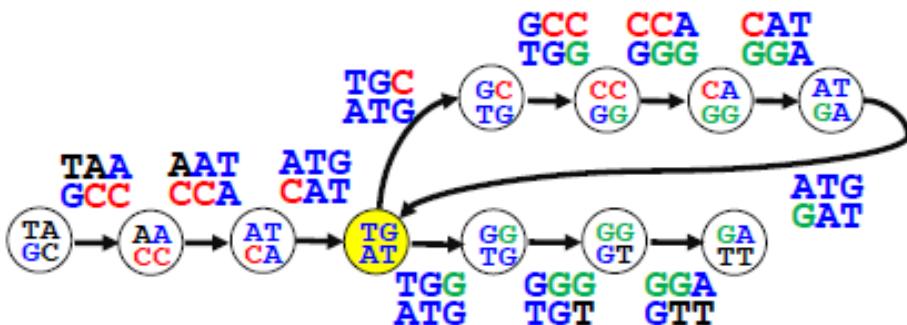
Izlaz: Niska $Text$ takva da je $PairedComposition_k(Text)$ jednak kolekciji uparenih k -grama.

Kako konstruisati upareni De Brojnov graf na osnovu uparenog k -gramskog sastava? Postupak je sličan prethodnom slučaju, kada nismo imali parove. Pretpostavimo da je dat genom (niska *Genome*). Posmatrajmo genom kao putanju u grafu obeleženom na osnovu njegovog uparenog k -gramskog sastava (videti sliku 3.12). Svaka grana obeležena je uparenim k -gramom, a svaki čvor uparenim prefiksom, odnosno, sufiksom k -grama.



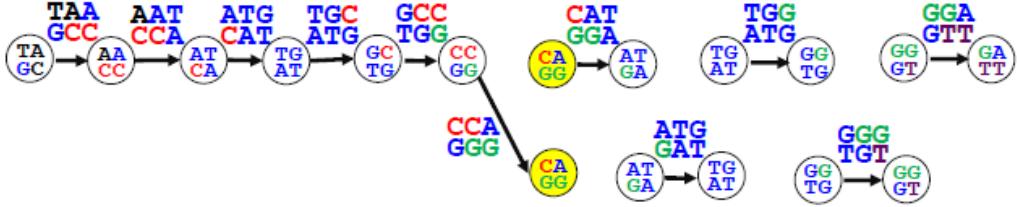
Slika 3.12: Grafički prikaz uparenog 3-gramskog sastava niske TAATGCCATGGGATGTT.

Potrebno je zlepiti čvorove sa istom oznakom, tako da svi čvorovi budu jedinstveno obeleženi. Postupak je identičan prethodnom slučaju, odnosno, kada nismo imali parove. Primetimo da sada imamo mnogo manje lepljenja jer imamo samo dva čvora sa istom oznakom (TG AT), (videti sliku 3.13).



Slika 3.13: Dva čvora sa oznakom (TG AT) spajaju se u jedan pri čemu su sve grane, incidentne sa tim čvorovima, očuvane.

Kao i u prethodnom slučaju, pretpostavili smo da je dat genom (niska *Genome*), što često nije slučaj. Posmatrali smo genom kao putanju u grafu obeleženom na osnovu njegovog uparenog k -gramskog sastava. Sada pretpostavimo da nije dat genom već samo upareni k -gramske sastav. Za svaki upareni k -gram pravimo dva čvora i jednu granu, zatim lepimo identične čvorove (videti 3.14), i na kraju dobijamo upareni De Brojnov graf, kao onaj na slici 3.13.



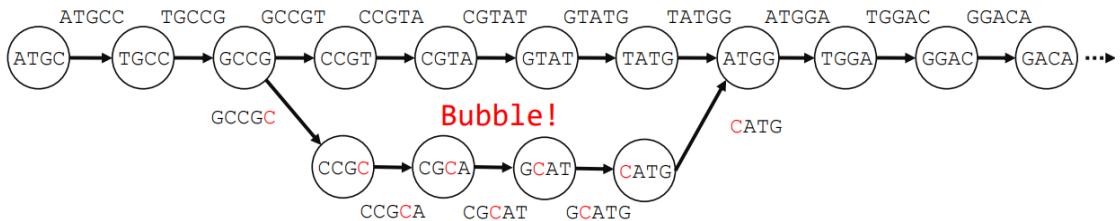
Slika 3.14: Konstrukcija uparenog De Brojnovog grafa na osnovu uparenih k -grama.

Dakle, upareni De Brojnov graf, na osnovu kolekcije uparenih k -grama, dobijamo tako što svaku granu označavamo jednim uparenim k -gramom. Zatim, svaki čvor označavamo prefiksima, odnosno, sufiksima odgovarajuće izlazne, odnosno, ulazne grane, redom. Na kraju, lepimo čvorove sa identičnim oznakama.

3.9 U realnosti

Ovde smo imali neke nerealne prepostavke:

- Savršena pokrivenost genoma očitavanjima (svaki k -gram iz genoma je očitan). Očitavanja dužine 250 nukleotida dobijena Illumina tehnologijom predstavljaju samo mali deo 250-grama unutar genoma. Rešenje je u razbijanju dobijenih očitavanja na kraće k -grame (kao na slici 3.16).
- Očitavanja ne sadrže greške. U ovom slučaju, ako bismo razbili na manje k -grame, onda bismo dobili više niski koje imaju pogrešno očitavanje. Postavljamo pitanje kako se ovakvi slučajevi manifestuju u konstrukciji DeBrojnovog grafa. Dolazi do stvaranja *balončića* (engl. *bubble*) u grafu (videti sliku 3.15). Jednostavan je slučaj kada govorimo o grešci na jednom očitavanju, međutim, ukoliko postoji više grešaka, onda dolazi do *eksplozije balončića* (videti sliku ??).
- Rastojanja između očitavanja u okviru parova očitavanja su egzaktna.
- itd.



Slika 3.15: Primer pojave balončića u De Brojnovom grafu usled pojave greške u očitavanju nukleotida T nukleotidom C.

atgccgtatggacaacgact	atgccgtatggacaacgact
atgccgtatg	atgcc
gccgtatgga	tgccg
gtatggacaa	gccgt
gacaacgact	ccgta
	cgtat
	gtatg
	tatgg
	atgga
	tggac
	ggaca
	gacaa
	acaac
	caacg
	aacga
	acgac
	cgact

Slika 3.16: Primer razbijanja dobijenih očitavanja na manje k-grame.

Glava 4

Kako sekvenciramo antibiotike?

U ovom poglavlju i dalje govorimo o sekvenciranju, ali ćemo proširiti pogled i pokazati različite načine za sekvenciranje peptida.

4.1 Otkriće antibiotika

Pre svega, krenućemo sa biološkim uvodom. Šta su to antibiotici? Sama reč antibiotik znači „onaj koji ubija život”, a tačnije, on predstavlja supstancu koja ubija bakterije. Kada ostavimo pomorandžu dugo negde gde je toplo, ona će da razvije čudne osobine kao što je bud. Šta to znači? Bud jeste jedna vrsta antibiotika što znači da se antibiotici nalaze u prirodi i da ih proizvode organizmi iz porodice gljiva (npr. bud) i bakterija.

Mi ćemo posmatrati antibiotike na molekularnom nivou koji nam govori od čega su oni zapravo izgrađeni. Od svih antibiotika posmatraćemo **tirocidin B1**, antibiotik koji proizvodi bakterija *Bacillus Brevis*. Tirocidin B1 na molekulskom nivou pripada *peptidima*, kratkim niskama aminokiselina, odnosno malim proteinima. Ovo je skok u odnosu na ono što smo do sada posmatrali – nukleotidne niske nad četverostrukom abzukom $\Sigma = \{A, C, G, T\}$, odnosno DNK.

Za DNK smo govorili da se pojavljuje u svakoj ćeliji svakog živog bića i da je veoma značajna supstanca jer sadrži recept (tačnije, nosi informaciju) za pravilno funkcionisanje i razvoj svakog živog bića. Da bi se svako živo biće pravilno razvijalo, neophodno je da njegove ćelije proizvode (sintetišu) u tačno određeno vreme određene supstance koje se nazivaju *proteini*. DNK nosi informaciju o tome kako treba neki protein da izgleda, od čega treba da se sastoji. Zašto je to bitno? Na primer, kada je dan, neke biljke treba da vrše fotosintezu, a za vršenje fotosinteze treba u samim ćelijama biljaka da se sintetišu određeni proteini.

Proteini su, nakon nukleinskih kiselina, druga značajna grupa molekula koja sa računarske tačke gledišta takođe predstavlja dugačke niske, ali ne nad abzukom od 4 karaktera, nego nad abzukom od 20 karaktera, a svaki karakter predstavlja molekul koji se naziva *aminokiselina*. Kao i nukleinske kiseline, aminokiseline se predstavljaju velikim latiničnim slovima $\{V, K, L, F, P, W, N, Q, Y, G, A, I, M, D, E, S, T, C, R, H\}$, a pored toga postoje i troslovne oznake $\{Val, Lys, Leu, Phe, Pro, Trp, Asn, Gln, Tyr, Gly, Ala, Ile, Met, Asp, Glu, Ser, Thr, Cys, Arg, His\}$. U prirodi postoji mnogo više od 20 aminokiselina, ali 20 njih najčešće učestvuje u sastavu proteina. DNK upravlja time kada će nastati protein u okviru ćelije. Recept za nastajanje svakog proteina je zapisan u DNK. Kako je taj recept zapisan, videćemo u nastavku.

Proteini se još nazivaju i *polipeptidi*. Dužina proteina je obično od 100 do nekoliko hiljada aminokiselina (proteini su kraći od genomske sekvence). Tirocidin B1 je peptid jer se sastoji iz malog broja aminokiselina, svega deset – $V, K, L, F, P, W, F, N, Q, Y$. Problem sekvenciranja antibiotika jeste problem određivanja aminokiselina koje ulaze u sastav tog antibiotika. U prethodnom poglavlju smo sekvencirali genom, ali tehnike iz prethodnog poglavlja nećemo moći da koristimo u sekvenciranju tirocidina B1, što će biti objašnjeno u poglavlju [4.2](#).

4.2 Kako bakterije prave antibiotike?

Pre rešavanja problema sekvenciranja antibiotika, govorićemo o zanimljivoj i kompleksnoj temi, a to je tema – kako se prave proteini? Već je pomenuto da se u okviru DNK nalazi recept za pravljenje proteina. Sada je vreme da se zapitamo kako je sve to zapisano u DNK pomoću A, C, G, T .

Znamo da je DNK dvostruki lanac čiji su krajevi označeni sa 5' i 3' (uvek čitamo lanac od 5' ka 3'). DNK jeste jedna vrsta nukleinskih kiselina koje postoji u ćeliji živih bića. Pored nje, postoje i različite vrste **ribonukleinskih kiselinina**, odnosno **RNK**. Ribonukleinske kiseline nisu predstavljene dvostrukim lancem, već jednostrukim. One se sastoje od nukleotida A, C, G, U . Umesto timina, kod RNK se pojavljuje nukleotid uracil koji se označava sa U .

DNK se **prepisuje** u RNK. Šta to znači? Da bi nastali proteini, neophodno je da se na osnovu dva lanca od DNK konstruiše RNK molekul. Pošto se RNK molekul sastoji od istih nukleotida kao i DNK, osim timina, onda kažemo da formiranje RNK na osnovu DNK predstavlja jednostavno *prepisivanje* nukleotida iz oba lanca DNK, uz zamenu nukleotida T sa nukleotidom U . Drugi naziv za prepisivanje jeste *transkripcija*. Ovo je prvi korak, i dalje nismo došli do aminokiseline, i dalje smo u abzuci nukleotida. RNK predstavlja jedan međukorak između DNK i samog proteina.

Drugi korak jeste *prevodenje*, odnosno *translacija*, prepisanog RNK u proteine. Imamo 4 nukleotida A, C, G, U i treba njih da prevedemo u nisku od 20 mogućih aminokiselina. To znači da mora da postoji neko preslikavanje, nekakav kod koji prevodi neke k -grame nukleotida u aminokiseline. Nad abzukom od 4 nukleotida postoji 16 različitih 2-grama, tj. bigrama. Da li možemo tih 16 bigrama da preslikamo u 20 aminokiselina? Tačnije, da li dva nukleotida možemo da preslikamo u jednu aminokiselinu? Ne možemo, jer moramo za svaku aminokiselinu da znamo koji je bigram označava.

Pošto ne možemo to da uradimo sa bigramima, da li možemo sa 3-gramima? Svi mogućih 3-grama nad abzukom od 4 nukleotida ima 64. To znači da će svaka od aminokiselina imati svoj kod, a neke od njih će možda imati i više kodova, tj. više trigrama može da ukazuje na jednu aminokiselinu. To je u redu, bitno je da je naša funkcija „na”, ne mora da bude „1 – 1”. Ali kako napraviti funkciju? Ne možemo svojevoljno da dodelimo trigramima određene aminokiseline. Ta funkcija je unapred utvrđena, odnosno, prirodnom determinisana i dokazana. U nastavku, koristićemo drugačiji naziv za naše 3-grame.

Definicija 4.1. *Kodon predstavlja jedan 3-gram (triplet) nukleotida.*

Preslikavanje o kojem je do sada bilo reči se naziva *genetski kod* i on je prikazan na slici ??.

Definicija 4.2. *Genetski kod predstavlja preslikavanje skupa kodona u skup aminokiselina.*

Vidimo da se, na primer, kodon *UGG* preslikava u aminokiselinu *Trp(W)*, dok se više kodona, *CUA, CUC, CUG, CUU, UUA, UUG*, preslikava u jednu aminokiselinu *Lys(L)*.

Redom, kodone iz RNK preslikavamo u aminokiseline. Ali, kako znamo da je kraj nekog proteina? U genetskom kodu je i tako nešto kodirano. Postoje tzv. **stop kodoni** koji označavaju da iza njih nema više aminokiselina koje čine taj protein. Ti stop kodoni su *UAA, UAG, UGA*.

Dolazimo do jednog veoma značajnog biološkog aksioma – **centralne dogme molekularne biologije**. Ona govori da se transkripcijom na osnovu DNK može dobiti RNK, a translacijom se iz RNK, na osnovu genetskog koda, dobijaju proteini. Ovu teoriju je predstavio Francis Krik (eng. *Francis Crick*) i prikazana je na slici ??.

Ono što želimo da saznamo jeste koje aminokiseline i kojim redom ulaze u sastav našeg malog peptida tirocidina B1. Pošto se on sastoji iz 10 aminokiselina, to znači da ga čine 30 nukleotida u genomu bakterije *Bacillus Brevis* koje će da se prepišu u RNK i da se prevedu iz RNK u tirocidin B1. Hiljade različitih 30-grama se može prevesti u tirocidin B1 jer se u genetskom kodu različiti kodoni mogu prevesti u istu aminokiselinu. Na slici 4.1 su prikazani neki od takvih 30-grama. Vidimo da oni nisu previše slični.

GT $\textcolor{green}{TAA}$ $\textcolor{red}{ATT}$ $\textcolor{black}{ATT}$ $\textcolor{black}{TCC}$ $\textcolor{black}{TGGTT}$ $\textcolor{green}{TAATCA}$ $\textcolor{red}{ATAT}$

GT $\textcolor{blue}{CAA}$ $\textcolor{red}{GCT}$ $\textcolor{black}{TTT}$ $\textcolor{blue}{CCC}$ $\textcolor{black}{TGGTT}$ $\textcolor{blue}{CAACCA}$ $\textcolor{blue}{GTAC}$

GT $\textcolor{red}{AAA}$ $\textcolor{red}{ACT}$ $\textcolor{black}{ATT}$ $\textcolor{black}{TCC}$ $\textcolor{black}{TGGTT}$ $\textcolor{red}{CAATCA}$ $\textcolor{red}{ATAT}$

Slika 4.1: Neki od 30-grama koji se mogu prevesti u tirocidin B1.

Treba uzeti u obzir da translacija može početi na bilo kojoj poziciji u genomu. To znači da bismo za datu poziciju, ako gledamo 30-gram, mogli da imamo 6 različitih tzv. **čitajućih okvira**, tj. 6 varijanti prepisivanja u RNK i onda prevodenja. Tri čitajuća okvira potiču iz tri nukleotida iz jednog kodona iz jednog prevedenog RNK lanca (ako krenemo da čitamo od prvog nukleotida, to je jedan čitajući okvir, iz drugog nukleotida je drugi čitajući okvir, iz trećeg nukleotida je treći čitajući okvir, a ako pročitamo od četvrtog nukleotida, to je već isti čitajući okvir kao prvi jer tu kreće novi kodon), a isto tako za drugi RNK lanac imamo tri čitajuća okvira sa druge strane.

Naš peptid tirocidin B1 jeste *cikličan*. Tih 10 aminokiselina koje ga čine idu nekim redom, ali su one povezane u krug, tako da imamo ukupno 10 različitih **linearnih reprezentacija** za tirocidin B1 u zavisnosti od toga koja nam je prva aminokiselina bila u samom receptu DNK. Koju god linearnu reprezentaciju pronađemo, rešili smo problem.

Ne odustajemo od pronalaženja 30-grama u genomu bakterije *Bacillus Brevis* koji kodira bar jednu linearnu reprezentaciju od svih 10 koje čine tirocidin B1. Prepostavimo da imamo na raspolaganju veoma moćan računar i neograničeno vreme. Doći ćemo do jednog čudnog rezultata. Nećemo uspeti da pronađemo nijedan 30-gram u genomu bakterije *Bacillus Brevis* koji kodira bar jednu linearnu reprezentaciju proteina tirocidina B1. Zašto? Stvari se komplikuju. Na ovom primeru je pokazano da centralna dogma ne važi uvek, odnosno ne važi da svaki protein u ćeliji nastaje na osnovu recepta koji je zapisan u DNK. Centralna dogma govori da se proces transkripcije izvršava pod uticajem enzima koji se zove *RNK polimeraza*, a translacija RNK u protein se vrši u ćelijskoj organeli koja se naziva *ribozom*. Postoje neki proteini koji ne nastaju na ovaj način, nego na specijalan način gde obično sekvenciranje genoma ne može da nam pomogne. Moramo da predložimo novi metod kako možemo da pronađemo odgovarajuću sekvencu aminokiselina.

Edvard Tejtum (eng. *Edward Tatum*), jedan od poznatih američkih genetičara, je 1963. godine inhibirao ribozom bakterije *Bacillus Brevis*. Šta ovo znači? S obzirom da se znalo da se u ribozomu vrši translacija RNK u protein, on je onemogućio da se bilo šta desi u ribozomu, isključio je funkcionalisanje te organele u ćeliji i očekivao je da se neće stvoriti nijedan protein, pa ni tirocidin B1. Međutim, suprotno očekivanjima, nastavljena je proizvodnja nekih peptida, uključujući i tirocidine. Ovo je bilo izuzetno iznenadjujuće otkriće.

Fric Lipman (eng. *Fritz Lipmann*), američko-nemački biohemičar, je 1969. godine pokazao da tirocidini spadaju u grupu **ne-ribozomalnih peptida (NRP-ova)**. To su peptidi za čiju sintezu nisu odgovorni ribozomi i RNK polimeraza već enzimi poznati pod nazivom **NRP sintetaze**, molekuli koji se takođe nalaze u ćeliji i utiču na različite procese koji se dešavaju u njoj. To znači da se stvaranje tirocidina razlikuje od većeg broja proteina u živim bićima.

Kako izgleda sinteza tirocidina B1 pomoću NRP sintetaze? Postoji veliki broj različitih NRP sintetaza, nije samo jedna odgovorna za stvaranje svih mogućih NRP-ova, nego za svaki ne-ribozomalni peptid postoji odgovarajuća NRP sintetaza. Ona NRP sintetaza koja je odgovorna za stvaranje tirocidina B1 se sastoji od 10 različitih podjedinica koje nazivamo *moduli*.

Svaki modul je odgovoran za nadovezivanje jedne aminokiseline na budući molekul tirocidina B1. Svaki od modula privuče jednu aminokiselinsku i spoji je sa prethodnom, a poslednji korak jeste cirkularizacija – spajanje aminokiselina nastalih uz pomoć prvog i poslednjeg modula radi kreiranja cikličnog peptida.

4.3 Sekvenciranje antibiotika razbijanjem na komade

Pošto nam sekvenciranje genomske sekvene i pronalaženje odgovarajuće podnische koja je zadužena za translaciju u aminokiseline odgovarajućeg peptida ne može pomoći u sekvenciranju tirocidina B1 (jer on ne nastaje na osnovu informacije zapisane u DNK), postavljamo pitanje da li postoji način na koji možemo da sekvenciramo antibiotike. Moramo direktno da sekvenciramo peptid. Jedan od načina jeste **sekvenciranje razbijanjem na komade** i biće predstavljen u ovoj sekciji.

U sekvenciranju antibiotika može nam pomoći mašina koja se naziva **maseni spektrometar** i koju možemo opisati kao skupu molekularnu vagu. Šta on radi? Za početak ćemo da se zapitamo kako možemo da merimo težinu, tačnije masu molekula.

Pošto se molekuli sastoje od atoma, prvo treba da govorimo o masi pojedinačnog atoma. U atomima postoje protoni, neutroni i elektroni. Protoni i neutroni su približno iste mase, dok su elektroni izuzetno mali i gotovo zanemarljive mase. Zato masu jednog atoma možemo svesti na masu protona, odnosno neutrona koji učestvuju u izgradnji konkretnog atoma koji posmatramo, pa se i masa molekula može izračunati kao suma masa atoma koji ga čine. Maseni spektrometar vraća masu molekula izračunatu u **Daltonima**.

$$\begin{aligned} 1 \text{ Dalton}(Da) &\approx \text{masa jednog protona/neutrona} \\ \text{Masa molekula} &\approx \text{suma masa protona/neutrona} \end{aligned}$$

Posmatrajmo masu jedne aminokiseline, recimo glicina. Glicin ima hemijsku formulu C_2H_3ON . Ugljenik ima masu 12, vodonik ima masu 1, kiseonik 16, a azot 14. Na osnovu ovih masa računamo masu celog molekula glicina.

$$\text{masa}(C_2H_3ON) = 12 * 2 + 1 * 3 + 16 * 1 + 14 * 1 \approx 57Da$$

Simbol \approx koristimo jer nam je stvarna masa nešto malo drugačija od celobrojne mase koju dobijamo ovde. Stvarna masa glicina iznosi 57.02Da. Podrazumevaćemo da je masa celog molekula upravo celobrojna masa koju smo dobili. Tabela masa svih aminokiselina data je u tabeli 4.1.

Tabela 4.1: Tabela masa 20 aminokiselina poređanih rastuće prema celobrojnim masama.

G	A	S	P	V	T	C	I,L	N	D	K,Q	E	M	H	F	R	Y	W
57	71	87	97	99	101	103	113	114	115	128	129	131	137	147	156	163	186

Primećujemo da neke aminokiseline imaju iste mase, npr. I i L, K i Q, pa za 20 aminokiselina imamo 18 celobrojnih masa.

Kada imamo mase aminokiselina, možemo da se zapitamo koja je masa tirocidina B1. Znajući da se tirocidin B1 sastoji iz 10 aminokiselina ovim redom: VKLFPWFNQY, masu računamo koristeći tabelu 4.1.

$$\text{masa(tirocidina B1)} = 99 + 128 + 113 + 147 + 97 + 186 + 147 + 114 + 128 + 163 = 1322$$

Vratimo se na maseni spektrometar o kojem smo govorili ranije. Zamislimo da imamo kratak peptid koji se sastoji od samo 4 aminokiseline *NQEL*. Uzorak ovog peptida ubacimo u maseni spektrometar. Šta se u njemu dešava? U njemu se nekim hemijskim procesima, u čije detalje nećemo ulaziti, generišu svi podpeptidi ulaznog peptida. Sa računarske tačke gledišta, maseni spektrometar generiše od zadate niske *NQEL* sve moguće podniske, podrazumevajući da je ulazni peptid cikličan. Tako se generišu podniske dužine jedan: *N, Q, E, L*, podniske dužine dva: *NQ, QE, EL, LN* i podniske dužine tri: *NQE, QEL, ELN, LNQ*. Maseni spektrometar za svaki od dobijenih podpeptida može da odredi molekulsku masu. Ono što mi dobijemo kao izlaz iz masenog spektrometra nisu podpeptidi, mi ne znamo koje podpeptide je dobio, niti kojim podpeptidima je prudružena koja masa. Izlaz iz masenog spektrometra jeste samo niz masa! Taj izlazni niz može da sadrži i dva ista broja jer neki podpeptidi mogu da imaju istu masu. Kako bismo formulisali računarski problem, moramo da definišemo šta je to *teorijski spektar*.

Definicija 4.3. *Teorijski spektar peptida predstavlja niz masa svih mogućih podpeptida tog peptida, uključujući nulu kao masu praznog peptida i masu samog peptida.*

Poznajući sastav peptida, lako možemo da izračunamo teorijski spektar. Suprotan smer je težak, odnosno teško je da na osnovu teorijskog spektra zaključimo kako je izgledao peptid. Upravo ovo jeste problem sekvenciranja ciklopeptida.

Problem sekvenciranja ciklopeptida. Rekonstruisati ciklični peptid na osnovu njegovog teorijskog spektra.

4.3.1 Sekvenciranje ciklopeptida grubom silom

Kada dobijemo spektar iz masenog spektrometra, najveća masa će označavati masu celog peptida. Želimo prvo da generišemo sve peptide sa masom jednakom masi celog peptida, zatim da za svaki tako generisan peptid formiramo teorijski spektar i uporedimo ga sa datim spektrom. Algoritam grube sile za problem sekvenciranje ciklopeptida dat je u nastavku.

```

1 BFCyclopeptideSequencing(Spectrum)
2 begin
3     // ParentMass(Spectrum) jeste najveća masa u spektru Spectrum
4     for every Peptide with Mass(Peptide) equal to ParentMass(Spectrum)
5         if Spectrum == CycloSpectrum(Peptide)
6             output Peptide
7 end

```

Vidimo da u ovom algoritmu ispitujemo sve kandidate (peptide sa istom masom kao dati peptid). Koliko imamo takvih kandidata? Grafik koji oslikava odgovor na ovo pitanje dat je na slici ??.

Vidimo da je ovaj algoritam grube sile eksponencijalne složenosti. Pod uslovom da imamo dovoljno brz računar, ovako nešto bismo i mogli da izračunamo. Ali, šta bi bili nedostaci ovog algoritma grube sile? Možemo da imamo dva peptida sa istom masom, a da su potpuno različiti. Na primer, peptid *NQEL* i peptid *TMDH* imaju masu 484. Kako možemo da isključimo pogrešan peptid? Za oba ova peptida možemo da generišemo teorijski spektar. Ispostavlja se da su njihovi spektri potpuno različiti. Želimo da ovu informaciju iskoristimo u sledećem pristupu rešavanja problema sekvenciranja ciklopeptida. Cilj nam je da ne idemo grubom silom već da ogroman broj kandidata od samog početka odstranimo.

4.3.2 *Branch-and-Bound* algoritam za sekvenciranje ciklopeptida

U ovom pristupu postepeno konstruišemo kandidate za rešenja od manjih linearnih peptida za razliku od prethodnog pristupa u kome smo odmah generisali ceo peptid koji postaje kandidat. Na taj način ćemo smanjiti ukupan broj linearnih peptida koje posmatramo. Ovakav pristup se koristi kod ***Branch-and-Bound* algoritama** koji će biti opisani u nastavku.

Kod *Branch-and-Bound* algoritama u celokupnom prostoru svih mogućih rešenja vršimo nekakva odsecanja. Počnemo od kratkog peptida dužine jedan, pa ga proširimo na sve moguće načine, odnosno, dodamo po jednu aminokiselinu i od toga napravimo sve moguće kandidate dužine dva. To je *branch grana* i predstavljena je na slici ???. *Bound grana* bi od postojećih kandidata, nastalih u *branch* granama, isključila neke potencijalne kandidate. *Bound grana* je predstavljena na slici ???. Postupak proširivanja i odsecanja ponavljamo sve dok ne dođemo do odgovarajućih vrednosti. Na ovaj način će nam ostati mnogo manje kandidata za rešenja nego u prethodnom pristupu grube sile.

Primenimo ovaj algoritam na konkretan problem. Recimo da nam je dat spektar

0 97 97 99 101 103 193 196 198 198 200 202 295 297 299 299 301 394 396 398 400 400 497.

Vidimo da se u datom spektru nalaze mase nekih aminokiselina, što nam govori koje aminokiseline ulaze u sastav traženog peptida. Te aminokiseline su *P, V, T, C* sa masama 97, 99, 101, 103. Ovo znači da možemo da počnemo ne sa svih 20 aminokiselina, nego sa 4 unigramma *P, V, T, C*. Ovo je unapred jedna *bound grana* jer smo 20 aminokiselina sveli na 4 kandidata aminokiselina. Zatim idemo na *branch grana*, širimo unigrame u sve moguće bigrame: *PA, PC, PD,..PY, VA, VC, VD,...,VY, TA, TC, TD,..., TY, CA, CC, CD,..., CY*. Proširujemo sa svih 20 aminokiselina jer ćemo kasniji videti da ovako zadati spektar jeste čisto teorijski spektar, a maseni spektrometar skoro nikada u praksi ne vraća teorijski spektar već spektar sa nekakvima greškama. Kako možemo da skratimo ovu listu, kako možemo da izvršimo korak *bound* u ovom trenutku? Posmatramo da li postoje odgovarajući bigrami koji se takođe pojavljuju u spektru. Zbog toga uvodimo pojam **konzistentnosti**.

Definicija 4.4. Za proizvoljan podpeptid p_1, \dots, p_n kažemo da je **konzistentan** sa spektrom S ukoliko se svaka masa iz teorijskog spektra podpeptida p_1, \dots, p_n nalazi u spektru S .

Na primer, *PV* je **konzistentno** sa spektrom ukoliko se masa od *P*, masa od *V* i masa od *PV* nalaze u spektru.

Konzistentnost ćemo koristiti u *bound* koraku, tačnije, izbacivši sve bigrame koji nisu konzistentni sa spektrom. Tako dobijemo listu konzistentnih bigrama *PV, PT, PC, VP, VT, VC, TP, TV, CP, CV* koju proširujemo u sve moguće 3-grame, a zatim svodimo na listu samo konzistentnih 3-grama. Postupak ponavljamo. Kada dođemo do liste konzistentnih pentagrama *PVCPT, PTPVC, PTPVC, PCVPT, VPTPC, VCPTP, TPCVP, CPTPV, CVPTP* vidimo da zapravo svi oni pokazuju na jedan isti ciklični peptid. Pseudokod opisanog algoritma dat je u nastavku.

```

1 CyclopeptideSequencing(Spectrum)
2 begin
3     Peptides = a set containing only the empty peptide
4     while Peptides is non-empty
5         // proširujemo sve peptide u skupu sa svim mogucim aminokiselinama
6         Peptides = Expand(Peptides)
7         for each Peptide in Peptides
8             // ParentMass(Spectrum) jeste najveca masa u spektru
9             if Mass(Peptide) = ParentMass(Spectrum)
10                if Cyclospectrum(Peptide) = Spectrum
11                    output Peptide
12                    remove Peptide from Peptides

```

```

13         else if Peptide is not consistent with Spectrum
14             remove Peptide from Peptides
15     end

```

Podsetimo se da je složenost algoritma grube sile, koji ovde pokušavamo da poboljšamo, eksponencijalna. Ispostavlja se da *Branch-and-Bound* algoritam takođe može biti eksponencijalne složenosti za neke peptide, ali je u praksi veoma brz.

4.4 Prilagođavanje sekvenciranja za spektre sa greškama

Spektar koji smo do sada definisali jeste teorijski spektar. Za razliku od njega, spektar koji izlazi iz masenog spektrometra, **eksperimentalni spektar**, često sadrže greške. O kakvim greškama se govori biće prikazano pomoću slike 4.2.

teorijski:	0	113	114	128	129	227	242	242	257	355	356	370	371	484	
eksperimentalni:	0	99	113	114	128		227		257	299	355	356	370	371	484

Slika 4.2: Primer teorijskog i eksperimentalnog spektra za *NQEL*.

Lažne mase jesu mase koje su na slici 4.2 prikazane zelenom bojom. To su mase koje su prisutne u eksperimentalnom spektru, ali nisu prisutne u teorijskom spektru.

Nedostajuće mase jesu mase koje su na slici 4.2 prikazane plavom bojom. To su mase koje se nalaze u okviru teorijskog spektra, ali ne i u okviru eksperimentalnog spektra.

Zbog pojave ovih otežavajućih okolnosti, tj. grešaka u spektru, neophodan je novi algoritam jer se kod dva predložena algoritma teorijski spektar peptida morao u potpunosti poklapati sa spektrom peptida koji je predstavljao rešenje problema. Sada moramo da olabavimo taj uslov pa uvodimo pojam **skor peptida**.

Definicija 4.5. *Skor peptida pokazuje koliko masa njegov teorijski spektar deli sa eksperimentalnim spektrom.*

Tako, za sliku 4.2, skor iznosi 11. Želimo da skor bude što veći. S obzirom da imamo nov način upoređivanja, moramo da unapredimo naš *Branch-and-Bound* algoritam, konkretno korak odsecanja.

Uzmimo primer golfa. U golfu, kada igrači prođu prvi krug takmičenja, u sledećim prolazima dalje samo igrači koji su konkurentni, oni koji imaju šanse da nešto osvoje. To znači da možemo da kažemo da nam je odsecanje takvo da, na primer, prva tri igrača sa najboljim skorom idu dalje, a ukoliko imamo još neke igrače koji imaju isti skor kao poslednji igrač, onda i oni prolaze dalje. Znači, zadržavaju se tri najbolja igrača „*with ties*”. Ovakav sistem primenjen na *Branch-and-Bound* algoritam prikazan je u sledećem pseudokodu.

```

1 LeaderboardCyclopeptideSequencing(Spectrum, N)
2 begin
3     Leaderboard = set containing only the empty peptide
4     LeaderPeptide = empty peptide
5
6     while Leaderboard is non-empty
7         // prosirujemo sve elemente koji se nalaze u okviru skupa Leaderboard
8         Leaderboard = Expand(Leaderboard)
9         for each Peptide in Leaderboard
10            // ParentMass(Spectrum) predstavlja najveću masu u spektru Spectrum
11            if Mass(Peptide) == ParentMass(Spectrum)
12                if Score(Peptide, Spectrum) > Score(LeaderPeptide, Spectrum)

```

```

13         LeaderPeptide = Peptide
14     else if Mass(Peptide) > ParentMass(Spectrum)
15         remove Peptide from Leaderboard
16     // odsecamo kandidate iz Leaderboard na osnovu njihovog skora
17     Leaderboard = Trim(Leaderboard, Spectrum, N)
18
19     output LeaderPeptide
20 end
21
22 Trim(Leaderboard, Spectrum, N, AminoAcid, AminoAcidMass)
23 begin
24     for j=1 to |Leaderboard|
25         Peptide = j-th peptide in Leaderboard
26         // LinearScore jeste skor nad linearnim spektrom
27         LinearScores[j] = LinearScore(Peptide, Spectrum)
28
29     sort Leaderboard according to the dec order of scores in LinearScores
30     sort LinearScores in dec order
31
32     for j=N+1 to |Leaderboard|
33         if LinearScores[j] < LinearScores[N]
34             remove all peptides starting from the j-th peptide from Leaderboard
35         return Leaderboard
36
37     return Leaderboard
38 end

```

Leaderboard pristup omogućava da bolje definišemo za eksperimentalni spektar kod *Branch-and-Bound* algoritma onu bound fazu kada treba da izbacimo neke kandidate.

4.4.1 Testiranje na spektru tirocidina B1

U ovom delu razmatraćemo rezultate testiranja na *Spectrum₁₀*, spektru sa 10% lažnih/nedostajućih masa. Kada primenimo *LeaderboardCyclopeptideSequencing* na spektar sa 10% loših vrednosti, tada zaista dobijemo peptid sa najvišim skorom *VKLFPWFNQY* koji odgovara tirocidinu B1. Međutim, ukoliko uzmemo spektar *Spectrum₂₅* koji ima 25% lažnih i nedostajućih vrednosti, spektar koji se još više udaljava od teorijskog spektra, onda se peptid sa najvišim skorom *VKLFPADFNQY* razlikuje od peptida *VKLFPWFNQY* koji želimo da dobijemo.

Ovo znači da *LeaderboardCyclopeptideSequencing* algoritam radi dobro kada nam je eksperimentalni spektar malo različit od teorijskog.

4.5 Od 20 do više od 100 aminokiselina

U ovoj sekciji biće reči o poboljšanju našeg algoritma uz uvođenje premlaza koje postoje u stvarnosti, a koje smo do sada zanemarivali da bismo dali neke početne načine za rešavanje.

Kada smo govorili o proteinima, rekli smo da 20 aminokiselina najčešće učestvuje u njihovoj izgradnji i da su za nas, sa računarske tačke gledišta, proteini niske nad abzukom od 20 karaktera i da postoji još veliki broj aminokiselina nezavisno od izgradnje proteina u celijama živilih bića. U gentskom kodu postoji samo za tih 20 aminokiselina, i u tabeli celobrojnih masa aminokiselina postoje mase samo za iste te aminokiseline.

S obzirom da u ovom poglavlju razmatramo NRP peptide, peptide koji ne nastaju prema pravilima centralne dogme, onda ovi peptidi mogu da sadrže i neke nestandardne aminokiseline, one aminokiseline koje se ne nalaze među standardnih 20 aminokiselina. Na primer, tirocidin B

sadrži nestandardnu aminokiselinu *Ornitin* (*Orn*). Za Ornitin ne postoji nukleotidni triplet u okviru genetskog koda na osnovu koga se ova aminokiselina dobija i ne postoji celobrojna masa u tabeli celobrojnih masa za aminokiseline. S ozbirom na to, možemo da pretpostavimo da bilo koji ceo broj između 57 i 200 (koliko nam iznosi najmanja i najveća masa standardnih aminokiselina) može biti masa neke nestandardne aminokiseline. Ovako nešto može da izgleda kao grubo ograničenje, ali je eksperimentalno potvrđeno da većina masa svih mogućih aminokiselina pripada ovom intervalu.

Spektar u kome nismo ograničeni na tabelu od samo 18 celobrojnih masa, već uzimamo u obzir da bilo koji celi broj između 57 i 200 može da označava neku aminokiselinu, nazivamo **prošireni spektar**. Kada primenimo Leaderboard algoritam na prošireni spektar sa 10% lažnih i nedostajućih masa, peptid koji dobijemo *VKLFPWFN* – 98 – 65 sadrži neke vrednosti za mase koje ne odgovaraju nijednoj aminokiselini. Pošto Leaderboard algoritam ovde ne daje ispravne vrednosti, moramo da primenimo jedan sasvim novi princip.

4.6 Spektralna konvolucija

Kod algoritma sa proširenim spektrom podrazumeva se da svi celi broevi između 57 i 200 odgovaraju masama aminokiselina. To znači da razmatramo 144 ili više (znamo da jednoj masi može da odgovara više aminokiselina, a sa druge strane postoje vrednosti kojima ne odgovara nijedna) aminokiselina u koje spadaju i standardne i nestandardne aminokiseline. Želimo da smanjimo broj aminokiselina koje razmatramo. Posmatrajmo eksperimentalni spektar za *NQEL*

0 99 113 114 128 227 257 299 355 356 370 371 484.

Mi znamo da je $\text{Mass}(E) = 129$ i vidimo da u spektru ne postoji ta vrednost. Sa druge strane, u spektru postoji $\text{Mass}(QE) = 257$ i $\text{Mass}(Q) = 128$. Razlika ove dve mase daje vrednost 129. Ova vrednost već deluje kao dobra vrednost za nedostajuću masu. U spektru postoji još ovakvih slučajeva. Recimo, $\text{Mass}(ELN) - \text{Mass}(LN) = 356 - 227 = 129$ i $\text{Mass}(NQEL) - \text{Mass}(LNQ) = 484 - 355 = 129$. Obe ove razlike ukazuju na masu od *E* koja nedostaje. Uvodimo tabelu koja se naziva **spektralna konvolucija**.

Definicija 4.6. *Spektralna konvolucija* je tabela koja pokazuje absolutnu vrednost razlike između svake dve mase u spektru.

Primer spektralne konvolucije za spektar čije su lažne vrednosti označene sa „false” prikazan je na slici ??.

Na preseku svake vrste i kolone u spektralnoj konvoluciji upisana je apsolutna vrednost razlike celobrojnih masa. Kako iskoristiti spektralnu konvoluciju? Tražimo vrednosti razlika koje se pojavljuju najveći broj puta, a da se nalaze između 57 i 200. Obojene vrednosti na slici ?? se pojavljuju veći broj puta. To su vrednosti 99, 113, 114, 128 i 129. Ove vrednosti odgovaraju masama aminokiselina, redom, *V, L, N, Q, E*. Od 5 najčešćih aminokiselina u konvoluciji 4 čine peptid *NQEL*.

Kako bi izgledao unapređeni algoritam za sekvenciranje ciklopeptida ukoliko uzmemo u obzir i nestandardne aminokiseline, odnosno proširenu tabelu celobrojnih masa aminokiselina? Pseudokod je dat u nastavku.

```

1 ConvolutionCyclopeptideSequencing(Spectrum, N, M)
2 begin
3   Formirati spektralnu konvoluciju spektra Spectrum.
4   Uzeti M najcescih elemenata u konvoluciji (izmedju 57 i 200).
5   Primeniti LeaderboardCyclopeptideSequencing, formirajuci peptide samo na
      ↪ osnovu ovih M celih brojeva.
6 end

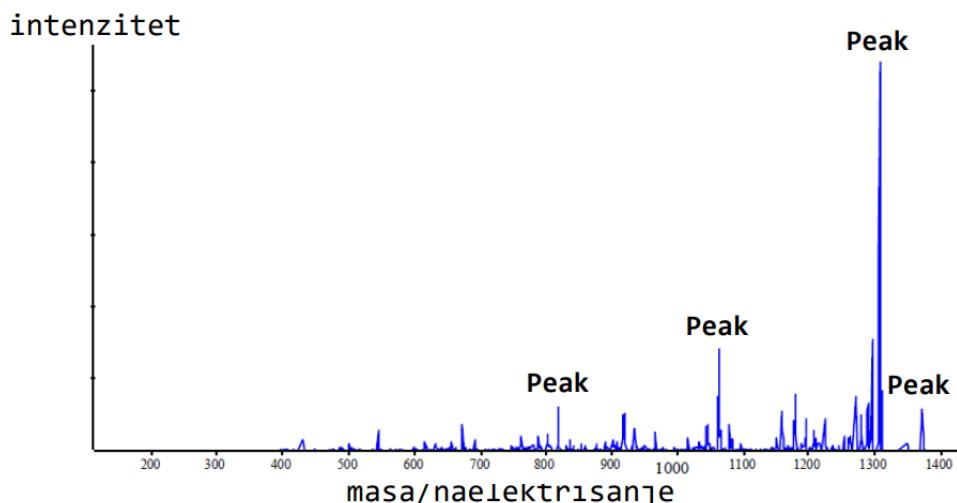
```

Algoritam *ConvolutionCyclopeptideSequencing* daje tačan rezultat i za spekture sa šumom od 10% i za spekture sa šumom od 25%, što pokazuje da je spektralna konvolucija odgovorila na sve izazove koji su postavljeni.

4.7 Spektri u realnosti

Kao što znamo, realnost je obično drugačija. Neke poteškoće iz realnosti smo zanemarivali. Koje?

- *Spectrum₂₅* je mnogo manje šumovit nego spektri dobijeni u praksi iz masenog spektrometra.
- Maseni spektrometar ne meri jednostavno fragmente podpeptida, već su postupci merenja mnogo komplikovani. Najpre se zaista vrši razbijanje datog peptida na fragmente. Zatim se oni sortiraju, korišćenjem elektromagnetskog polja, prema svojoj masi. Ono što maseni spektrometar meri jeste zapravo **odnos mase i naelektrisanja** za svaki fragment (znači nije baš masa) i određuje **intenzitet** (kao broj jona) u svakom odnosu mase i naelektrisanja. Šta to znači? To znači da kao izlaz iz masenog spektrometra ne dobijamo eksperimentalni spektar koji smo do sada imali prilike da vidimo, nego grafik intenziteta prema odnosu mase i naelektrisanja sa vrhovima na određenim mestima. Primer ovakvog grafika dat je na slici 4.3. Na osnovu vrhova na grafiku, određivaćemo sam sastav peptida. Ovaj grafik se



Slika 4.3: Primer grafika intenziteta prema odnosu mase i naelektrisanja.

naziva **realni spektar**. Rekonstrukcija peptida na osnovu realnog spektra biće obrađena u poglavlju 11.

Glava 5

Kako poredimo biološke sekvene?

Zašto uopšte želimo da poredimo biološke sekvene? U prethodnom poglavlju govorili smo o stvaranju proteina. Prvo smo posmatrali nukleotidne sekvene kao niske nad azbukom od 4 karaktera {A, C, G, T}. Pored toga, u biološke sekvene spadaju i proteini - sekvene nad azbukom od uglavnom 20 aminokiselina. Govorili smo o tome kako nastaju proteini od DNK (centralna dogma). Međutim, zaključili smo da ne nastaju baš svi proteini na taj način (prisetiti se primera sa *Bacillus breavis*). Neki proteini nastaju neribozomalnim procesom pomoću enzima koji se nazivaju *neribozomalne peptidne sintetaze* (NRP).

5.1 Biološki uvid u poređenje sekvenci

Kao što je već pomenuto, neribozomalni proteini nastaju pod dejstvom NRP sintetaze. NRP sintetaza je kao mašina za nadovezivanje aminokiselina. Podsetimo se da za svaki NRP protein postoji posebna NRP sintetaza koja ga pravi. Svaka NRP sintetaza se sastoji od modula koji određuju koju aminokiselinu treba dodatina peptid koji se sintetiše. Svaki modul sastoji se od različitih podjedinica (tzv. domena), a za sintezu su najznačajni adelacioni domeni (skraćeno A-domeni).

NRP sintetaza koja kodira antibiotik Tirocidin B1 (sastavljen od 10 aminokiselina) uključuje 10 modula, a svaki sadrži A-domén koji je odgovoran za dodavanje jedne aminokiseline u Tirocidin B1.

Kada je sekpcioniranje proteina bilo usavršeno, biolozi su žeeli da porede različite A-domene kako bi pokušali da pronađu sličnosti koje će nam ukazati koje aminokiseline će se povezati u okviru peptida pod njihovim uticajem. Na slici 5.1 prikazana su 3 A-domena različitih bakterija, koje redom kodiraju Asp, Orn i Val. Naravno, očekivali su da će naći neke delove koji se poklapaju, ali očekivano je bilo i da će se pojaviti i neke različitosti obzirom da se utiču na tri različite aminokiseline.

Bez poravnjanja ova tri A-domena nemaju mnogo sličnosti. Različitim su dužina i kada ih poređamo jednu ispod druge ne vidimo neke veće delove koji su jednaki. Svega 3 karaktera se poklapaju. Kolone koje sadrže istu aminokiselinu nazivamo **konzervirane kolone** (u smislu, nisu izmenjene). Međutim, nakon poravnjanja (što je prikazano na slici) vidimo da ima čak 19 poklapanja odnosno, konzerviranih kolona. Sad ne možemo reći da ove sekvene nisu slične. Crvene kolone predstavljaju konzervirano jezgro koje dele mnogi A-domeni.

Kada su ovi delovi A-domena uzeti u obzir, znalo se da je svaki od njih zaslužan za kodiranje jedne aminokiseline, odnosno za njeno dodavanje. Kako sva tri A-domena imaju istih 19 kolona, na osnovu čega znamo da će baš prva dodati *Aps*, druga *Orn*, a treća *Val*? Detektovane su pozicije koje sadrže aminokiseline koje kodiraju druge aminokiseline koje treba da nastanu. To

je niz od 8 aminokiselina koji nazivamo **signatura**. Na slici 5.1, označene su plavom bojom i one definišu neribozomalni kod. Signatura i dalje nije poznata za sve aminokiseline.

```
YAFDLYT1CMFPV2LLGGELHIVQKETY3TAPDEIAHYI4KEHGITYIKLTPSLFHTIVNTASFAFDANFES5RL6TVL7GGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIIKKYDITIF8EATPALVIPLMEYI-YEQKLDISQLQIL9IVGSDSCSME10DFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAP11LLNGGTVV12CIDYYTTIDIKALEAVFKQHHIRGAM13LPPALLKQCLVSA14--PTMISSLEIL15FAAGDRLSSQDAILARRAVGSGV-Y-NAYGPTENTVLS
```

Slika 5.1: A-domeni tri različite bakterije

Kako su biološke sekvence podložne promeni, umetanju i brisanju, čest je slučaj da i -ti simbol jedne sekvence odgovara simbolu na drugoj poziciji druge sekvence. U tom slučaju, cilj je postići najbolje poklapanje simbola. Na primer, *ATGCATGC* i *TGCATGCA* nemaju delove koji se poklapaju, pa je njihova Hamingova udaljenost 8:

$$\begin{array}{c} ATGCATGC \\ TGCATGCA \end{array}$$

Ali ako ih malo drugačije poravnamo, ove dve niske imaju 6 poklapajućih pozicija:

$$\begin{array}{c} ATGCATGC- \\ -TGCATGCA \end{array}$$

Stringovi ATGCTTA i TGCATTA imaju manje uočljive sličnosti:

$$\begin{array}{c} ATGC - TTA- \\ -TGCATTAA \end{array}$$

Ovi primjeri navode nas da definišemo dobro poravnanje kao ono koje ima najveći mogući broj poklapanja. Povećanje broja poklapanja simbola možemo posmatrati kao igricu u kojoj u svakom potezu imamo dva izbora. Možemo da uklonimo oba simbola i osvojimo poen ako su oni isti ili možemo ukloniti simbol iz jedne od niski, ne osvojimo poene, ali omogućimo da u daljem igranju osvojimo više poena. Cilj je da maksimizujemo broj poena.

5.2 Igra poravnjanja i najduža zajednička podsekvence

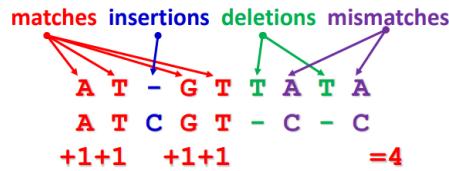
Kod **Igre poravnanja** cilj je ukloniti sve simbole iz sekvenci tako da pritom sakupimo što više poena :

- Uklanjanje prvog simbola iz svake sekvence
 - 1 poen ako se simboli poklapaju,
 - 0 ako se simboli ne poklapaju
- Uklanjanje prvog simbola iz jedne sekvene - 0 poena

Poravnanje dve sekvene predstavlja matricu koja ima dva reda. Prvi red matrice popunjeno je simbolima prve sekvene (redom), drugi red matrice popunjeno je simbolima druge sekvene (redom). Simboli razmaka ('-') mogu biti ubaćeni u oba reda, bitno je da se dva takva simbola ne nađu u istoj koloni.

Poravnanje predstavlja jedan mogući scenario u kom se prva niska pretvara u drugu. Kolone koje sadrže karaktere sa istim karakterima, nazivamo poklapanje (match), a one sa različitim karakterima nazivamo promašaj (missmatch) i one predstavljaju zamenu jednog nukleotida drugim. Kolone koje sadrži razmake nazivamo indel-i: ako je razmak u prvom redu onda je u pitanju

insercija, dok razmak u drugom redu predstavlja deleciju. Na slici 5.2 vidimo 4 poklapanja, 2 promašaja, 1 inserciju i 2 delecije.



Slika 5.2: Poravnanje

Poklapanja (matches) u poravnanju dve sekvene (u primeru 5.2 to je ATGT) formiraju njihovu zajedničku podsekvencu (nije baš podniska jer se karakteri ne nalaze svi jedan do drugog). Želimo da imamo što više poklapanja odnosno želimo da imamo najdužu zajedničku podsekvencu.

Problem najduže zajedničke podsekvence. Naći najdužu zajedničku podsekvencu dve niske.
 Ulaz: Dve niske.
 Izlaz: Najduža zajednička podsekvenca ovih niski

5.3 Problem turiste na Menhetnu

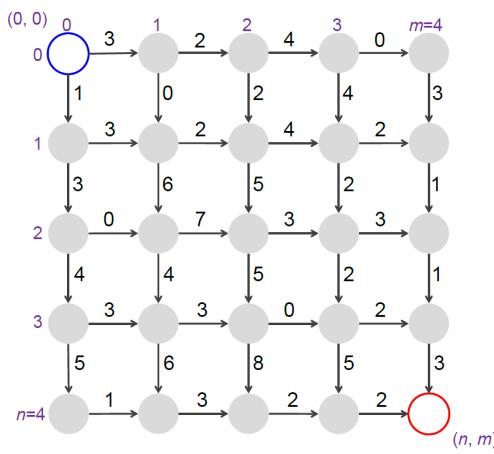
Zamislimo da smo turisti na Menhetnu, nalazimo se na uglu 59. ulice i 8. Avenije (gore levo) i želimo da stignemo do ugla 3. avenije i 42. ulice a da na putu prođemo što više znamenitosti. Prepostavimo da se sve ulice seku pod pravim uglom. Mapu grada modelovaćemo grafom koji za svaku raskrsnicu sadrži čvor, a ulice su grane. Dodatno, želimo da za svaku granu dodamo broj koji će predstavljati broj znamenitosti koji se može videti u toj ulici. Želimo da pronađemo put u grafu od čvora $(0, 0)$ - izvor; do čvora (n, m) - ponor; takav da je njegova vrednost najveća.

Pre svega postavimo problem:

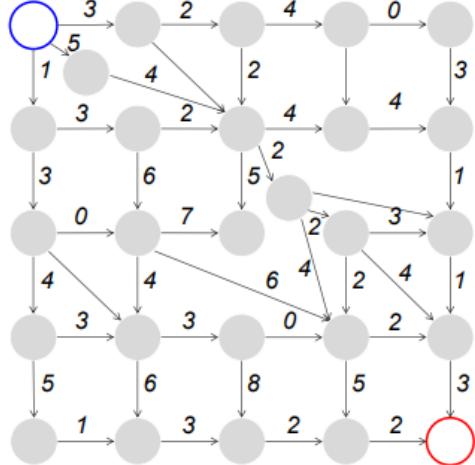
Problem turiste na Menhetnu. Naći najdužu putanju u pravougaonoj mreži gradskih ulica.
 Ulaz: Usmeren težinski mrežni graf.
 Izlaz: Najduža putanja od početnog (source) do krajnjeg čvora (sink) u mrežnom grafu.

Na slici 5.3 grafički je prikazan problem turiste na Menhetnu. Cilj je stići od plavog do crvenog kruga i pri tom sakupiti što više poena. Dozvoljeno kretanje je dole i desno. Možemo koristiti pohlepni algoritam i tako doći do cilja. Ideja pohlepnog algoritma jeste da u svakom koraku bira granu koja je otežana najvećom vrednošću. Da li smo tako sakupili najviše poena?

Prikazali smo primer sa određenom topologijom grafa. Imali smo samo horizontalne i vertikalne grane usmerene na desno odnosno na dole. Dodatna izmena grafa bi bila da imamo i neke dijagonalne grane (5.4).



Slika 5.3: Problem turiste na Menhetnu



Slika 5.4: Nepravilna mreža

Zbog toga definишемо проблем за usmereni graf. Kasnije ћemo видети да нам неће одговарати сваки usmereni graf.

Problem najduže putanje u usmerenom grafu. Naći najdužu putanju između dva čvora u težinskom usmerenom grafu.

Ulas: Usmereni težinski graf sa označenim čvorovima source i sink.

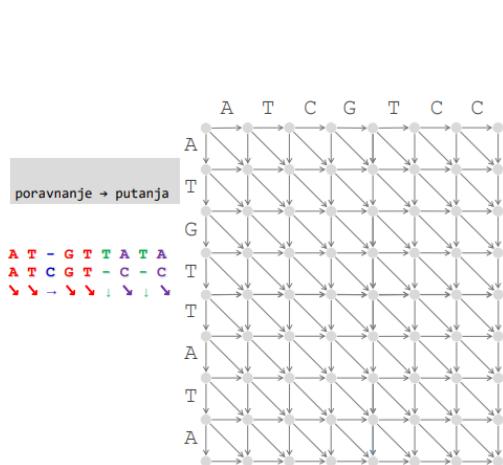
Izlaz: Najduža putanja od čvora source do čvora sink u usmerenom težinskom grafu.

Kakve veze ima turista i igra poravnjanja? Mogli bismo svakoj koloni u matrici dodeliti jednu granu u grafu. Za poklapanje/promašaj dodajemo dijagonalne grane, za inserciju dodajemo horizontalnu, a za deleciju vertikalnu granu. Precizno uputstvo za izgradnju grafa dato je u sledećim koracima:

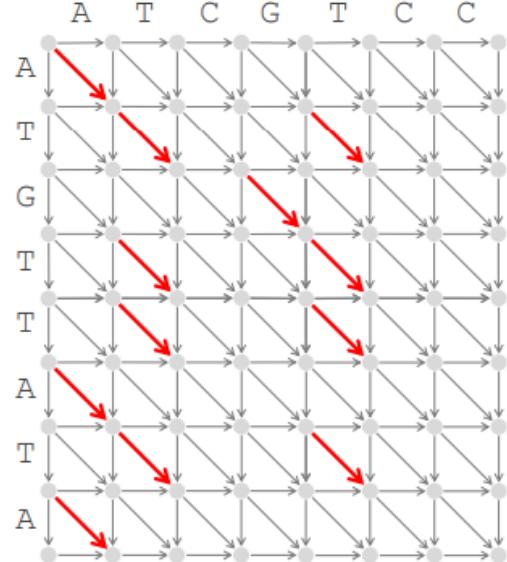
- Vrste označimo aminokiselinama iz prve niske
- Kolone označimo aminokiselinama iz druge niske
- U svaku presečnu tačku postavimo jedan čvor
- Gde god je moguće, postaviti vertikalne (insercija), horizontalne (delecija) i dijagonalne grane (match ili mismatch)
- Dijagonalne grane otežati koeficijentom 1, ostale koeficijentom 0
- Problem najduže zajedničke podsekvence se svodi na problem nalaženja najduže putanje između dva data čvora u usmerenom grafu

Poravnjanje predstavlja jednu putanju od izvora do ponora. Kada nađemo poravnjanje najvišeg skora našli smo i najdužu putanju u mrežnom grafu.

Dijagonalne crvene grane odgovaraju poklapanju simbola i imaju skor 1 (5.6). Podsetimo se da je pronalaženje najduže zajedničke podsekvence dve niske ekvivalentno poravnjanju tih niski sa maksimalnim brojem pogodaka.



Slika 5.5: Poravnanje → Putanja



Slika 5.6: Poravnanje → Putanja

5.4 Problem kusura

Upoznajmo se sa sledećim problemom:

Problem vraćanja kusura. Naći minimalan broj novčića neophodnih za vraćanje kusura.

Ulaz: Ceo broj $money$ i niz pozitivnih celih brojeva ($coin_1, coin_2, \dots, coin_d$).

Izlaz: Minimalan broj novčića ($coin_1, coin_2, \dots, coin_d$) u apoenima koji rasitnjava sumu $money$.

5.4.1 Pohlepni algoritam

Najzastupljeniji način vraćanja kusura širom sveta podrazumeva iterativno traženje sledećeg najvećeg novčića. To bi značilo da bismo za kusur od 40 dinara dobili sledeće novčiće: $25 + 10 + 52$. Ovakav način vraćanja kusura opisuje takozvani pohlepni algoritam.

```

1 GreedyChange(money)
2 begin
3   change ← empty collection of coins
4   while money > 0
5     coin ← largest denomination that does not exceed money
6     add coin to change
7     money ← money - coin
8   return change
9 end

```

Međutim, ako malo bolje razmislimo ovo rešenje zapravo nije najbolje. Kusur bismo mogli vratiti i sa manje novčića na sledeći način: $40 = 20 + 20$. **Zaključak:** GreedyChange ne daje optimalno rešenje!

5.4.2 Rekurzivni algoritam

Pokušajmo sada da problem rešimo na drugačiji način koristeći rekurziju. Za zadate apoene 6, 5, 1, koji je najmanji broj novčića neophodnih za vraćanje kusura od 9 centi?

<i>money</i>	1	2	3	4	5	6	7	8	9	10	11	12
<i>MinNumCoins</i>			?	?				?	?			

Slika 5.7: Vraćanje kusura - rekurzija

Problem rešavamo tako što prvo od 9 oduzmemos 6 i dobijemo 3 kao ostatak kusura. Dakle, 9 se može vratiti od jednog novčića od 6 apoena i još plus broj novčića koji je potreban za preostali deo kusura od 3 centa. U istoj iteraciji analogno računamo za preostale apoene. Na slici 5.7 crvenim znakom pitanja označeno je traženo rešenje koje dobijemo rešavanjem manjih problema za kusure 3, 4 i 8.

$$\text{MinNumCoins}(9) = \min \begin{cases} \text{MinNumCoins}(9 - 6) + 1 = \text{MinNumCoins}(3) + 1 \\ \text{MinNumCoins}(9 - 5) + 1 = \text{MinNumCoins}(4) + 1 \\ \text{MinNumCoins}(9 - 1) + 1 = \text{MinNumCoins}(8) + 1 \end{cases}$$

Na osnovu prethodnog, moguće je izvesti opštu formulu:

$$\text{MinNumCoins}(\text{money}) = \min \begin{cases} \text{MinNumCoins}(\text{money} - \text{coin}_1) + 1 \\ \dots \\ \text{MinNumCoins}(\text{money} - \text{coin}_d) + 1 \end{cases}$$

Hajde sada da vidimo kako bismo to isprogramirali:

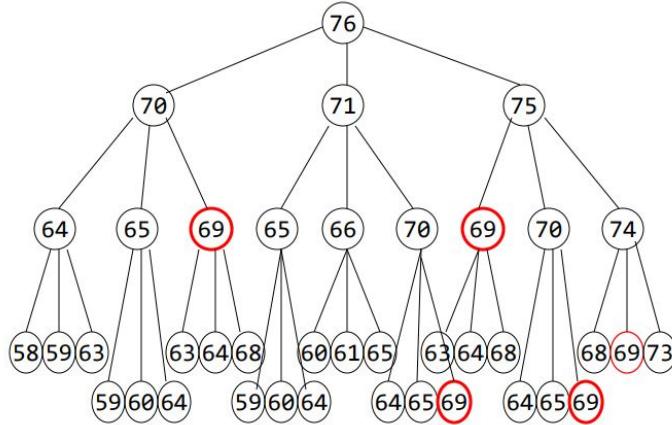
```

1 RecursiveChange(money, coins)
2 begin
3     if money = 0
4         return 0
5     MinNumCoins ← infinity
6     for i ← 1 to |coins|
7         if money ≥ coini
8             NumCoins ← RecursiveChange(money - coini, coins)
9             if numCoins + 1 < MinNumCoins
10                MinNumCoins ← numCoins + 1
11
12    return MinNumCoins
13 end

```

Reklo bi se da smo sada dobili odgovarajući algoritam za naš problem, hajde to da proverimo. Postavlja se pitanje, koliko je brz RecursiveChange? Pokušajmo na konkretnom primeru da dođemo do rešenja. Neka naš problem sada bude vraćanje kusura od 76 centi. Pomoću rekurzivnog stabla demonstrirajmo ponašanje našeg algoritma (5.8).

Ono što se odmah može primetiti jeste višestruko pozivanje algoritma za vrednost od 69 centi, čak 6 puta! Daljim procenama možemo doći do zaključka da se optimalna kombinacija novčića za 30 centi izračunava milijardama puta! Sada je očigledno da nam rekurzija ne rešava problem na najbolji mogući način.



Slika 5.8: Vracanje kusura - ponašanje rekurzivnog algoritma

5.4.3 Vraćanje kusura dinamičkim programiranjem

Cilj nam je da izbegnemo višestruka izračunavanja vraćanja kusura za istu vrednost, tako da bi ideja bila da imamo objekat koji će pamtiti sva računanja i iz koga ćemo čitati već izračunate vrednosti. Dakle, umesto vremenski zahtevnih poziva `RecursiveChange(money - coini, coins)`, jednostavno bismo potražili vrednosti iz unapred izračunate tabele `MinNumCoins(money - coini)`.

```

1 DPChange(money, coins)
2 begin
3     MinNumCoins(0) ← 0
4     for m ← 1 to money
5         MinNumCoins(m) ← infinity
6         for i ← 1 to |coins|
7             if m ≥ coini
8                 if MinNumCoins(m - coini) + 1 < MinNumCoins(m)
9                     MinNumCoins(m) ← MinNumCoins(m - coini) + 1
10    return MinNumCoins(money)
11 end

```

5.5 Dinamičko programiranje i putokazi za povratak

Vratimo se na jednostavniji, Menhetn graf, koji nema dijagonalne grane. Prepostavimo da do čvora sink možemo doći samo na dva načina: kretanjem južno ↓ ili kretanjem istočno →. I dalje želimo da dođemo od izvora do ponora i to najdužom putanjom. Kako bismo to rešili rekurzivno?

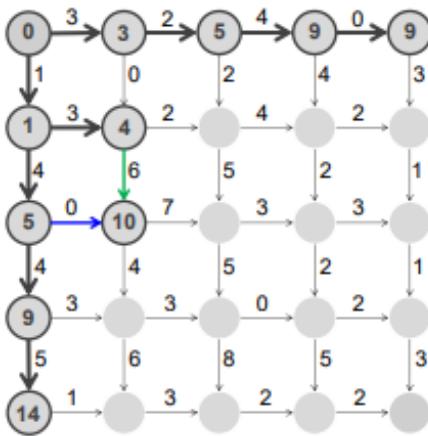
```

1 SouthOrEast(n, m)
2 if n=0 and m=0
3     return 0
4 x ← -infinity, y ← -infinity
5 if n > 0
6     x ← SouthOrEast(n-1,m)+weight of edge "↓" into (n, m)
7 if m > 0
8     y ← SouthOrEast(n,m-1)+ weight of edge "→" into (n,m)
9 return max{x, y}

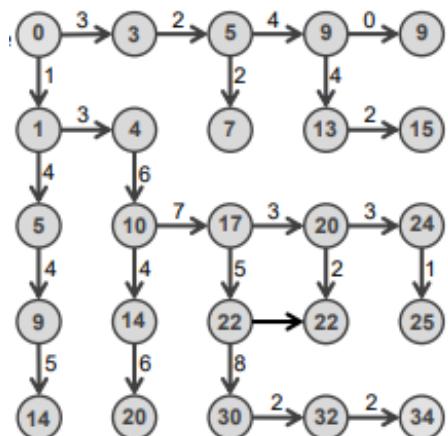
```

Nailazimo na isti problem kao kod rekurzivnog algoritma za vraćanje kusura. Ovaj algoritam se poziva za svaki čvor u grafu veličine $m \times n$, a pri tom se dešava da za jedan isti čvor računamo više puta. Zbog toga je ovaj pristup previše spor, pa prelazimo na dinamičko programiranje.

Krenućemo od početnog čvora. I prvo ćemo obraditi grane koje se nalaze na obodu grafa, a onda ćemo preći na ostale. Čvorovi na obodu su jednostavnji jer imamo jedinstven put do njih. U svaki čvor (i, j) upisujemo dužinu maksimalne putanje od $(0, 0)$ do (i, j) . Ostale čvorove obrađujemo kolonu po kolonu. Za svaki čvor (i, j) , koji se ne nalazi na obodu, biramo da li ćemo do njega doći sa severa ili sa zapada. Naravno, odabir zavisi od težine grana koje vode do čvora, pri čemu se bira veća vrednost. Može se desiti da nam obe grane daju istu vrednost akda je sve jedno koju ćemo odabrati.



Slika 5.9: Računanje vrednosti puta za svaki čvor.



Slika 5.10: Cene za svaki čvor su određene, odabranе grane су podebljane.

Na slici 5.10 prikazane su podebljane grane koje predstavljaju putokaze za povratak od čvora sink do čvora source.

Sve ovo treba predstaviti matematički, a to najlakše možemo učiniti rekurentnom relacijom. Označimo sa $s_{i,j}$ dužinu najduže putanje od izvora do čvora (i, j) . Onda važi sledeća rekurentna relacija:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ "into}(i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ "into}(i,j) \end{cases}$$

Na osnovu ove rekurentne relacije lako možemo napisati algoritam koji rešava problem turiste na Menhetnu.

```

1 ManhattanTourist(n, m, Down, Right)
2  $s_{0,0} \leftarrow 0$ 
3 for i  $\leftarrow 1$  to n
4    $s_{i,0} \leftarrow s_{i-1,0} + down_{i,0}$ 
5 for j  $\leftarrow 1$  to m
6    $s_{0,j} \leftarrow s_{0,j-1} + right_{0,j}$ 
7 for i  $\leftarrow 1$  to n
8   for j  $\leftarrow 1$  to m
9      $s_{i,j} \leftarrow \max \{ s_{i-1,j} + down_{i,j}, s_{i,j-1} + right_{i,j} \}$ 
10 return  $s_{n,m}$ 
```

5.6 Od Menhetna do grafa poravnjanja

Prikazali smo kako se dinamičko programiranje može primeniti na Menhetn graf. Međutim, ono što nama zaista treba jesti algoritam koji će raditi sa grafom poravnjanja. Osim horizontalnih i vertikalih grana, u grafu poravnjanja imamo i dijagonalne grane. Pored toga, za dijagonalnu granu moramo razlikovati situacije kada se desilo poklapanje i kada se desio promašaj. Rekurentna relacija dinamičkog programiranja kod grafa poravnjanja u opštem slučaju bi bila:

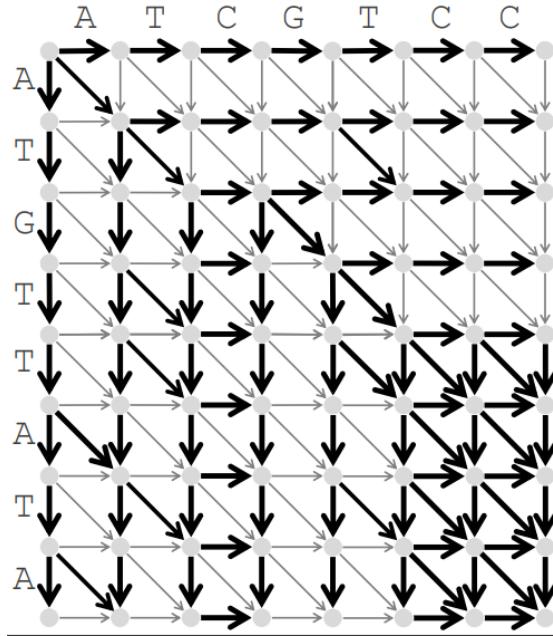
$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } " \downarrow " \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } " \rightarrow " \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } " \searrow " \text{ into } (i,j) \end{cases}$$

Kada bismo hteli da otežamo grane u skladu sa prethodnim pristupom, odnosno da poklapanje nosi vrednost 1, a ostalo 0, dobili bismo sledeću rekurentnu realciju:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + 0 \\ s_{i,j-1} + 0 \\ s_{i-1,j-1} + 1, v_i = w_j \\ s_{i-1,j-1} + 0, v_i \neq w_j \end{cases}$$

Crvene grane imaju težinu 1, a ostale grane težinu 0.

Na slici 5.11 se vide boldovane grane koje su nastale primenom pravila rekurentne relacije. One predstavljaju putokaze za povratak (backtrack) kod grafa za najdužu zajedničku podsekvencu.



Slika 5.11: Putokazi za povratak (backtrack)

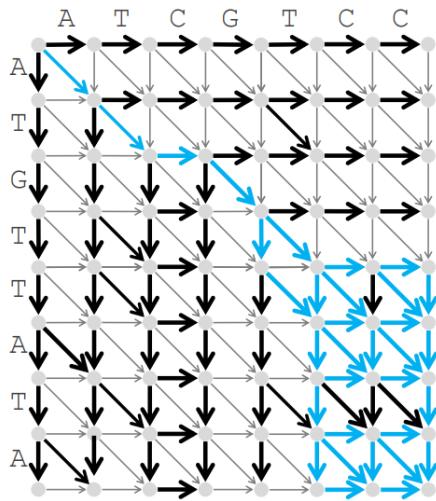
5.6.1 Računanje putokaza za povratak

Kada bismo sproveli isti pristup kao kod Menhetn grafa i kada bismo beležili grane koje smo birali, i ovde bismo da te grane koristimo kao putokaze i dođemo od ponora do izvora odnosno mogli bismo da rekonstruišemo najdužu putanju. Razlika je što ovde ima više podebljanih grana, tj. putokaza, što nam govori i da postoji više puteva koji imaju maksimalnu vrednost.

Ako nam je $s_{i,j}$ dato prethodno definisanom rekurentnom relacijom, putokaze možemo računati preko sledeće rekurentne relacije:

$$\text{backtrack}_{i,j} \leftarrow \max \begin{cases} " \rightarrow ", s_{i,j} = s_{i,j-1} \\ " \downarrow ", s_{i,j} = s_{i-1,j} \\ " \searrow ", \text{otherwise} \end{cases}$$

Podsetimo se sada kako bismo rekonstruisali putanje preko putokaza kod Menhetn grafa? Krenuli bismo od krajnjeg čvora (sink) i pratili putokaze u obrnutom smeru do početnog čvora (source). Na slici 5.12 možemo videti povratak (backtracking) kod grafa za najdužu zajedničku podsekvencu.



Slika 5.12: Backtracking

U nastavku je dat pseudokod za određivanje najduže zajedničke podsekvence (LCS – longest common subsequence) korišćenjem putokaza za povratak.

```

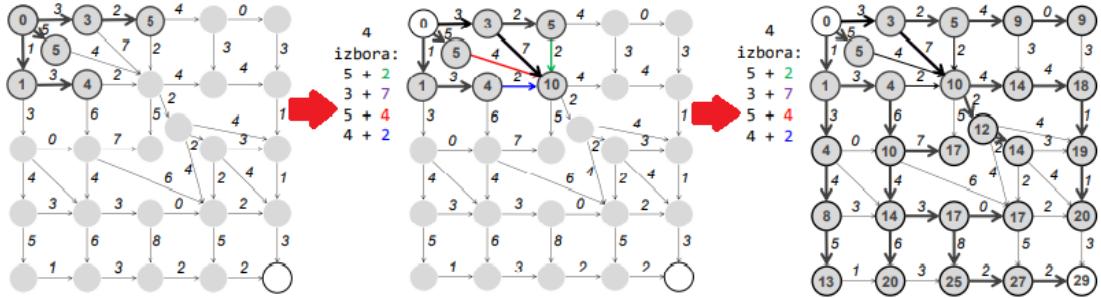
1 OutputLCS (backtrack, v, i, j)
2 if i = 0 or j = 0
3     return
4 if backtracki,j = "→"
5     OutputLCS (backtrack, v, i, j-1)
6 else if backtracki,j = "↓"
7     OutputLCS (backtrack, v, i-1, j)
8 else
9     OutputLCS (backtrack, v, i-1, j-1)
10    output vi
```

Do sada smo prepostavljali da graf u kom tražimo najdužu putanje ima samo tri vrste grana. Da li se OutputLCS može generalizovati tako da važi i za grafove koji nemaju tako specifičnu topologiju? Kako se rekurentna relacija dinamičkog programiranja menja za ovakav graf? 5.13

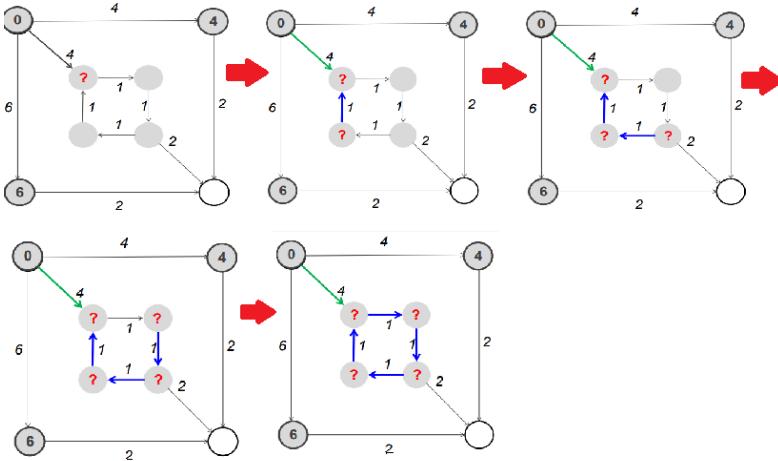
$$s_a = \max_{\text{all predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$$

Kod ovakve rekurentne relacije, važno je da pri računanju s_a imamo izračunate s_b za sve čvorove prethodnike b (čvorovi za koje postoji grana do čvora a) Da li je to moguće u bilo kom usmerenom težinskom grafu? Odgovor je **nije**. Da bismo kod svakog čvora mogli da izračunamo skor za sve njegove prethodnike, usmereni težinski graf mora biti acikličan - DAG (Directed Acyclic Graph).

Ako je dat usmereni aciklični graf, da li njegove čvorove možemo poređati u niz tako da



Slika 5.13: Prikaz algoritma na opštem usmerenom težinskom grafu



Slika 5.14: Računanje skora za SVE prethodnike

njihov redosled u nizu osigurava uslov da pri računaju s_a imamo izračunate s_b za sve čvorove prethodnike b (čvorovi za koje postoji grana do čvora a)? Odgovor je **da**, moguće je poređati sve čvorove grafa u niz i taj niz topološki sortirati.

Topološko sortiranje : Sortiranje čvorova DAG-a u nizu tako da sve grane u takvom nizu idu s leva na desno.

Teorema: Svaki DAG se može topološki sortirati.

Topološko sortiranje svakog DAG-a se obavlja za $O(\#edges)$ koraka. Kada imamo topološko sortiranje čvorova, možemo odrediti najužu putanju od izvora do ponora tako što ćemo proći čvorove u redosledu koji diktira topološko sortiranje. Algoritam za nalaženje najduže putanje u DAG-u dat je u nastavku.

```

1 LongestPath(Graph, source, sink)
2 for each node a in Graph
3      $s_a \leftarrow -\infty$ 
4  $s_{source} \leftarrow 0$ 
5 topologically order Graph
6 for each node a (from source to sink in topological order)
7  $s_a \leftarrow \max_{all \text{ predecessors } b \text{ of node } a} \{s_b + \text{weight of edge from } b \text{ to } a\}$ 
8 return  $s_{sink}$ 
```

Pošto svaka grana učestvuje tačno jednom, složenost je $O(\#edges)$. LongestPath vraća dužinu

najdužeg zajedničkog podniza ali ne rekonstruiše putanju.

5.7 Od globalnog do lokalnog poravnjanja

Do sada smo podrazumevali da dužine niski koje poredimo ne odstupaju previse jedna od druge (da ne poredimo nisku od npr. 10 sa niskom od 100 karaktera). Ako to podrazumevamo, onda radimo globalno poravnanje. Međutim, i druga situacija je moguća, ali moramo malo izmeniti pristup problemu i bliže ga razmotriti.

Do, sada, skor poravnjanja računali smo kao broj poklapanja - *matches*. Međutim, u slučaju kada su niske potpuno različitih dužina, ovaj način računanja skora neće biti sasvim prikladan, moraćemo da ga malo profinimo. Dakle, do sad je poklapanje nosilo 1, a ostale situacije vrednost 0. Sada, uvodimo kazne za nepoklapanje (μ) i za indel-e(σ). Skor sa mismatch i indel kaznama računamo po formuli $matches - \mu * mismatches - \sigma * indels$

U primeru koji je dat prilikom opisa igre poravnjanja, skor je bio 4. Primenom nove formule dobijamo skor -7.

Uvodimo pojam matrice skora. Vrste (odnosi se na karaktere prve niske) i kolone (odnosi se na karaktere druge niske) označimo vrednostima karaktera iz abecede i praznim simbolom. Elementi matrice imaju vrednosti kazne za svaku kombinaciju karaktera. Na dijagonali će se naći jedinice jer tada dolazi do poklapanja, poslednja vrsta i kolona imaju vrednost $-\sigma$ jer su to indel-i, a ostale vrednosti su promašaji pa su vrednosti $-\mu$.

Možemo napraviti generalniju matricu, gde možemo uvesti dodatne vrednosti za različita nepoklapanja. Ovaka matrica se može odrediti eksperimentalno. Posmatra se koje mutacije se dešavaju često, koje ređe i na osnovu toga određujemo skor za različite kombinacije. Primer je dat na slici 5.15.

$ \begin{array}{ccccccccc} \textcolor{red}{A} & \textcolor{red}{T} & - & \textcolor{green}{G} & \textcolor{red}{T} & \textcolor{teal}{T} & \textcolor{purple}{A} & \textcolor{red}{T} & \textcolor{blue}{A} \\ \textcolor{red}{A} & \textcolor{red}{T} & \textcolor{red}{C} & \textcolor{green}{G} & \textcolor{red}{T} & - & \textcolor{red}{C} & - & \textcolor{red}{C} \\ +1 & +1 & -2 & +1 & +1 & -2 & -3 & -2 & -3 = -7 \end{array} $	
$ \begin{array}{cccccc} \textcolor{black}{A} & \textcolor{black}{C} & \textcolor{black}{G} & \textcolor{black}{T} & - & \\ \textcolor{red}{A} & \textcolor{red}{+1} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{blue}{-\sigma} \\ \textcolor{red}{C} & \textcolor{purple}{-\mu} & \textcolor{red}{+1} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{blue}{-\sigma} \\ \textcolor{red}{G} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{red}{+1} & \textcolor{purple}{-\mu} & \textcolor{blue}{-\sigma} \\ \textcolor{red}{T} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{purple}{-\mu} & \textcolor{red}{+1} & \textcolor{blue}{-\sigma} \\ - & \textcolor{blue}{-\sigma} & \textcolor{blue}{-\sigma} & \textcolor{blue}{-\sigma} & \textcolor{blue}{-\sigma} & \end{array} $ <p>Matrica skora</p>	$ \begin{array}{cccccc} \textcolor{black}{A} & \textcolor{black}{C} & \textcolor{black}{G} & \textcolor{black}{T} & - & \\ \textcolor{red}{A} & \textcolor{red}{+1} & \textcolor{purple}{-3} & \textcolor{purple}{-5} & \textcolor{purple}{-1} & \textcolor{blue}{-3} \\ \textcolor{red}{C} & \textcolor{purple}{-4} & \textcolor{red}{+1} & \textcolor{purple}{-3} & \textcolor{purple}{-2} & \textcolor{blue}{-3} \\ \textcolor{red}{G} & \textcolor{purple}{-9} & \textcolor{purple}{-7} & \textcolor{red}{+1} & \textcolor{purple}{-1} & \textcolor{blue}{-3} \\ \textcolor{red}{T} & \textcolor{purple}{-3} & \textcolor{purple}{-5} & \textcolor{purple}{-8} & \textcolor{red}{+1} & \textcolor{blue}{-4} \\ - & \textcolor{blue}{-4} & \textcolor{blue}{-2} & \textcolor{blue}{-2} & \textcolor{blue}{-1} & \end{array} $ <p>Još generalnija matrica skora</p>

Slika 5.15: Matrica skora

Matrica skora može se definisati i za proteinske sekvene i tu možemo videti koliko se često dešava da jedna aminokiselina mutira u drugu. Postoji više vrsta ovakvih matrica za proteinske sekvene.

U okviru algoritma dinamičkog programiranja bila je u fokusu rekurentna relacija koja je rešavala problem. Hajde i sada da odredimo rekurentnu relaciju koja će nam rešiti problem.

Počinjemo od prethodno definisane rekurentne relacije:

$$s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{weight of edge } \downarrow \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } \rightarrow \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } \searrow \text{ into } (i,j) \end{cases}$$

Uместо 0, dodajemo negirano σ ili μ :

$$s_{i,j} = \max \begin{cases} s_{i-1,j} - \sigma \\ s_{i,j-1} - \sigma \\ s_{i-1,j-1} + 1, v_i = w_j \\ s_{i-1,j-1} - \mu, v_i \neq w_j \end{cases} \quad s_{i,j} = \max \begin{cases} s_{i-1,j} + \text{score}(v_i, -) \\ s_{i,j-1} + \text{score}(-, w_j) \\ s_{i-1,j-1} \text{ score}(v_i, w_j) \end{cases}$$

Isto možemo zapisati pomoću funkcije `score()`, koja je zgodna ako imamo generalniju matricu skora:

Podsetimo se problema globalnog poravnjanja.

Problem globalnog poravnjanja. Naći poravnanje sa najvišim skorom između dve niske za datu matricu skora.

Ulaz: Niske v i w , kao i matrica skora score

Izlaz: Poravnanje niski v i w čiji je skor poravnjanja (prema matrici skora) maksimalan od svih mogućih poravnanja v i w .

Zbog čega uopšte razmatramo druga poravnjanja? Rekli smo da ćemo nekada želeti da poredimo niske različitih dužina odnosno, neku malu u odnosu na neku priličnu veliku sekvencu.

Ograničenja globalnog poravnjanja su **homeobox geni**. Analiza homeobox gena daje primer problema za koji globalno poravnanje neće uspeti da otkrije biološki bitne sličnosti. Ovi geni regulišu razvoj embriona i prisutni su kod velikog broja vrsta, od muva do ljudi. Homobox geni su dugački i veoma se razlikuju od vrste do vrste, ali postoje veoma kratki regioni, koje nazivamo **homeodomeni**, koji su čvrsto konzervirani. Ukoliko bismo pokušali da pronađemo taj kratak homeodomen kod dve različite vrste. Dva gena u različitim vrstama mogu biti slična u kratkim, konzerviranim regionima, a različita u ostalim delovima. Homeobox geni sadrže kratak region homeodomen koji je čvrsto konzerviran među različitim vrstama. Globalno poravnanje može da propusti nalaženje homeodomena jer pokušava da poravna sekvence u celosti. Uporedimo sledeća dva poravnanja:

Slika 5.16: Globalno poravnanje
score = 22(matches) - 2(indent) = 2

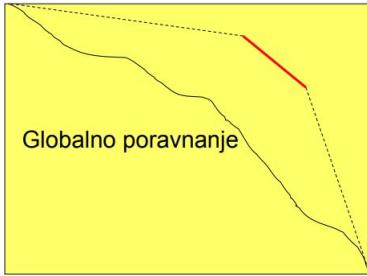
Slika 5.17: Lokalno poravnanje
score = 17 (matches) - 30(indent) = -13

Vidimo da poravnanje na slici 5.17 ima 17 poklapanja (u odnosu na 22, na slici 5.16), i ima čak 30 indel-1 (dok na prvoj ima 20). Skor je dosta lošiji, iznosi -13, u odnosu na 2. Međutim, 5.17 poravnanje ima dugačak zajednički deo koji se poklapa, pronašli smo kratak konzervirani region. Tako da, ako se zapitamo koje od ovih poravnanja je bolje, iz priloženog zaključujemo da je to lokalno poravnanje.

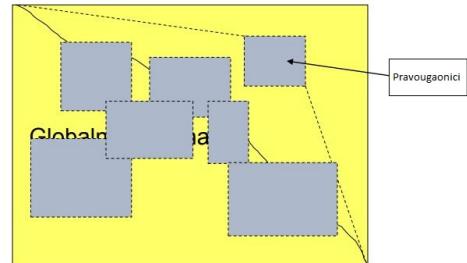
Kada su biološki bitne sličnosti prisutne u nekim delovima sekvenci v i w i odsutne u ostalim, biolozi pokušavaju da ignorišu globalno poravnanje i umesto toga poravnaju podnische v i w , čime se dobija lokalno poravnanje ovih niski. Problem pronalaženja podniski koje maksimizuju skor globalnog poravnjanja naziva se **Problem lokalnog poravnjanja**.

5.7.1 Lokalno poravnjanje

Lokalno poravnjanje računamo kao globalno poravnjanje u pravougaoniku, pogledajmo sliku 5.19. Odnosno, tražimo neke delove niske u kojima ćemo tražiti pokalapanja umesto u celoj niski.



Slika 5.18



Slika 5.19

Da bismo dobili lokalno poravnanje potrebno je da izračunamo globalno poravnjanje u okviru svakog pravougaonika. Algoritam lokalnog poravnjanja ponovićemo između svaka dva čvora, ne samo između početnog (source) i krajnjeg (sink), što radimo za globalno poravnjanje. Stoga, broj ponavljanja algoritma će biti $\#nodes^2$ puta.

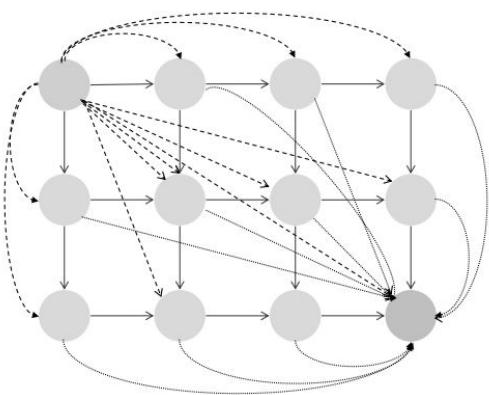
Problem lokalnog poravnjanja. Naći lokalno poravnjanje najvećeg skora između dve niske.

Ulaz: Niske v i w , kao i matrica skora score

Izlaz: Podnische niski v i w čije je globalno poravnjanje (prema matrici skora) maksimalno među svim globalnim poravnanjima svih podniski v i w .

Besplatna taksi vožnja

Kako ovo možemo da realizujemo? Zamislimo da postoji taxi koji bi nas besplatno vozio do tačke početka lokalnog poravnjanja, i od tačke završetka lokalnog poravnjanja pa do kraja. Na taj način ne bismo sku-pili negativne poene dok ne pronađemo odgovarajući pravougaonik, već samo pozitivne. Ovakva vožnja nam daje samo skor lokalnog poravnjanja kao što smo i želeli. Kako bi izgle-dao Menhetn graf za naš problem? Dodamo grane težine 0 od $(0,0)$ do svakog čvora, i od svakog čvora do (n,m) . Ukupan broj dodatih grana je $O(|v| + |w|)$, pa algoritam ostaje brz.



Slika 5.20: Menhetn graf za lokalno poravnjanje

Zahvaljujući besplatnoj vožnji, ne moramo da tražimo najduži put između svaka dva čvora u grafu. Ukupan broj grana u grafu je $O(|v| \cdot |w|)$, što je i dalje malo. Pošto vreme izvršavanja zavisi od broja grana, ovaj algoritam biće brz. Što se tiče računanja vrednosti $s_{i,j}$, dodavanje grana

težine 0 od čvora $(0, 0)$ do svakog čvora čini izvor prethodnikom svih čvorova. Zato, sada imamo četiri grane koje ulaze u čvor (i, j) , a rekurentna relacija se samo proširuje jednim elementom:

$$s_{i,j} = \max \begin{cases} 0 \\ s_{i-1,j} + \text{weight of edge } " \downarrow " \text{ into } (i,j) \\ s_{i,j-1} + \text{weight of edge } " \rightarrow " \text{ into } (i,j) \\ s_{i-1,j-1} + \text{weight of edge } " \searrow " \text{ into } (i,j) \end{cases}$$

Takođe, pošto smo dodali grane od svakog čvora do ponora, svaki čvor predstavlja prethodnika ponora pa će najveća vrednost $s_{n,m}$ najvećoj vrednosti iz matrice.

5.7.2 Kažnjavanje praznina

U globalnom poravnanju je fiksna kazna σ bila dodeljena svakom indel-u. Međutim, ova fiksna kazna može biti preoštra kod lokalnog poravnanja kada možemo imati 100 uzastopnih indel-a. Niz od k uzastopnih indel-a često predstavlja jedan isti evolucijski događaj, ne k različitim, (5.21). Šta to znači? Prepostavlja se da kada se jedna vrsta razdvajala od druge, znači da se tih k indel-a odjednom dogodilo, a ne u više navrata. Zbog toga uvodimo afinu kaznu za praznine (skup uzastopnih crtica).

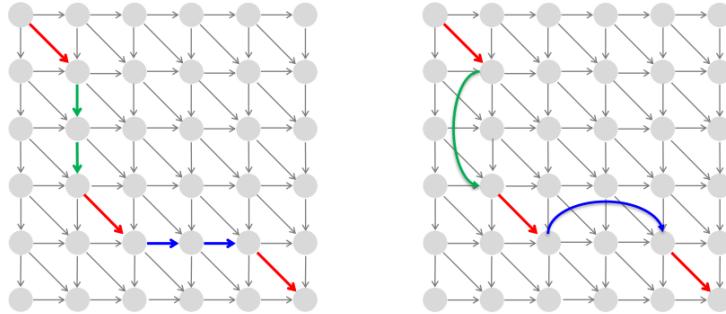
dve praznine (niži skor)	GATCCAG	GATCCAG	jedna praznina (viši skor)
	GA-C-AG	GA--CAG	

Slika 5.21

Afini kazni za prazninu dužine k definisana je formulom $\sigma + \varepsilon * (k - 1)$, gde je

- σ kazna za **otvaranje** praznine
- ε kazna za **proširenje** praznine
- $\sigma > \varepsilon$, jer otvaranje praznine treba kazniti više nego njeno proširenje

Kako ćemo to modelovati grafom? Na slici 5.22 prikazano je modelovanje afinskih kazni za praznine pomoću dugih grana.

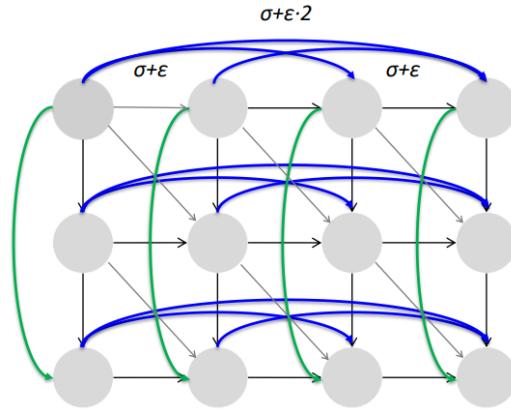


Slika 5.22: Modelovanje afinskih kazni za praznine pomoću dugih grana

5.7.3 Izgradnja Menhetn grafa sa afnim kaznama za praznine

Dodajemo nove grane za sve nesusedne čvorove sa dužinama $\sigma + \varepsilon \cdot k$. Pre ovoga smo imali kvadratni broj grana, sada smo prešli na kub.

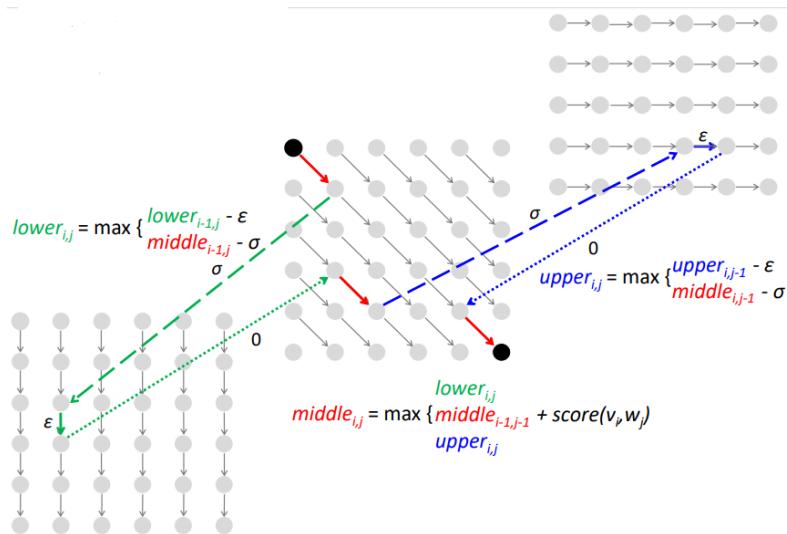
Vremenska složenost je direktno proporcionalna broju grana, zbog čega želimo da smanjimo broj grana u grafu (trenutno je $O(n^3)$, 5.23). Jedan način za smanjivanje broja grana je povećanje broja čvorova u grafu. Zato delimo Menhetn graf na tri nivoa (5.24).

Slika 5.23: Dodali smo $O(n^3)$ grana

Kada smo pravili Menhetn graf, dodavali smo grane horizontalno, vertikalno i dijagonalno, gde god je to bilo moguće. Sada želimo da razdvojimo sve to, tako da donji nivo sadrži samo vertikalne (insercije), srednji samo dijagonalne (poklapanje/promašaj) i gornji nivo sadrži samo horizontalne grane (delecija).

Kako se ponašamo sad prikazano je na slici 5.24. Prelazak sa gornjeg ili donjeg nivoa na srednji ima cenu nula (zatvaranje praznine, besplatna taksi vožnja), dok prelazak sa srednjeg na gornji ili donji ima vrednost σ . Krećemo na srednjem nivou, imamo poklapanje i krećemo se dijagonalnom granom. Zatim dolazi do insercije i zelenom granom prelazimo na donji nivo. Tada od trenutne vrednosti oduzimamo σ , odnosno, otvaramo prazninu. Dalje, imamo još jednu inserciju, a pošto je praznina već otvorena, nećemo ponovo oduzimati veliku vrednost σ , već onu manju, ε . Nakon toga ponovo imamo poklapanje, vraćamo se nazad na srednji nivo. Onda imamo deleciju, prelazimo plavom granom na gornji nivo (cena σ), nakon čega sledi još jedna delecija sa cenom ε . Na kraju imamo još jedno poklapanje, plavom granom, cene 0, vraćamo se nazad na srednji nivo.

Na ovaj način povećali smo broj čvorova, ali smanjen je broj grana, koji je ponovo kvadratan, što nam vraća efikasnost.



Slika 5.24: Simulacija Menhetn grafa na 3 nivoa

Promenljiva $lower_{i,j}$ računa skor optimalnog poravnanja između i -prefiksa niske v i j -prefiksa niske w koje se završava delecijom, odnosno, rupom u w . Sa druge strane, promenljiva $upper_{i,j}$ računa skor optimalnog poravnanja ovih prefiksa koje se završava insercijom, odnosno, rupom u v . Promenljiva $middle_{i,j}$ računa skor optimalnog poravnanja koje rezultuje pogotkom ili promašajem. U formulama za računanje promeljivih matrica $lower$ i $upper$, prvi red odnosi se na proširenje rupe, dok drugi red predstavlja otvaranje rupe.

5.8 Prostorno efikasno poravnanje sekvenci

Da li možemo poravnati NPR sintetaze iz dve različite bakterije? Uzmimo u obzir sledeće činjenice. NPR sintetaze su obično veoma dugi proteini, približno 20 000 aminokiselina. Vremenska složenost poravnanja je približno jednaka broju ivica (#edges), odnosno kvadratna. Prostorna složenost poravnanja je približno jednaka broju čvorova (#nodes), odnosno takođe je kvadratna. Memorija je često usko grlo pri poređenju dugih sekvenci.

Iz prethodnog zaključujemo da nam prostorna složenost pravi problem i da bi za prosečan računar ovo bilo nemoguće da izračuna. Stoga, potreban nam je drugi pristup koji će nam rešiti problem. Potreban nam je algoritam koji zahteva linearnu prostornu složenost i udvostručenu vremensku složenost. U te svrhe najčešće se koristi algoritam koji radi po principu **podeli-povladaj**.

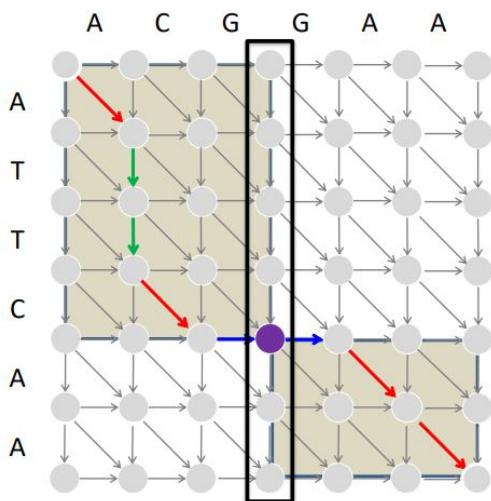
Vraćamo se na graf poravnanja i male nukleotidne sekvence, radi ilustracije. Uvedimo nove pojmove:

- Srednja kolona poravnanja (middle) = #columns/2, ??
- Srednji čvor poravnanja - čvor u preseku putanje optimalnog poravnanja i srednje kolone, slika ??

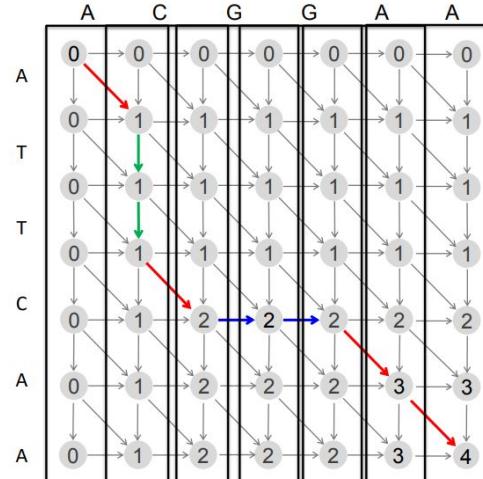
Koristeći navedene pojmove, demonstrirajmo algoritam **Podeli-pa-vladaj** za poravnanje sekvenci, slika 5.25:

```

1 AlignmentPath(source, sink)
2   find middleNode
3   AlignmentPath(source, middleNode)
4   AlignmentPath(middle, sink)
```



Slika 5.25: Srednja kolona poravnanja



Slika 5.26: Recikliranje prostora za kolone u grafu poravnanja

Za nalaženje najduže putanje u grafu poravnajanja traži se čuvanje svih putokaza za - $O(nm)$ prostora. Međutim, za nalaženje dužine najduže putanje u grafu poravnajanja ne traži se čuvanje svih putokaza - $O(n)$ prostora, slika 5.26. Dobijamo da je prostor potreban za algoritam $2 * n \sim O(n)$, odnosno, linearan.

Hoćemo da pokažemo da, da bismo izračunali koliko iznosi dužina najduže putanje, nije potrebno da čuvamo ceo graf, tj. sve njegove čvorove, već je dovoljno da čuvamo samo dve kolone. Prvu kolonu popunimo nulama pa gledamo za drugu kako možemo doći do svakog od čvorova. Kod prvog čvora u toj koloni nema razmišljanja, možemo doći samo horizontalnom granom. Za drugi imamo više izbora, a biramo dijagonalnu granu jer postoji poklapanje. Za sve ostale čvorove ove kolone koristimo vertikalnu granu jer više nema poklapanja koja će uvećati dužinu putanje do nekog od tih čvorova.

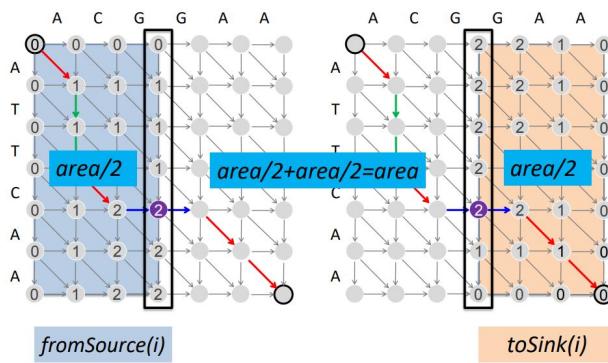
Kao što vidimo, za određivanje težina u drugoj koloni bila nam je potrebna samo jedna prethodna kolona. Kako se krećemo kroz graf, kolonu po kolonu, biće nam potrebna samo prethodna kolona. Pokazali smo da je prostor potreban za podeli-pa-vladaj poravnanje $2 * n \sim O(n)$, odnosno, linearan. Da vidimo kakva će biti vremenska složenost.

Da bismo smanjili prostornu složenost na linearanu, morali smo da se odrekнемo konstrukcije najduže putanje. Pitanje je da li, i kako, možemo da nađemo srednji čvor ako ne konstruišemo putanju.

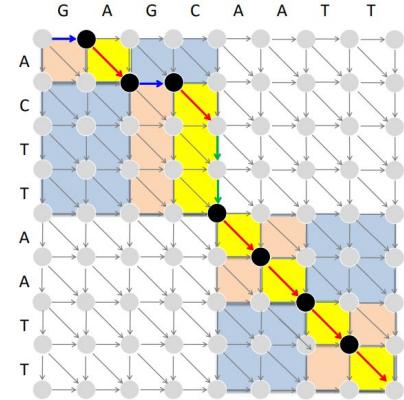
Neka je **i-putanja** najduža putanja od svih putanja koja posećuje i-ti čvor u srednjoj koloni i neka je $length(i)$ dužina i-putanje. Na slici 5.27 vidimo da je $length(0) = 2$ i $length(4) = 4$. Dužinu označavamo sa $length(i)$ i možemo računati po formuli:

$$length(i) = fromSource(i) + toSink(i)$$

Na slici 5.27 vidimo koliko je potrebno vremena za nalaženje srednjeg čvora - čak $O(nm)$ vremena za nalaženje samo jednog čvora!



Slika 5.27: Računanje vremena za nalaženje srednjeg čvora



Slika 5.28: Računanje vremena za nalaženje 7 tačaka

Svaki problem se može rešiti u vremenu proporcionalnom broju grana tj. površini koju zauzima. Vreme potrebno za rešavanje naredna dva potproblema: $area/4 + area/4 = area/2$, znači $O(nm + nm/2)$ vremena za nalaženje 3 čvora. Dakle, vreme potrebno za nalaženje svih čvorova je: $area + area/2 + area/4 + area/8 + \dots < 2 * area$! Slika 5.28 vizuelno prikazuje vreme potrebno za nalaženje 7 tačaka. Pokazali smo da je vremenska složenost $2 * n * m \sim O(n * m)$.

5.9 Višestruko poravnanje sekvenci

Do sada su u poravnanju učestvovalo samo dve sekvene. Slaba sličnost između dve sekvene postaje značajna ako je prisutna i u drugim sekvencama. Višestruka poravnanja mogu otkriti

suptilne sličnosti koje dvostruka poravnanja ignorišu. Na slici 5.29 je prikazano poravnanje tri A-domena.

```

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIKKYDITIFEAUTPALVPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA---PTMISSLEILFAAGDRLSQDAILARRAVGSGV-Y-NAYGPTENTVLS

```



```

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIKKYDITIFEAUTPALVPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA---PTMISSLEILFAAGDRLSQDAILARRAVGSGV-Y-NAYGPTENTVLS

```



```

YAFDLGYTCMFPVLLGGGELHIVQKETYTAPDEIAHYIKEHGITYIKLTPSLFHTIVNTASFAFDANFESLRLIVLGGEKIIPIDVIAFRKMYGHTE-FINHYGPTEATIGA
-AFDVSAGDFARALLTGGQLIVCPNEVKMDPASLYAIKKYDITIFEAUTPALVPLMEYI-YEQKLDISQLQILIVGSDSCSMEDFKTLVSRFGSTIRIVNSYGVTEACIDS
IAFDASSWEIYAPLLNGGTVVCIDYYTTIDIKALEAVFKQHHIRGAMLPPALLKQCLVSA---PTMISSLEILFAAGDRLSQDAILARRAVGSGV-Y-NAYGPTENTVLS

```

Slika 5.29: Poravnanja 3 A-domena

Poravnanje 2 sekvene predstavili smo matricom od 2 reda. Tako će poravnanje 3 sekvene biti matrica od 3 reda (slika 5.30). Na odgovarajuća mesta su ponovo umetnute crtice.

A T - G C G -
A - C G T - A
A T C A C - A

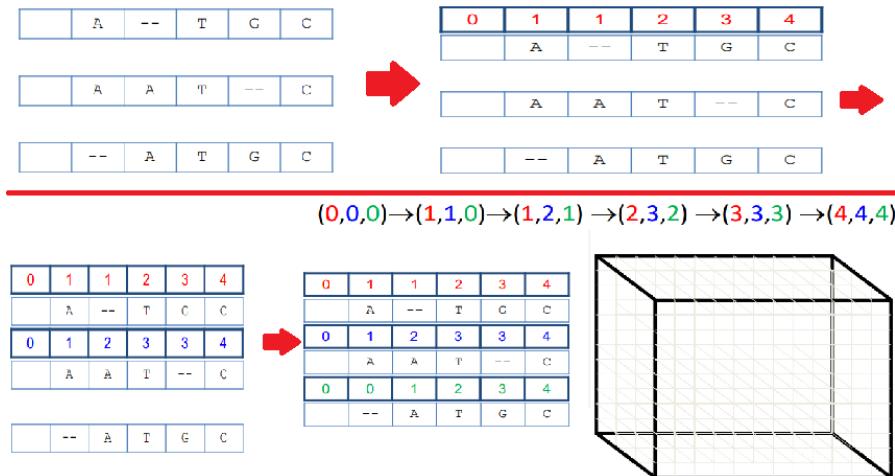
Slika 5.30

Funkcija skora treba da dodeljuje visok skor poravnajima sa konzerviranim kolonama.

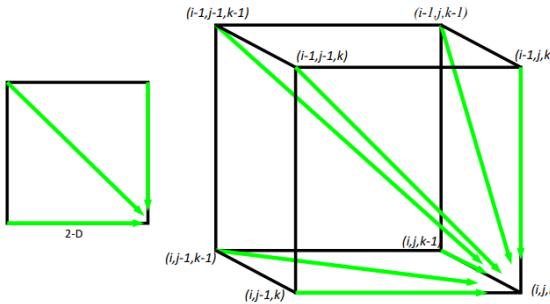
Poravnanje dve niske bilo je predstavljeno mrežom kvadrata. Poravnanje tri sekvene predstavljamo mrežom kockica i govorimo od 3D-putanja.

Poravnanje sekvenci ATGC, AATC i ATGC predstavljen je na slici 5.31. Prvo smo napravili matricu od tri reda, a onda posebno obeležili svaki red. Oznake u dodatnim redovima predstavljaju broj simbola do date pozicije u odgovarajućoj niski (crtica se ne računa kao simbol!). Čemu služi to prebrojavanje?

Ako procitamo sve vrednosti po kolnama, dobićemo 3D putanju, kojom se krećemo od čvora (0, 0, 0) do čvora (4, 4, 4). Kako to izgleda na grafu prikazano je na slici 5.32. Ono što je kod dvostrukog poravnanja bilo predstavljeno kvadratom, kod trostrukog poravnanja predstavljeno je kockom.



Slika 5.31: Poravnanje sekvenci ATGC, AATC i ATGC



Slika 5.32: 2-D poravnanje u odnosu na 3-D poravnanje

Da vidimo kako se rekurentna relacija dinamičkog programiranja za višestruko poravnanje komplikuje sa povećanjem dimenzije:

$$s_{i,j,k} = \max \begin{cases} s_{i-1,j-1,k-1} + \delta(v_i, w_j, u_k) \\ s_{i-1,j-1,k} + \delta(v_i, w_j, -) \\ s_{i-1,j,k-1} + \delta(v_i, -, u_k) \\ s_{i,j-1,k-1} + \delta(-, w_j, u_k) \\ s_{i-1,j,k} + \delta(v_i, -, -) \\ s_{i,j-1,k} + \delta(-, w_j, -) \\ s_{i,j,k-1} + \delta(-, -, u_k) \end{cases}$$

gde je $\delta(x, y, z)$ element 3-D matrice skora.

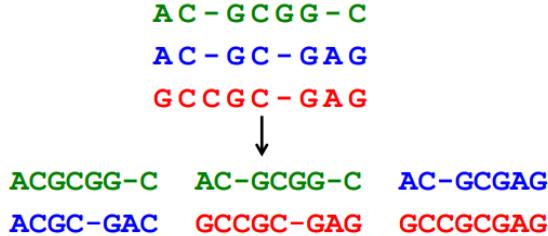
5.9.1 Vremenska složenost dinamičkog algoritma za višestruko poravnanje

Kao kod dvostrukog poravnajanja, vremenska složenost je proporcionalna broju grana $O(\#edges)$. Za 3 sekvene n, vremenska složenost je proporcionalna $7n^3$ (pomoću 7 različitih grana možemo doći do svakog čvora). Za poravnanje k sekveni, potrebno je izgraditi k-dimenzionalni Menhetn graf sa:

- n^k čvorova
- većina čvorova će imati $2^k - 1$ ulaznih grana
- Vremenska složenost: $O(2^k n^k)$

Ovo važi ukoliko primenjujemo algoritam globalnog poravnjanja na višestruko poravnanje Viđimo da složenost nije najbolja i pitamo se kako to može biti bolje, efikasnije.

Pogledajmo sledeće. Određujemo trostruko poravnanje. Možemo da primetimo da ono sadrži i neka dvostruka poravnanja. Iz trostrukog poravnjanja možemo da izvučemo dvostruka poravnanja (5.33). Da li može biti obrnuto?



Slika 5.33

Pitamo se, da li za dati skup proizvoljnih dvostrukih poravnjanja možemo konstruisati višestruko poravnanje iz kog su izvedeni (slika 5.34).

AAAATTTT---- ----TTTGGGG	---AAAATTTT -----GGGGAAAA	TTTTGGGG---- ----GGGGAAAA
------------------------------------	-------------------------------------	-------------------------------------

Slika 5.34

Tu će nam pomoći profilna reprezentacija višestrukog poravnjanja (slika 5.35). Imamo matricu koja ima 5 redova (simboli azbuke i praznina). Broj kolona direktno zavisi od broja kolona u višestrukom poravnanju. Kako popunjvamo matricu? Na isti način kao u poglavlju 2, kada smo pravili profilne matrice motiva.

-	A	G	G	C	T	A	T	C	A	C	C	T	G	
T	A	G	-	C	T	A	C	C	A	-	-	-	G	
C	A	G	-	C	T	A	C	C	A	-	-	-	G	
C	A	G	-	C	T	A	T	C	A	C	-	G	G	
C	A	G	-	C	T	A	T	C	G	C	-	G	G	
<hr/>														
A	0	1	0	0	0	0	1	0	0	.8	0	0	0	0
C	.6	0	0	1	0	0	.4	1	0	.6	.2	0	0	
G	0	0	1	.2	0	0	0	0	.2	0	0	.4	1	
T	.2	0	0	0	1	0	.6	0	0	0	0	.2	0	
-	.2	0	0	.8	0	0	0	0	0	.4	.8	.4	0	

Slika 5.35: Profilna matrica višestrukog poravnjanja

Do sada smo poravnavali sekvencu u odnosu na sekvencu. Možemo li poravnati sekvencu u odnosu na profil? Možemo li poravnati profil u odnosu na profil? Predstavićemo jedan od algoritama za višestruko poravnanje koje bi pratilo pohlepni pristup.

Imamo nekoliko sekvenci. Izabratemo dve najsličnije sekvence tako što će njihovo dvostruko poravnanje dati najveći skor (poravnaćemo svaku sa svakom). Onda te sekvene kombinjemo u profil i tako smo smanjili broj sekvenci sa k na $k - 2$ i jedan profil. Iteriramo postupak dok ne

dođemo do zajedničkog profila.

Primer pohlepnog pristupa:

Imamo sekvence: GATTCA, GTCTGA, GATATT, GTCAGC koje želimo da poravnamo. Odredićemo 6 dvostrukih poravnanja sa primitivnom šemom skorovanja - poklapanje ima vrednost 1, dok promašaj i indel-i imaju vrednost -1 (slika 5.36).

<i>s₂</i> GTCTGA <i>s₄</i> GTCAGC (score = 2)	<i>s₁</i> GATTCA-- <i>s₄</i> G-T-CAGC (score = 0)	}	
<i>s₁</i> GAT-TCA <i>s₂</i> G-TCTGA (score = 1)	<i>s₂</i> G-TCTGA <i>s₃</i> GATAT-T (score = -1)		<i>s₂</i> GTCTGA <i>s₄</i> GTCAGC } <i>s_{2,4}</i> = GTCT/aGa/c
<i>s₁</i> GAT-TCA <i>s₃</i> GATAT-T (score = 1)	<i>s₃</i> GAT-ATT <i>s₄</i> G-TCAGC (score = -1)		Slika 5.37

Slika 5.36

Pošto su s_2 i s_4 najbliže, od njih pravimo profil. Kako pravimo profil? Tamo gde smo imali poklapanja, ti karakteri ostaju nepromenjeni. Gde ima nepoklapanja, oba uvrstimo na to mesto (prikaz na slici 5.37). Time smo dobili novi skup od 3 sekvence za poravnanje: s_1 = GATTCA, s_3 = GATATT i $s_{2,4}$ = GTCT/aGa/c

Glava 6

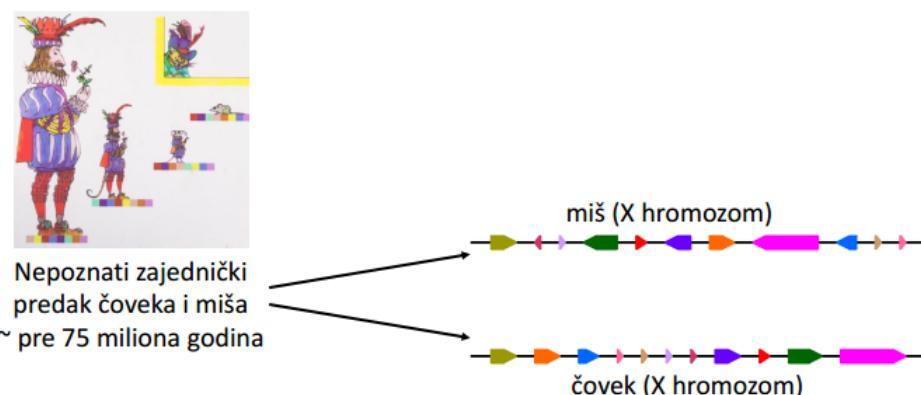
Postoje li osetljivi delovi u ljudskom genomu?

Kada smo razmatrali genomke sekvene rekli smo da je kod različitih organizama DNK organizovana u različit broj hromozoma i da se u svakom nalaze geni u kojima je upisano kako se razvija biće. Ono što nam sledi u nastavku jeste razmatranje sličnosti između različitih biljnih i životinjskih vrsta. Posotje neke poznate relacije, npr. čovek je relativno blizak majmunima. Govorimo o evoluciji kao mehanizmu za razdvajanje vrsta. Odnosno na koji su se način razdvojili od zajedničkog pretka.

Posmatrajmo X-hromozoma čoveka i miša. Pokazano je da postoje sličnosti između ovih X-hromozoma. Sada govorimo o sličnosti na drugačiji način. Nećemo poravnavati sekvene i slično već poređimo delove hromozoma.

Na slici 6.1 prikazana su ova dva X-hromozoma. Primetimo da više nisu predstavljeni karakterima nad abzukom {A, C, G, T} već su neke celine grupisane u blokove različitih boja. Ono što možemo da primetimo sa slike jeste da se neki blokovi, koji se pojavljuju u X-hromozomu čoveka, takođe pojavljuju u X-hromozomu miša u drugačijem redosledu i sa eventualno drugačijom orijentacijom.

Ono što nas sada zanima jeste koji blokovi genoma su slični i kako da ih nađemo? Kakav bi bio evolucijski scenario za transformisanje jednog genoma u drugi?

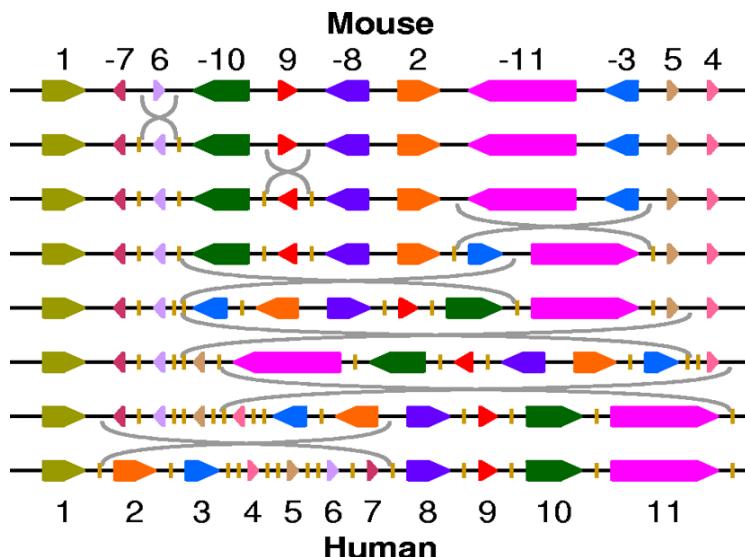


Slika 6.1: X hromozom miša i čoveka

Zapravo, ako bismo isekli 23 ljudska hromozoma na 280 delova i pomerali ove fragmente DNK, a zatim zlepili delove zajedno u novom redosledu, formiralo bi se 20 mišjih hromozoma. Međutim, evolucija nije koristila samo operaciju *cut-and-paste*, već manju promenu poznatu kao

preuređenje genoma, što će biti naš fokus u ovom poglavlju.

6.1 Blokovi sintenije



Slika 6.2: Blokovi sintenije

Svaki od jedanaest obojenih segmenata na slici 6.1 predstavlja blok sličnih gena i naziva se **blok sintenije**. U nastavku će biti objašnjeno kako se izgrađuju blokovi sintenije i šta označavaju levi i desni pravac blokova.

Slika 6.2 prikazuje niz 7 promena koje transformišu mišiji X hromozom u ljudski X hromozom. Nažalost, ovaj niz od 7 promena predstavlja samo jedan od 1.070 različitih scenarija od 7 promena koji transformišu X hromozom miša u X hromozom čoveka. Možemo li pretvoriti X hromozom miša u ljudski X hromozom koristeći samo šest promena?

Bez obzira na to koliko promena razdvaja mišije i ljudske X hromozome, promene moraju biti *retki genomski događaji*. Zapravo, obično preuređeni genomi uzrokuju smrt ili sterilnost mutiranog organizma, čime se sprečava prenošenje preuređenja na narednu generaciju. Međutim, mali deo preuređenja genoma može imati pozitivan efekat na preživljavanje i propagirati se kroz vrstu kao rezultat prirodne selekcije. Kada stanovništvo postane izolovano od ostatka njene vrste dovoljno dugo, preuređenja mogu čak stvoriti i novu vrstu.

Promenu možemo zamisliti kao prekid genoma sa obe strane hromozomskog intervala, pomeranje intervala, a zatim lepljenje rezultujućih segmenata u novom redosledu.

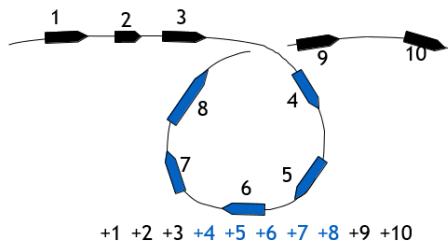
Slika 6.2 predstavlja transformaciju mišijeg X hromozoma u ljudski X hromozom sa sedam promena. Svaki blok sintenije je jedinstveno obojen i označen celim brojem između 1 i 11. Pozitivni ili negativni znak svakog celog broja ukazuje na smer sintenog bloka (pokazivanje desno ili levo, respektivno). Dva kratka vertikalna segmenta obeležavaju krajnje tačke obrnutog intervala u svakom preokretu.

Prepostavimo da je evolucijski scenario tačan i recimo peti blok sintenije od vrha predstavlja uređenje hromozoma pretka. Zatim su se desile prve četiri promene na evolucionom putu od miša do zajedničkog pretka, a poslednje tri promene su se desile na evolucionom putu od zajedničkog pretka ka čoveku.

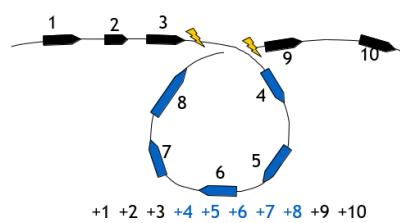
6.2 Sortiranje po promenama

Zamislimo gen ako linearan niz blokova. Prilikom evolucije može se dogoditi zavrtanje genoma kao na slici 6.3, i tada može doći do promena. Može se desiti da popučaju veze između nekih delova i da se uspostave druge veze.

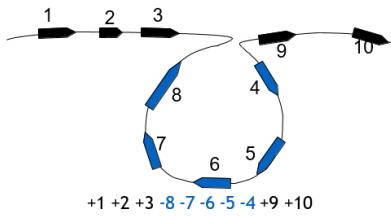
Glavni problem je, kao što je pomenuto u uvodu, nalaženje minimalnog broja promena koje omogućavaju transformaciju X hromozona miša u X hromozom čoveka. Možemo posmatrati niz blokova sintenije numerisanih kao na slici 6.3.



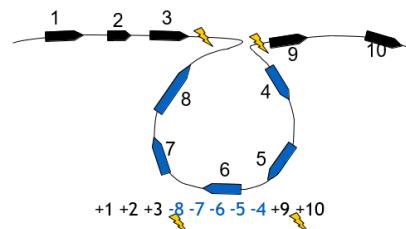
Slika 6.3: Blokovi sintenije



Slika 6.4



Slika 6.5: Nakon prekida



Slika 6.6: 2 tačke prekida

Nakon izvršene promene, dobijamo preuređen niz blokova sintenije u genomu to vidimo na slikama 6.4 i 6.5. Promene u genomu su dovele do stvaranja dve tačke prekida koji predstavljaju poremećaj u redosledu gena u genomu (slika 6.6).

Posmatraćemo 2 scenarija sa različitim brojem promena. Na slici 6.7 vidimo scenario sa 5 promena, a na slici 6.8 sa 4 promene. Ono što je podvučeno to su blokovi koji se obrću. Obrtanje samo jednog bloka nam pomoguće da mu promenimo orijentaciju.

Step 0:	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	2	3	4	5	6	7	8	1
Step 3:	2	3	4	5	6	7	8	-1
Step 4:	-8	-7	-6	-5	-4	-3	-2	-1
Step 5:	1	2	3	4	5	6	7	8

Slika 6.7: Scenario sa 5 promena

Step 0:	2	-4	-3	5	-8	-7	-6	1
Step 1:	2	3	4	5	-8	-7	-6	1
Step 2:	-5	-4	-3	-2	-8	-7	-6	1
Step 3:	-5	-4	-3	-2	-1	6	7	8
Step 4:	1	2	3	4	5	6	7	8

Slika 6.8: Scenario sa 4 promena

Definicija 6.1. *Rastojanje premutacija* je najmanji broj promena potrebnih za transformisanje jedne premutacije u drugu.

Naredni problem koji posmatramo je **problem sortiranja po promenama** koji predstavlja izračunavanje rastojanja između date permutacije i identične permutacije (+1 +2 ... +n).

Problem sortiranja po promenama Izračunavati rastojanje između date permutacije i identične permutacije (+1 +2 ... +n).

Ulaz: Permutacija P.

Izlaz: Rastojanje između permutacije P i identične permutacije.

6.3 Pohlepno sortiranje po promenama

Prva aproksimacija rastojanja između 2 permutacije je **pohlepno sortiranje po promenama**(primer je dat na slici 6.9). Želimo da redom smestamo blokove na odgovarajuće pozicije (i -ti blok na i -tu poziciju, ako indeksiramo od 1) i odgovarajuće orientacije (svi treba da budu pozitivni). Vidimo da se blok 1 nalazi na odgovarajućoj poziciji (pozicija 1) i sa odgovarajućom orientacijom (+), pa prelazimo odmah na blok 2. Kada njega smestimo prelazimo na blok 3, i tako dok svi blokovi nisu smešteni na svoje mesto.

```
(+1 -7 +6 -10 +9 -8 +2 -11 -3 +5 +4)
(+1 -2 +8 -9 +10 -6 +7 -11 -3 +5 +4)
(+1 +2 +8 -9 +10 -6 +7 -11 -3 +5 +4)
(+1 +2 +3 +11 -7 +6 -10 +9 -8 +5 +4)
(+1 +2 +3 -4 -5 +8 -9 +10 -6 +7 -11)
(+1 +2 +3 +4 -5 +8 -9 +10 -6 +7 -11)
(+1 +2 +3 +4 +5 +8 -9 +10 -6 +7 -11)
(+1 +2 +3 +4 +5 +6 -10 +9 -8 +7 -11)
(+1 +2 +3 +4 +5 +6 -7 +8 -9 +10 -11)
(+1 +2 +3 +4 +5 +6 +7 +8 -9 +10 -11)
(+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 -11)
(+1 +2 +3 +4 +5 +6 +7 +8 +9 +10 +11)
```

Slika 6.9: Pohlepno sortiranje

Definicija 6.2. Element k u permutaciji $P = (p_1, p_2, \dots, p_n)$ je **sortiran**, ako je $p_k = k$, a u suprotnom je **nesortiran**.

Definicija 6.3. Permutacija P je **k -sortirana**, ako je prvih $k-1$ elemenata sortirano, a k -ti element nesortiran.

Sledeći primer pokazuje da je pohlepno sortiranje loša aproksimacija rastojanja između dve permutacije(slika 6.10) jer je ponekad moguće naći mnogo jednostavniji način (slika 6.11).

$$\begin{aligned}
 & (-6 +1 +2 +3 +4 +5) \\
 & (-1 +6 +2 +3 +4 +5) \text{ step 1} \\
 & (+1 +6 +2 +3 +4 +5) \text{ step 2} \\
 & (+1 -2 -6 +3 +4 +5) \text{ step 3} \\
 & (+1 +2 -6 +3 +4 +5) \text{ step 4} \\
 & \dots \\
 & (+1 +2 +3 +4 + -6 +5) \text{ step 9} \\
 & (+1 +2 +3 +4 + -5 +6) \text{ step 10} \\
 & (+1 +2 +3 +4 + +5 +6)
 \end{aligned}$$

Slika 6.10: Pohlepno sortiranje

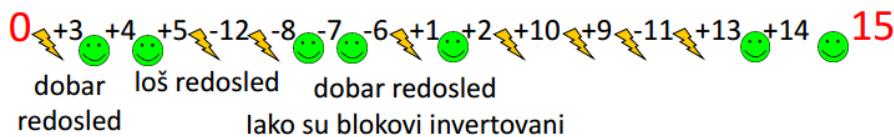
$$\begin{aligned}
 & (-6 +1 +2 +3 +4 +5) \\
 & (-5 -4 -3 -2 -1 +6) \\
 & (+1 +2 +3 +4 +5 +6)
 \end{aligned}$$

Slika 6.11: Kraći način

6.4 Teorema u prekidnoj tački

Uočimo da su uzastopni elementi (npr. $(+12 +13)$) poželjni, jer se javljaju u istom redosledu kao i u identičnoj permutaciji. Takođe, i $(-11 -10)$ su poželjni, jer se mogu inverzijom postaviti u pravi redosled. Ova dva para elemenata imaju zajedničku osobinu da je drugi element za 1 veći od prvog. Stoga, definišemo pojam suseda.

Definicija 6.4. (p_i, p_{i+1}) u permutaciji $P = (p_1, p_2, \dots, p_n)$ predstavljaju **susede**, ako je $p_{i+1} - p_i = 1$, a u suprotnom čine **prekid** (primere vidimo na slici 6.12).



Slika 6.12: Susedi i prekidi

Važi:

$$\text{adjacencies}(P) + \text{breakpoints}(P) = |P| + 1.$$

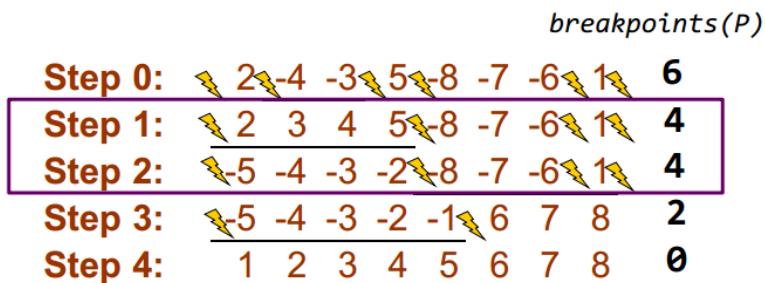
Što je permutacija bliža identičkoj permutaciji to ima manje tačaka prekida!

6.4.1 Sortiranje po promenama eliminacijom prekidnih tacaka

Koliko prekidnih tačaka može biti eliminisano u jednoj promeni? Odgovor nam daje sledeća teorema.

Teorema 6.5. *Teorema u prekidnoj tački: Rastojanje između permutacija nije manje od polovine broja prekidnih tačaka.*

U svakoj promeni možemo eliminisati najviše dve tačke prekida. Nema garancije da će svaka promena eliminisati 2 prekidne tačke (step 2), a najveći broj promena bi bio za permutaciju $(+n + (n - 1) \dots + 1)$ i iznosi $n + 1$ (to je gornja granica). Donja granica je $(n + 1)/2$. Velika razlika između donje i gornje granice nam sugerira da moramo preći na drugi način za rešavanje ovog problema.

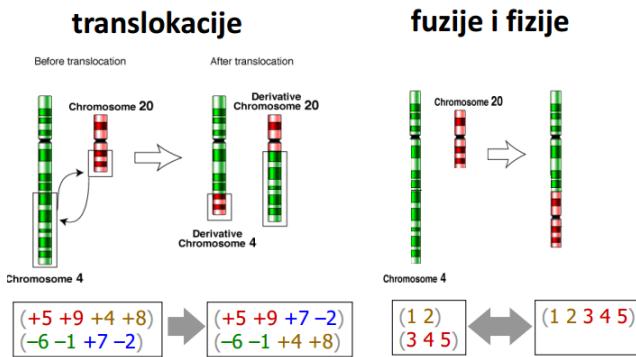


Slika 6.13: Sortiranje po promenama eliminacijom prekidnih tačaka

6.5 Preuređivanje u multihromozomalnim genomima

Umesto što posmatramo preuređivanje gena u okviru jednog hromozoma (hromozom X kod čoveka i miša), generalizujemo problem i posmatramo sve hromozome genoma. U ovoj generalizaciji će biti više oblika preuređivanja blokova u genomu (do sada su bila samo obrtanja). Problem je naizgled komplikovaniji, u nastavku će se ispostaviti da nije tako.

6.5.1 Translokacije, fuzije i fizije



Slika 6.14: Translokacije, fuzije i fizije

Za modeliranje translokacija posmatramo multihromozomalni genom sa k hromozoma kao permutaciju koja je podeljena na k delova.

Na primer, genom $(+1 +2 +3 +4 +5 +6) (+7 +8 +9 +10 +11)$ je sastavljen od dva hromozoma $(+1 +2 +3 +4 +5 +6)$ i $(+7 +8 +9 +10 +11)$.

Translokacija razmenjuje segmente različitih hromozoma, npr. translokacija dva hromozoma $(+1 +2 +3 +4 +5 +6) (+7 +8 +9 +10 +11)$ može dovesti do sledeća 2 hromozoma $(+1 +2 +3 +4 +9 +10 +11) (+7 +8 +5 +6)$.

Možemo zamišljati translokaciju kao prvo cepanje svakog od hromozoma $(+1 +2 +3 +4 +5 +6) (+7 +8 +9 +10 +11)$ na 2 dela, $(+1 +2 +3 +4) (+5 +6) (+7 +8) (+9 +10 +11)$, a zatim lepljenje rezultujućih segmenata u 2 nova hromozoma, $(+1 +2 +3 +4 +9 +10 +11) (+7 +8 +5 +6)$.

Preuređenja u multihromozomalnim genomima nisu ograničena na promene i translokacije. Ona takođe uključuju hromozomske fuzije, koje spajaju 2 hromozoma u 1, kao i fisije, koje dele 1 hromozom na 2 hromozoma (slika 6.14).

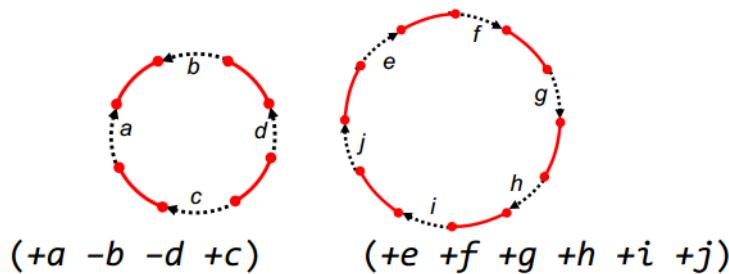
Na primer, 2 hromozoma ($+1+2+3+4+5+6$) ($+7+8+9+10+11$) mogu biti fuzionisani (spojeni) u 1 hromozom ($+1+2+3+4+5+6+7+8+9+10+11$). Sledеća fisiјa ovog hromozoma moze dovesti do 2 hromozoma ($+1+2+3+4$) ($+5+6+7+8+9+10+11$).

Pre pet miliona godina, ubrzo nakon razdvajanja чoveka i šimpanze, fuzija dva hromozoma (nazvana 2A i 2B) u jednom od naših predaka stvorila je ljudski hromozom 2 i smanjila broj hromozoma sa 24 na 23.

6.6 Problem rastojanja 2-prekida

6.6.1 Od linearnih do cirkularnih hromozoma

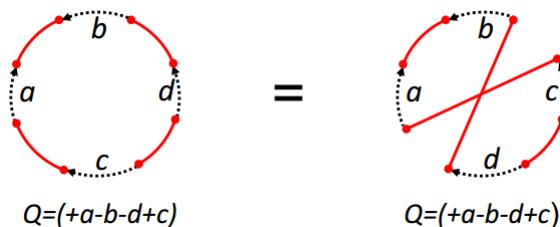
Sada se fokusiramo na jedan od hromozoma u multihromozomalmnom genomu i razmotrimo transformacije promene kružnog hromozoma $P = (+a -b -c +d)$ u $Q = (+e +f +g +h +i +j)$. Uvodimo pojam crnih usmerenih i crvenih neusmerenih grana. Posmatraćemo hromozome P i Q sa slike 6.15.



Slika 6.15: Hromozomi P i Q

Crne usmerene grane predstavljaju blokove sintenije. **Crvene neusmerene grane** povezuju susedne blokove sintenije.

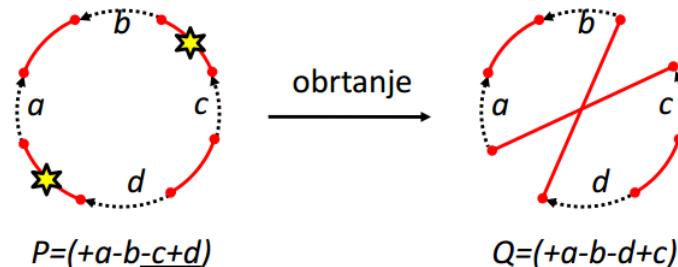
Možemo nacrtati Q na različite načine, zavisno od toga kako se odlučimo da uredimo crne grane. Slika 6.16 prikazuje dve takve ekvivalentne reprezentacije. Crvene grane određuju šta je susedno.



Slika 6.16: Ekvivalentne reprezentacije hromozoma Q

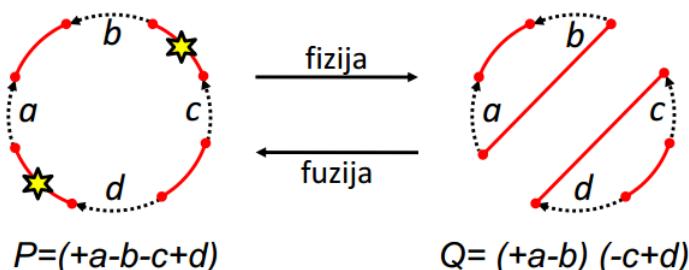
Iako je prvi crtež Q na slici njegova najprirodnija reprezentacija, koristićemo drugu reprezentaciju, jer su joj crne grane raspoređene kružno u potpuno istom redosledu kako se pojavljuju u prirodnoj reprezentaciji $P = (+a -b -c +d)$.

Kao što je prikazano na slici 6.17, fiksiranje crnih grana omogućava nam da vizualizujemo efekat promena. Kao što možemo videti, promena briše dve crvene grane iz P (povezivanje b sa c i d sa a) i zamenjuje ih sa dve nove crvene grane (povezivanje b sa d i c sa a). Ova promena se naziva **obrtanje**.



Slika 6.17: Obrtanje

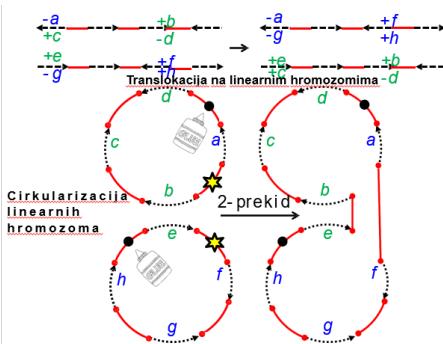
Slika 6.18 ilustruje fiziju $P = (+ a -b -c + d)$ u $Q = (+ a -b) (- c + d)$. Inverzna operacija fiziji odgovara fuziji dva hromozoma iz Q u hromozom P. Operacije fuzije i fizije, kao i promene, odgovaraju brisanju dve grane u jednom genomu i njihovim zamenjivanjem sa 2 nove grane u drugom genomu.



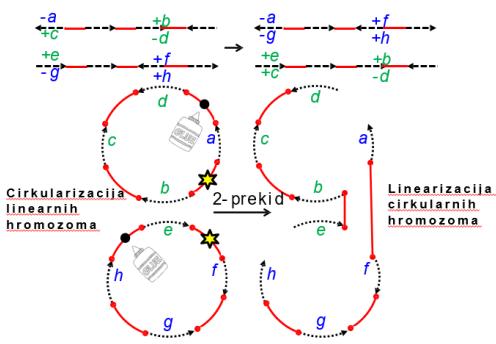
Slika 6.18: Fizija i fuzija

Definicija 6.6. *2-prekid: preuređenje koje zamenjuje dve crvene grane sa dve nove crvene grane (čvorovi ostaju isti).*

Translokacija koja uključuje dva linearne hromozome takođe se može simulirati cirkularizacijom ovih hromozoma, a zatim zamenjivanjem dve crvene grane sa dve različite crvene grane, kao što je prikazano na slici 6.18. Zbog toga se mogu objediniti ova 4 različita tipa preuređenja (slika 6.19). Svi oni se mogu posmatrati kao cevanje 2 crvene grane grafa genoma i zamena sa dve nove crvene grane na ista 4 čvora. Iz tog razloga definisemo opštu operaciju na grafu genoma koja zamenjuje crvenu granu sa dve nove crvene grane pri čemu čvorovi ostaju isti i nazivamo je **2-prekid**.



Slika 6.19: 2-prekid



Slika 6.20: Objedinjavanje 4 preuredjenja

Definicija 6.7. *Rastojanje 2-prekida $d(P, Q)$: Minimalni broj 2-prekida koji transformišu genom P u genom Q .*

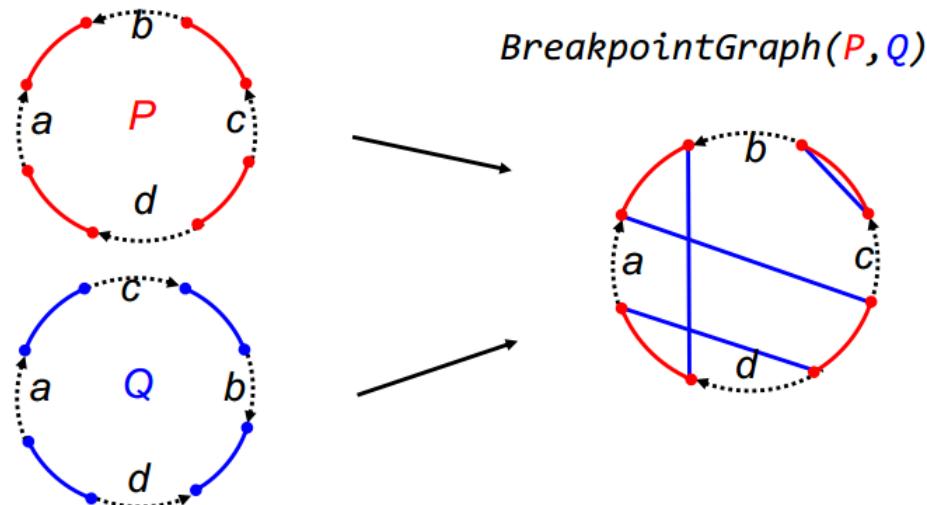
Problem rastojanja 2-prekida: Naći rastojanje 2-prekida između dva genoma.

Ulaz: Dva genoma nad istim skupom blokova sintenije.

Izlaz: Rastojanje 2-prekida između ovih genoma.

6.7 Grafovi prekidnih tačaka

Hoćemo da poredimo dva genoma, P (crveno) i Q (plavo). Za izračunavanje rastojanja 2-prekida konstruisaćemo graf za upoređivanje dva genoma. Preuredićemo blokove u genomu Q tako da budu u istom redosledu kao u P . Zatim, nadgradnjom genoma P i Q dobijamo $\text{BreakpointGraph}(P, Q)$ (slika 6.21).

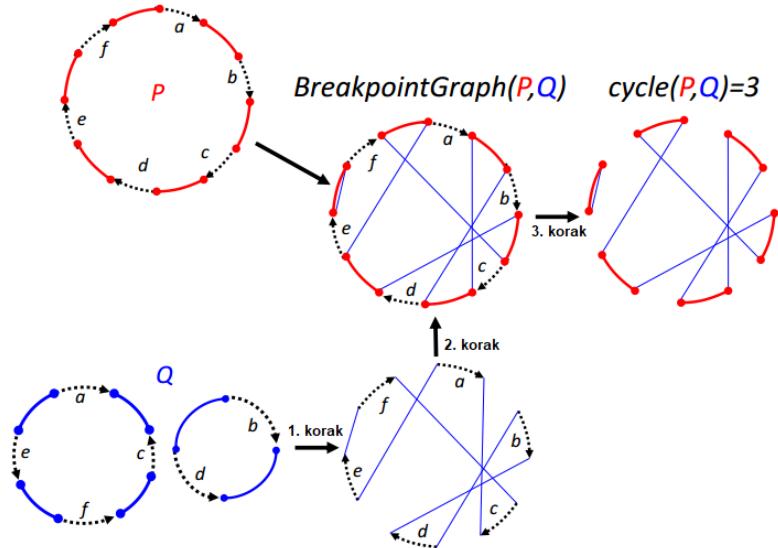
Slika 6.21: Nadgradnja genoma P i Q

Crvene i crne grane u grafu prekidnih tačaka formiraju genom P . Plave i crne grane u grafu prekidnih tačaka formiraju genom Q . Crvene i plave grane formiraju **alternirajuće crveno-**

plave cikluse. Ako bismo početne i krajnje tačke grana smatrali čvorovima kretanjem naizmeđu crvenim i plavim granama, napravili bismo ciklus. Takvih ciklusa u grafu može biti više. U našem primeru imamo dva takva ciklusa.

Koristimo oznaku $\text{cycle}(P, Q)$ za broj alternirajućih crveno-plavih ciklusa. Pitamo se šta predstavlja $\text{cycle}(P, Q)$?

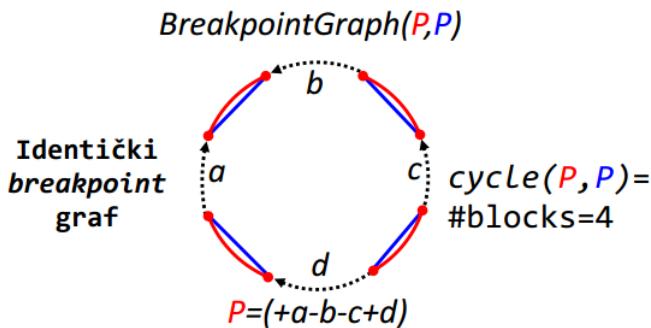
Posmatraćemo grafove genoma P i Q na slici 6.22. Ponovo, prvo poređamo crne grane genoma Q u isti redosled kao u genomu P . Zatim, vršimo nadgradnju genoma P i Q u jedan. Na kraju izvršimo povezivanje. Postupak je prikazan na slici 6.22. Vidimo da imamo 3 ciklusa.



Slika 6.22: $\text{cycle}(P, Q)$

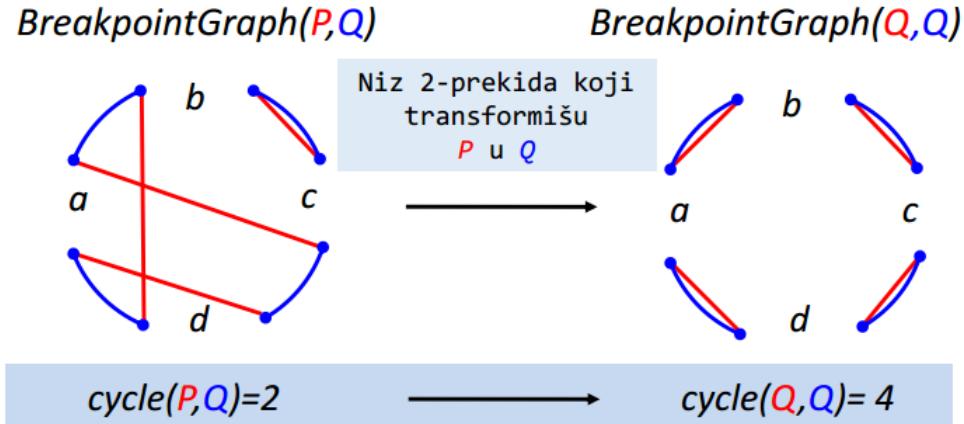
Za dato P (slika 6.23), koje Q maksimizuje $\text{cycle}(P, Q)$? Broj ciklusa je maksimalan kad imamo dva identična genoma!

U slučaju da P i Q imaju isti broj blokova sintenije, označimo taj broj sa $\text{Blocks}(P, Q)$. Ako su P i Q identični, njihov graf prekidnih tačaka se sastoji od $\text{Blocks}(P, Q)$ ciklusa dužine 2 od kojih svaki sadrži jednu crvenu i jednu plavu granu. Cikluse dužine 2 nazivamo **identičkim ciklusima**, a graf prekidnih tačaka formiran na osnovu identičkih genoma nazivamo **identičkim grafom prekidnih tačaka** (slika 6.37).



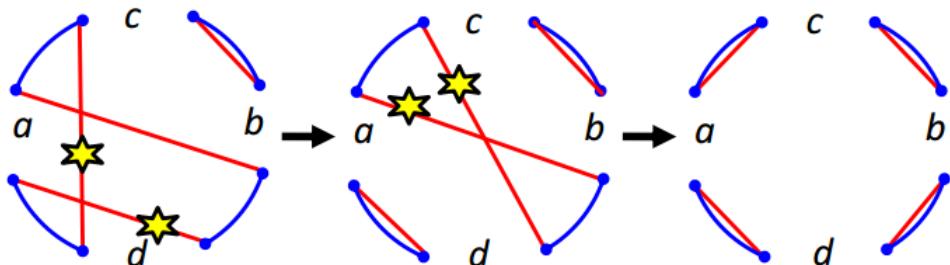
Slika 6.23: Idenički graf prekidnih tačaka

Preuređenje genoma utiče na crveno-plave cikluse. Svaka transformacija $P \rightarrow Q$ (slika 6.24) odgovara transformaciji:

Slika 6.24: Trasnformacija $P \rightarrow Q$

Slika 6.25 prikazuje kako preuređenja genoma utiču na $cycle(P, Q)$. U svakom koraku povećavamo broj ciklusa.

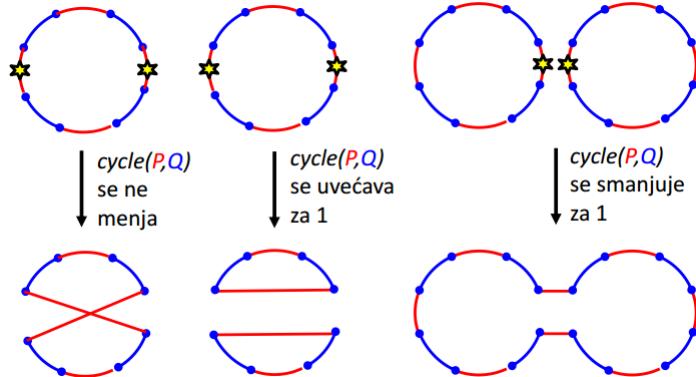
$$\begin{aligned}
 P = (+a -b -c +d) &\rightarrow P' = (+a -\mathbf{b} -c -d) \rightarrow P'' = Q = (+a +c +b -d) \\
 \text{BreakpointGraph}(P,Q) \rightarrow \text{BreakpointGraph}(P',Q) \rightarrow \text{BreakpointGraph}(Q,Q) \\
 \text{cycle}(P,Q)=2 &\rightarrow \text{cycle}(P',Q)=3 \rightarrow \text{cycle}(Q,Q)=4
 \end{aligned}$$

Slika 6.25: Uticaj preuređenja na $cycle(P,Q)$

6.8 Teorema o rastojanju 2-prekida

Želimo sada da generalizujemo problem sortiranja po 2-prekidima. Neka nam je dat niz 2-prekida koji transformiše P u Q . Na taj način će se prekidni graf za P i Q transformisati u graf za Q i Q , odnosno u identički prekidni graf. Isto tako, broj ciklusa će se povećavati dok ne dostigne maksimum. Broj blokova koji učestvuje u izgradnji P i Q označićemo sa $blocks(P, Q)$. Broj crveno-plavih ciklusa uvećava se za $blocks(P, Q) - cycle(P, Q)$.

Koliko može svaki 2-prekid da doprinese ovom uvećanju? 2-prekid može izmeniti $cycle(P, Q)$ najviše za 1. Posmatrajmo sliku 6.26. Na početku imamo 1 ciklus. U prvom prekidu broj ciklusa se ne menja. U drugom prekidu, broj ciklusa se uvećao za 1. Međutim, može se desiti i da se broj ciklusa smanji (u slučaju fuzije), kao u trećem prekidu.



Slika 6.26

Teorema o rastojanju 2-prekida nam govori sledeće:

- Svaki 2-prekid povećava broj ciklusa najviše za 1.
- Za svaki 2-prekid postoji povećanje broja ciklusa za tačno 1.
- Svako sortiranje po 2-prekidima mora povećati broj ciklusa za $\text{blocks}(P, Q) - \text{cycle}(P, Q)$.
- 2-prekid rastojanje između genoma P i Q:

$$d(P, Q) = \text{blocks}(P, Q) - \text{cycle}(P, Q)$$

Evo kako to izgleda na konkretnom primeru. Posmatramo rastojanje 2-prekida između genoma čoveka i miša:

- Genomi čoveka i miša se mogu rastaviti na 280 blokova sintenije (dužine bar pola miliona nukleotida)
- Graf prekidnih tačaka nad ovim blokovima ima ukupno 35 ciklusa
- Na osnovu teoreme o rastojanju 2-prekida:

$$d(H, M) = \text{blocks}(H, M) - \text{cycle}(H, M) = 280 - 35 = 245$$

- Postoje različite verzije scenarija sa 245 koraka.
- Pravi evolutivni scenario je možda imao i više od 245 koraka.

Shortest Rearrangement Scenario je pristup unutar filogenetske analize koji ima pristup da, iako smo možda imali više od 245 koraka, uzima u obzir da je bilo tačno 245 koraka, odnosno $\text{blocks}(H, M) - \text{cycle}(H, M)$ 2-prekida, i razmatra situacije kako se moglo doći od jednog do drugog genoma. Jedan deo pseudokoda dat je u nastavku. 2 – *BreakOnGenome(P, i, i', j, j')* uklanja grane (i, i') i dodaje grane (i, j) i (i', j') (genom predstavljen grafom prekidnih tačaka).

ShortestRearrangementScenario(P, Q)

```

output P
RedEdges ← ColoredEdges(P)
BlueEdges ← ColoredEdges(Q)
BreakpointGraph ← the graph formed by RedEdges and BlueEdges
while BreakpointGraph has a non-trivial cycle Cycle
     $(j, i') \leftarrow$  an arbitrary edge from BlueEdges in a nontrivial red

```

blue cycle
 $(i, j) \leftarrow$ an edge from *RedEdges* originating at node j
 $(i', j') \leftarrow$ an edge from *RedEdges* originating at node i'
 $RedEdges \leftarrow RedEdges$ with edges (i, j) and (i', j') removed
 $RedEdges \leftarrow RedEdges$ with edges (j, i') and (j', i) added
 $BreakpointGraph \leftarrow$ the graph formed by *RedEdges* and *BlueEgdes*
 $P \leftarrow \text{2-BreakOnGenome}(\mathbf{P}, i, i', j, j')$
output \mathbf{P}

Glava 7

Koja životinja nam je donela SARS?

7.1 Uvod

Neka od glavnih pitanja koja možemo postaviti su: Koja životinja nam je donela SARS? Kako smo se prvo bitno zarazili? Kako se SARS širio po svetu? Sva ova pitanja spadaju u domen filogenetske analize koja se bavi rekonstrukcijom evolutivnih stabala.

7.2 Izbijanje epidemije

Kao jedna od bolesti koja se najbrže širila po svetu, ova misteriozna bolest je uspela da iz Hong Konga pređe preko Tihog Okeana za svega sedam dana, gde je nasuprot njoj bolestima kao što je Kuga bilo neophodno da prođe četiri godine da bi samo prešla iz Istanbula do Kijeva, a HIV-u je bilo potrebno da prođe dve decenije da bi obišla ceo svet!

Kada je utvrđeno za novu bolest da je u pitanju virus, primećeno je posmatrajući je pod mikroskopom da pripada porodici Korona virusa, virusa koji izgledaju kao pomračenje sunca ???. (sunčeva korona), po čemu je i dobio ime. Korona virusi su već bili poznati naučnicima i lekarima, samo što je bilo čudno što je nikada nije imao čovek, već samo životinje.

Korona virusi pripadaju virusima koji sadrže RNK-a umesto DNK-a. Kod RNK-a je mnogo veći nivo mutacija nego kod DNK-a, što znači da prilikom udvajanja virusa mnogo cešće dolazi do greške pri replikaciji. Korona virusi mutiraju često i brzo. Za takve viruse veoma je teško naći lek. Na osnovu simptoma ova bolest je dobila naziv: Teški akutni respiratorni sindrom (eng. Severe Acute Respiratory Syndrome, SARS).

Ostaje pitanje, koja životinja, koja je oboljevala od ovog virusa, je uspela da prenese tu bolest ljudima tokom evolucije? Kako je ta životinja uspela da prenese čoveku? Kako se SARS širio po svetu? Na ova pitanja odgovor će nam da ti filogenetska analiza.

7.3 Transformacija matrice rastojanja u evolutivno stablo

Govorili smo o različitim vrstama poravnjanja, između ostalog i o višestrukom poravnjanju. Sada nećemo da ulazimo u to kako smo došli do višestrukog poravnjanja, prepostavimo samo da ga imamo.

7.3.1 Konstrukcija matrice rastojanja

Na osnovu višestrukog poravnjanja dobijamo matricu rastojanja iz koje želimo da konstruišemo evolutivno stablo. Definišimo najpre šta predstavlja svaki element ove matrice. Svaka kolona i svakak vrssta odgovara jednoj vrsti. Na glavnoj dijagonali su nule, a ostali elementi matrice označavaju broj različitih simbola u višestrukom poravnjanju genoma dve vrste. Matrica je simetrična. Na slici 7.1 možemo videti jedan primer matrice rastojanja.

Definicija 7.1. *Jedan element matrice u oznaci $D_{i,j}$ predstavlja broj različitih simbola u i -tom i j -tom redu višestrukog poravnjanja.*

Vrsta	Poravnanje	Matrica rastojanja			
		Šimpanza	Čovek	Foka	Kit
Šimpanza	ACGTAGGCCT	0	3	6	4
Čovek	ATGTAAGACT	3	0	7	5
Foka	TCGAGAGCAC	6	7	0	2
Kit	TCGAAAGCAT	4	5	2	0

Slika 7.1: Prikaz konstrukcije matrice rastojanja

Kada se radi filogenetska analiza obično se uzima protein za koji se zna da obavlja funkciju koja je izuzetno važna i koja se neće menjati tokom ogromnog broja godina. Takav protein se bira jer je on najviše konzerviran odnosno, u njemu se desilo najmanje mutacija.

Za sve vrste korona virusa važi da se sastoje od 6 proteina, i od svih tih proteina odabran je tzv. *spike* protein.

7.3.2 Stabla

Dakle, biramo spike protein ovih vrsta, zatim konstruišemo matricu rastojanja na osnovu koje ćemo formirati evolutivno stablo.

Stablo je povezani aciklični graf. Za povezana aciklična stabla se može pokazati da svako stablo sa bar dva čvora sadrži bar dva lista i svako stablo sa n čvorova sadrži tačno $n - 1$ grana.

U evolutivnom stablu listovi predstavljaju današnje vrste (i oni su stepena 1), dok unutrašnji čvorovi predstavljaju izumrle vrste (stepena većeg od 1). Koreni čvor predstavlja najdaljeg zajedničkog predaka. Koreni čvor predstavlja najdaljeg zajedničkog pretka. Od njega su sve ostale vrste potekle.

Problem filogeneze na osnovu rastojanja: Konstruisatio evolutivno stablo na osnovu matrice rastojanja.

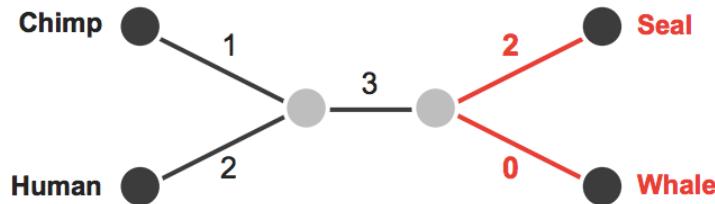
Ulaz: Matrica rastojanja.

Izlaz: Nekoreno stablo koje odgovara matrici rastojanja.

Na slici 7.2 se može videti primer evolutivnog stabla koji odgovara matrici. Na koji način stablo odgovara matrici? Hajde da vidimo za čoveka i šimpanzu, vidimo da je cena puta od jednog do drugog jednak 3, a ista ta vrednost upisana je u matricu. To važi i za sve ostale listove. Može se pojaviti grana težine 0. Ukoliko se to desi između unutrašnjih čvorova, oni se spajaju. Međutim, u našem primeru to se desilo između unutrašnjeg čvora i lista. Ako bismo ih spojili to bi značilo da je kit izumrela vrsta.

Uvodimo pojam rastojanja između dva lista i i j u evolutivnom stablu u oznaci $d_{i,j}$.

	Chimp	Human	Seal	Whale
Chimp	0	3	6	4
Human	3	0	7	5
Seal	6	7	0	2
Whale	4	5	2	0



Slika 7.2: Prikaz matrice rastojanja i njenog odgovarajućeg evolutivnog stabla

Problem izračunavanja rastojanja između listova: Za dato težinsko stablo, izračunati rastojanje između listova.

Ulaz: Težinsko stablo sa n listova.

Izlaz: Matrica $n \times n$ ($d_{i,j}$), gde je $d_{i,j}$ dužina putanje između listova i i j .

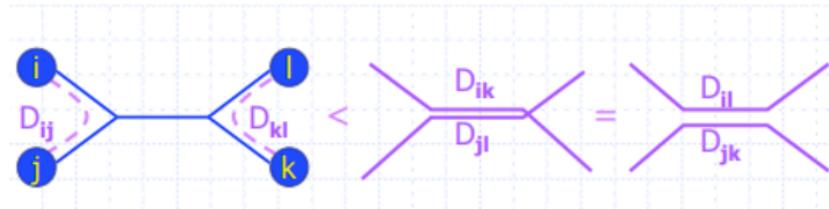
Ovaj problem jeste lako rešiv, međutim, zanima nas kako da za datu matricu rastojanja konstruišemo evolutivno stablo. S obzirom na to da je moguće da iz jedne matrice dobijemo više različitih stabala, neophodno je da matrice zadovoljavaju određena svojstva, kako bi za jednu matricu dobili tačno jedno stablo. Da bi to bilo moguće matrica mora biti aditivna.

Definicija 7.2. Za dato evolutivno stablo, matricu koja opisuje rastojanja između njegovih listova zovemo aditivnom matricom

Teorema 7.3. Matrica D je aditivna akko za proizvoljna četiri indeksa u matrici i, j i k, l važi:

$$D_{ij} + D_{kl} \leq D_{ik} + D_{jl} = D_{il} + D_{jk}$$

Na osnovu ove teoreme i definicije možemo zaključiti na koji način da povežemo listove u evolutivnom stablu 7.3.



Slika 7.3: Prikaz povezivanja listova u evolutivnom stablu

Pošto je za jednu matricu moguće konstruisati više evolutivnih stabala, uvek je bolje izabrati prosto stablo.

Definicija 7.4. *Prosto stablo:* stablo bez čvorova stepena 2.

Teorema 7.5. Postoji tačno jedno prosto stablo koje odgovara aditivnoj matrici.

Sada možemo formulisati **problem filogeneze na osnovu rastojanja**:

Problem filogeneze na osnovu rastojanja: Konstruisati evolutivno stablo na osnovu aditivne matrice rastojanja.

Ulaz: Aditivna matrica rastojanja.

Izlaz: Prosto stablo koje odgovara dатој матрици rastojanja.

7.4 Prema algoritmu za rekonstrukciju filogenetskog stabla na osnovu rastojanja

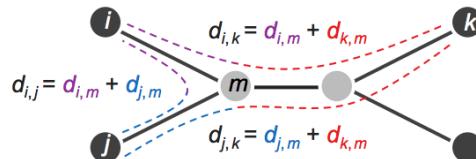
Primetimo da minimalna pozitivna vrednost matrice rastojanja odgovara listovima u stablu koje povezuje zajednički roditelj. Takve listove nazivamo **susednim listovima**. Susedni listovi imaju zajedničkog pretka. U primeru 7.2 foka i kit su susedi (jer dele istog roditelja). Šta ako ne postoje susedni listovi u stablu? Sledeća teorema nam pomaže:

Teorema 7.6. Za svako prosto stablo sa bar dva čvora postoji bar jedan par susednih listova.

Dakle, možemo prepostaviti da ćemo uvek imati par susednih listova. Mi znamo da su *kit* i *foka* susedni listovi i da dele zajedničkog roditelja, međutim, ne znamo koja se vrednost pridružuje svakoj od ovih grana. Da bismo dobili rastojanje između susednih listova i i j od roditelja m potrebno je da imamo još neki list unutar takvog stabla, recimo list k . Razmatramo sledeća rastojanja: $d_{i,k}$, $d_{j,k}$, $d_{i,j}$ su rastojanja između listova koja su data u matrici rastojanja, dok rastojanja $d_{i,m}$ i $d_{j,m}$ koja predstavljaju rastojanja između lista i unutrašnjeg čvora, nisu data 7.4.

Da bismo dobili rastojanje između čora i i čvora j treba da pratimo ljubičasto i plavo rastojanje 7.4. Slično računamo i $d_{i,k}$ i $d_{j,k}$, a $d_{k,m}$ dobijamo sabiranjem kao na slici 7.4. Sa velikim \mathbf{D} ($D_{i,k}, D_{j,k}, D_{i,j}$) označavamo elemente matrice rastojanja, dok sa malim \mathbf{d} označavamo rastojanje između bilo koja dva čora u evolutivnom stablu.

Kada znamo rastojanje od $d_{k,m}$ sada možemo da dobijemo $d_{i,m}$ kao: $(D_{i,k} + D_{i,j}D_{j,k}) / 2$. Analogno dobijamo i za drugog suseda $d_{j,m}$. Obratimo pažnju da je čvor k proizvoljan - bilo koji list koji je različit od listova čija rastojanja do roditelskog čvora tražimo.



$$d_{k,m} = [(d_{i,m} + d_{k,m}) + (d_{j,m} + d_{k,m}) - (d_{i,m} + d_{j,m})] / 2$$

Slika 7.4: Prikaz rekonstrukcije filogenetskog stabla na osnovu rastojanja

Formula za rastojanje $d_{k,m}$, data na slici 7.4, nije nam pogodna jer zavisi od čvora m . Čvor m je unutrašnji čvor i za njega nemamo rastojanja u matrici. Zbog toga prezapisujemo formulu na sledeći način:

$$d_{k,m} = (d_{i,k} + d_{j,k} - d_{i,j})/2 = (D_{i,k} + D_{j,k} - D_{i,j})/2$$

Sada su nam sva rastojanja poznata, odnosno koristimo samo rastojanja između listova. To znači da vrednost između lista i unutrašnjeg čvora možemo da izračunamo preko listova. Na ovaj način, kada znamo rastojanje između k i m , možemo i da izračunamo rastojanje od i do m , sledećom formulom:

$$d_{i,m} = D_{i,k} - (D_{i,k} + D_{j,k} - D_{i,j})/2 = (D_{i,k} + D_{i,j} - D_{j,k})/2$$

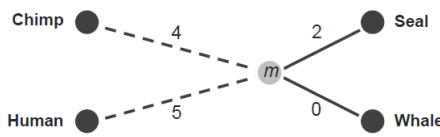
Obratimo pažnju da je k proizvoljan čvor. Uzeli smo bilo koji list koji je bio različit od susednih lvorova i i j .

Posmatrajmo sada opet evolutivno stablo. Foka će biti čvor i , j će biti susedan čvor kit. Za k možemo uzeti po algoritmu, bilo koji čvor, biramo šimpanzu. Sada važi:

$$d_{Seal,m} = (D_{Seal,Chimp} + D_{Seal,Whale} - D_{Whale,Chimp})/2 = 2$$

Kada smo odredili vrednost grane od foke do unutrašnjeg čvora, treba da odredimo i vrednost od unutrašnjeg čvora do kita. To je lako jer znamo ukupno rastojanje od foke do kita, koje iznosi 2, i upravo smo odredili rastojanje od foke do unutrašnjeg čvora, koje takođe iznosi 2. Zbog toga, rastojanje od unutrašnjeg čvora do kita mora biti 0.

Kada su određene ove vrednosti, uvodimo novu vrstu i kolonu u matricu za čvor m . Ona sadrži rastojanja do listova kit i foka. Sada u matrici više ne gledamo kita i foku, već gledamo samo čvor m i računamo ostala rastojanja. Smanjili smo matricu rastojanja na tri vrste, kao na slici 7.6. Dalje ponavljamo postupak.



Slika 7.5: Vrednosti za Seal i Whale su određene i više ih ne koristimo.

	Chimp	Human	m
Chimp	0	3	4
Human	3	0	5
m	4	5	0

Slika 7.6: Uveden je novi čvor m koji posmatramo umesto Seal i Whale.

Ovakav rekurzivni pristup nije moguce primeniti za svaku matricu. U tim situacijama koristimo *AdditivePhylogeny* algoritam.

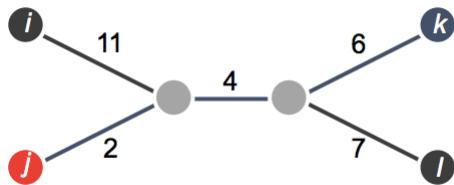
7.5 AdditivePhylogeny algoritam

Na slici 7.7 prikazana je matrica za koju ne možemo da применimo rekurzivni algoritam. Problem je što ćemo u nekom trenutku dobiti granu koja ima negativnu vrednost. Zašto algoritam ne radi? Možemo videti da je minimalni element $D_{j,k}$, a j i k nisu susedi, zato ne možemo primeniti prethodni pristup, iako je matrica ultrametrična. Međutim, ova matrica ima odgovarajuće stablo i ono je prikazano na slici 7.7. Umesto traženja suseda, pokušajmo da izračunamo dužinu krajnjih grana, onih grana koje vode do listova. Formule iz rekurzivnog algoritma (slika 7.4) važe ako su i i j susedni čvorovi. Ovde će važiti slična formula, a sledeća teorema nam govori tome:

Teorema 7.7. O dužini spoljnih grana: $LimbLength(i)$ je jednako minimalnoj vrednosti $(D_{i,k} + D_{i,j} - D_{j,k})/2$ po svim listovima j i k .

Isključujemo prepostavku koju smo imali kod susednih listova, dakle računamo rastojanje samo za jedan list, jer ne znamo koji joj je susedni čvor, pa zato uzimamo sve kombinacije za j i k .

	<i>i</i>	<i>j</i>	k	<i>l</i>
<i>i</i>	0	13	21	22
<i>j</i>	13	0	12	13
k	21	12	0	13
<i>l</i>	22	13	13	0



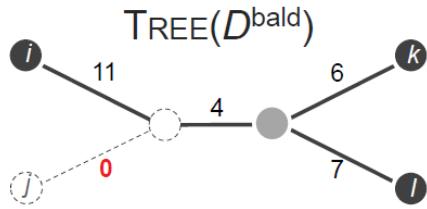
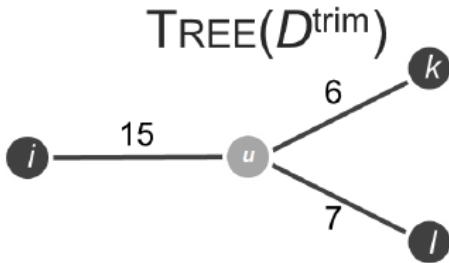
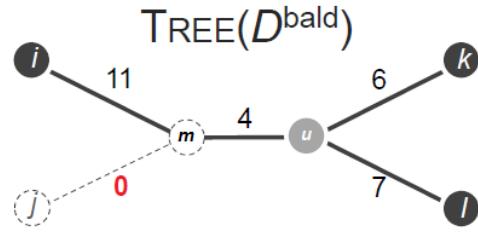
Slika 7.7: Primer matrice za koju ne možemo primeniti prethodni pristup

Prikažimo AdditivePhylogeny algoritam:

1. Izaberemo proizvoljno list, npr. *j*.
2. Izračunamo dužinu njegove krajnje grane, $\text{LimbLength}(j)$. Uzećemo sve moguće kombinacije za preostala dva čvora (*i* i *k*, *i* i *l*, *k* i *l*) i izračunaćemo minimum. Tako dobijemo vrednost 2.
3. Oduzmemosmo $\text{LimbLength}(j)$ od svake grane i dobijemo matricu D^{bald} u kojoj do lista *j* vodi ogoljena (bold) grana (dužine 0). Dakle, *j*-toj vrsti i *j*-toj koloni umanjujemo vrednosti za $\text{LimbLength}(j)$. Stanje nakon promene prikazano je na slici 7.8.
4. Uklonimo *j*-ti red i kolonu iz matrice i dobijemo $(n - 1) \times (n - 1)$ matricu D^{trim} .
5. Konstruišemo $\text{Tree}(D^{trim})$ - rekurzivno.
6. Identifikujemo tačku u $\text{Tree}(D^{trim})$ gde list *j* treba da se nalazi. Ovom koraku ćemo se posebno posvetiti.
7. Dodamo list *j* povezujuci ga granom dužine $\text{LimbLength}(j)$ kako bismo formirali $\text{Tree}(D)$.

Tačku povezivanja za list *j* odredićemo na osnovu teoreme 7.7. Tačka treba da se nađe na putanji između listova *i* i *k* na rastojanju $D_{i,j}^{bald}$ od *i*. Dakle važi: $D_{i,j}^{bald} + D_{j,k}^{bald} = D_{i,k}^{bald}$. Kada smo pronašli tačku treba dodati čvor *j*. Prvo grana koja je povezivala čvor *i* sa unutrašnjim čvorom (označimo ga sa *u*) mora da se ukloni. Zatim se dodaje novi unutrašnji čvor (označimo ga sa *m*) na koji se povezuju *i* i *j*. Trenutno smo u D^{bald} stablu pa je dužina od *j* do *m* jednaka 0, a dužina od *i* do *m* jednaka je $D_{i,j}^{bald}$. Dužina od *m* do drugog unutrašnjeg čvora iznosi $D_{i,u}^{trim} - D_{i,m}^{bald}$. Prikaz postupka može se videti na slikama 7.9 i 7.10.

	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>
<i>i</i>	0	11	21	22
<i>j</i>	11	0	10	11
<i>k</i>	21	10	0	13
<i>l</i>	22	11	13	0

Slika 7.8: Izgled matrice i stabla nakon umanjivanja vrednosti za $LimbLength(j)$ Slika 7.9: Tražimo tačku gde treba da umetnemo čvor j .Slika 7.10: Dodajemo čvor j u stablo.

Dобра strana ovog algoritma je da kreira stablo koje odgovara aditivnoj matrici, a loša da ne radi za neaditivne matrice.

7.6 Metod najmanjih kvadrata

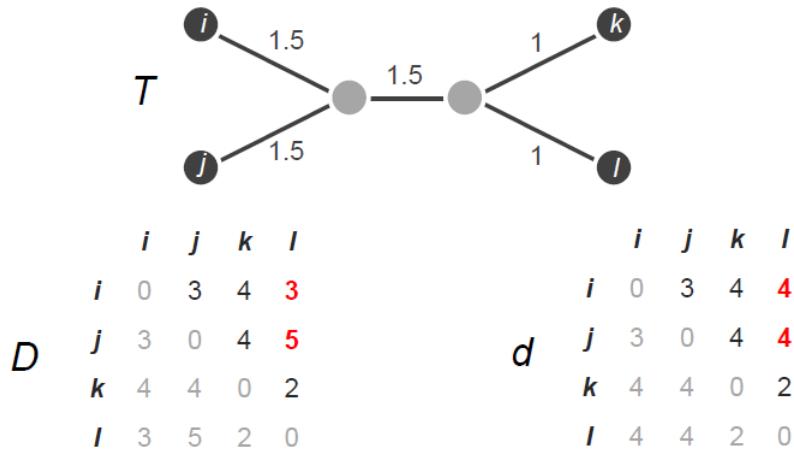
U slučaju da matrica nije aditivna, treba je aproksimirati nekom aditivnom matricom. Na slici 7.11 prikazana je jedna neaditivna matrica (D) za stablo T . Matrica d je njena aproksimacija koju je, u ovom slučaju, bilo lako odrediti. Samo smo izmenili crvene vrednosti. Šta se radi u komplikovanijem slučaju? Računamo diskrepancu za stablo T i odgovarajuću matricu D :

$$\text{Discrepancy}(T, D) = \sum_{1 \leq i < j \leq n} (d_{i,j}(T) - D_{i,j})^2$$

Diskrepanca određuje koliko je d dobra aproksimacija za D . U ovom slučaju matrica d nam je bila poznata. Šta da radimo ako nemamo aditivnu matricu? Treba dodeliti dužine granama u stablu T tako da veličina $\text{Discrepancy}(T, D)$ bude minimalna.

U opštem slučaju, za stablo date topologije postoji algoritam polinomijalne složenosti koji će dodeliti dužine granama stabla tako da diskrepanca bude minimalna. Međutim, u praktičnim primenama neće biti poznata topologija stabla pa stoga moramo računati minimum po svim mogućim stablima. Sa dodavanjem svakog lista u stablo, broj različitih topologija stabala raste eksponencijalno. Problem minimizacije diskrepance po svim mogućim stablima je NP kompletan. U nastavku, razmotrićemo dve heuristike za konstrukciju stabla na osnovu neaditivnih matrica.

Prva heuristika tiče se posebne vrste stabala, a to su ultrametrična evolutvna stabla. Pre nego što ih opišemo prvi da se upoznamo sa još nekim pojmovima.



Slika 7.11: Matrica D nije aditivna i ne možemo primeniti prethodni algoritam. Možemo je aproksimirati aditivnom matricom.

7.6.1 Modelovanje specijacije

U praktičnim primenama, istraživači često prepostavljaju da svaki unutrašnji čvor odgovara *specijaciji* kada se jedna vrsta deli na dve. Podsetimo se da u listovima se nalaze organizmi koje pripadaju vrste koje postoje danas, a u unutrašnjim čvorovima su izumrele vrste. Čvor koji ima dva potomka odgovara evolutivnom događaju specijacije kada je jedna vrsta izumrela ali su sve dve vrste tada razdvojile.

Definicija 7.8. *Nekoreno binarno stablo:* svaki čvor je stepena 1 ili 3.

Definicija 7.9. *Koreno binarno stablo:* nekoreno binarno stablo sa korenom (jedini će biti stepena 2) postavljenom na jednoj od grana.

7.6.2 Ultrametrična stabla

Recimo da smo odredili korensko stablo. Sada želimo da svakom čvoru dodelimo startost. Listovi imaju starost 0 (starost u smislu pre koliko miliona godina su izumrele).

Definicija 7.10. *Molekularni sat:* dodeljuje starost svakom čvoru u stablu (starost listova = 0).

Definicija 7.11. *Težine grana:* razlika u starosti čvorova koje povezuju.

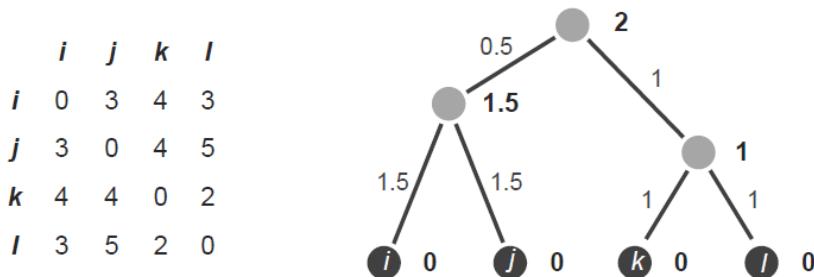
Definicija 7.12. *Ultrametrično stablo:* koreno stablo u kom važi da udaljenost od korena do bilo kog lista je ista i predstavlja starost korena.

UPGMA: heurističko klasterovanje

1. Formirati klaster za svaku današnju vrstu. Svaki klaster sadrži jedan list.
2. Naći dva najbliža klastera C_1 i C_2 na osnovu prosečnog rastojanja između njihovih članova $D_{avg}(C_1, C_2) = \sum_{i \text{ in } C_1, j \text{ in } C_2} D_{i,j} / |C_1| \bullet |C_2|$ gde $|C|$ označava broj elemenata u klasteru C . Za jednočlane klastere rastojanje je dato u matrici rastojanja.
3. Spojiti C_1 i C_2 u jedinstveni klaster C .
4. Formirati novi čvor za klaster i granama povezati ga za čvorovima. Postaviti starost čvora C na $D_{avg}(C_1, C_2)/2$.

5. Ažurirati matricu rastojanja tako što ubacimo novi čvor, izbacimo čvorove koje on sadrži i izračunamo rastojanja kao prosečna rastojanja između svaka dva para klastera.
6. Iteriramo dok ne dođemo do jednog klastera koji sadrži sve vrste.

Dobra strana ovog algoritma je da kreira stablo za svaku matricu, a loša da stablo ne mora da odgovara aditivnoj matrici. Na slici 7.12 je prikazan primer stabla dobijen ovim algoritmom. Vidimo da stablo ne odgovara datoj aditivnoj matrici.



Slika 7.12: Stablo dobijeno primenom heuristike UPGMA.

7.7 Neighbour-Joining teorema

Za datu matricu rastojanja D dimenzije $n \times n$, njena **neighbour-joining** matrica u označi D^* definiše se kao:

$$D_{i,j}^* = (n-2) \bullet D_{i,j} - \text{TotalDistance}_D(i) - \text{TotalDistance}_D(j)$$

gde je $\text{TotalDistance}_D(i)$ suma rastojanja od i do svih ostalih listova.

Teorema 7.13. Neighbour-joining: ako je matrica D aditivna, onda minimalni element matrice D^* odgovara susednim listovima u stablu $\text{Tree}(D)$.

1. Konstruišemo neighbour-joining matricu D^* na osnovu matrice D .
2. Nađemo minimalni element $D_{i,j}^*$ matrice D^* .
3. Izračunamo $\Delta_{i,j} = (\text{TotalDistance}_d(i) - \text{TotalDistance}_d(j)) / (n-2)$.
4. Našli smo da i i j imaju zajedničkog pretka i povezujemo ih u jedan čvor. Treba da odredimo težine grana do novog čvora. Postavimo $\text{LimbLength}(i)$ na $1/2(D_{i,j} + \Delta_{i,j})$ i $\text{LimbLength}(j)$ na $1/2(D_{i,j} + \Delta_{i,j})$.
5. Formiramo matricu D' tako što uklonimo i -ti i j -ti red/kolonu iz D i dodamo m -ti red/kolonu tako da za svako k važi $D_{k,m} = (D_{i,k} + D_{j,k} - D_{i,j}) / 2$. Ovim smo smanjili dimenziju našeg problema za 1.
6. Primenimo *NeighborJoining* rekurzivno na D' da dobijemo $\text{Tree}(D')$.
7. Vratimo krajnje grane do čvorova i i j i dobijemo $\text{Tree}(D)$.

Iako je *Neighbor Joining* algoritam bio revolucionarno otkriće. Međutim, metodi zasnovani na rastojanju imaju i mane. Dakle, mi smo birali nekakve reprezentativne sekvence, zatim smo ih poravnali. Kada višestruko poravnanje zamenimo matricom rastojanja, gubimo informacije o sekvencama iz poravnjanja. Zbog toga ne možemo da zaključimo kakva je sekvenca odgovarala vrstama iz unutrašnjih čvorova. Samim tim ne možemo da vidimo do kakvih je to mutacija tačno došlo tokom evolucije.

7.8 Rekonstrukcija stabla na osnovu karakteristika

Pre oko pedeset godina, istraživači su konstruisali filogenetska stabla na osnovu anatomsko-fizioloških osobina organizama koje su nazvane **karakteristikama**. U odnosu na karakteristike konstruisali su tabele karakteristika. Jedna takva tabela prikazana je na slici 7.13. One koji imaju slične karakteristike smatramo bližim evolutivnim rođacima.

	wings	legs
winged stick insect	Yes	6
wingless stick insect	No	6
giant centipede	No	42



Slika 7.13: Tabela karakteristika

Ono što bi mogao da bude problem jeste to što se mi u ovom poglavljju bavimo virusima. Za njih baš nemamo tako očigledne karakteristike, kao što su imaju/nema krila, broj nogu i slično. Ovako nešto nije primenjivo na njih.

7.8.1 Filogeneza na osnovu karakteristika

Problem filogeneze zasnovane na karakteristikama: Rekonstruisati evolutivno stablo na osnovu karakteristika.

Ulez: Tabela karakteristika n x m za n vrsta i m karakteristika.

Izlaz: Stablo kog kog su vrste sa sličnim karakteristikama blizu jedna drugoj.

Kako bismo konstruisali evolutivno stablo na osnovu karakteristika? Ako bismo radili sa insektima, na jednu stranu bismo stavili one koji npr. imaju krila, a sa druge one koji nemaju krila.

Dolooov zakon o nepovratnosti evolucionih procesa (1893): *evolucija ne izmišlja dva puta isti organ (npr. krila kod insekata).*

Ipak, pojavili su se izuzeci Dolooovog zakona. Ispitivanje je rađeno na velikom uzorku insekata. Tokom studije pronađen je izuzetak Dolooovog zakona, desilo se da je vrsta izgubila krila tokom evolucije, a kasnije tokom evolucije krila su se ponovo razvila. Kako je to moguće?

Vratimo se na DNK, na osnovu svega. Unutar DNK je upisano kako će izgledati svaki organ koji neki organizam poseduje. To što je nešto zapisano u okviru DNK ne znači da će se to ispoljiti. Tako kod insekata, jedna vrsta možda sadrži gen za krila, ali on se ne mora ispoljiti baš kod te vrste, može i kod nekog njenog potomka. Do ekspresije može, i ne mora da dođe.

Došlo se do ideje da se poravnjanje koristi kao tabela karakteristika (poravnanje DNK ili RNK sekvenci). Tako bismo rešili problem karakteristika kod virusa.

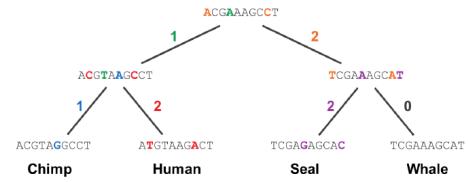
7.9 Problem male parsimonije

Dato nam je binarno stablo. U listovima su organizmi. Ono što nas zanima jesu sekvene predaka. Na slici 7.14 data nam je tabela karakteristika i stablo. Ono što je poznato jeste koji su listovi susedni. Zanima nas do kakvih je mutacija došlo tokom evolucije, odnosno, zanima nas koja je sekvenca njihovog zajedničkog pretka.

Chimp	ACGTAGGCCT
Human	ATGTAAGACT
Seal	TCGAGAGCAC
Whale	TCGAAAGCAT

```

graph TD
    Root["?????????"] --- L1L["?????????"]
    Root --- L1R["?????????"]
    L1L --- Chimp["Chimp"]
    L1L --- Human["Human"]
    L1R --- Seal["Seal"]
    L1R --- Whale["Whale"]
  
```



Slika 7.15: Skor parsimonije

Slika 7.14: U listovima su sekvene različitih vrsta i želimo da otkrijemo sekvene njihovih predaka.

Na slici 7.15 dato je jedno moguće rešenje. Kvalitete rešenja određuje nam skor. Skor parsimonije je suma Hamingovih rastojanja duž svake grane. Za svaku granu možemo da odredimo Hamingovo rastojanje niski koje se nalaze na krajevima te grane. Skor parsimonije na ovoj slici jednak je 8.

Problem male parsimonije: Odrediti označe za unutrašnje čvorove korenog stabla.

Ulaz: Koreno binarno stablo gde je svaki list označen jednim simbolom.

Izlaz: Oznake za sve ostale čvorove stabla takve da minimizuju skor parsimonije stabla.

Algoritam dinamičkog programiranja: Neka je T_v podstablo stabla T sa korenom u čvoru v . Neka je $s_k(v)$ minimalni skor parsimonije stabla T_v za sva moguća obeležavanja, pod pretpostavkom da je čvor v obeležen simbolom k . Minimalni skor parsimonije stabla jednak je minimalnoj vrednosti $s_k(\text{root})$ po svim simbolima k . Neka je $\delta_{i,j}$ Kronekerov delta simbol:

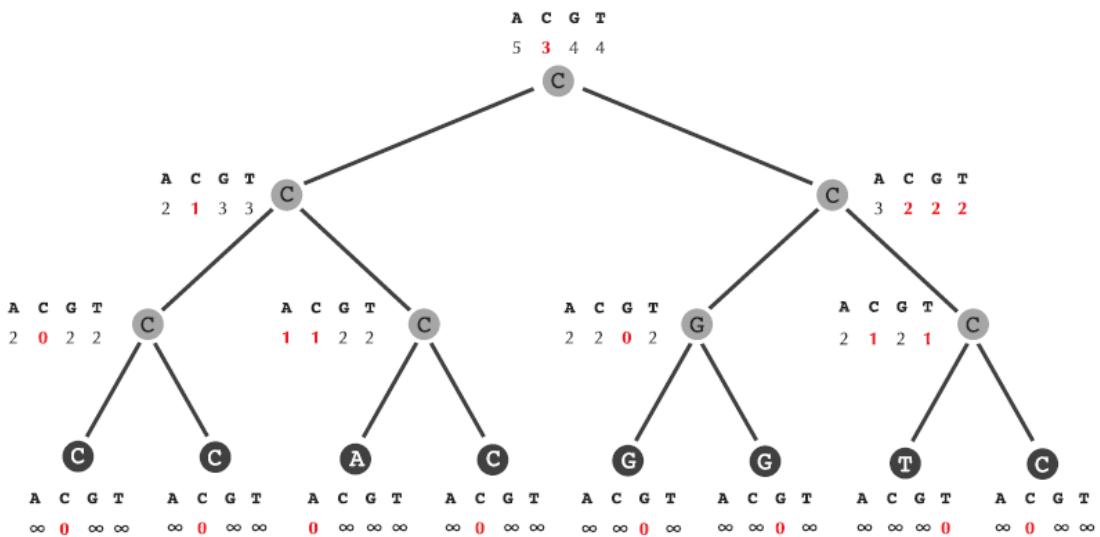
- $\delta_{i,j} = 0$ ako $i = j$
- $\delta_{i,j} = 1$ inače

Važi sledeća rekurentna relacija:

$$s_k(v) = \min_{\text{all symbols } i} s_i(\text{Daughter}(v)) + \delta_{i,k} + \min_{\text{all symbols } j} s_j(\text{Son}(v)) + \delta_{j,k}$$

gde $\text{Daughter}(v)$ označava levo podstablo, a $\text{Son}(v)$ desno podstablo sa korenom u čvoru v .

Pogledajmo na primeru kako to funkcioniše. Na početku imamo po jedan nukleotid u svakom listu. Njegova vrednost je obeležena sa 0, dok je kod ostalih vrednost ∞ . Onda za svaki nukleotid računamo skor parsimonije i uzimamo vrednost koja je minimalna. To radimo dok ne stignemo do korena. Kada stignemo do korena radimo backtraking, ondnosno idemo unazad kako bismo odredili oznake za unutrašnje čvorove. Biramo ih tako da bude što manje mutacija, a to je nukleotid kome odgovara najmanja vrednost. Može se desiti da više nukleotida ima najmanju vrednost i tada je u suštini sve jedno koji ćemo odabrat. Međutim, ako je neki od tih nukleotida predak najbolje je da njega odaberemo kako ne bi bilo mutacije. Na slici 7.16 prikazan je jedan primer izvršavanja ovog algoritma.



Slika 7.16: Primena algoritma dinamičkog programiranja na problem male parsimonije.

7.10 Problem velike parsimonije

Kod problema male parsimonije, filogenetska stabla su uvek binarna i uvek korena. Algoritam za rešavanje podrazumeva da nam je poznata topologija stabla, odnosno, znamo gde koji čvor stoji. Nekada to nije poznato, ne znamo koji listovi treba da budu susedni. Ovo je problem velike parsimonije i u nastavku ćemo ga detaljnije opisati.

Problem velike parsimonije: Za dati skup niski, naći stablo čiji su listovi označeni ovim niskama koje ima najmanji skor parsimonije.

Ulaz: Kolekcija niski jednakе дужине.

Izlaz: Koreno binarno stablo T koje minimizuje skor parsimonije po svim mogućim korenim binarnim stablima čiji su listovi označeni datim niskama.

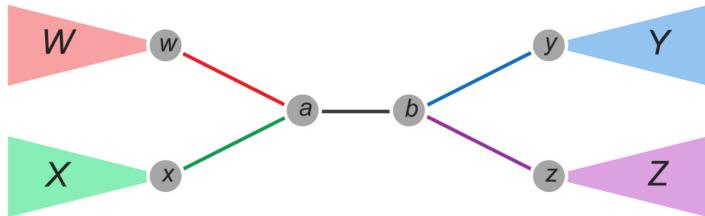
Ovaj problem je NP-kompletan.

7.10.1 Pohlepna heuristika za veliku parsimoniju

Trenutno posmatramo nekoreno stablo, bitno je da ćemo lako odrediti koreno. Od celog stabla izdvojili smo jednu unutrašnju granu koja spaja unutrašnje čvorove a i b (sa po najviše 2 potomka). Svaki od ovih čvorova može imati dalje potomke pa definisemo četiri podstabla.

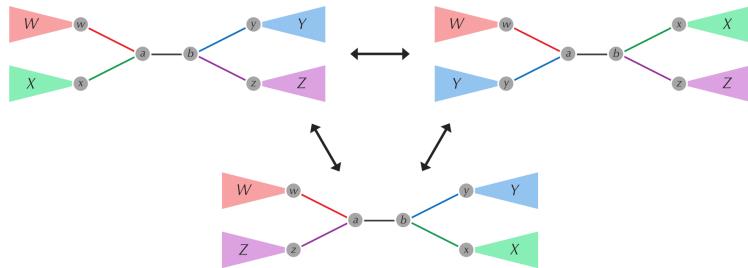
Primetimo da uklanjanje unutrašnje grane od a do b (i oba čvora sa ostalim granama koje su povezane na njih) dovodi do stvaranja četiri podstabla (W , X , Y , Z) 7.17.

Pre nego što smo uklonili ove grane, prepostavili smo raspored kakav je na slici 7.17. Dakle, prepostavili smo neku topologiju. Međutim, to je nešto što nam nije poznato kod problema velike parsemonije.



Slika 7.17: Prikaz stvaranja četiri podstabla

Preuređenje rasporeda ovih stabala se naziva razmena najbližih suseda 7.18.



Slika 7.18: Prikaz razmene najbližih suseda

Problem najbližih suseda u stablu: Za datu granu u binarnom stablu, generisati dva suseda ovog stabla.

Ulaz: Unutrašnja grana binarnog stabla.

Izlaz: Dva najbliža suseda ovog stabla za datu unutrašnju granu.

Heuristika za razmenu najbližih suseda:

1. Postaviti trenutno stablo na koreno binarno stablo proizvoljne strukture.
2. Proći kroz sve unutrašnje grane (a ne jednu kao malo pre) i izvršiti sve moguće razmene najbližih suseda.
3. Rešiti problem male parsimonije za svako takvo stablo.
4. Ako stablo ima skor parsimonije bolje od optimalnog stabla, postaviti da to bude trenutno stablo; inače, vratiti trenutno stablo.

Glava 8

Kako locirati mutacije koje izazivaju bolesti?

8.1 Mapiranje očitavanja

Cena sekvencionisanja genoma je od 2001. u konstantnom padu, i teži se tome da ono postane potpuno pristupačno običnom čoveku, kao i da bude sastavni deo lekarske usluge.

Oko 1% dece rodi se sa mentalnom retardacijom, ali uzroci ove pojave i dan-danas nisu razjašnjeni, jer do nje može dovesti niz različitih genetskih poremećaja. Jedan od njih je i Ohdo sindrom, koji izaziva bezličan „maskoliki” izraz lica. Biolozi su 2011. godine uspeli da pronađu niz zajedničkih mutacija kod pacijenata, koje su kasnije iskorišćene za identifikovanje jedinstvene mutacije proteina, odgovorne za nastanak ovog sindroma. Razumevanje suštinskog uzroka Ohdo sindroma samo je jedno od otkrića do kojeg se došlo proučavanjem genetskih poremećaja upotrebom **mapiranja očitavanja**. Kod ove metode, porede se sekvencionisana očitavanja DNK uzeta od pojedinaca sa *referentnim ljudskim genomom*.

Referentni ljudski genomom zamišljen je da bude „prosečan” ljudski genom izračunat na određenom broju uzoraka. Trenutni referentni genom baziran je na genomima 13 dobrovoljaca iz SAD-a; i dalje se vrši njegovo usavršavanje ispravljanjem grešaka i popunjavanjem rupa (trenutno ih je preko stotinu). U proseku, razlika između individualnog i referentnog genoma je u oko 3 miliona mutacija.

Pitanje je kako možemo efikasno sastaviti individualne genome koristeći referentne. Možemo koristiti **asempliranje**, ali konstrukcija de Brojnovog grafa zahteva mnogo memorije. Možemo koristiti postojeću strukturu referentnog genoma kao pomoć u sekvencioniranju genoma pacijenta.

Mapiranje očitavanja predstavlja određivanje pozicije u referentnom genomu sa kojima svako očitavanje ima visoku sličnost. Podsetimo se da su očitavanja izlaz sekvencera, dakle, neki sitni delovi genoma.

```
CTGAGGATGGACTACGCTACTACTGATAGCTGTTT  
GAGGA      CCACG      TGA-A
```

Slika 8.1: Primer mapiranja očitavanja; gornja niska predstavlja referentni genom, a donje niske očitavanja individualnog genoma

8.1.1 Egzaktno uparivanje šablonu

Potrebno je pronaći gde se očitavanja egzaktno poklapaju sa referentnim genomom. Postoji jednostruko i višestruko uparivanje šablonu. Egzaktno u smislu da nema insercija i delecija.

Problem jednostrukog uparivanja šablonu:

Ulaz: Niske *Pattern* i *Genome*.

Izlaz: Sve pozicije u niski *Genome* gde se niska *Pattern* pojavljuje kao podniska.

Problem višestrukog uparivanja šablonu:

Ulaz: Kolekcija niski *Patterns* i *Genome*.

Izlaz: Sve pozicije u niski *Genome* gde se niske iz kolekcije *Patterns* pojavljuju kao podnische.

8.1.2 Rešenje grubom silom

Rešenje koje nam prvo pada na pamet je rešavanje problema grubom silom. Algoritam se sastoji u tome da se linearno krećemo kroz genom i proveravamo da li se dati šablon poklapa sa podniskom genoma iste dužine, koja počinje na toj poziciji.

panamabananas
pana

Slika 8.2: Uparivanje šablonu grubom silom
- nepoklapanje

panamabanananas
nana

Slika 8.3: Uparivanje šablonu grubom silom
- poklapanje

Vreme izvršavanja algoritma u slučaju jednostrukog *Pattern-a* je $O(|Genome| * |Pattern|)$, dok je u slučaju višestrukog *Patterns-a* je $O(|Genome| * |Patterns|)$, gde je $|Patterns|$ suma dužina elemenata liste *Patterns*.

Međutim, problem je u tome što genomi mogu biti veoma dugi. U slučaju ljudskog genoma (3 GB), ukupna dužina svih očitavanja može biti veća od 1 TB; kao rezultat toga, algoritam složenosti $O(|Genome| * |Patterns|)$ je previše spor.

U nastavku smo u potrazi za efikasnijim algoritmom, ali i pogodnijom strukturu podataka. U algoritmu svakom koraku radimo poređenje iznova, što dovodi do neefikasnosti.

8.1.3 Sufiksna stabla

Razlog velike neefikasnosti prethodnog algoritma jeste u tome što paterni prolaze kroz genom nezavisno jedan od drugog. Ako nisku *Genome* zamislimo kao put, onda bi izvršavanje algoritma grube sile bilo analogno vožnji svakog paterna po putu u zasebnom automobilu. Ono što želimo jeste da sve paterne smestimo u jedan „autobus”, čime bi nam bio dovoljan samo jedan prolazak kroz nisku *Genome*. Zbog toga paterne organizujemo u strukturu podataka nalik na usmereni aciklički graf, koju nazivamo **Trie** i koja ima sledeće osobine:

- Trie ima jedinstven čvor sa ulaznim stepenom nula, koji nazivamo koren
- Svaka grana Trie je obeležena jednim slovom
- Grane koje izlaze iz jednog čvora obeležene su različitim slovima

- Svaki sufiks neke niske dobija se nadovezivanjem slova duž neke putanje grafa, idući od korena naniže
- Svaka putanja stabla od korena do lista, ili do čvora sa izlaznim stepenom 0, predstavlja jedan element iz liste *Patterns*

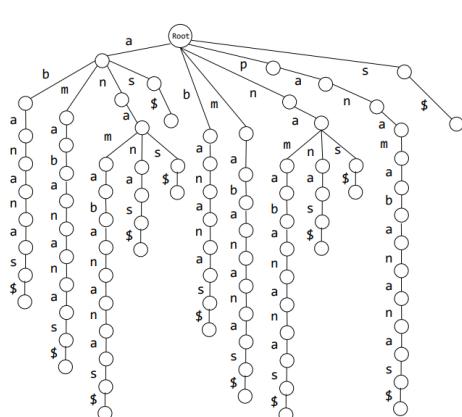
Najjednostavniji način konstrukcije strukture Trie jeste iterativno dodavanje niski iz niza *Patterns* u rastuću strukturu. Za date niske Text i Trie(*Patterns*) možemo brzo proveriti da li se neki element niza Patterns predstavlja prefiks niske Text. Dovoljno je da krenemo da spelujemo Text i da slovo po slovo prolazimo kroz Trie od korena naniže. Za svako slovo iz Text-a gledamo da li iz trenutnog čvora postoji grana obeležena tim slovom; ukoliko postoji, nastavljamo sa pretragom; u suprotnom obustavljamo pretragu i zaključujemo da nijedan element niza Patterns nije prefiks niske Text. Ukoliko stignemo do lista, onda je pretraga bila uspešna.

Da bismo pronašli da li se neki patern nalazi u genomu, potrebno je da u $\|Text\|$ iteracija pokrećemo prethodno opisani algoritam, pri čemu u svakoj iteraciji iz niske Text izbacujemo početni simbol, sve dok ona ne postane prazna.

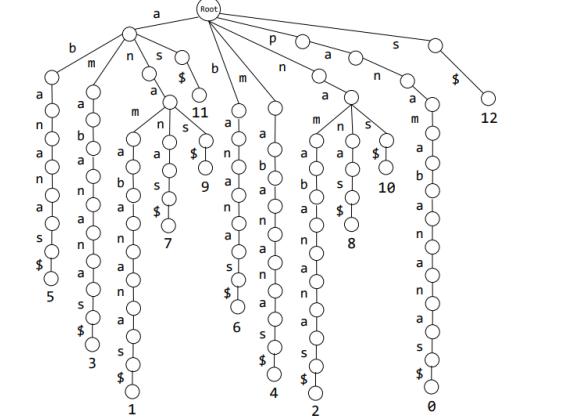
Iako je prethodno opisani postupak vremenski efikasan, njegova mana leži u velikom zauzeću memorije. Naime, veličina strukture Trie proporcionalna je ukupnom broju simbola niza Patterns, a pošto veličina kolekcije očitavanja ljudskog genoma može dostići 1 TB, memorija potrebna za čuvanje ove strukture je prevelika.

Bolji pristup je da se struktura Trie pravi na osnovu niske Text, tj. genoma. Za ovo će nam biti struktura poznata kao **sufiksni trie**. **Sufiksni trie** date niske predstavlja trie formiran na osnovu svih sufiksa te niske. Pre konstrukcije sufiksног stabla, na kraj niske dodajemo simbol \$, kako bismo kasnije znali kad smo stigli do kraja.

Proveru da li se dati patern nalazi u tekstu vršimo tako što tražimo put u sufiksnom stablu, speljući slova paterna do kraja. Ukoliko dođe do nepoklapanja, pretraga je neuspela. U suprotnom, pretraga je uspešna. Primer jednog nekompresovanog sufiksnog stabla nalazi se na slici 8.4.



Slika 8.4: Nekompresovanog sufiksno stablo



Slika 8.5: Sufiksno stablo sa numerisanim prefiksima

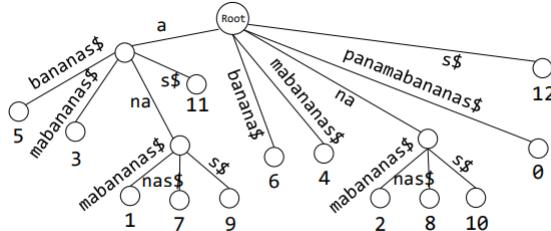
Ovim postupkom možemo utvrditi da li se pattern pojavljuje u genomu, ali ne i na kojoj poziciji. Za to moramo dodati još informacija u stablu. Na svakom listu dodamo početnu poziciju u niski Genome sufiksa koji se završava u tom listu, kao na slici 8.5.

Sad se postavlja pitanje, kada pronađemo uparivanje, kako da znamo na kojoj poziciji se ono nalazi. To je sada lako, kada pronađemo uparivanje, nastavimo sa kretanjem naniže do lista, gde se nalazi pozicija odakle počinje pojavljivanje podniske.

Da bismo smanjili prostornu složenost, možemo kompresovati svaku putanju koja se ne grana u jednu granu. Ovakva struktura podataka naziva se **sufiksno stablo**. Kompresovano sufiksno

stablo prikazano je na slici 8.6.

Za svaku nisku *Genome* važi da je ukupan broj čvorova manji od dvostrukih dužina niske *Genome*, tj. $\#nodes < 2|Genome|$. Ovo važi na osnovu činjenice da je broj listova jednak dužini genoma, tj. $\#leaves = |Genome|$, odnosno da je broj unutrašnjih čvorova manji od dužine genoma umanjene za jedan, tj. $\#internalnodes < |Genome| - 1$.



Slika 8.6: Kompresovano sufiksno stablo

Vremenska složenost za konstrukciju sufiksнog stabla tako što se prvo konstruiše nekompre-
sovano sufiksno stablo je $O(|Genome|^2)$, a za nalaženje uparivanja $O(|Patterns|)$. **Prostorna
složenost** za konstrukciju sufiksнog stabla tako što se prvo konstruiše nekompresovano sufiksno
stablo je $O(|Genome|^2)$, a za čuvanje sufiksнog stabla $O(|Patterns|)$.

Postoje algoritmi sa linearном prostornom i vremenskom složenoшćу. Vremenska složenost za
konstrukciju sufiksнog stabla direktno $O(|Genome|)$, a za nalaženje uparivanja je $O(|Patterns|)$
što ukupno iznosi $O(|Genome| + |Patterns|)$. Prostorna složenost za konstrukciju sufiksнog
stabla direktno $O(|Genome|^2)$, a za čuvanje sufiksнog stabla $O(|Patterns|)$ što ukupno iznosi
 $O(|Genome|)$.

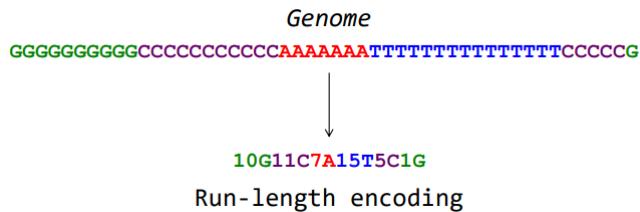
8.2 Kompresija niski i Barouz-Vilerova transformacija

Najveći problem koji se javlja sa prethodnom rešenjem je to što O -notacija ignoriše konstante,
a najpoznatija implementacija sufiksнih stabala zahteva $20 * |Genome|$ (npr. veličina humanog
genoma je 3GB $=>$ 60 GB; i dalje unapređenje u odnosu na 1TB). Postavlja se pitanje da li
možemo smanjiti faktor konstante. Odgovor nam daje kompresija genoma.

8.2.1 Kompresija genoma

Glavna ideja ovog rešenja jeste da se smanji količina memorije potrebna za čuvanje niske
Genome. Za ovo su nam potrebne metode za kompresiju niske velikih dužina, što je naizgled
sasvim drugačiji problem.

U ovom genomu imamo nekoliko uzastopnih ponavljanja jedne aminokiseline (ranovi, runs):
prvo uzastopna ponavljanja aminokiseline G, pa C i tako dalje), a u nekim imamo uzastopna
ponavljanja nizova aminokiselina (ripitsi, repeats): prvo uzastopna ponavljanja GAC, pa CATT
i tako dalje.

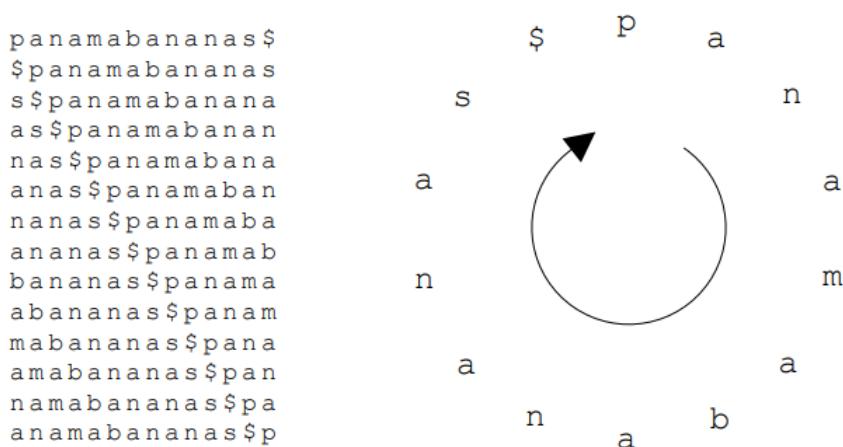


Slika 8.7

Prva ideja pri rešavanju ovog problema jeste da kodiramo dužine ranova. Problem kod ovog pristupa jeste to što u genomu nema mnogo ranova. Međutim, ima mnogo ripita. Postavlja se pitanje kako izvesti transformaciju ripita u ranove. Znamo kako da kodiramo dužinu ranova. Imamo genomsku sekvencu *Genome* i ona sigurno sadrži neke ripite. Međutim, za njih ne znamo kodiranje. Možemo konvertovati nisku u *Genome** koja predstavlja nisku u kojoj su se ripiti konvertovali u ranove. A, kada to uradimo, onda znamo kako da kodiramo ranove i dobijamo *CompressedGenome*.

Kako onda kodiramo ripite u ranove? Odgovor na ovo pitanje daje nam Barouz-Vilerova transformacija (The Burrows-Wheeler Transform, skraćeno BWT).

Ideja kod ovog algoritma je da se na početku formiraju sve ciklične rotacije date niske.



Slika 8.8: Scenario sa 4 promene

Takvu kolekciju niski sortiramo leksikografski (\$ je na početku). BWT je u ovako sortiranoj kolekciji niski poslednja kolona. Možemo primetiti da poslednja kolona sadrži veliki broj ranova. Međutim, isti slučaj je i sa prvom kolonom. Prvo ćemo se pozabaviti dekompresijom dobijene niska, pa ćemo se posle vratiti na ovo pitanje.

8.3 Inverzna BWT

Pogledajmo primer BWT-a za nisku „banana”. Ako sortiramo karaktere poslednje kolone „annb\$aa”, dobićemo prvu kolonu matrice. Na osnovu toga znamo 2-gramske sastav cirkularne niske *banana\$*. Sortiranjem niski dobijamo prve dve kolone matrice. Sada imamo dve kolone cikličnih niski. Zatim ponavljamo postupak - dodamo poslednju koju znamo, itd. Na kraju dobijamo rekonstruisanu celu matricu

\$b	anana	a \$	\$b
a \$	banan	na	a \$
ana \$	ban	na	an
anana \$	b	ba	an
banana \$	2-mers	\$b	ba
na \$	ban a	an	na
n a	na \$	an	na

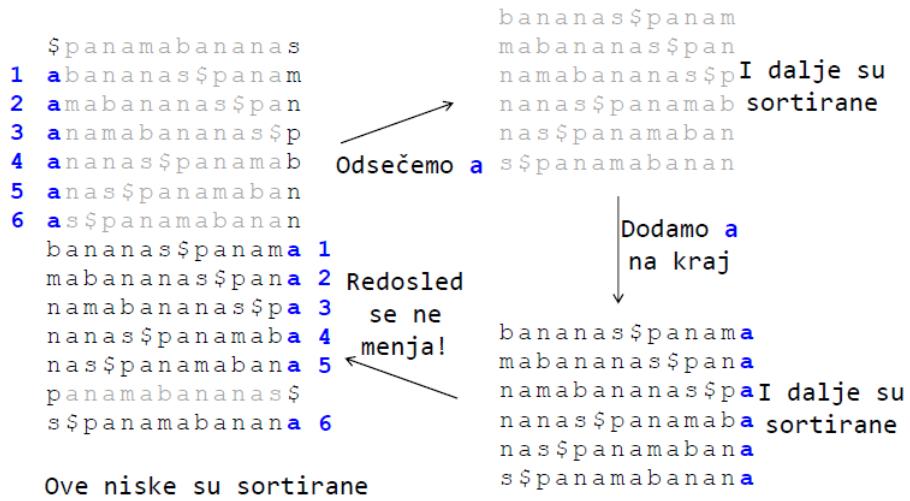
Slika 8.9

Nisku banana\\$ dobijamo tako što uzmemo sve elemente iz prvog reda posle \$. **Prostorna složenost:** Rekonstrukcija niske Genome na osnovu BWT(Genome) zahteva čuvanje $|Genome|$ kopija niske Genome, što iznosi $O(|Genome|^2)$. Poboljšanje složenosti je moguće ako primetimo neobično svojstvo.

First-Last svojstvo: k-to pojavljivanje simbola u FirstColumn i k-to pojavljivanje simbola u LastColumn odgovaraju istoj poziciji simbola u niski Genome.

Ako na slici 8.10 posmatramo prvi karakter 'a' u prvoj koloni, i prvi karakter 'a' u poslednjoj koloni, primetićemo da se te dve niske razlikuju samo za to 'a', odnosno razlikuju se po tome što je karakter 'a' kod prve niske u prvoj koloni, a kod druge u poslednjoj koloni. To važi i za ostale niske.

Hajde da proverimo da li ovo uvek važi. Posmatramo prvu kolonu i vidimo da se karakter 'a' pojavljuje u 6 cikličnih rotacija na prvom mestu. Izdvojimo te niske i prvu kolonu (tj. karakter 'a') i dobijamo kolekciju koja je i dalje sortirana. Zatim dodamo karakter 'a' na kraj, i dalje imamo sortirane niske. I to su niske koje se nalaze u ostatku kolekcije, gde je 'a' na kraju.



Slika 8.10: First-Last svojstvo

8.3.1 Efikasnija BWT dekompresija

Krenemo od simbola \$ (prvi u nizu cikličnih niski) u FirstColumn, zatim pogledamo koji simbol je u LastColumn u tom redu, nađemo ga u FirstColumn, onda za taj red nađemo koji simbol je u tom redu u LastColumn, itd. U jednom trenutku ćemo doći do \$ u LastColumn i tada smo okrenuli ceo krug i rekonstruisali celu Genome nisku. Prostorna složenost je $2|Genome| = O(|Genome|)$.

8.3.2 Korišćenje BWT za uparivanje šablonu

Da se podsetimo, uparivanje šablonu korišćenjem sufiksnih stabala zahtevalo je vremensku složenost od $O(|\text{Genome}| + |\text{Patterns}|)$, prostorna $O(|\text{Genome}|)$. Problem je bio što je sufiksno stablo tražilo $20 * |\text{Genome}|$ prostora. Poboljšanje možemo dobiti ako umesto sufiksnog stabla koristimo BWT(*Genome*) kao strukturu podataka.

Postupak se sastoji od toga da krenemo od kraja niske koju tražimo i u FirstColumn nađemo taj karakter. Zatim u odgovarajućem redu u LastColumn tražimo drugi od pozadi karakter od tih kojima je u FirstColumn poslednji iz uzorka.

Zatim nađemo u FirstColumn gde su ti iz LastColumn i gledamo naredni karakter. Ako se poklapa sa trećim od pozadi, nastavljamo dalje, ako ne, nema ga. I tako dok ne pređemo ceo uzorak od kraja ka početku.

Pogledajmo to na primeru. Tražimo nisku „ana” u niski „panamabananas\$”. U prvoj koloni imamo 6 niski koje počinju karakterom ‘a’. Posmatramo poslednju kolonu i tražimo slovo ‘n’. Takvih kolona ima tri, ostale nećemo dalje razmatrati. Karakteri ‘a’ kojima prethodni ‘n’ obeleženi su redom brojevima 2, 5 i 6. Tražimo tri niske koje počinju karakterom ‘n’ (obeleženi su sa n1, n2 i n3). Gledamo šta se nalazi iza karaktera ‘n’, to su sve karakteri ‘a’ (oni obeleženi brojevima 2, 5 i 6), što je očekivano jer su to oni karakteri ‘a’ koje smo malo pre pronašli u prvoj koloni. Zatim, posmatramo poslednju kolonu i odbacujemo niske koje nam ne odgovaraju. Kako se sve tri niske završavaju karakterom ‘a’, nećemo nijednu isključiti. Ovi karakteri ‘a’ obeleženi su brojevima 3, 4, 5. Prelazimo u redove gde se nalaze a3, a4 i a5 i pronalazimo da sve tri sadrže nisku „ana”.

Kao što vidimo, možemo da koristimo BWT reprezentaciju za uparivanje šablonu. Međutim, nismo dobili odgovor na pitanje na kojim pozicijama se nalaze ova pojavljivanja. Možemo dobiti odgovor da li se nalazi i koliko puta se pojavljuje.

```
$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3namabananas$p1
a4nanas$panamab1
a5nas$panamaban2
a6s$panamabanan3
b1ananas$panama1
m1abananas$pana2
n1a3mabananas$p3
n2a4nas$panamab4
n3a5s$panamaban5
p1anamabananas$1
s1$panamabanana6
```

Slika 8.11: First-Last svojstvo

```
$1panamabananas1
a1bananas$panam1
a2mabananas$pan1
a3na3mabananas$p1
a4na4nas$panamab1
a5nas$panamaba2
a6s$panamabanan3
b1ananas$panama1
m1abananas$pana2
n1amabananas$p3
n2anas$panamab4
n3as$panamaban5
p1anamabananas$1
s1$panamabanana6
```

Slika 8.12: First-Last svojstvo

8.3.3 Pronalaženje uparenih šablonata

Problem višestrukog uparivanja šablonata:

Ulaz: Kolekcija niski Patterns i niska Genome.

Izlaz: Sve pozicije u niski Genome gde se niske iz kolekcije Patterns pojavljuju kao podnische.

Treba da nađemo pozicije. BWT ne daje ovaj podatak. Na primer, na gornjem primeru „ana“ se pojavljuje 3 puta, ali na kojim pozicijama? To ćemo odrediti pomoću sufiksnog niza.

Sufiksni niz je niz koji čuva početnu poziciju za svaki sufiks (niz karaktera u svakom redu matrice do simbola \$) i prikazan je na slici 8.13.

13	\$ ₁ panamabanana s ₁
5	a ₁ bananas\$panam ₁
3	a ₂ mabananas\$pan ₁
1	a ₃ namabananas\$p ₁
7	a ₄ nanas\$panamab ₁
9	a ₅ nas\$panamaban ₂
11	a ₆ s\$panamaban ₃
6	b ₁ ananas\$panama ₁
4	m ₁ abananas\$pana ₂
2	n ₁ amabananas\$p ₃
8	n ₂ anas\$panamaba ₄
10	n ₃ as\$panamabana ₅
0	p ₁ anamabananas\$ ₁
12	s ₁ \$panamabanana ₆

Slika 8.13: Sufiksni niz

Sa slike vidimo da se „ana“ iz prethodnog primera pojavljuje na pozicijama 1, 7 i 9.

Da li smo pokvarili složenost? Prostorna složenost je $4 * |Genome|$ (ako koristimo 4B za cele brojeve kao elemente niza), što je bolje nego $20 * |Genome|$. Sufiksni niz jeste poboljšanje, ali nije najbolje poboljšanje. Postoji prisup sa $|Genome|$ prostornom složenosti, a koristi parcijalni sufiksni niz. Ovim se nećemo dalje baviti.

8.4 Približno preklapanje

Do sad smo definisali problem egzaktnog poklapanja i način na koji ga možemo rešiti. Ponekad je neophodno pronaći približna uparivanja šablonata. Približno znači da se neće sve pozicije poklapati. Baš te pozicije predstavljaju mutacije do kojih je došlo zbog čega nam je približno poklapanje značajno.

Približno uparivanje šablonata:

Ulaz: Niska Pattern, niska Genome, ceo broj d (kod višestrukog uparivanja ulaz je kolekcija niski Patterns).

Izlaz: Sve pozicije niske Genome, gde se niska Pattern pojavljuje kao podniska sa najviše d razlika.

Analogno egzaktnom uparivanju, postoji i problem višestrukog približnog uparivanja šablonu koji opisujemo u nastavku.

Višestruko približno uparivanje šablonu:

Ulaz: Kolekcija niski Pattern, niska Genome, ceo broj d (kod višestrukog uparivanja ulaz je kolekcija niski Patterns).

Izlaz: Sve pozicije niske Genome, gde se niska Pattern pojavljuje kao podniska sa najviše d razlika.

Hajde da vidimo na primeru kako možemo primeniti BWT na približno preklapanje. Pretpostavimo da je $d = 1$. Traženje preklapanja radimo kao i pre, samo što sada prihvatamo i kad imamo različite karaktere. Zbog toga, sada zadržavamo i crvena slova, kao na slici 8.14, sve dok je broj razlika $\leq d$. Čuvaćemo informaciju o broju nepoklapanja i svuda gde smo odabrali crveno slovo broj nepoklapanja se uvećava za jedan.

Sada tražimo niske koje počinju karakterima koje smo odabrali u prethodnom koraku ($m_1, n_1, p_1, b_1, n_2, n_3$), odnosno tražimo ih u prvoj koloni. Naravno, te vrste se završavaju sa a. Treba da utvrdimo šta se nalazi ispred, odnosno na kraju, u poslednjoj koloni. Sve niske, osim one koja počinje sa p_1 , završavaju se sa 'a', što nam ne povećava broj nepoklapanja. Niska koja počinje sa p_1 završava sa '\$', što uvećava broj nepoklapanja na 2. Tu nisku ćemo odbaciti jer je premašen broj nepoklapanja.

Tako smo pronašli pet 3-grama koji imaju najviše 1 nepoklapanje, a na osnovu sufiksnog niza možemo videti gde su početne pozicije tih 3-grama.

	# Mismatches	Suffix Array
\$ ₁ panamabananas ₁		\$ ₁ panamabananas ₁
a ₁ bananas\$panam ₁	1	a ₁ b ₁ ananas\$panam ₁
a ₂ mabananas\$pan ₁	0	a ₂ m ₁ abananas\$pan ₁
a ₃ namabananas\$p ₁	1	a ₃ n ₁ amabananas\$p ₁
a ₄ anas\$panamab ₁	1	a ₄ n ₁ as\$panamab ₁
a ₅ nas\$panamaba ₂	0	a ₅ n ₁ as\$panamaba ₂
a ₆ s\$panamaban ₃	0	a ₆ s\$panamaban ₃
b ₁ ananas\$panama ₁		b ₁ ananas\$panama ₁
m ₁ abananas\$pana ₂		m ₁ abananas\$pana ₂
n ₁ amabananas\$pa ₃		n ₁ amabananas\$pa ₃
n ₂ anas\$panamaba ₄		n ₂ anas\$panamaba ₄
n ₃ as\$panamabana ₅		n ₃ as\$panamabana ₅
p ₁ anamabananas\$ ₁		p ₁ anamabananas\$ ₁
s ₁ \$panamabana ₆		s ₁ \$panamabana ₆

Slika 8.14: Traženje približnog preklapanja za $d = 1$

Slika 8.15: Pozicije u genomu gde se javljaju približna preklapanja

Glava 9

Zašto naučnici i dalje nisu razvili vakcinu za HIV

9.1 Uvod

9.1.1 Klasifikacija HIV fenotipa

1984. godine, američka ministarka zdravljva Margaret Hekler je objavila da će vakcina za HIV biti dostupna u narednih 2 godine. 1997. godine, Bil Clinton je otvorio novi centar za istraživanje na Nacionalnom institutu zdravlja, sa ciljem da se razvije vakcina za HIV. Kompanija Merck je 2005. počela kliničko ispitivanje vakcine za HIV, ali je odustala posle 2 godine, jer su rezultati pokazali da je vakcina zapravo povećala rizik od dobijanja HIV-a kod nekih primalaca vakcine.

Danas, uprkos ogromnim investicijama i istraživanjima, daleko smo od razvijanja vakcine za HIV, a 35 miliona ljudi žive s tom bolešću. Naučnici su napravili ogroman napredak u razvoju antiretroviralne terapije, koja predstavlja mešavinu lekova koji stabilizuju simptome zaraženog pacijenta. Međutim, ova terapija ne leči sidu i ne može da zaustavi širenje HIV-a tako da ne predstavlja pravu vakciju za sidu.

Klasične vakcine protiv virusa su često napravljene od proteina virusa. Ove vakcine stimulišu čovekov imuni sistem da prepozna virusne omotače proteina kao strane, da ih uništi i da sačuva podatke o njima, da bi imuni sistem mogao kasnije da ih identifikuje i iskoreniti.

Međutim, virusni omotači proteina HIV virusa mogu biti ekstremno promenljivi, zato što virus mora da mutira brzo da bi preživeo. Virus HIV-a kod neke osobe evoluira vrlo brzo da bi izbegao imuni sistem čoveka. Takođe, uzorci HIV-a uzeti od različitih pacijenata su pokazali da oni imaju podtipove koji se veoma razlikuju. Dakle, uspešna vakcina za HIV mora biti dovoljno širokog spektra da pokrije sve ove različitosti.

HIV ima samo devet gena i u ovom poglavlju se fokusiramo na *env* gen koji brzo mutira. Protein koji kodira *env* gen ulazi u **glycoprotein gp120** i **glycoprotein gp41**.

S obzirom da HIV mutira tako brzo, različiti izolati HIV-a mogu imati različite fenotipe, koji onda zahtevaju različite mešavine lekova. HIV virusi se mogu podeliti na brzo replicirajuće (SI) izolate i sporo replicirajuće (NSI) izolate. Tokom infekcije, proteini virusa kao što je gp120 koje HIV koristi da uđe u ćelije se prenose do površine ćelije, gde mogu da prouzrokuju da se ta ćelija spoji sa susednom ćelijom. To uzrokuje da desetine ljudskih ćelija spoje svoje ćelijske membrane u jedan veliki, nefunkcionalni syncytium ili u abnormalnu multinukleaturnu ćeliju. Na ovaj način, inficirajući samo jednu ćeliju, biće ubijene mnoge ljudske ćelije.

Ograničenja u poravnanju sekvenci

Pre nego što biolozi uopšte mogu da počnu da proučavaju pitanje predviđanja HIV fenotipa koristeći gp120 sekvene, oni se suočavaju sa problemom konstrukcije preciznog poravnjanja ovih

sekvenci. Čak i jedno pogrešno poravnanje, koje postavlja amino kiselinu na poziciju utičući na SI/NSI fenotip, može prouzrokovati pogrešnu klasifikaciju HIV fenotipa. Iz poglavlja 5, već znamo da je konstrukcija višestrukog poravnanja sekvenci koje divergiraju težak algoritamski problem.

Problem formulacije višestrukog poravnanja uveden u poglavlju 5 ne pruža adekvatnu translaciju biološkog problema PHV klasifikacije u algoritamski problem. Zbog toga moramo smisliti novu formulaciju problema poravnanja sekvenci koja će dovesti statistički solidne analize gp120 proteina.

9.2 Nepoštena kockarnica

9.2.1 Kockanje sa Jakuzama

Japanska kriminalna organizacija pod nazivom jakuza potiče od grupe putujućih kockara iz 18. veka koji su se nazivali bakut („jakuza” je naziv za gubitničku ruku u Japanskoj kartaškoj igri). Jedna od najpopularnijih igara koju su bakuto organizovali u svojim kazinima se zvala Čo-Han. U ovoj igri, koja se bukvalno prevodi kao „jednake šanse”, krupije baca dve kockice, a igrač se kladi na to da li će suma kockica biti paran ili neparan broj.

Iako je igranje Čo-Han igre u jakuzinim kockarnicama veoma zanimljivo, možemo takođeigrati i igru koja se zove „glava ili pismo”, tako što se baca novčić u vazduh i pogoda se ishod. Prepostavimo da se iz nekog razloga više ljudi kladi na pismo nego na glavu u ovoj igri. U tom slučaju bi nepošten krupije mogao da iskoristi otežani novčić koji ima veću verovatnoću da padne na glavu nego na pismo. Mi ćemo prepostaviti da otežani novčić ima verovatnoću da padne na glavu 3/4.

Pitanje: Recimo da igramo igru pismo ili glava 100 puta, i novčić padne na glavu 63 puta. Da li možemo da kažemo da krupije vara? Da li je korišćen fer ili otežan novčić? **Nagoveštaj:** 63 je bliže 75 nego 50!

Pitanje nije dobro formulisano, jer bilo koji novčić može da proizvede bilo koji niz bacanja. Da li možemo da utvrdimo koji novčić je verovatnije korišćen? Zapišimo verovatnoću padanja pisma („T”) i glave („H”) za fer novčić (F) kao:

$$Pr_F(H'') = 1/2 \quad Pr_F(T'') = 1/2 \quad (9.1)$$

i verovatnoća otežanog novčića (B):

$$Pr_B(H'') = 3/4 \quad Pr_B(T'') = 1/4 \quad (9.2)$$

Kako su bacanja novčića nezavisni događaji, verovatnoća da će n bacanja fer novčića proizvesti dati niz $x = x_1x_2\dots x_n$ sa k pojavljivanja glave je:

$$Pr(x|F) = \prod_{i=1}^n Pr_F(x_i) = (1/2)^n. \quad (9.3)$$

Verovatnoća da će otežani novčić da proizvede isti niz je:

$$Pr(x|B) = \prod_{i=1}^n Pr_B(x_i) = (1/4)^{(n-k)} \cdot (3/4)^k = 3^k / 4^n. \quad (9.4)$$

Ako je $Pr(x|F) > Pr(x|B)$, onda je veća verovatnoća da je krupije koristio fer novčić, a ako je $Pr(x|F) < Pr(x|B)$, onda je obrnuto. Kada su jednaki $Pr(x|F) = Pr(x|B)$ tada je:

$$(1/2)^n = (1/4)^{(n-k)} \cdot (3/4)^k \rightarrow 2^n = 3^k \rightarrow k = \log_2 3 \cdot n \rightarrow k \approx 0.632 \cdot n \quad (9.5)$$

Dakle ukoliko je mera odnosa $k/n < 1/\log_2 3$, tada je i $Pr(x|F) > Pr(x|B)$. Iako je 63 bliže 75 nego 50, fer novčić će sa većom verovatnoćom dati 63 glave u 100 bacanja.

9.2.2 Dva novčića u krupijeovom rukavu

U bakuto kockarnicama, Čo-Han krupije bi skinuo svoju majicu u toku igre, kako bi skinuo sa sebe sumnju da mulja sa kockicama. Mi ćemo ipak pretpostaviti da u igri glava ili pismo, nepošteni krupije nosi majicu i drži oba novčića u svom rukavu i može da ih neprimetno menja u bilo kom trenutku. Pošto ne želi da bude uhvaćen kako zamenjuje novčice, on ih menja samo povremeno, pretpostavimo sa verovatnoćom 0.1 nakon svakog bacanja.

Pitanje: Nakon niza bacanja novčića, možemo li reći kada je krupije koristio fer coin a kada otežani novčić?

Kazino problem: Za dati niz bacanja novčića, odrediti kada je krupije koristio fer a kada otežani novčić.

Ulaz: Niz $x = x_1 x_2 \dots x_n$ bacanja dobijenih od novčića F (fer) i B (otežani).

Izlaz: Niz $\pi = \pi_1 \pi_2 \dots \pi_n$, gde je svako π_i jednako ili F ili B što znači da je x_i dobijeno bacanjem fer ili otežanog novčića, redom.

Nažalost, ni ovo nije dobro definisan problem. Svaki ishod bacanja novčića može biti dobijen bilo kojom permutacijom fer i otežanog novčića! HHHHH je moglo biti dobijeno od BBBBB, FFFFF, FBFBF, itd.

Neophodan je način za ocenjivanje različitih scenarija: BBBBB, FFFFF, FBFBF, itd. zavisno od toga koliko je svaki od njih verovatan.

Kako da ispitamo i ocenimo 2^n mogućih scenarija? Jedan pristup ovom problemu da pogodimo koji novčić je verovatnije krupije koristio u svakom bacanju je da uzmem „okvir prozora“ (dužine $t < n$) duž niza bacanja $x = x_1 \dots x_n$ i da izračunamo meru odnosa fer i otežanog novčića (kao u gornjem primeru) u okviru svakog prozora. Ukoliko je mera odnosa u okviru prozora manja od nule, onda je veća verovatnoća da krupije koristi otežani novčić u okviru prozora, inače je obrnuto. Posmatramo odnos **log-odds** sekvence x , odnosno logaritam tog odnosa (kako ne bi došlo do potkoračenja, obzirom da su verovatnoće obično mali brojevi).

$$\log_2 Pr(x|F)/Pr(x|B) = \log_2 [(1/2)^n / (3/4)^k \cdot (1/4)^{n-k}] = \#Tosses - \log_2 3 \#Heads$$

U zavisnosti od **logg-odds** odnosa možemo da zaključimo koji je novčić verovatnije korišćen. Pa, ako je vrednost manja od nule, rekli bismo da je verovatnije korišćen otežani novčić, a ako je vrednost pozitivna onda je verovatnije korišćen fer novčić.

Postoji dva problema sa metodom „okvira prozora“. Prvo je da nemamo vidan način da odaberemo dužinu okvira prozora. Drugo, okviri prozora koji se preklapaju mogu klasifikovati isti ishod uzrokovani i otežanim i fer novčićem. Na primer, ako je $x = „HHHHHTTHHHTTTT“$, onda prozor $x_1 \dots x_{10} = „HHHHHTTHH“$ ima negativnu meru odnosa, a onda prozor $x = x_6 \dots x_{15} = „TTHHHTTTTT“$ ima pozitivnu meru odnosa. Koji je novčić krupije koristio na bacanjima $x_6 \dots x_{10}$?

9.3 Pronalaženje CG ostrva

Početkom dvadesetog veka, Phoebus Levene je otkrio četiri nukleotida od kojih se sastoji DNK. U to vreme, vrlo malo se znalo o DNK. Zbog toga, Levene je sumnjao da DNK može da čuva genetske informacije koristeći samo četiri slova i postavio je hipotezu da se u DNK nalazi gotovo jednak broj adenina, citozina, guanina i timina. Jedan vek kasnije, znamo da komplementarni nukleotidi na suprotnim obalama DNK imaju jednaku frekvenciju osnovnog uparivanja, ignorujući ekstremno retke greške osnovnog uparivanja. Međutim, nije tačno da su frekvencije nukleotida približno iste na jednoj obali DNK. Različite vrste imaju različite **CG-sadržaje**, ili procenat citozina i guanina u genomu.

Mogli bismo očekivati da se svaki od dinukleotida CC, CG, GC, GG u ljudskom genomu javlja sa frekvencijom od $0.21 * 0.21 + 4.41\%$. Međutim, frekvencija CG u ljudskom genomu je samo 1%. Ovaj dinukleotid je toliko redak zbog **metilacije**.

Definicija 9.1. *Metilacija je dodavanje metil (CH_3) grupe na citozin (često u okviru CG dinukleotida).*

Rezultujući metilovani citozin ima tendenciju da deaminuje u timin. Kao rezultat metilacije, CG je najredi dinukleotid u mnogim genomima. Metilacija je često izostavljena u genima u regionima pod nazivom **CG-ostrva** (CG se često pojavljuje).

U prvom pokušaju za nalaženje ovakvih gena, kako bismo tražili CG-ostrva? Naivan pristup traženju CG-ostrva bi bio da se pomera prozor kroz genom, proglašavajući prozore sa većom frekvencijom CG, potencijalnim CG-ostrvima. Mane ovog pristupa bi bile iste kao i pri pomeranju prozora da bi se odredio koji novčić je krupije koristio u kom trenutku. Ne znamo koliki će prozor biti i zato nije jasno kako odabratи veličinu prozora za detekciju CG-ostrva. Takođe, različiti prozori mogu klasifikovati iste pozicije u genomu različito.

9.4 Skriveni Markovljevi modeli

9.4.1 Od bacanja novčića do Skrivenog Markovljevog Modela

Naš cilj je da razvijemo koncept koji modeluje nepoštenog krupijea i potragu za CG-ostrvima u genomu. Krupijea možemo posmatrati kao mašinu koja ima k skrivenih stanja (F i B). U svakom koraku, emituje simbol (H ili T) iz jednog od svojih skrivenih stanja. Dok je u određenom stanju, mašina donosi dve odluke:

- Koji simbol će emitovati?
- U koje skriveno stanje će nakon toga preći?

Mašina odgovara na prvo pitanje tako što izabere proizvoljan broj između stanja F i B , sa verovatnoćom 0.9 da će ostati u trenutnom stanju i verovatnoćom 0.1 da će promeniti stanje. Mašina odgovara na drugo pitanje tako što bira između simbola H i T sa verovatnoćama koje zavise od toga u kom stanju je trenutno. Naš cilj je da zaključimo koja je najverovatnija sekvenca stanja maštine analizirajući sekvence simbola koje emituje.

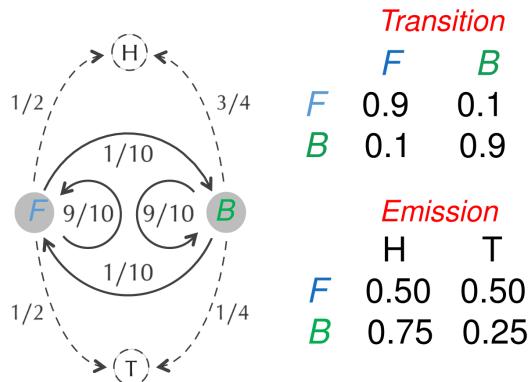
Na ovaj način smo pretvorili krupijea u apstraktnu mašinu koja se zove **Skriveni Markovljev Model** (HMM). Razlika između maštine za bacanje novčića i generalnog koncepta HMM-a je što će HMM kasnije imati proizvoljan broj stanja i može da ima proizvoljnu distribuciju verovatnoća, odlučujući u koje stanje da pređe i koje simbole da emituje.

9.4.2 HMM dijagrami

HMM se definiše kao skup četiri objekta:

- azbuka emitovanih simbola Σ (H i T)
- skup skrivenih stanja (F i B)
- Matrica verovatnoće prelaska: $Transition = (transition_{l,k})$: $|States| \times |States|$ matrica **verovatnoća prelaska** (iz stanja l u stanje k)
- Matrica emisionih verovatnoća: $Emision = (emision_k(b))$: $|States| \times |\Sigma|$ matrica **emisionih verovatnoća** (emitovanje simbola b u stanju k)

Kao što je prikazano na slici 9.1, HMM se može prikazati kao HMM dijagram, graf gde je svako stanje predstavljeno jednim punim čvorom. Usmerene pune grane povezuju svaki par čvorova, kao i svaki čvor sa sobom. Svaka takva grana je obeležena verovatnoćom prelaska iz jednog stanja u drugo. HMM dijagram takođe ima i isprekidane grane koje povezuju svako stanje sa svojim isprekidanim čvorom. Svaka takva grana je obeležena verovatnoćom da će HMM emitovati taj simbol dok je u tom stanju.



Slika 9.1: HMM dijagram

Definicija 9.2. *Skrivena putanja* je niz $\pi = \pi_1, \dots, \pi_n$ stanja kroz koje HMM prolazi.

Slika 9.1 prikazuje primer gde nepošteni krupije HMM proizvodi sekvencu $x = „THTHHHTHTTH”$ sa skrivenom putanjom $n = FFFBBBFF$. Fer novčić je korišćen za prva tri bacanja i za poslednja tri bacanja, a otežani novčić se koristi za pet bacanja između.

Uvedimo sledeće jednakosti:

- $Pr(x, \pi)$: zajednička verovatnoća da dati HMM polazi kroz stanja π i emituje nisku $x = x_1x_2\dots x_n$.
- $Pr(x|\pi)$: uslovna verovatnoća da HMM emituje nisku x nakon prolaska kroz skrivenu putanju π .
- $Pr(\pi_{i-1} \rightarrow \pi_i)$: verovatnoća da je HMM prešao iz stanja π_{i-1} u stanje π_i ($transition_{\pi_{i-1}, \pi_i}$).
- $Pr(x, \pi) = Pr(x|\pi) * Pr(\pi)$

Da bi se izračunalo $Pr(x, \pi)$, prvo moramo da izračunamo $Pr(\pi)$. Neka $Pr(\pi_i \rightarrow \pi_{i+1})$ označava verovatnoću prelaska HMM-a iz stanja π_i u stanje π_{i+1} . Verovatnoća za π je jednaka proizvodu verovatnoća prelaska

$$Pr(\pi) = \prod_{i=1}^n Pr(\pi_{i-1} \rightarrow \pi_i) = \prod_{i=1}^n transition_{\pi_{i-1}, \pi_i} \quad (9.6)$$

Problem verovatnoće skrivene putanje: Izračunati verovatnoću skrivene putanje HMM-a.

Ulaz: Skrivena putanja π i model HMM ($\Sigma, States, Transition, Emission$).

Izlaz: Verovatnoća date putanje, $Pr(\pi)$.

Da bismo izračunali $Pr(x|\pi)$ za neki HMM, označićemo sa $Pr(x_i|\pi_i)$ verovatnoću emitovanja $emission_{\pi_i}(x_i)$ da je x_i emitovan kada je HMM bio u stanju π_i . Kao rezultat toga, za neku putanju π , HMM emituje string x sa verovatnoćom jednakom proizvodu verovatnoća emitovanja na toj putanji,

$$Pr(x, \pi) = \prod_{i=1}^n Pr(x_i|\pi_i) = \prod_{i=1}^n emission_{\pi_i}(x_i) \quad (9.7)$$

Problem verovatnoće ishoda za datu skrivenu putanju: Izračunati verovatnoću da dati HMM emitiše datu nisku za datu skrivenu putanju.

Ulaz: Niska $x = x_1, \dots, x_n$ koju emitiše dati HMM($\Sigma, States, Transition, Emission$) i skrivena putanja $\pi = \pi_1, \dots, \pi_n$.

Izlaz: Uslovna verovatnoća $Pr(x|\pi)$ da će dati HMM emitovati nisku x prateći skrivenu putanju π .

9.5 Problem dekodiranja

Koja bi bila optimalna skrivena putanja? Kako ćemo je odrediti?

Problem dekodiranja: Naći optimalnu skrivenu putanju sa kojom je dati HMM emitovao datu nisku.

Ulaz: Niska $x = x_1 \dots x_n$ koju emitiše HMM($\Sigma, States, Transition, Emission$).

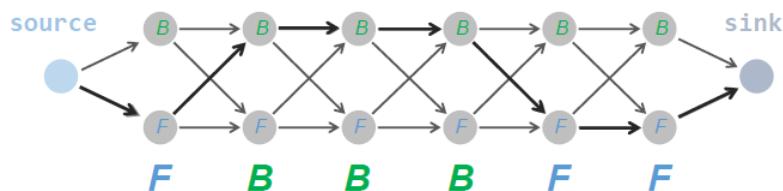
Izlaz: Putanja π koja maksimizuje verovatnoću $Pr(x, \pi)$ po svim mogućim putanjama π za ovaj HMM.

Verovatnoću $Pr(x, \pi)$ znamo da izračunamo:

$$Pr(x, \pi) = \prod_{i=1, n} Pr(x|\pi_i) * Pr(\pi_{i-1} \rightarrow \pi_i) = \prod_{i=1, n} emission_{\pi_i(x_i)} * transition_{\pi_{i-1}, \pi_i}$$

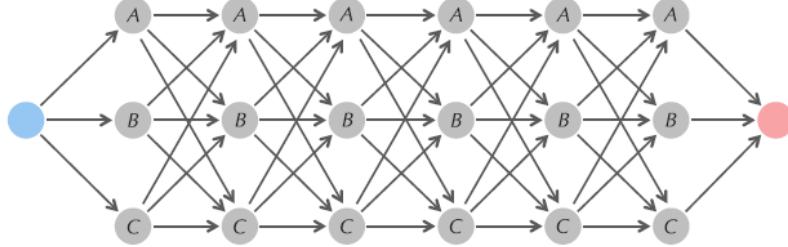
9.5.1 Viterbi graf

Da bi rešio problem dekodiranja, Andrew Viterbi je koristio Menhetn graf inspirisan HMM-om. Za HMM koji emitiše string od n simbola $x = x_1 \dots x_n$, čvorovi HMM-ovog Viterbi grafa se dele na $|States|$ vrsta i n kolona (slika 9.3). Dakle, čvor (k, i) reprezentuje stanje k i i -ti emitovani simbol. Svaki čvor je povezan sa svim čvorovima iz kolone s njegove desne strane; grana koja povezuje $(l, i-1)$ sa (k, i) odgovara prelasku iz stanja l u stanje k (sa verovatnoćom $transition_{l,k}$) i zatim emitovanju simbola x (sa verovatnoćom $emission_k(x_i)$). Dodatno, imao čvorove *source* i *sink*. Kao rezultat toga, sve putanje koja povezuje čvor u prvoj koloni Viterbi grafa sa čvorom u poslednjoj koloni, odgovara skrivenoj putanji $\pi = \pi_1 \dots \pi_n$.



Slika 9.2: Menhetn graf za problem nepoštenog kazina i rekonstrukcija putanje za datu ishod.

Kod problema dekodiranja imamo više od 2 stanja. Na slici 9.3 možemo videti kako to izgleda za tri stanja. Broj redova jednak je 3, jer imamo tri stanja, a broj kolona jednako je broju emitovanih simbola.



Slika 9.3: Menhetn graf za problem dekodiranja

Kako da odredimo težine grana u ovom granu? Grana $(l, k, i - 1)$ izvora $(l, i - 1)$ u čvor (k, i) nam govori da se prelazi iz stanja l u stanje k , sa verovatnoćom $transition_{l,k}$, i da se trenutno emituje i -ti simbol (sa verovatnoćom $emission_k(i)$). Težina grane koja povezuje $(l, i - 1)$ i (k, i) u Viterbi grafu je jednaka

$$Weight_i(l, k) = transition_{\pi_i, \pi_{i-1}} * emission_{\pi_i}(x_i) \quad (9.8)$$

Zatim, definišemo **proizvod težina** putanja u Viterbi grafu, kao proizvod težina njegovih grana. Za putanju od najlevlje kolone do najdesnije kolone u Viterbi grafu koja odgovara skribovenoj putanji n , ovaj proizvod je jednak proizvodu $n - 1$ članova,

$$\begin{aligned} Pr(x, \pi) &= \prod_{i=1,n} emission_{\pi_i(x_i)} * transition_{\pi_{i-1}, \pi_i} \\ &= \prod_{i=1,n} \text{weight of the } i\text{-th edge path } \pi \\ &= \prod_{i=1,n} weight(\pi_{i-1}, \pi_i, i - 1) \end{aligned}$$

Znamo da odredimo Menhetn graf, znamo da odredimo težine grana, ali sve to znamo kada nam je data sekvenca π . U nastavku ćemo videti kako do nje možemo doći.

9.5.2 Viterbi algoritam

Primenićemo algoritam dinamičkog programiranja da bismo rešili problem dekodiranja. Prvo, definisimo $score_{k,i}$, što predstavlja proizvod težina optimalne putanje (putanje sa najvećom težinom proizvoda) od početnog čvora $source$ do čvora (k, i) . Viterbi algoritam je zasnovan na činjenici da prvih $i - 1$ grana optimalne putanje od izvora do (k, i) moraju formirati optimalnu putanju od izvora do $(l, i - 1)$ za neko (nepoznato) stanje l . Ovo zapažanje proizvodi sledeću jednačinu:

$$\begin{aligned} score_{k,i} &= \max_{all states l} \{ score_{l,i-1} \cdot (weight of edge between nodes (l, i - 1) and (k, i)) \} \\ &= \max_{all states l} \{ score_{l,i-1} \cdot Weight_i(l, k) \} \\ &= \max_{all states l} \{ score_{l,i-1} \cdot transition_{\pi_{i-1}, \pi_i} \cdot emission_{\pi_i}(x_i) \} \end{aligned} \quad (9.9)$$

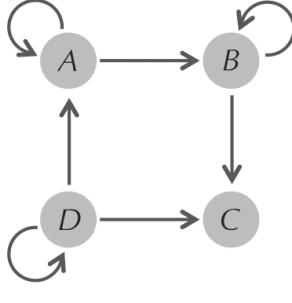
Skor početnog čvora biće 1, kako ne bi uticao na proizvod. Skor za krajnji čvor je maksimum po svim putanjama od početnog do krajnjeg čvora.

$$score_{sink} = \max_{\text{all states } l} score_{l,n}$$

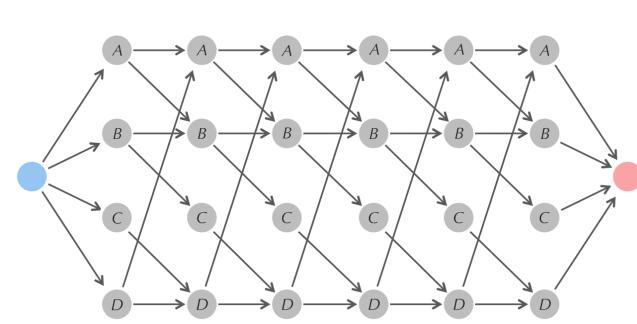
9.5.3 Brzina Viterbi algoritma

Možemo da posmatramo problem dekodiranja kao još jednu instancu problema najduže putanje u DAG problemu iz 5. poglavlja, zato što putanja π koja maksimizira proizvod težina $\prod_{i=1}^n Weight_i(\pi_{i-1}, \pi_i)$ takođe maksimizira logaritam ovog proizvoda, koji je jednak $\sum_{i=1}^n \log(Weight_i(\pi_{i-1}))$. Prema tome, možemo da zamenimo težine svih grana u Viterbi grafu njihovim logaritmima. Na-laženje najdužeg puta u rezultujućem grafu će odgovarati putanji maksimalnih težina proizvoda u originalnom Viterbi grafu. Iz ovog razloga, vreme izvršavanja Viterbi algoritma je linearno u odnosu na broj grana u Viterbi grafu. Broj ovih grana je $|States|^2 \cdot n$ gde je n broj emitovanih simbola.

U praksi, mnogo HMM-ova ima **zabranjene prelaze** između nekih stanja. Za takve prelaze, možemo da obrišemo odgovarajuće grane iz HMM dijagrama (slika 9.4). Ova operacija dovodi do ređeg Viterbi grafa (slika 9.5), što dovodi do smanjenja vremena izvršavanja Viterbi algoritma, s obzirom da je vreme izvršavanja algoritma za pronalaženje najduže putanje u DAG-u linearno u odnosu na broj grana u tom DAG-u.



Slika 9.4: HMM dijagram sa nekim zabranjenim stanjima kao npr. od A do D ili od C do samog sebe



Slika 9.5: Viterbi graf za HMM sa prethodne slike koji emituje string dužine 6

Kako nam je $score$ proizvod verovatnoća njegova vrednost može da postane veoma mala i postoji opasnost da dođe do potkoračenja. Iz tog razloga ćemo računati njegov logaritam, kao što smo to radili kod veličine **log-odds**. Nova formula za računanje skora je

$$\log(score_{k,i}) = \max_{\text{all states } l} \{\log(score_{l,i-1}) + \log(weight(l, k, i - 1))\}$$

9.6 Računanje najverovatnijeg ishoda HMM-a

Vreovatnoća da HMM prati skrivenu putanju π i emituje nisku x obležena je sa $Pr(x, \pi)$. Dinamičko programiranje nam pomaže da odgovorimo na pitanje proširivanja HMM-a i preko najverovatnije skrivene putanje. Mi možemo da izračunamo verovatnoću $Pr(\pi)$ skrivene putanje π . Ali šta je sa $Pr(x)$, koja je verovatnoća da HMM emituje string x ?

Problem verovatnoće ishoda: Izračunati verovatnoću da HMM emituje datu nisku.

Ulaz: Niska $x = x_1 \dots x_n$ koju emituje HMM($\Sigma, States, Transition, Emission$).

Izlaz: Verovatnoća $Pr(x)$ da model HMM emituje nisku x .

Već smo zaključili da je $Pr(x)$ jednak sumi $Pr(x, \pi)$ za sve skrivene putanje π . Međutim, broj putanja u Viterbi grafu je eksponencijalan u odnosu na broj emitovanih stringova x , tako da možemo da koristimo dinamičko programiranje kao brži način da izračunamo $Pr(x)$.

Neka je $forward_{k,i}$ proizvod svih putanja od *izvora* do čvora (k, i) u Viterbi grafu; treba uočiti da je $forward_{sink}$ jednak $Pr(x)$. Da bismo izračunali $forward_{k,i}$, podelićemo sve putanje koje povezuju *izvor* i čvor (k, i) na $|States|$ podskupova, gde svaki podskup sadrži one putanje koje prolaze kroz čvor $(l, i-1)$ (sa težinom proizvoda $forward_{l,i-1}$), dok ne dođemo do (k, i) za neko l između 1 i $|States|$. Dakle, $forward_{k,i}$ je suma $|States|$ članova,

$$\begin{aligned} forward_{k,i} &= \sum_{\text{all states } l} forward_{l,i-1} \cdot \text{težina grane koja povezuje } (l, i-1) \text{ i } (k, i) \\ &= \sum_{\text{all states } l} forward_{l,i-1} \cdot Weight_i(l, k) \end{aligned} \quad (9.10)$$

Treba primetiti da je jedina razlika između ove jednačine i Viterbi jednačine,

$$score_{k,i} = \max_{\text{all states } l} \{s_{l,i-1} \cdot Weight_i(l, k)\}, \quad (9.11)$$

u tome što se maksimizacija u Viterbi algoritmu menja u sumaciju. Sada možemo rešiti problem verovatnoće ishoda računajući $forward_{sink}$, što je jednako

$$\sum_{\text{all states } k} forward_{k,n} \quad (9.12)$$

Sada možemo da izračunamo $Pr(x)$ za emitovani string x , logično pitanje je pronaći najverovatniji takava string. Za problem nepoštenog krupije, ovo odgovara pronalasku najverovatnije sekvene bacanja novčića za sve moguće sekvene, za fer i otežani novčić koji krupije može koristiti.

Problem najverovatnijeg ishoda: Naći najverovatniji string koji emituje HMM.

Ulaz: HMM($\Sigma, States, Transition, Emission$) i ceo broj n .

Izlaz: Najverovatniji string $x = x_1 \dots x_n$ koji emituje HMM, odnosno, string koji maksimizira verovatnoću $Pr(x)$ da će HMM emitovati x .

9.7 Profilni algoritmi za poravnjanje sekvenci

9.7.1 Kako su HMM povezani za poravnjanje sekvenci?

Za datu familiju povezanih proteina, možemo proveriti da li nova sekvenca proteina pripada ovoj familiji, konstruišući parno poravnanje između novo sekvenciranog proteina i svakog člana familije. Ako jedno od rezultujućih poravnanja da rezultat iznad nekog strogog praga, onda možemo pretpostaviti da novi protein pripada familiji. Međutim, ovaj pristup može neuspešno da identificuje proteine koji su udaljeno povezani. Ako sekvenca ima slabe povezanosti sa velikim brojem članova familije, onda ona najverovatnije pripada toj familiji.

Problem je poravnati novi protein sa *svim* članovima familije odjednom. Da bismo ovo postigli, moramo da pretpostavimo da već imamo konstruisano višestruko poravnanje familije proteina. Srećom, često će biti očigledno da dva proteina dolaze iz iste familije. Shodno tome,

biolozi često počinju konstruišući poravnanje proteina koji su nesumnjivo povezani, koje je obično lako poravnati, čak i koristeći jednostavne metode poravnanja koje smo predstavili u poglavlju 5.

Slika 9.6 (prvi deo) prikazuje 5×10 poravnanje *Alignment* koje predstavlja hipotetičku familiju proteina. Primetimo da 6. i 7. kolona ovog poravnanja sadrže mnogo praznih “-“ simbola, i verovatno ne predstavljaju značajne karakteristike familije. Shodno tome, biolozi često ignoriraju kolone za koje je deo ovih praznih “-“ simbola veći ili jednak **pragu brisanja kolone** θ . Brisanje kolona rezultuje semenom poravnanju *Alignment** 5×8 predstavljenom na slici 9.6 (drugi deo).

Dato semeno poravnanje *Alignment** predstavlja familiju povezanih proteina i naš cilj je da izgradimo HMM koji realistično modelira sklonosti simbola u *Alignment** koji je predstavljen profilnom matricom PROFILE(*Alignment**) na slici 9.6 (treći deo). Umesto da razmišljamo o poravnavanju postojećeg semenog poravnanja do datog *Text-a* (koji predstavlja novi protein), mi ćemo umesto da razmišljamo o tome da izračunamo verovatnoću da HMM emituje *Text*. Ako je HMM dobro dizajniran, onda što je slicciji *Text* nizu u *Alignement**, to će verovatnije biti emitovan od strane HMM-a.

	1	2	3	4	5	6	7	8	
<i>Alignment</i>	A	C	D	E	F	A C	A	D	F
	A	F	D	A	-	— C	C	F	
	A	—	—	E	F	D — F	D	C	
	A	C	A	E	F	— — A	—	C	
	A	D	D	E	F	A A A	D	F	
<i>Alignment*</i>	A	C	D	E	F	A	D	F	
	A	F	D	A	—	C	C	F	
	A	—	—	E	F	F	D	C	
	A	C	A	E	F	A	—	C	
PROFILE(<i>Alignment*</i>)	A	D	D	E	F	A	D	F	
	A	1	0	0	1/5	0	3/5	0	0
	C	0	2/4	0	0	0	1/5	1/4	2/5
	D	0	1/4	3/4	0	0	0	3/4	0
	E	0	0	0	4/5	0	0	0	0
	F	0	1/4	0	0	1	1/5	0	3/5


```

graph LR
    M1((M1)) --> M2((M2))
    M2 --> M3((M3))
    M3 --> M4((M4))
    M4 --> M5((M5))
    M5 --> M6((M6))
    M6 --> M7((M7))
    M7 --> M8((M8))
  
```

Slika 9.6: 5×10 višestruko poravnanje (prvi deo). Uklanjamo kolone ukoliko broj praznina “-“ prelazi θ . 5×8 semenom poravnanje (drugi deo). Izbacili smo kolone. Profilna matrica semenog poravnanje (treći deo) i jednostavni HMM dijagram koji modelira gornji PROFIL. Semeni algoritam je dobijen od originalnog poravnanja ignorisanjem kolona (osenčenih sivom bojom). U ovom slučaju ignoriramo kolone čiji je deo praznih “-“ simbola veći ili jednak pragu $\theta = 0.35$. Kako bismo jasnije prikazali vezu između poravnanja i semenog poravnanja, razdvojili smo prvih 5 kolona u semenom poravnanju od poslednje 3 kolone i numerisali te kolone iznad originalnog poravnanja. Stanja pogotka MATCH(i) su skraćena kao M_i . HMM ima samo jednu moguću putanju; u svom početnom stanju MATCH(1), verovatnoća prelaska iz stanja MATCH(i) do stanja MATCH(i+1) je jednaka 1 za svako i svi drugi prelasci su zabranjeni. Emisione verovatnoće su jednake frekvencijama u profilu, npr. emisione verovatnoće za M_2 su 0 za A, $2/4$ za C, $1/4$ za D, 0 za E i $1/4$ za F.

Prvo ćemo konstruisati jednostavan HMM koji tretira kolone *Alignement*-a* kao k sekvenčijalno povezanih stanja koja ćemo nazvati **stanja pogotka** (slika 9.6 četvrti deo), označeni MATCH(1), ..., MATCH(k). Kada HMM uđe u stanje MATCH(i), tada emituje simbol x_i sa verovatnoćom jednakom frekvenciji ovog simbola u i-toj koloni PROFILE(*Alignment**). HMM se

onda prebacuje u stanje MATCH($i+1$) sa prelaznom verovatnoćom jednakom 1.

Slicnost pogotka između $Alignment^*$ -a i $Text$ -a je verovatnoća $\Pr(Text)$ da HMM za $Alignment^*$ emituje $Text$. Ovaj rezultat je jednak proizvodu frekvencija u $PROFILE(Alignment^*)$ koji odgovaraju svakom simbolu iz $Text$ -a. Na primer, verovatnoća da HMM na slici 9.6 emituje ADDAFFDF je:

$$1 \cdot \frac{1}{4} \cdot \frac{3}{4} \cdot \frac{1}{5} \cdot 1 \cdot \frac{1}{5} \cdot \frac{3}{4} \cdot \frac{3}{5} = 0.003375. \quad (9.13)$$

HMM koji smo prikazali rezultuje svaku kolonu na slici 9.6 drugačije i do određenog stepena, što je sličniji $Text Alignment^*$ -u, to je veći rezultat sličnosti. Međutim, ovaj HMM ima samo jednu skrivenu putanju, i nudi jednostavan pogled višestrukih poravnjanja zato što ne uračunava inserciju i brisanja. Na kraju, $Text$ se može „poravnati” u odnosu na $Alignment^*$ ako je dužina $Text$ -a tačno jednak broju kolona u $Alignment^*$ -u (slika 9.7). Ipak mi ćemo iskoristiti ovaj ograničeni HMM kao temelj za moćnije HMM-ove.

	A	C	D	E	F	A	D	F
	A	F	D	A	–	C	C	F
$Alignment^*$	A	–	–	E	F	F	D	C
	A	C	A	E	F	A	–	C
	A	D	D	E	F	A	D	F
$Text$	A	D	D	A	F	F	D	F
emission probability	1	1/4	3/4	1/5	1	1/5	3/4	3/5

Slika 9.7: Poravnanje $Text = ADDAFFDF$ u odnosu na semeno poravnanje $Alignment^*$ predstavljeno kao jednostavan HMM na slici 9.6. Ovaj HMM je ograničen zato što nismo u mogućnosti da poravnamo nizove dužine različite od 8.

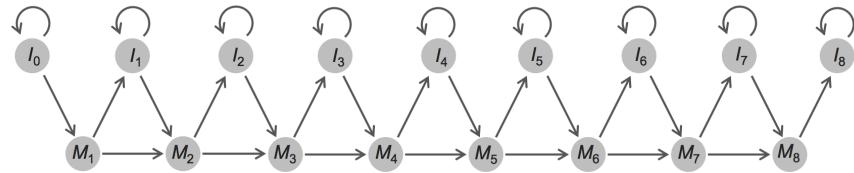
9.7.2 Građenje profilnog HMM-a

Poboljšani HMM koji ćemo predstaviti se zove **profilni HMM**. Sa datim višestrukim poravnanjem $Alignment$ i pragom brisanja kolona θ koji koristimo da dodemo do $Alignment^*$ -a, označićemo ovaj profilni HMM kao $HMM(Alignment, \theta)$. Za dati niz $Text$ da se poravna u odnosu na postojeće semeno poravnanje, naš cilj je da pronađemo optimalni skriveni put u profilnom HMM-u tako što ćemo rešiti Problem Dekodiranja za ovaj HMM i emitovani niz $Text$.

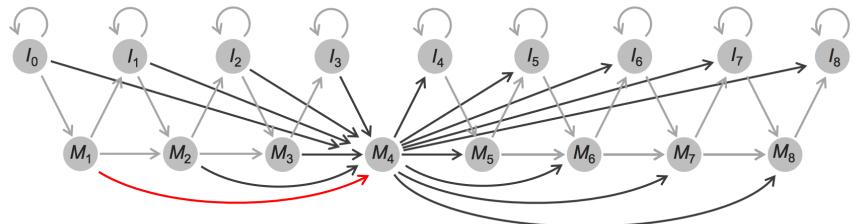
Prvo dodajemo $k + 1$ **insercionih stanja**, označenih kao $INSERTION(0), \dots, INSERTION(k)$ (slika 9.8). Ulaženje u $INSERTION(i)$ dopušta profilnom HMM-u da emituje dodatni simbol nakon posećivanja i -te kolone $PROFILE(Alignment^*)$ -a i pre ulaženja u $(i + 1)$ -tu kolonu. Povezujemo $MATCH(i)$ sa $INSERTION(i)$ i $INSERTION(i)$ sa $MATCH(i + 1)$. Kako bismo dopustili višestruke insertovane simbole između kolona $PROFILE(Alignment^*)$ -a, povezaćemo $INSERTION(i)$ samu sa sobom.

Nakon modelovanja insercija novih simbola u $PROFILE(Alignment^*)$, treba da modelujemo i „delecije” koja omogućavaju profilnom HMM-u da preskoči kolone $PROFILE(Alignment^*)$ -a. Jedan način modeliranja ovih delecija je da dodamo ivice koje povezuju svako stanje u profilnom HMM-u sa svakim stanjem desno od njega (slika 9.9).

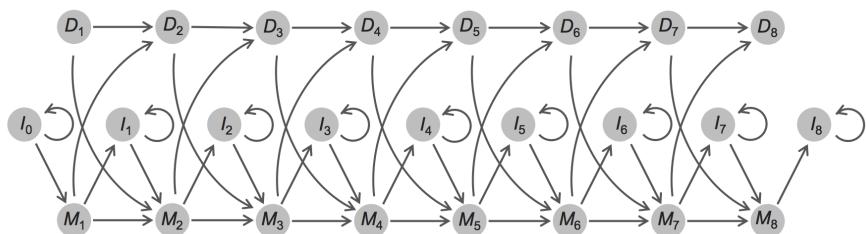
Delecije na ovaj način zahtevaju veliki broj grana. Zbog toga ćemo dodati stanja delecije, kao za insercije, što je prikazano na slici 9.10.



Slika 9.8: HMM dijagram za semeno poravnjavanje sa slike 9.6 sa pogotcima i insercionim stanjima, skraćeni kao M i I, redom. Stanja I_0 i I_8 modeluju insercije simbola koje se dešavaju pre početka i kraja $Alignment^*$ -a, redom.

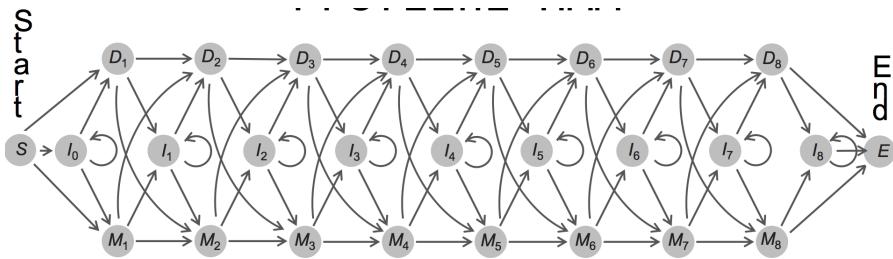


Slika 9.9: Dodavanjem ivica koje povezuju svako stanje u profilnom HMM-u sa svakim stanjem desno od njega, možemo preskočiti kolone $Alignment$ -a kada poredimo $Text$ u odnosu na ovo poravnjavanje. Gornji HMM dijagram označava da sve ivice vode u i iz $MATCH(4)$.



Slika 9.10: Dodavanje stanja deleciji (skraceno kao D_i) profilnom HMM dijagramu.

Imamo match, mismatch, insertion i deletion grane u grafu. Da li nam još nešto nedostaje? Pa treba da dodamo grane između stanja insercije i delecije, kao na slici 9.11. Ono što smo dobili jeste profilni HMM dijagram. Ono što se sad pitamo jeste šta su nam matrice prelaska i emisiona matrica, kako da ih izračunamo preko dijagrama?



Slika 9.11: Dodavanje prelaza od insertionih stanja do deletionih stanja i obrnuto upotpunjava profilni HMM dijagram za profilnu matricu na slici 9.6. Početno i završno stanje su označeno kao S i E , redom.

Problem profilnog HMM-a: Konstruisati profilni HMM na osnovu višestrukog poravnjanja.

Ulaz: Višestruko poravnanje *Alignment* i parametar θ (maksimalni udeo insercija po koloniji).

Izlaz: Emisiona i tranziciona matrica profilnog HMM $HMM(Alignment, \theta)$.

Neka nam je dato višestruko poravnanje niski kao na slici 9.12, i neka je određen HMM dijagram za svaku od niski, kao na slici 9.13. Da vidimo kako se određuju matrica prelaska i emisiona matrica.

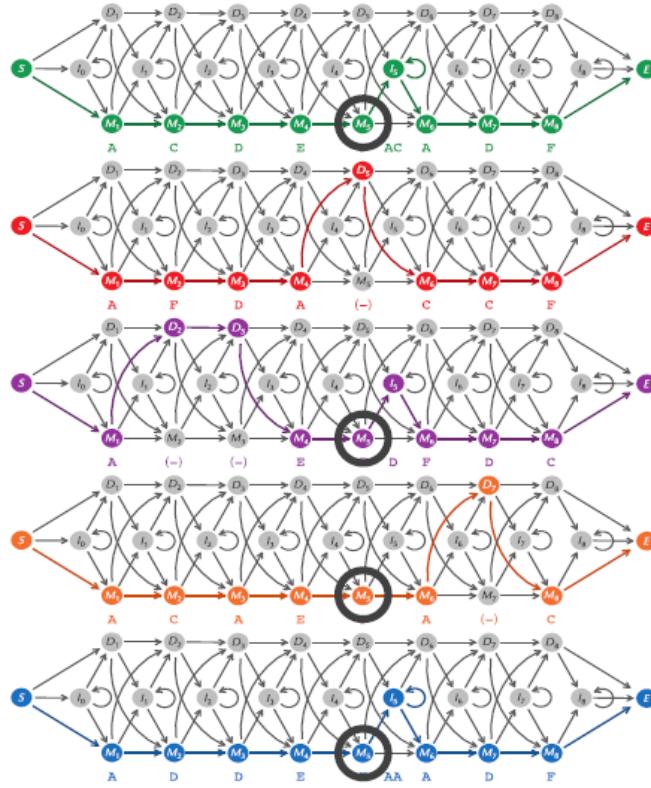
A	C	D	E	F	A	C	A	D	F
A	F	D	A	-	- -	C	C	F	
A	-	-	E	F	D	-	F	D	C
A	C	A	E	F	- -	A	-	-	C
A	D	D	E	F	A	A	A	D	F

Slika 9.12: Višestruko poravnanje pet niski.

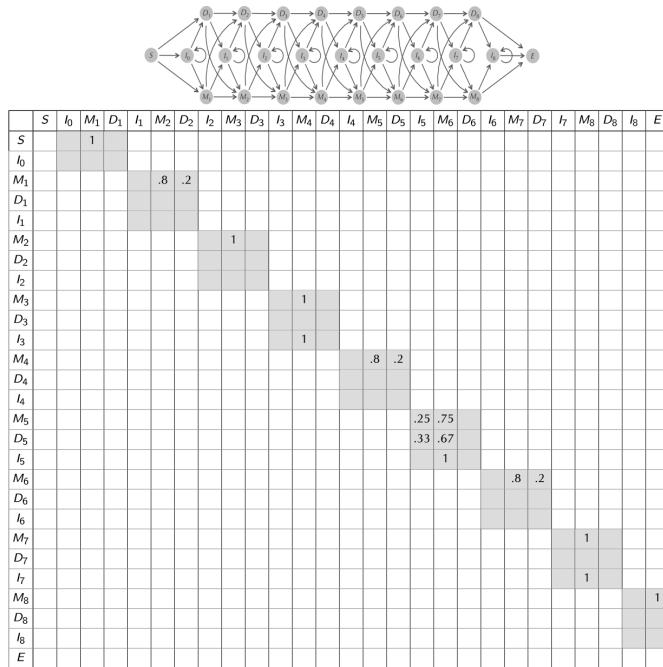
Posmatrajmo M_5 . Vidimo da imamo 4 prelaska iz stanja M_5 u druga, i to su 3 insercije, jedno poklapanje, a delecija nije bilo. Petog prelaska nema jer se pre toga desila delecija pa nije došlo do emitovanja simbola na poziciji 5. Na ovaj način možemo za svako stanje da odredimo prelaska.

Što se tiče emisione matrice, možemo posmatrati M_2 . Na ovom delu imamo 5 simbola. Dva puta se emitovao simbol C, po jednom siboli D i F, a simboli A i E se nisu emitovali na ovoj poziciji. Za svaku poziciju ponovimo postupak i dobili smo emisionu matricu.

Kada bismo pogledali verovatnoće prelaska videli bismo sledeće (9.14). Sive celije su grane u HMM dijagramu, a prazne su zabranjeni prelasci. Vidimo da ima mnogo praznog prostora, odnosno, da je matrica retka. Imamo i vrednosti 0 u sivim celijama. Zbog toga primenjujemo Laplasovo pravilo, a σ bi bio dodatan parametar za profilni dijagram.



Slika 9.13: Verovatnoće prelaska u profilnom HMM

Slika 9.14: Zabranjeni prelasci. **Sive ćelije:** grane u HMM dijagramu. **Prazne ćelije:** zabranjeni prelasci.