UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

Dunja Čitlučanin

PRONALAŽENJE POČETNOG REGIONA REPLIKACIJE DNK - ELEKTRONSKA LEKCIJA

master rad

Beograd, 2026.

Mentor:

dr Jovana Kovačević, vanredni profesor
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Mirjana Maljković Ružičić, docent
Univerzitet u Beogradu, Matematički fakultet

Nevena Ćirić, asistent
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Mom tati, koji je uvek verovao u mene

Naslov master rada: Pronalaženje početnog regiona replikacije DNK - elektronska lekcija

Rezime: Ovaj rad predstavlja elektronsku lekciju posvećenu algoritmima za identifikaciju početnog regiona replikacije u DNK sekvencama. Lekcija obuhvata teorijsko objašnjenje osnovnih bioinformatičkih koncepata i algoritama, uključujući *Frequent Words*, *Clump Finding*, *GC-skew* analizu i pristupe sa dozvoljenim odstupanjima, uz analizu njihovog rada i vremenske složenosti. Interaktivni deo realizovan je kao veb aplikacija korišćenjem tehnologija *FastAPI* i *React*, koja omogućava pokretanje algoritama implementiranih u programskom jeziku *Python*. Korisnicima je omogućeno da eksperimentišu sa različitim ulaznim podacima i prate rezultate izvršavanja algoritama, čime se olakšava razumevanje njihovih principa rada i primene u bioinformatici. Ovakva elektronska lekcija može služiti kao edukativni alat za studente i nastavnike u oblasti bioinformatike.

Ključne reči: DNK, bioinformatika, algoritmi, replikacija, *oriC*, *Frequent Words*, *Clump Finding*, *GC-skew*, *mismatches*

Sadržaj

Glava 1

Uvod

Dezoksiribonukleinska kiselina (DNK) je osnovni nosilac genetske informacije kod živih organizama. Jedan od značajnih izazova u bioinformatici predstavlja identifikacija funkcionalno relevantnih regiona unutar DNK sekvenci, pri čemu se ova analiza često zasniva na primeni algoritamskih metoda.

Posebno interesantan problem predstavlja određivanje početnog regiona replikacije (*oriC*, eng. *origin of replication*), koji ima ključnu ulogu u procesu umnožavanja genetskog materijala. U okviru velikih genoma, koji mogu sadržati milione nukleotida, ovaj region je relativno kratak i nije eksplicitno označen[1]. Stoga se njegovo identifikovanje zasniva na analizi obrazaca u DNK sekvenci i prepoznavanju karakterističnih signala koji ukazuju na početak replikacije.

Jedan od ključnih uvida jeste da se u ovim regionima često pojavljuju ponavljajuće sekvence, kao i specifične statističke nepravilnosti u raspodeli nukleotida. Ove osobine omogućavaju primenu algoritamskih metoda za identifikaciju kandidata za *oriC* region.

U ovom radu analizirani su algoritamski pristupi koji se zasnivaju na učestalosti podniski, njihovoj lokalnoj raspodeli i statističkim svojstvima genoma. Cilj rada je da se prikaže kako se kombinacijom ovih metoda može efikasno pristupiti problemu identifikacije početnog regiona replikacije. Rad je organizovan na sledeći način. U drugom poglavlju predstavljene su osnovne biološke i bioinformatičke osnove koje su neophodne za razumevanje problema. U trećem poglavlju obrađeni su algoritamski pristupi za pronalaženje početnog regiona replikacije. U četvrtom poglavlju

[1] U DNK sekvenci ne postoje unapred definisane oznake koje bi direktno ukazivale na početak, kraj ili lokaciju *oriC* regiona. Kada bi region bio eksplicitno označen, njegova lokacija bila bi jasno definisana i mogla bi se neposredno očitati iz podataka, bez potrebe za dodatnom analizom.

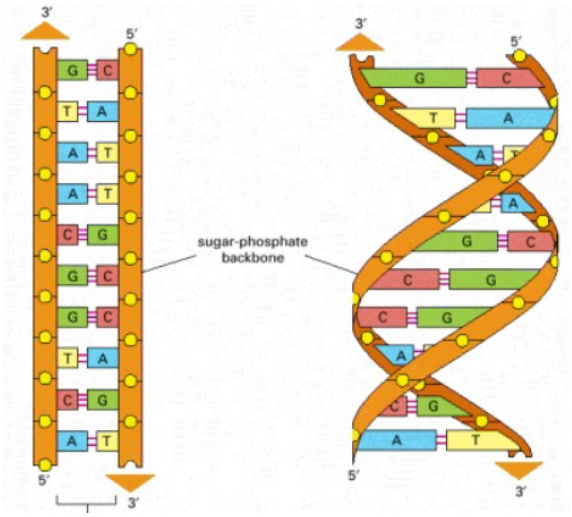
prikazana je implementacija elektronske lekcije i tehnologije korišćene za realizaciju interaktivnog dela rada.

Glava 2

Teorijske osnove

2.1 Struktura DNK

DNK je molekul koji se sastoji od dva međusobno povezana lanca koji formiraju dvostruku spiralu. Svaki lanac je građen od nukleotida, pri čemu svaki nukleotid sadrži jednu od četiri azotne baze: adenin (A), timin (T), guanin (G) ili citozin (C). Kao što je prikazano na Slici 2.1, baze se međusobno uparuju po principu komplementarnosti, tako da se adenin vezuje za timin, a guanin za citozin. Ovakva struktura omogućava stabilnost molekula, kao i precizno kopiranje genetske informacije tokom procesa replikacije [10].



Slika 2.1: Struktura DNK [10]

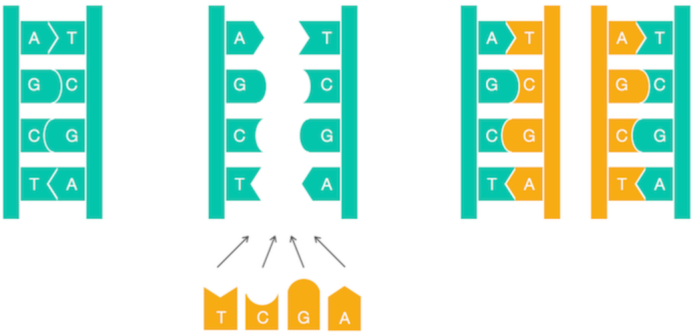
DNK je u ćelijama organizovana u strukture koje se nazivaju hromozomi. Skup svih DNK molekula jednog organizma, odnosno kompletna informacija sadržana u

njegovim hromozomima, naziva se genom. Različiti organizmi imaju različit broj hromozoma, kao i genom različite dužine, koji može varirati od nekoliko stotina hiljada do više miliona nukleotida.

Sa stanovišta bioinformatike, DNK sekvenca se može predstaviti kao konačna niska karaktera nad alfabetom $\{A, C, G, T\}$. Podniske te niske, dužine k, nazivaju se k-grami i predstavljaju osnovne jedinice koje se analiziraju u bioinformatici. Analiza učestalosti i raspodele k-grama u genomu omogućava otkrivanje obrazaca koji mogu ukazivati na funkcionalno značajne regione DNK. Ovakav model omogućava primenu algoritamskih metoda za analizu sekvenci, prepoznavanje obrazaca i rešavanje različitih problema u obradi genetskih podataka.

2.2 Replikacija DNK

Replikacija DNK predstavlja proces umnožavanja genetskog materijala koji se odvija pre deobe ćelije. Tokom ovog procesa, dva lanca DNK se razdvajaju, a zatim se za svaki od njih sintetiše novi komplementarni lanac. Kao rezultat replikacije nastaju dve identične kopije originalnog DNK molekula, koje se raspodeljuju u ćerke ćelije. Slika 2.2 ilustruje opisano.

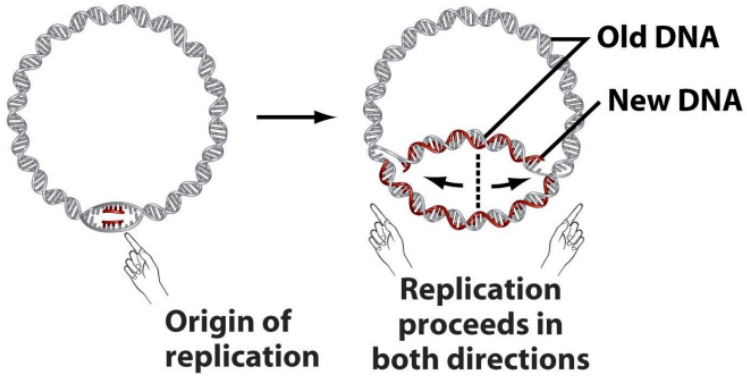


Slika 2.2: Replikacija DNK[14]

Ovaj proces je strogo regulisan, jer njegovo pokretanje i odvijanje kontrolišu različiti enzimi i drugi proteini kako bi replikacija započela na odgovarajućim lokacijama u genomu. Nakon toga, replikacija se odvija u oba smera duž DNK molekula, pri čemu dolazi do razdvajanja roditeljskih lanaca i sinteze novih komplementarnih nukleotidnih lanaca.

2.3 Početni region replikacije (*oriC*)

Početni region replikacije, označen kao *oriC*, predstavlja specifičan segment DNK u kojem započinje proces replikacije. Kod bakterija, genom je najčešće organizovan u obliku kružnog hromozoma i sadrži samo jedan takav region iz kojeg se replikacija odvija u oba smera, kao što je prikazano na Slici 2.3.



Slika 2.3: Početni region replikacije [12]

Ovaj region sadrži sekvence koje omogućavaju vezivanje proteina neophodnih za početak replikacije i time ima ključnu ulogu u pokretanju celokupnog procesa umnožavanja genetskog materijala. Iako je *oriC* relativno kratak u poređenju sa dužinom celog genoma, koji može sadržati milione nukleotida, on u DNK sekvenci nije eksplicitno označen, odnosno ne postoji jedinstvena sekvenca ili oznaka koja direktno ukazuje na njegovu lokaciju. Zbog toga, njegovo pronalaženje predstavlja složen i netrivijalan zadatak.

2.4 *DNK* polimeraza, *DnkA* proteini i boksovi

Početak replikacije DNK započinje vezivanjem specifičnih proteina za određene delove genoma. Jedan od najvažnijih proteina u ovom procesu je *DnkA*, koji ima sposobnost da prepozna i veže se za kratke sekvence unutar početnog regiona replikacije. Ove kratke sekvence poznate su kao *DnkA* boksovi i karakteriše ih to što se pojavljuju više puta unutar *oriC* regiona. Njihova učestalost na ograničenom regionu unutar DNK omogućava proteinima da identifikuju mesto na kojem treba da započne replikacija. Nakon vezivanja *DnkA* proteina dolazi do lokalnog razdvajanja DNK lanca, čime se omogućava delovanje *DNK polimeraze*, enzima koji funkcioni-

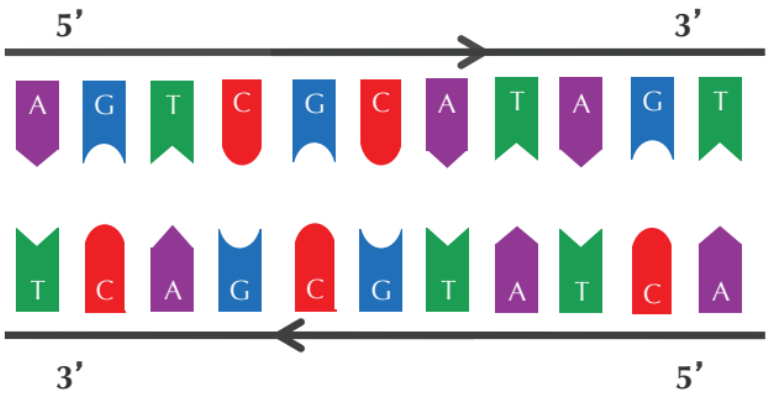
še kao molekularna mašina za kopiranje genetskog materijala i vrši sintezu novih, komplementarnih DNK lanaca. Upravo zbog svoje ponavljajuće prirode, *DnkA* boksovi se mogu posmatrati kao karakteristični obrasci u DNK sekvenci, što omogućava njihovo pronalaženje primenom algoritamskih metoda.

2.5 Obrnuti komplement

Zbog komplementarne prirode DNK molekula, svaki lanac DNK povezan je sa drugim lancem prema pravilima uparivanja baza, pri čemu se adenin (A) uparuje sa timinom (T), dok se guanin (G) uparuje sa citozinom (C). Pored toga, dva lanca DNK molekula imaju suprotnu orijentaciju (antiparalelni su), odnosno jedan je orijentisan u smeru $5' \rightarrow 3'$, a drugi u smeru $3' \rightarrow 5'$. Oznake $5'$ i $3'$ predstavljaju krajeve DNK lanca i određuju njegovu orijentaciju. Po konvenciji, DNK sekvence se zapisuju i čitaju u smeru $5' \rightarrow 3'$, što omogućava jednoznačno predstavljanje i poređenje sekvenci.

Na slici 2.4 prikazana su dva komplementarna DNK lanca i način uparivanja nukleotida. Može se uočiti da svakoj bazi jednog lanca odgovara tačno određena komplementarna baza na drugom lancu.

Na osnovu ovih osobina definiše se pojam obrnutog komplementa (eng. *reverse complement*). Komplement jedne DNK sekvence dobija se zamenom svake baze njenom komplementarnom bazom. Zbog suprotne orijentacije DNK lanaca, komplementarna sekvenca čita se u smeru $5' \rightarrow 3'$, čime se dobija obrnuti komplement.



Slika 2.4: Komplementarni DNK lanci i suprotna orijentacija lanaca [14]

Na primer, za sekvencu AGTC, komplementarna sekvenca je TCAG, dok je njen obrnuti komplement GACT.

S obzirom na dvostruku strukturu DNK molekula, biološki značajne sekvence mogu se pojaviti na bilo kom od njegova dva lanca. Zbog toga je pri analizi genoma neophodno uzeti u obzir oba lanca DNK molekula. Kako se DNK sekvence po konvenciji zapisuju u smeru $5' \rightarrow 3'$, drugi lanac predstavlja se njegovim obrnutim komplementom. Ovo je od posebnog značaja u algoritmima za pronalaženje početnog regiona replikacije, jer omogućava prepoznavanje karakterističnih obrazaca bez obzira na lanac na kojem se nalaze.

Glava 3

Algoritamski pristupi za pronalaženje početnog regiona replikacije

Biološke osobine početnog regiona replikacije, kao što su prisustvo ponavljajućih sekvenci i statističke nepravilnosti u raspodeli nukleotida, omogućavaju primenu algoritamskih metoda za njegovo pronalaženje. Analizom učestalosti podniski, njihove lokalne raspodele i statističkih karakteristika genoma moguće je identifikovati potencijalne kandidate za *oriC* region. U ovom poglavlju biće predstavljeni algoritamski pristupi koji koriste ove osobine za analizu DNK sekvenci.

3.1 Problem čestih reči

Kako određeni obrasci u DNK sekvenci mogu ukazivati na funkcionalno važne regione, prirodan pristup jeste analiza učestalosti pojavljivanja podniski. Sekvence koje se pojavljuju značajno češće od ostalih mogu imati posebnu biološku ulogu, pa se problem svodi na pronalaženje najčešće zastupljenih podniski određene dužine.

Polazeći od pretpostavke da se DNK sekvenca može posmatrati kao niska karaktera nad alfabetom $\{A, C, G, T\}$, moguće je primeniti metode analize teksta u cilju identifikacije karakterističnih obrazaca.

Podniske dužine k jedne biološke sekvence (DNK, njenog segmenta ili proteina) nazivaju se k-grami (eng. *k-mers*) i predstavljaju osnovne jedinice analize bioloških sekvenci. Analizom njihovog broja pojavljivanja moguće je identifikovati obrasce koji mogu ukazivati na prisustvo funkcionalno značajnih regiona, kao što je *oriC*.

Neka je data niska $Text$ i podniska $Pattern$ dužine k. Definišemo funkciju:

$$PatternCount(Text, Pattern)$$

kao broj pojavljivanja podniske $Pattern$ u niski $Text$, pri čemu se uzimaju u obzir i preklapajuća pojavljivanja.

Algoritam za prebrojavanje pojavljivanja

Za izračunavanje funkcije $PatternCount(Text, Pattern)$ koristi se pristup zasnovan na kliznom prozoru (eng. *sliding window*). Ideja je da se prolaskom kroz tekst proveri svaka podniska dužine k.

Algoritam 1 prikazuje pseudokod za izračunavanje ove funkcije.

Algoritam 1: Pattern Count

```
Funkcija PatternCount(Text, Pattern)
    count ← 0
    for i ← 0 to |Text| − |Pattern| do
        if Text[i .. (i + |Pattern| − 1)] = Pattern then
            count ← count + 1
        end
    end
    return count
```

3.1.1 Algoritam

Problem čestih reči (eng. *Frequent Words Problem*) može se formalno definisati na sledeći način:

- Ulaz: niska karaktera $Text$ i ceo broj k
- Izlaz: skup svih k-grama koji maksimizuju $PatternCount(Text, Pattern)$

Kažemo da je k-gram najčešći ukoliko se nijedan drugi k-gram ne pojavljuje više puta od njega u posmatranoj sekvenci.

Ideja i opis algoritma

Osnovna ideja algoritma za pronalazak čestih reči zasniva se na prolasku kroz celu DNK sekvencu i prebrojavanju koliko puta se svaka podniska dužine k pojavljuje. Nakon toga se izdvajaju one podniske koje se pojavljuju maksimalan broj puta.

Naivni pristup rešavanju ovog problema može se opisati kroz sledeće korake:

- Proći kroz sve moguće pozicije u niski $Text$.
- Za svaku moguću početnu poziciju u niski izdvojiti podnisku dužine k koja počinje na toj poziciji.
- Za svaku podnisku izračunati $PatternCount(Text, Pattern)$ primenom algoritma 1.
- Odrediti maksimalan broj pojavljivanja.
- Izdvojiti sve podniske koje imaju tu frekvenciju.

Algoritam 2 prikazuje pseudokod algoritma za pronalaženje najčešćih reči.

Algoritam 2: Frequent Words

Funkcija `FrequentWords(`*Text, k*`)`
- $najcesciKgrami \leftarrow$ prazan skup
- $count \leftarrow$ niz dužine $|Text| - k + 1$ inicijalizovan nulama
- **for** $i \leftarrow 0$ **to** $|Text| - k$ **do**
  - $pattern \leftarrow Text[i..i + k - 1]$
  - $count[i] \leftarrow$ PatternCount$(Text, pattern)$
- **end**
- $maxBroj \leftarrow$ maksimalna vrednost u nizu $count$
- **for** $i \leftarrow 0$ **to** $|Text| - k$ **do**
  - **if** $count[i] = maxBroj$ **then**
    - dodaj $Text[i..i + k - 1]$ u $najcesciKgrami$
  - **end**
- **end**
- ukloni duplikate iz $najcesciKgrami$
- **return** $najcesciKgrami$

Primer

Posmatrajmo DNK sekvencu:

`ACTGACTCCCACCCC`

Za $k = 3$ izdvajaju se svi 3-grami i računa se broj njihovih pojavljivanja u sekvenci.

Neki od izdvojenih 3-grama su:

`ACT`, `CTG`, `TGA`, `GAC`, `ACT`, ...

Algoritam ***PatternCount*** za svaki obrazac prolazi kroz celu sekvencu i određuje koliko puta se dati 3-gram pojavljuje. Važno je napomenuti da algoritam ***PatternCount*** uzima u obzir i preklapajuća pojavljivanja podniski. To znači da nova pojava podniske može započeti pre nego što se prethodna potpuno završi.

Na primer, u nizu:

`ATATATA`

podniska `ATA` pojavljuje se na pozicijama 0, 2 i 4, pri čemu se ova pojavljivanja međusobno preklapaju.

Primenom algoritma ***PatternCount*** za svaki izdvojeni 3-gram računa se broj njegovih pojavljivanja u celoj sekvenci. Na slici 3.1 prikazan je niz ***Count*** kreiran primenom algoritma 2.

Text	**A**	**C**	**T**	G	**A**	**C**	**T**	C	C	C	A	C	C	C	C
COUNT	**2**	1	1	1	**2**	1	1	3	1	1	1	3	3		

Slika 3.1: Primer izdvajanja 3-grama i određivanja njihovih frekvencija [14].

Na primer:

$$\text{PatternCount}(\texttt{Text}, \texttt{ACT}) = 2$$

jer se obrazac `ACT` pojavljuje dva puta u sekvenci.

Slično tome:

$$\text{PatternCount}(\texttt{Text}, \texttt{CCC}) = 3$$

što znači da je `CCC` najčešći 3-gram u datoj sekvenci.

3.1.2 Vremenska složenost

Algoritam ***PatternCount*** prolazi kroz sve moguće pozicije u niski $Text$, kojih ima $|Text|-k+1$. Za svaku poziciju vrši se poređenje podniske dužine k, što zahteva do k poređenja karaktera. Zbog toga je vremenska složenost ovog algoritma:

$$O((|Text| - k + 1) \cdot k)$$

što se za velike vrednosti $|Text|$ može aproksimirati kao:

$$O(|Text| \cdot k)$$

Algoritam ***FrequentWords*** generiše $|Text| - k + 1$ k-grama i za svaki od njih poziva funkciju ***PatternCount***. Stoga je ukupan broj operacija:

$$(|Text| - k + 1) \cdot ((|Text| - k + 1) \cdot k)$$

što daje ukupnu vremensku složenost:

$$O((|Text| - k + 1)^2 \cdot k)$$

Za velike vrednosti ulaza, ova složenost se može pojednostaviti na:

$$O(|Text|^2 \cdot k)$$

Drugim rečima, algoritam ima kvadratnu složenost u odnosu na dužinu sekvence, uz dodatni faktor k koji odgovara dužini analiziranih podniski.

Ovakva složenost brzo postaje nepraktična za duge DNK sekvence, koje mogu sadržati milione nukleotida. Zbog toga je neophodno razmotriti optimizovane pristupe, poput korišćenja niza frekvencija ili sortiranja, koji omogućavaju značajno efikasnije rešavanje ovog problema. Ovi pristupi biće detaljno analizirani u sekcijama 3.1.3 i 3.1.5.

3.1.3 Optimizovani pristup korišćenjem niza frekvencija

Naivni algoritam za rešavanje problema čestih reči ima visoku vrsemensku složenost jer za svaki k-gram ponovo prolazi kroz celu sekvencu. Da bi se ovaj problem rešio efikasnije, uvodi se optimizovani pristup zasnovan na korišćenju niza frekvencija.

Osnovna ideja je da se kroz sekvencu prolazi samo jednom, pri čemu se za svaki k-gram beleži broj njegovih pojavljivanja. Umesto ponovnog prebrojavanja, rezultati se čuvaju u strukturi podataka koja omogućava brz pristup.

Leksikografsko indeksiranje i numerička reprezentacija k-grama

Kako bi se omogućilo efikasno skladištenje i pretraživanje, svaki k-gram se mapira na jedinstveni indeks. S obzirom na to da DNK alfabet sadrži četiri simbola, postoji ukupno 4^k mogućih k-grama.

Ovi k-grami se mogu poređati leksikografski (kao u rečniku), pri čemu se svakom dodeljuje jedinstveni broj iz opsega $[0, 4^k - 1]$.

Za ovu transformaciju koriste se funkcije ***PatternToNumber*** i ***NumberToPattern***, koje omogućavaju prelazak između simboličke i numeričke reprezentacije k-grama.

DNK alfabet $\{A, C, G, T\}$ može se mapirati na sledeći:

$$A \to 0, \quad C \to 1, \quad G \to 2, \quad T \to 3$$

Funkcija ***SymbolToNumber*** predstavlja pomoćnu funkciju koja svakom simbolu DNK alfabeta (`A`, `C`, `G`, `T`) dodeljuje odgovarajuću numeričku vrednost (`0`, `1`, `2`, `3`), čime omogućava numeričku reprezentaciju k-grama u sistemu sa osnovom 4.

Funkcija ***PatternToNumber*** preslikava dati k-gram u odgovarajući broj koristeći zapis u brojnom sistemu sa osnovom 4. Pri tome se prefiks k-grama posmatra kao već izračunata vrednost, koja se množi sa 4 i uvećava za numeričku vrednost poslednjeg simbola. Ovaj postupak odgovara standardnoj konverziji iz pozicionog brojnog sistema i i omogućava da se svaki k-gram jednoznačno predstavi celobrojnom vrednošću. U algoritmu zasnovanom na nizu frekvencija, dobijena celobrojna vrednost predstavlja indeks odgovarajućeg elementa niza frekvencija, u kojem se čuva broj pojavljivanja datog k-grama. Algoritam 3 prikazuje pseudokod funkcije ***PatternToNumber***.

Obrnuta transformacija vrši se funkcijom ***NumberToPattern***, koja dati indeks pretvara nazad u odgovarajući k-gram. Funkcija ***NumberToSymbol*** vrši obrnutu transformaciju u odnosu na funkciju ***SymbolToNumber***, tako što svakoj numeričkoj vrednosti iz skupa `(0,1,2,3)` dodeljuje odgovarajući simbol DNK alfabeta (`A`, `C`, `G`, `T`). Algoritam 4 prikazuje pseudokod funkcije ***NumberToSymbol***, koriste se operacije celobrojnog deljenja i ostatka pri deljenju sa 4: količnik određuje prefiks,

Algoritam 3: PatternToNumber

Funkcija `PatternToNumber(`*Pattern*`)`
- **if** *Pattern je prazan* **then**
  - **return** 0
- **end**
- *symbol* ← poslednji simbol u *Pattern*
- $prefix$ ← *Pattern* bez poslednjeg simbola
- **return** 4· *PatternToNumber*($prefix$) + *SymbolToNumber*(*symbol*)

dok ostatak određuje poslednji simbol. Na taj način se postupno rekonstruiše ceo k-gram.

Algoritam 4: NumberToPattern

Funkcija `NumberToPattern(`*index, k*`)`
- **if** $k = 1$ **then**
  - **return** NumberToSymbol(*index*)
- **end**
- $prefixIndex \leftarrow \lfloor index/4 \rfloor$
- $r \leftarrow index \bmod 4$
- *symbol* ← NumberToSymbol(r)
- $prefixPattern \leftarrow$ NumberToPattern($prefixIndex, k-1$)
- **return** konkatenacija $prefixPattern$ i *symbol*

Ove dve funkcije zajedno omogućavaju efikasan prelazak između simboličke i numeričke reprezentacije k-grama, što je ključno za implementaciju algoritama zasnovanih na nizovima frekvencija.

Algoritam

Algoritam za izračunavanje frekvencija prolazi kroz tekst samo jednom i za svaki k-gram povećava odgovarajući element niza frekvencija. Pseudokod prikazuje Algoritam 5.

Nakon što je niz frekvencija izračunat, najčešći k-grami se dobijaju pronalaženjem maksimalne vrednosti u nizu i identifikacijom svih indeksa i njima odgovarajućih k-grama koji imaju tu vrednost. Algoritam 6 prikazuje pseudokod funkcije *Faster Frequent Words*.

Algoritam 5: Computing Frequencies

```
Funkcija ComputingFrequencies(Text, k)
    for i ← 0 to 4^k − 1 do
        FrequencyArray[i] ← 0
    end
    for i ← 0 to |Text| − k do
        Pattern ← Text[i..i + k − 1]
        j ← PatternToNumber(Pattern)
        FrequencyArray[j] ← FrequencyArray[j] + 1
    end
    return FrequencyArray
```

Algoritam 6: Faster Frequent Words

```
Funkcija FasterFrequentWords(Text, k)
    najcesciKgrami ← prazan skup
    FrequencyArray ← ComputingFrequencies(Text, k)
    maxCount ← maksimalna vrednost u FrequencyArray
    for i ← 0 to 4^k − 1 do
        if FrequencyArray[i] = maxCount then
            Pattern ← NumberToPattern(i, k)
            dodaj Pattern u skup rezultata
        end
    end
    return najcesciKgrami
```

Primer

Za slučaj $k = 2$ postoji ukupno $4^2 = 16$ mogućih 2-grama. Oni se mogu poređati leksikografski, pri čemu svaki 2-gram dobija jedinstveni indeks iz opsega od 0 do 15.

Na slici 3.2 prikazan je primer leksikografskog uređenja svih mogućih 2-grama, zajedno sa odgovarajućim indeksima i nizom frekvencija za posmatranu DNK sekvencu.

Na primer, 2-gram `GT` ima indeks `11` što znači da funkcija: *`PatternToNumber(GT)`* =11. Obrnuto, funkcija: *`NumberToPattern`*(11,2) vraća `GT`.

Na ovaj način omogućava se efikasno mapiranje između simboličke i numeričke reprezentacije k-grama, što predstavlja osnovu za konstrukciju niza frekvencija.

Niz frekvencija prikazan na Slici 3.2 predstavlja broj pojavljivanja svakog 2-grama u posmatranoj sekvenci. Na primer, vrednost 3 na poziciji koja odgovara 2-gramu `AA` pokazuje da se ovaj obrazac pojavljuje tri puta u sekvenci. Slično tome,

2-gram GG takođe ima frekvenciju 3, dok se neki obrasci ne pojavljuju nijednom, pa je njihova frekvencija jednaka 0. Na ovaj način niz frekvencija omogućava efikasno određivanje najčešćih k-grama bez ponovnog prolaska kroz celu sekvencu.

k-mer	AA	AC	AG	AT	CA	CC	CG	CT	GA	GC	GG	GT	TA	TC	TG	TT
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
frequency	3	0	2	0	1	0	0	0	0	1	3	1	0	0	1	0

Slika 3.2: Leksikografsko indeksiranje 2-grama i primer niza frekvencija [14].

Vremenska složenost optimizovanog pristupa

U ovom pristupu niska *Text* se obrađuje samo jednom, pri čemu se za svaki k-gram vrši njegova obrada i konverzija u numerički oblik, što daje složenost $O(|Text| \cdot k)$.

Međutim, dodatni trošak predstavlja inicijalizacija niza frekvencija dimenzije 4^k, kao i iteracija kroz ceo niz prilikom traženja maksimuma i rekonstrukcije odgovarajućih k-grama. Ukupna složenost algoritma iznosi:

$$O(|Text| \cdot k + 4^k \cdot k)$$

Za male vrednosti k, izraz 4^k može se smatrati konstantom, pa složenost postaje približno linearna u odnosu na dužinu teksta.

Prostorna složenost ovog pristupa određena je dimenzijom niza frekvencija, koji sadrži 4^k elemenata. Zbog toga prostorna složenost iznosi:

$$O(4^k)$$

Ovaj pristup je značajno efikasniji od naivnog algoritma kada je k relativno malo u odnosu na $|Text|$. Međutim, zbog eksponencijalnog rasta izraza 4^k, ovaj metod postaje nepraktičan za veće vrednosti k, što motiviše razvoj dodatnih optimizacija.

3.1.4 Optimizovani pristup korišćenjem heš mape

Prethodno opisani pristup korišćenjem niza frekvencija omogućava efikasnije rešavanje problema čestih reči u odnosu na naivni algoritam, ali ima jedan važan nedostatak. Dimenzija niza frekvencija iznosi 4^k, jer se unapred rezerviše mesto za

svaki mogući k-gram nad DNK alfabetom. Kada je vrednost k veća, broj mogućih k-grama raste eksponencijalno, pa ovaj pristup može zahtevati veliku količinu memorije.

Da bi se ovaj problem ublažio, uvodi se optimizovani pristup zasnovan na korišćenju heš mape, odnosno rečnika. Umesto da se čuva frekvencija za svaki mogući k-gram, u memoriji se čuvaju samo brojači onih k-grama koji se zaista pojavljuju u posmatranoj DNK sekvenci.

Osnovna ideja je ista kao u prethodno objašnjenom pristupu, odnosno, kroz nisku *Text* prolazi se samo jednom. U svakom koraku izdvaja se trenutni k-gram, a zatim se u heš mapi ažurira broj njegovih pojavljivanja. Ključ u heš mapi predstavlja k-gram, dok vrednost predstavlja broj njegovih pojavljivanja u tekstu.

Ako se posmatrani k-gram prvi put pojavljuje, on se dodaje u heš mapu sa vrednošću 1. Ukoliko već postoji u heš mapi, njegova vrednost se uvećava za 1. Na taj način se izbegava ponovno prebrojavanje pojavljivanja istog k-grama, kao i čuvanje frekvencija za k-grame koji se uopšte ne pojavljuju u sekvenci.

Algoritam

Algoritam za izračunavanje frekvencija pomoću heš mape prolazi kroz tekst i za svaki pronađeni k-gram ažurira odgovarajuću vrednost u rečniku. Pseudokod je prikazan u Algoritmu 7.

Algoritam 7: Computing Frequencies With Dictionary

```
Funkcija ComputingFrequenciesWithDictionary(Text, k)
    FrequencyMap ← prazna heš mapa
    for i ← 0 to |Text| − k do
        Pattern ← Text[i..i + k − 1]
        if Pattern nije ključ u FrequencyMap then
            FrequencyMap[Pattern] ← 1
        end
        else
            FrequencyMap[Pattern] ← FrequencyMap[Pattern] + 1
        end
    end
    return FrequencyMap
```

Nakon što su frekvencije svih pronađenih k-grama izračunate, najčešći k-grami se dobijaju pronalaženjem maksimalne vrednosti u heš mapi. Kao što je prikazano

u Algoritmu 8, svi obrasci čija je vrednost jednaka maksimumu dodaju se u skup rezultata [15].

Algoritam 8: Faster Frequent Words With Dictionary

```
Funkcija FasterFrequentWordsWithDictionary(Text, k)
    FrequentPatterns ← prazan skup
    FrequencyMap ← ComputingFrequenciesWithDictionary(Text, k)
    maxCount ← maksimalna vrednost u FrequencyMap
    foreach Pattern u FrequencyMap do
        if FrequencyMap[Pattern] = maxCount then
            dodaj Pattern u FrequentPatterns
        end
    end
    return FrequentPatterns
```

Primer

Neka je data DNK sekvenca `ACTGACTCCCACCCC` i neka je $k = 3$. Algoritam posmatra sve 3-grame koji se pojavljuju u tekstu pomeranjem prozora dužine 3 za jedno mesto udesno.

Na početku je heš mapa prazna. Prvi posmatrani 3-gram je `ACT`, pa se on dodaje u heš mapu sa vrednošću 1. Zatim se posmatra 3-gram `CTG`, koji se takođe dodaje sa vrednošću 1. Kada se kasnije ponovo pojavi 3-gram `ACT`, njegova vrednost u heš mapi se uvećava sa 1 na 2.

Na ovaj način heš mapa postepeno beleži broj pojavljivanja svakog 3-grama koji se zaista javlja u sekvenci. Na kraju obrade, najčešći 3-gram je onaj čija je vrednost u heš mapi najveća. Ako više različitih 3-grama ima istu maksimalnu vrednost, svi oni predstavljaju rezultat problema čestih reči.

Vremenska složenost optimizovanog pristupa korišćenjem heš mape

U ovom pristupu tekst se obrađuje samo jednom, pri čemu se u svakom koraku izdvaja jedan k-gram i ažurira njegova vrednost u heš mapi. Izdvajanje podniske dužine k zahteva $O(k)$ operacija, dok je pristup i ažuriranje vrednosti u heš mapi u prosečnom slučaju $O(1)$.

S obzirom na to da se ova operacija izvršava za svih $|Text| - k + 1$ pozicija, ukupna složenost ove faze iznosi:

$$O(|Text| \cdot k)$$

Nakon toga, potrebno je pronaći maksimalnu vrednost u heš mapi, što zahteva prolazak kroz sve njene elemente. U najgorem slučaju, broj različitih k-grama je najviše $|Text| - k + 1$, pa je za ovaj korak potrebno:

$$O(|Text|)$$

Kombinovanjem svih koraka dolazi se do ukupne vremenske složenosti:

$$O(|Text| \cdot k)$$

Za razliku od pristupa sa nizom frekvencija, ovde nije potrebno inicijalizovati niz dimenzije 4^k, niti prolaziti kroz sve moguće k-grame. Pretražuju se samo oni obrasci koji su se stvarno pojavili u niski *Text*.

Prostorna složenost zavisi od broja različitih k-grama u posmatranoj sekvenci. U najgorem slučaju, svi izdvojeni k-grami mogu biti različiti, pa je broj elemenata u heš mapi najviše $|Text| - k + 1$. Kako svaki k-gram zauzima $O(k)$ memorije, ukupna prostorna složenost iznosi:

$$O(|Text| \cdot k)$$

Ovaj pristup je posebno koristan za veće vrednosti k, jer izbegava eksponencijalni memorijski trošak koji nastaje kod niza frekvencija. Zbog toga predstavlja efikasniju optimizaciju kada nije potrebno čuvati informacije o k-gramima koji se ne pojavljuju u ulaznoj sekvenci.

3.1.5 Pronalaženje najčešćih k-grama sortiranjem

Pored pristupa zasnovanog na nizu frekvencija, problem čestih reči može se rešiti i korišćenjem sortiranja. Osnovna ideja ovog pristupa je da se svi k-grami najpre transformišu u numeričke vrednosti, nakon čega se sortiraju, čime se identični elementi grupišu jedan do drugog.

Neka je data niska $Text$. Za svaku poziciju i izdvajamo k-gram $Text[i..i+k-1]$ i transformišemo ga u odgovarajući broj korišćenjem funkcije ***PatternToNumber***. Na taj način dobijamo niz indeksa koji zatim sortiramo.

Nakon sortiranja, identični k-grami se pojavljuju u uzastopnim segmentima niza, što omogućava jednostavno prebrojavanje njihovih pojavljivanja. Algoritam 9 prikazuje pseudokod ovog pristupa.

Algoritam 9: Finding Frequent Words by Sorting

```
Funkcija FindingFrequentWordsBySorting(Text, k)
    FrequentPatterns ← prazan skup
    for i ← 0 to |Text| − k do
        Pattern ← Text[i..i + k − 1]
        Index[i] ← PatternToNumber(Pattern)
        Count[i] ← 1
    end
    SortedIndex ← Sort(Index)
    for i ← 1 to |Text| − k do
        if SortedIndex[i] = SortedIndex[i − 1] then
            Count[i] ← Count[i − 1] + 1
        end
    end
    maxCount ← maksimalna vrednost u Count
    for i ← 0 to |Text| − k do
        if Count[i] = maxCount then
            Pattern ← NumberToPattern(SortedIndex[i], k)
            dodaj Pattern u FrequentPatterns
        end
    end
    return FrequentPatterns
```

Primer

Posmatrajmo DNK sekvencu:

`AAGCAAAGGTGGG`

Za $k = 2$ izdvajaju se svi 2-grami redom kojim se pojavljuju u sekvenci i transformišu se u odgovarajući numerički indeks korišćenjem funkcije ***PatternToNumber***. Na taj način dobija se niz prikazan na Slici 3.3.

Nakon sortiranja identični indeksi se grupišu jedan do drugog (Slika 3.4). Zatim se prolaskom kroz sortirani niz određuje dužina svakog uzastopnog segmenta identičnih indeksa. Segment sa najvećim brojem elemenata odgovara k-gramu koji se pojavljuje najveći broj puta u posmatranoj sekvenci.

2-mer	AA	AG	GC	CA	AA	AA	AG	GG	GT	TG	GG	GG
INDEX	0	2	9	4	0	0	2	10	11	14	10	10

Slika 3.3: Izdvajanje i transformacija 2-grama [14].

2-mer	AA	AA	AA	AG	AG	CA	GC	GG	GG	GG	GT	TG
SORTEDINDEX	0	0	0	2	2	4	9	10	10	10	11	14
COUNT	1	2	3	1	2	1	1	1	2	3	1	1

Slika 3.4: Primer sortiranja indeksa i grupisanja identičnih k-grama [14].

U ovom primeru indeks `0` odgovara 2-gramu `AA`, dok indeks `10` odgovara 2-gramu `GG`. Kako se oba indeksa pojavljuju tri puta, zaključuje se da su `AA` i `GG` najčešći 2-grami u datoj sekvenci.

Vremenska složenost pristupa zasnovanog na sortiranju

U ovom pristupu generiše se niz od $|Text| - k + 1$ elemenata, nakon čega se vrši sortiranje tog niza. Dominantna operacija je sortiranje, čija složenost iznosi:

$$O(|Text| \log |Text|)$$

Dodatne operacije, poput transformacije k-grama i prolaska kroz niz, imaju složenost $O(|Text| \cdot k)$.

Ukupna vremenska složenost algoritma je:

$$O(|Text| \log |Text| + |Text| \cdot k)$$

3.1.6 Poređenje različitih pristupa

Na osnovu poređenja prikazanog u Tabeli 3.1 može se zaključiti da izbor algoritma zavisi od vrednosti parametra k, dužine ulazne sekvence *Text* i raspoloživih memorijskih resursa. Pristup zasnovan na nizu frekvencija efikasan je za manje vrednosti parametra k, dok je za veće vrednosti pogodniji pristup zasnovan na heš mapi, jer ne zahteva memoriju veličine 4^k. Pristup zasnovan na sortiranju predstavlja kompromis između vremenske i prostorne složenosti, dok je naivni algoritam prvenstveno pogodan za ilustraciju problema i obradu kraćih sekvenci.

Pristup	**Vremenska složenost**	**Prostorna složenost**	**Karakteristike**
Naivni algoritam	$O(\|Text\|^2 \cdot k)$	$O(\|Text\|)$	Jednostavna implementacija, ali veoma spor za duže sekvence.
Niz frekvencija	$O(\|Text\| \cdot k + 4^k \cdot k)$	$O(4^k)$	Efikasan za male vrednosti k, ali zahteva memoriju veličine 4^k.
Heš mapa	$O(\|Text\| \cdot k)$	$O(\|Text\| \cdot k)$	Čuvaju se samo k-grami koji se pojavljuju u sekvenci.
Sortiranje	$O(\|Text\| \log \|Text\| + \|Text\| \cdot k)$	$O(\|Text\|)$	Ne zahteva niz veličine 4^k, ali je sporiji od pristupa sa heš mapom zbog sortiranja.

Tabela 3.1: Poređenje različitih pristupa za rešavanje problema čestih reči

3.2 Problem pronalaženja grupa

Sama učestalost podniski na nivou celog genoma nije uvek dovoljna za preciznu identifikaciju funkcionalno značajnih regiona. U praksi je često mnogo značajnije posmatrati lokalnu frekvenciju podniski unutar kraćih segmenata genoma. Na primer, određeni k-gram može se pojavljivati relativno često u celom genomu, ali ukoliko su njegova pojavljivanja grupisana u kratkom regionu, takav obrazac može ukazivati na biološki značajan signal. Ovakva pojava je posebno izražena kod *DnkA* boksova, koji se često pojavljuju u grupama unutar početnog regiona replikacije, odnosno *oriC* regiona. Zbog toga analiza lokalnih ponavljanja predstavlja prirodno proširenje problema čestih reči.

Motivacija i veza sa biološkim problemom

Nakon identifikacije čestih obrazaca u poznatim *oriC* regionima, postavlja se ključno pitanje: kako pronaći slične regione u genomu kada početni region replikacije nije unapred poznat? U takvoj situaciji nije dovoljno posmatrati samo globalno najčešće k-grame, jer se biološki relevantan signal često ogleda u njihovom lokalnom grupisanju.

Analiza *oriC* regiona bakterije *Vibrio cholerae*, prikazanog na Slici 3.5, pokazuje da se određeni 9-grami pojavljuju više puta u tom regionu.

```
atcaatgatcaacgtaagcttctaagcatgatcaaggtgctcacacagtttatccacaacctgagtgg
atgacatcaagataggtcgttgtatctccttcctctcgtactctcatgaccacggaaagatgatcaag
agaggatgatttcttggccatatcgcaatgaatacttgtgacttgtgcttccaattgacatcttcagc
gccatattgcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgttctgttt
atcttgttttgactgagacttgttaggatagacggtttttcatcactgactagccaaagccttactct
gcctgacatcgaccgtaaattgataatgaatttacatgcttccgcgacgatttacctcttgatcatcg
atccgattgaagatcttcaattgttaattctcttgcctcgactcatagccatgatgagctcttgatca
tgtttccttaaccctctatttttttacggaagaatgatcaagctgctgctcttgatcatcgtttc
```

Slika 3.5: Početni region replikacije bakterije Vibrio cholerae [12]

Konkretno, u tom regionu izdvajaju se sledeći obrasci: `ATGATCAAG` i `CTTGATCAT` (Slika 3.6). Ova dva obrasca predstavljaju međusobne reverzne komplemente, što znači da nose istu biološku informaciju posmatrano na komplementarnim DNK lancima.

```
atcaatgatcaacgtaagcttctaagcATGATCAAGgtgctcacacagtttatccacaacctgagtgg
atgacatcaagataggtcgttgtatctccttcctctcgtactctcatgaccacggaaagATGATCAAG
agaggatgatttcttggccatatcgcaatgaatacttgtgacttgtgcttccaattgacatcttcagc
gccatattgcgctggccaaggtgacggagcgggattacgaaagcatgatcatggctgttgttctgttt
atcttgttttgactgagacttgttaggatagacggtttttcatcactgactagccaaagccttactct
gcctgacatcgaccgtaaattgataatgaatttacatgcttccgcgacgatttacctCTTGATCATcg
atccgattgaagatcttcaattgttaattctcttgcctcgactcatagccatgatgagctCTTGATCA
TgtttccttaaccctctatttttttacggaagaATGATCAAGctgctgctCTTGATCATcgtttc
```

Slika 3.6: Početni region replikacije bakterije Vibrio cholerae sa označenim 9-gramima koji se najčešće pojavljuju [12]

Ono što ih čini posebno interesantnim jeste činjenica da se oba pojavljuju više puta i to u neposrednoj blizini, unutar istog kratkog regiona genoma. Važno je naglasiti da ovakva pojava nije verovatna u slučajnoj DNK sekvenci. Verovatnoća da se isti 9-mer pojavi više puta u kratkom segmentu genoma je veoma mala. Na primer, verovatnoća da se neki 9-mer pojavi tri ili više puta u slučajno generisanom DNK nizu dužine 500 iznosi približno 1/1300. Zbog toga se ovakvi obrasci smatraju statistički značajnim, a upravo takvi grupisani motivi predstavljaju *DnkA* boksove.

Važno je naglasiti da *DnkA* boksovi nisu univerzalni i da je nukleotidni sastav *DnkA* boksova specifičan za različite organizme. Na primer, analiza *oriC* regiona bakterije *Thermotoga petrophila*, prikazanog na Slici 3.7, pokazuje da se obrasci `ATGATCAAG` i `CTTGATCAT`, koji su karakteristični za *Vibrio cholerae*, u ovom genomu uopšte ne pojavljuju. Umesto toga, izdvajaju se drugačiji često zastupljeni 9-grami, kao što su `AACCTACCA`, `ACCTACCAC`, `CCTACCACC` i drugi slični motivi.

Slika 3.7: Početni region replikacije bakterije Thermotoga petrophila [12]

Da bismo formalizovali problem i započeli sa njegovim rešavanjem, ključna informacija je da se ovi obrasci pojavljuju lokalno grupisano. Zbog toga se uvodi problem pronalaženja grupa (eng. *Clump Finding Problem*), čiji je cilj identifikacija k-grama koji se pojavljuju više puta unutar ograničenog regiona genoma.

3.2.1 Algoritam

Problem pronalaženja grupa formalno se definiše na sledeći način:

- Ulaz: niska *Genome* i celi brojevi k, L i t
- Izlaz: svi različiti k-grami koji formiraju (L, t)-grupe u niski *Genome*

Kažemo da k-gram formira (L, t)-grupu ukoliko postoji podniska genoma dužine L u kojoj se posmatrani k-gram pojavljuje najmanje t puta.

Drugim rečima, traže se sve podsekvence dužine k koje su lokalno učestale u makar jednom segmentu genoma dužine L.

Naivni pristup

Osnovna ideja algoritma zasniva se na korišćenju kliznog prozora dužine L koji se pomera duž cele sekvence *Genome*. Za svaki položaj prozora posmatra se odgovarajući segment genoma, određuju se frekvencije svih k-grama u njemu, a zatim proverava da li se neki od njih pojavljuje najmanje t puta.

Ako određeni k-gram zadovoljava ovaj uslov u makar jednom prozoru, smatra se da formira grupu i uključuje se u skup rezultata.

Najdirektniji pristup rešavanju ovog problema jeste da se za svaki prozor dužine L iznova izračuna niz frekvencija svih k-grama. Nakon toga se prolazi kroz sve moguće indekse od 0 do $4^k - 1$ i proverava da li odgovarajući k-gram ima frekvenciju veću ili jednaku t.

Algoritam 10 prikazuje ovu osnovnu varijantu.

Algoritam 10: Clump Finding

```
Funkcija ClumpFinding(Genome, k, L, t)
    FrequentPatterns ← prazan skup
    for i ← 0 to |Genome| − L do
        Text ← Genome[i..i + L − 1]
        FrequencyArray ← ComputingFrequencies(Text, k)
        for index ← 0 to 4^k − 1 do
            if FrequencyArray[index] ≥ t then
                Pattern ← NumberToPattern(index, k)
                dodaj Pattern u FrequentPatterns
            end
        end
    end
    return FrequentPatterns
```

Primer

Pretpostavimo da se duž genoma pomera prozor dužine $L = 500$ i da se u svakom takvom segmentu posmatraju 9-grami, odnosno da je $k = 9$. Ukoliko se određeni 9-gram pojavi najmanje $t = 3$ puta unutar nekog prozora, tada kažemo da on formira $(500, 3)$-grupu.

Na primer, ukoliko se obrazac `ATGATCAAG` tri puta pojavi unutar jednog segmenta dužine 500 nukleotida, taj obrazac predstavlja kandidat za biološki značajnu sekvencu. Slično važi i za njegov reverzni komplement. Upravo ovakvi lokalno grupisani obrasci predstavljaju signal koji može ukazivati na prisustvo *oriC* regiona.

3.2.2 Vremenska složenost naivnog pristupa

Vremenska složenost određuje se na osnovu broja prozora i broja operacija koje se izvršavaju unutar svakog od njih.

Broj prozora dužine L u niski $Genome$ jednak je:

$$|Genome| - L + 1$$

Za svaki prozor najpre se računa niz frekvencija. Funkcija ***ComputingFrequencies*** zahteva inicijalizaciju niza dimenzije 4^k, što ima složenost $O(4^k)$, a zatim prolazak kroz prozor dužine L, pri čemu se za svaki od približno L k-grama vrši transformacija pomoću funkcije ***PatternToNumber***. Kako ova transformacija obrađuje k simbola, njen trošak je $O(k)$, pa ukupno računanje niza frekvencija zahteva:

$$O(4^k + L \cdot k)$$

Kombinovanjem ovih koraka dobija se da je ukupna složenost jedne iteracije prozora:

$$O(4^k + L \cdot k)$$

Pošto se takva obrada vrši za svih $|Genome| - L + 1$ prozora, ukupna vremenska složenost naivnog algoritma iznosi:

$$O((|Genome| - L + 1) \cdot (4^k + L \cdot k))$$

Kako je uobičajeno da važi $L \ll |Genome|$, broj prozora $(|Genome| - L + 1)$ može se aproksimirati sa $|Genome|$, pa se vremenska složenost često zapisuje i kao

$$O(|Genome| \cdot (4^k + L \cdot k)).$$

Za velike genome ovaj pristup postaje veoma skup, jer se za svaki prozor iznova računa niz frekvencija bez korišćenja prethodnih rezultata. U tipičnom slučaju bakterijskih genoma, gde je $|Genome|$ veoma veliko, a k i L fiksirani, ova složenost je i dalje previsoka za efikasnu obradu.

3.2.3 Optimizacija kliznog prozora

Glavni nedostatak naivnog pristupa jeste to što se niz frekvencija pri svakom pomeranju prozora računa iz početka, iako se između dva uzastopna prozora menjaju frekvencije za veoma mali broj k-grama.

Kada se prozor pomeri za jednu poziciju udesno:

- jedan k-gram izlazi iz prozora,

- jedan novi k-gram ulazi u prozor.

Sve ostale frekvencije ostaju nepromenjene. To znači da nije potrebno iznova računati ceo niz frekvencija, već je dovoljno ažurirati samo dve njegove vrednosti:

- smanjiti frekvenciju k-grama koji izlazi iz prozora,
- povećati frekvenciju k-grama koji ulazi u prozor.

Na taj način dobija se efikasniji algoritam koji koristi isti princip kliznog prozora, ali izbegava redundantna izračunavanja.

Optimizovani algoritam

Algoritam 11 prikazuje optimizovanu verziju pronalaženja grupa. U prvom koraku računa se niz frekvencija za početni prozor dužine L, a zatim se pri svakom pomeranju prozora ažuriraju samo vrednosti koje odgovaraju prvom i poslednjem k-gramu u novom položaju prozora.

3.2.4 Vremenska složenost optimizovanog pristupa

Optimizovani algoritam najpre računa niz frekvencija za prvi prozor, što zahteva:

$$O(4^k + L \cdot k)$$

Nakon toga, za svaki naredni prozor vrše se dve transformacije pomoću funkcije ***PatternToNumber*** i dve izmene u nizu frekvencija. Kako svaka transformacija zahteva $O(k)$ vremena, ukupni trošak jedne iteracije iznosi $O(k)$.

S obzirom na to da postoji $|Genome| - L$ pomeranja prozora, ukupni trošak ažuriranja frekvencija iznosi:

$$O((|Genome| - L) \cdot k)$$

Kombinovanjem svih koraka dobija se ukupna vremenska složenost optimizovanog algoritma:

$$O(4^k \cdot k + L \cdot k + (|Genome| - L) \cdot k)$$

Pošto važi

Algoritam 11: Better Clump Finding

```
Funkcija BetterClumpFinding(Genome, k, t, L)
    FrequentPatterns ← prazan skup
    Text ← Genome[0..L − 1]
    FrequencyArray ← ComputingFrequencies(Text, k)
    for i ← 0 to 4^k − 1 do
        if FrequencyArray[i] ≥ t then
            Pattern ← NumberToPattern(i, k)
            dodaj Pattern u FrequentPatterns
        end
    end
    for i ← 1 to |Genome| − L do
        FirstPattern ← Genome[i − 1..i + k − 2]
        indexFirst ← PatternToNumber(FirstPattern)
        FrequencyArray[indexFirst] ← FrequencyArray[indexFirst] − 1
        LastPattern ← Genome[i + L − k..i + L − 1]
        indexLast ← PatternToNumber(LastPattern)
        FrequencyArray[indexLast] ← FrequencyArray[indexLast] + 1
        if FrequencyArray[indexLast] ≥ t then
            Pattern ← NumberToPattern(indexLast, k)
            dodaj Pattern u FrequentPatterns
        end
    end
    return FrequentPatterns
```

$$L \cdot k + (|Genome| - L) \cdot k = |Genome| \cdot k,$$

ukupna složenost može se zapisati kao

$$O(4^k \cdot k + |Genome| \cdot k).$$

Ovaj rezultat predstavlja značajno poboljšanje u odnosu na naivni pristup, jer se niz frekvencija više ne generiše iz početka za svaki prozor, čime se eliminiše multiplikativni faktor L.

3.3 Asimetrija replikacije DNK i *Skew* dijagram

Proces replikacije DNK započinje u specifičnom regionu genoma poznatom kao *oriC*, odakle se replikacija odvija u oba smera duž hromozoma, sve do regiona ozna-

čenog kao *terC* (eng. *termination of chromosomal replication*). Tokom ovog procesa formiraju se dve replikacione viljuške (eng. *replication forks*), duž kojih se replikacija odvija u suprotnim smerovima.

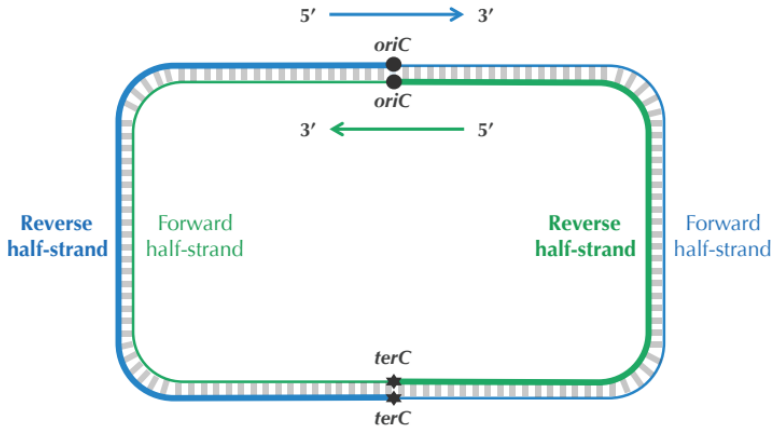
DNK polimeraza sintetiše novi lanac na osnovu postojećeg, ali može da nadovezuje nukleotide isključivo u smeru $5' \rightarrow 3'$. Postavlja se pitanje na koji način se onda vrši sinteza novog lanca duž drugog lanca koji ima usmerenje $3' \rightarrow 5'$. Uslovljena orijentacija sinteze novog lanca dovodi do asimetrije u procesu replikacije i predstavlja osnovu za kasnije nastajanje razlika u raspodeli nukleotida duž genoma, o čemu će biti više reči u nastavku.

3.3.1 Okazaki fragmenti i asimetrija replikacije

Ograničenje da DNK polimeraza može da sintetiše novi lanac isključivo u smeru $5' \rightarrow 3'$ ima značajne posledice na proces replikacije. Iako se oba roditeljska lanca replikuju istovremeno, smer nastajanja duplikata nije isti.

Na osnovu smera replikacije, kružni bakterijski hromozom može se podeliti na dve polovine označene kao zaostajući (eng. *forward half-strand*) i vodeći lanac (eng. *reverse half-strand*). Na vodećem lancu DNK polimeraza može kontinuirano da prati otvaranje DNK molekula i sintetiše novi lanac bez prekida. Nasuprot tome, na zaostajućem lancu smer kretanja DNK polimeraze se razlikuje od orijentacije lanca, pa se sinteza ne može odvijati kontinualno.

Kao što je prikazano na Slici 3.8, hromozom se može podeliti na vodeći i na zaostajući lanac u odnosu na položaj početka replikacije (*oriC*).

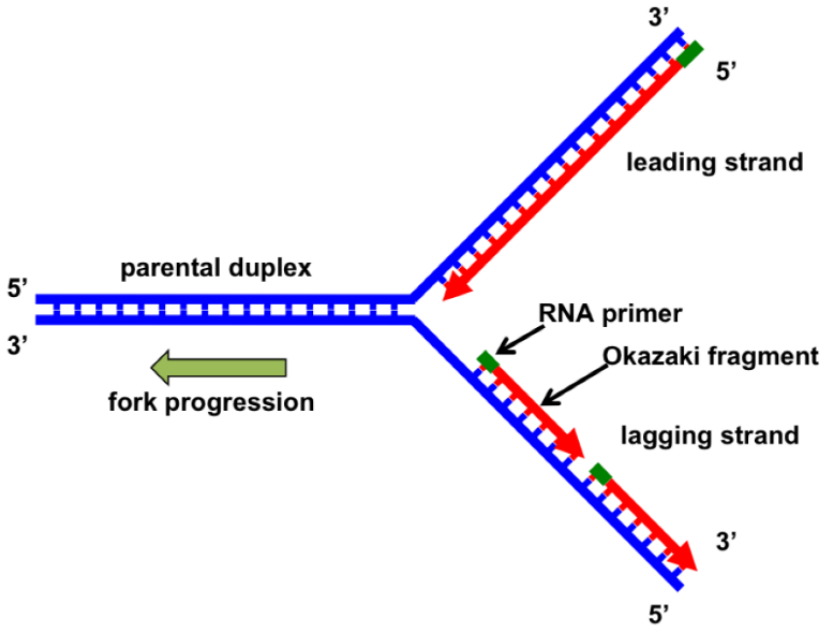


Slika 3.8: Vodeći i zaostajući lanac tokom procesa replikacije DNK [14].

Pošto DNK polimeraza ne može da sintetiše DNK u smeru $3' \rightarrow 5'$, na zaostajućem lancu neophodno je višestruko započinjanje sinteze. Svaki novi segment započinje formiranjem RNK prajmera, odnosno kratkog lanca RNK koji sintetiše enzim primaza i koji obezbeđuje početni $5'$ kraj sa koga DNK polimeraza može započeti sintezu. Ovi kratki DNK segmenti nazivaju se **Okazaki fragmenti**.

Kako se replikaciona viljuška pomera duž DNK molekula, nastaju novi Okazakijevi fragmenti, a DNK polimeraza periodično započinje sintezu novih fragmenata. Po završetku replikacije, RNK prajmeri se uklanjaju, njihovo mesto popunjavaju komplementarnim nukleotidima, a enzim DNK ligaza povezuje susedne fragmente u kontinuirani lanac. Zbog isprekidane sinteze, na zaostajućem lancu potreban je poseban RNK prajmer za svaki novonastali Okazaki fragment. Nasuprot tome, na vodećem lancu sinteza se odvija kontinualno, pa je dovoljan samo jedan RNK prajmer [10].

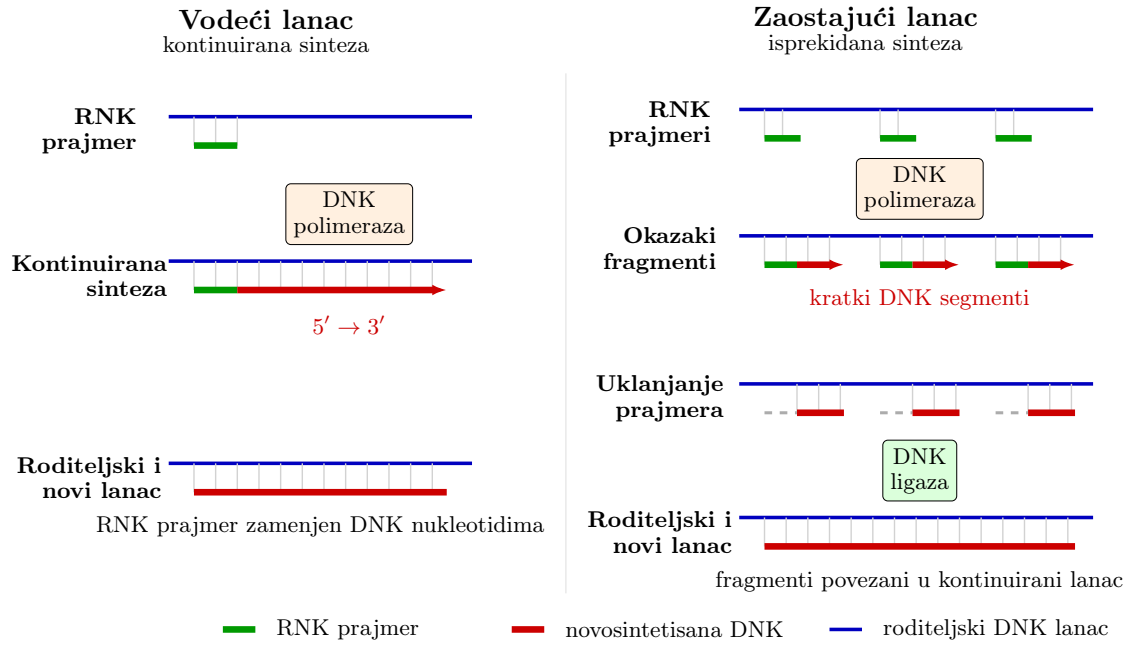
Na Slici 3.9 prikazana je asimetrija replikacije DNK na vodećem i zaostajućem lancu, dok Slika 3.10 prikazuje ilustraciju koraka tokom replikacije.



Slika 3.9: Asimetrija replikacije na vodećem (leading) i zaostajućem (lagging) lancu [13]

Na Slici 3.11 prikazan je proces formiranja Okazaki fragmenata i njihovo povezivanje u kontinuirani DNK lanac.

Zbog stalnog započinjanja i prekidanja sinteze, zaostajući lanac provodi znatno



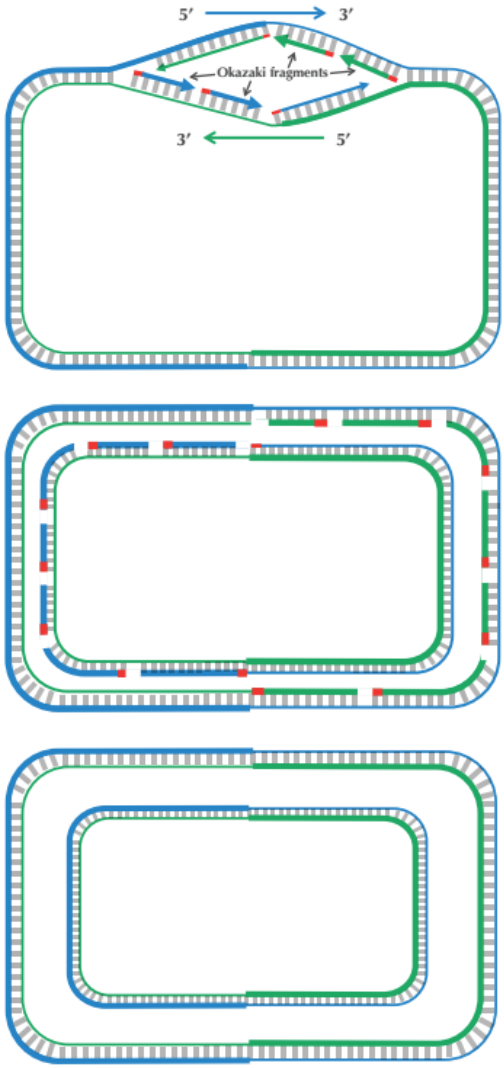
Slika 3.10: Uporedni prikaz asimetrije replikacije DNK na vodećem i zaostajućem lancu. Plavi segment označava roditeljski DNK lanac koji služi kao matrica, zeleni segmenti označavaju RNK prajmere, a crveni segmenti novosintetisanu DNK.

više vremena u jednolančanom stanju od vodećeg lanca. Ova razlika između kontinuirane i isprekidane sinteze predstavlja osnovu za nastanak statističke asimetrije nukleotidnog sastava DNK u *oriC* regionu.

Jednolančana DNK je hemijski nestabilnija i podložnija mutacijama. Posebno je značajan proces deaminacije, odnosno hemijski proces u kojem se iz molekula uklanja amino grupa. U ovom procesu dolazi do transformacije citozina u timin, što doprinosi promenama u nukleotidnom sastavu DNK. Ova mutacija se češće dešava na lancu koji duže ostaje jednolančan. Kao posledica toga, zaostajući lanac vremenom gubi citozin, što dovodi do neravnomerne raspodele nukleotida između dva dela genoma, posebno između baza guanin (G) i citozin (C).

Ova asimetrija u raspodeli nukleotida predstavlja važan signal koji se može iskoristiti za identifikaciju početnog regiona replikacije. Naime, raspodela nukleotida razlikuje se između dve polovine hromozoma usled različitih uslova sinteze vodećeg i zaostajućeg lanca. U regionu *oriC*, gde započinje replikacija, dolazi do promene ovog obrasca raspodele nukleotida, zbog čega se upravo taj region može identifikovati analizom nukleotidne asimetrije.

Da bi se ovaj fenomen formalizovao, uvodi se grafik koji grafički prikazuje razliku između pojavljivanja G i C duž genoma.



Slika 3.11: Nastanak Okazaki fragmenata tokom replikacije i njihovo povezivanje u kontinuirani DNK lanac [14].

3.3.2 Iskrivljeni dijagram

GC-skew predstavlja meru razlike između broja pojavljivanja baza guanin (G) i citozin (C) u prefiksu DNK sekvence.

Formalno, za poziciju i definišemo meru:

$$Skew(i) = \#G - \#C,$$

gde $\#G$ i $\#C$ predstavljaju broj pojavljivanja baza G i C u prvih i nukleotida sekvence.

Početna vrednost funkcije je:

$$Skew(0) = 0$$

Konstrukcija iskrivljenog dijagrama

Iskrivljeni dijagram (eng. *Skew Diagram*) se konstruiše prolaskom kroz sekvencu i inkrementalnim ažuriranjem vrednosti funkcije.

Za svaku poziciju važi:

- ako je baza G → vrednost se povećava za 1
- ako je baza C → vrednost se smanjuje za 1
- ako je baza A ili T → vrednost ostaje nepromenjena

Na ovaj način dobija se niz vrednosti koji se može grafički prikazati, gde x-osa predstavlja poziciju u genomu, a y-osa vrednost funkcije $Skew(i)$.

Algoritam 12 prikazuje pseudokod za izračunavanje *GC-skew* vrednosti.

Algoritam 12: Izračunavanje GC-skew

```
Funkcija GC-skew(Genome)
    Skew[0] ← 0
    for i ← 1 to |Genome| do
        Skew[i] ← Skew[i − 1]
        if Genome[i] = G then
            Skew[i] ← Skew[i] + 1
        end
        else if Genome[i] = C then
            Skew[i] ← Skew[i] − 1
        end
    end
    return Skew
```

Interpretacija rezultata

Vrednost funkcije $Skew(i)$ pokazuje karakterističan obrazac prilikom prolaska kroz genom. U jednom delu genoma vrednosti opadaju, dok u drugom počinju da rastu.

Tačka u kojoj funkcija dostiže minimum predstavlja mesto promene ovog trenda i u zapravo odgovara početnom regionu replikacije ($oriC$). Intuitivno, minimum

označava prelaz između dva dela genoma sa različitim statističkim osobinama, što je direktna posledica asimetrije procesa replikacije.

Problem minimalnog *skew*-a

Na osnovu prethodne analize može se definisati problem pronalaženja minimalne vrednosti funkcije $Skew(i)$, koji predstavlja osnovu za algoritamsko određivanje lokacije *oriC*. Cilj je pronaći sve pozicije u genomu na kojima funkcija dostiže svoju minimalnu vrednost, jer te pozicije predstavljaju kandidate za početni region replikacije (*oriC*). Problem minimalnog *skew*-a formalno se definiše na sledeći način:

- **Ulaz:** DNK sekvenca *Genome*.
- **Izlaz:** sve pozicije i za koje je vrednost $Skew(i)$ minimalna.

Primer

Na Slici 3.12 prikazan je *GC-skew* dijagram za genom bakterije *Escherichia coli*.

Uočava se karakterističan obrazac u kome vrednost funkcije najpre raste, zatim opada i dostiže minimum, nakon čega ponovo počinje da raste. Ova promena trenda odražava prelaz između dva dela genoma sa različitim statističkim osobinama.

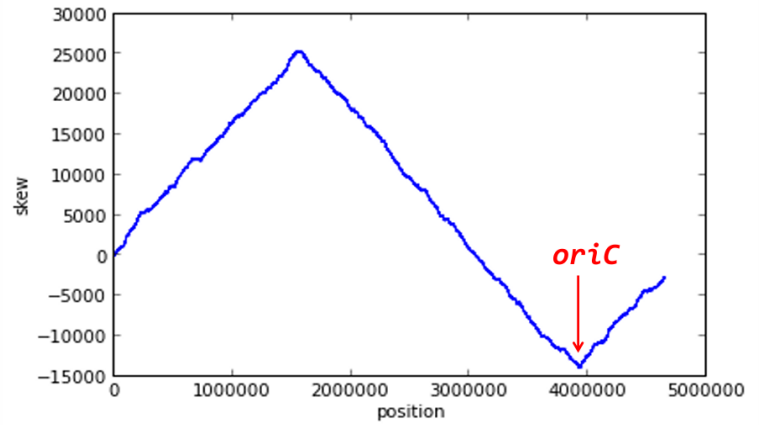
Minimum funkcije dostiže se približno na poziciji od oko 4 miliona nukleotida i predstavlja kandidat za lokaciju početnog regiona replikacije (*oriC*). Razlog za to je što upravo na tom mestu dolazi do promene između zaostajućeg i vodećeg lanca, odnosno do promene u raspodeli baza G i C.

Takođe, u suprotnom delu genoma funkcija dostiže maksimum, što odgovara regionu približno nasuprot *oriC* na kružnom hromozomu. Ova simetrična struktura dijagrama posledica je same prirode replikacije, koja se odvija u oba smera od početne tačke.

3.3.3 Vremenska i prostorna složenost

Algoritam za izračunavanje *GC-skew* vrednosti zasniva se na jednom prolazu kroz DNK sekvencu. Za svaku poziciju u genomu vrši se ažuriranje vrednosti funkcije $Skew(i)$ na osnovu prethodne vrednosti, pri čemu se proverava samo trenutni nukleotid.

U svakom koraku izvršava se konstantan broj operacija, što znači da složenost po jednoj iteraciji iznosi $O(1)$.



Slika 3.12: GC-skew dijagram za genom bakterije Escherichia coli [12]

Kako se ovaj postupak ponavlja za svih $|Genome|$ pozicija u sekvenci, ukupna vremenska složenost algoritma iznosi:

$$O(|Genome|)$$

Pored toga, potrebno je memorisati niz vrednosti dužine $|Genome|$, pa je prostorna složenost takođe:

$$O(|Genome|).$$

3.3.4 Prednosti i ograničenja

Jedna od glavnih prednosti ovog pristupa jeste njegova jednostavna implementacija, kao i činjenica da se zasniva na jednom prolazu kroz genom, što omogućava efikasno izračunavanje čak i za veoma duge sekvence. Dodatno, metoda ne zahteva unapred poznate obrasce u DNK, što je čini pogodnom za analizu genoma kod kojih relevantni motivi još uvek nisu identifikovani.

Sa druge strane, *GC-skew* analiza ima i određena ograničenja. Pre svega, ona se oslanja na statističke osobine koje mogu varirati između različitih organizama, pa ne garantuje uvek precizno određivanje lokacije *oriC* regiona. Takođe, uspešnost metode zavisi od izraženosti asimetrije u raspodeli nukleotida, koja u nekim genomima može biti slabo naglašena. Zbog toga se ovaj pristup, u praksi, najčešće koristi u kombinaciji sa algoritmima zasnovanim na analizi podniski, čime se postiže pouzdanija identifikacija biološki značajnih regiona.

3.4 Približno poklapanje sekvenci i česte reči sa propustima

U prethodnim poglavljima razmatrani su algoritmi zasnovani na tačnom poklapanju podniski u DNK sekvenci. Takav pristup podrazumeva da se funkcionalno značajni obrasci pojavljuju u potpuno identičnom obliku. Međutim, u realnim biološkim sistemima ova pretpostavka često nije zadovoljena zato što su DNK sekvence podložne mutacijama. Tada dolazi do zamene pojedinačnih nukleotida, pa se isti biološki motiv može pojaviti u više varijanti koje se razlikuju na manjem broju pozicija, ali i dalje zadržavaju svoju funkciju.

Zbog toga algoritmi zasnovani isključivo na tačnom poklapanju mogu propustiti značajne obrasce. Da bi se ovaj problem prevazišao, uvodi se model približnog poklapanja sekvenci, koji omogućava poređenje uz dozvoljeni broj odstupanja. Ovaj model predstavlja prirodno proširenje prethodnih pristupa i čini osnovu za rešavanje problema čestih reči sa propustima.

3.4.1 Hamingovo rastojanje

Osnovna mera razlike između sekvenci jednake dužine jeste Hamingovo rastojanje (eng. *Hamming distance*). Za dve sekvence p i q dužine k, Hamingovo rastojanje definiše se kao broj pozicija na kojima se one razlikuju.

Ako su date sekvence

$$p = p_1 p_2 \dots p_k \quad \text{i} \quad q = q_1 q_2 \dots q_k,$$

onda se Hamingovo rastojanje može formalno zapisati kao:

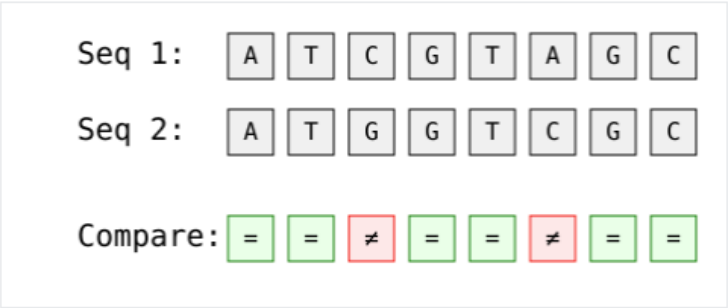
$$HammingDistance(p, q) = |\{i \mid p_i \neq q_i\}|.$$

Na primeru sa Slike 3.13, sekvence `ATCGTAGC` i `ATGGTCGC` razlikuju se u dve pozicije, pa je njihovo Hamingovo rastojanje jednaka 2.

Ova mera predstavlja osnovni alat za procenu sličnosti između podniski. Što je Hamingovo rastojanje manje, to su dve sekvence međusobno sličnije.

Algoritam za računanje Hamingovog rastojanja

Najjednostavniji način računanja Hamingovog rastojanja zasniva se na poređenju odgovarajućih karaktera u dve sekvence iste dužine. Svaki put kada se simboli razlikuju, brojač se uvećava za 1. Algoritam 13 prikazuje pseudokod ove funckije.



Slika 3.13: Poređenje dve DNK sekvence sa naglašenim razlikama po pozicijama [11]

Algoritam 13: Hamming Distance

```
Funkcija HammingDistance(p, q)
    d ← 0
    for i ← 1 to |p| do
        if p_i ≠ q_i then
            d ← d + 1
        end
    end
    return d
```

Ovaj algoritam prolazi kroz obe sekvence samo jednom i pri svakoj iteraciji proverava da li se simboli na istoj poziciji razlikuju. Zbog toga, njegova vremenska složenost je linearna u odnosu na dužinu sekvence.

3.4.2 Približno poklapanje obrazaca

Na osnovu Hamingovog rastojanja definiše se problem približnog poklapanja obrazaca (eng. *Approximate Pattern Matching*). Za razliku od tačnog poklapanja, gde se zahteva potpuno podudaranje sekvenci, ovde se dozvoljava ograničen broj odstupanja.

Problem približnog poklapanja obrazaca formalno se definiše na sledeći način:

- **Ulaz:** niske $Text$ i $Pattern$, i ceo broj d
- **Izlaz:** sve pozicije u $Text$ na kojima se $Pattern$ pojavljuje sa najviše d odstupanja

Drugim rečima, traže se sve podniske dužine $|Pattern|$ čije je Hamingovo rastojanje od obrasca $Pattern$ manja ili jednaka d.

Algoritam za približno poklapanje

Najjednostavniji pristup rešavanju ovog problema zasniva se na kliznom prozoru. Prozor dužine $|Pattern|$ pomera se kroz sekvencu $Text$, a za svaku poziciju računa se Hamingovo rastojanje između obrasca i odgovarajuće podniske. Algoritam 14 prikazuje pseudokod ove funcije.

Algoritam 14: Approximate Pattern Matching

```
Funkcija ApproximatePatternMatching(Text, Pattern, d)
    positions ← prazan skup
    for i ← 0 to |Text| − |Pattern| do
        Pattern′ ← Text[i..i + |Pattern| − 1]
        if HammingDistance(Pattern, Pattern′) ≤ d then
            dodaj i u positions
        end
    end
    return positions
```

Objašnjenje rada algoritma

Algoritam sistematski razmatra sve podniske dužine $|Pattern|$ u tekstu. Za svaku od njih vrši se poređenje sa obrascem karakter po karakter, pri čemu se prebrojava broj pozicija na kojima se razlikuju.

Ukoliko broj razlika ne prelazi zadatu vrednost d, smatra se da na toj poziciji postoji približno poklapanje, pa se indeks te pozicije beleži u skup rezultata.

Važno je naglasiti da se i u ovom slučaju uzimaju u obzir preklapajuća pojavljivanja, jer se klizni prozor pomera za jednu poziciju u svakom koraku.

Primer

Posmatrajmo DNK sekvencu:

`AACAAGCATAAACATTAAAGAG`

i obrazac:

`AAAAA`,

uz dozvoljeno $d = 2$.

Algoritam koristi klizni prozor dužine 5 i za svaku poziciju i posmatra podnisku $Text[i..i+4]$, nakon čega računa Hamingovo rastojanje (d_H) u odnosu na obrazac.

Na primer:

- Za $i = 0$: podniska je `AACAA`. Poređenjem sa `AAAAA` dobijamo razliku na jednoj poziciji, pa je $d_H = 1$ i ova pozicija se prihvata.
- Za $i = 1$: podniska je `ACAAG`. Razlikuje se u dve pozicije, pa je $d_H = 2$ i ova pozicija se prihvata.
- Za $i = 2$: podniska je `CAAGC`. Razlikuje se u više od dve pozicije, pa se odbacuje.
- Za $i = 8$: podniska je `AAAAC`. Razlikuje se u jednoj poziciji, pa je $d_H = 1$ i ova pozicija se prihvata.

Na ovaj način algoritam prolazi kroz celu sekvencu i vraća sve indekse na kojima se pojavljuje približno poklapanje, odnosno gde je Hamingovo rastojanje manje ili jednako d.

3.4.3 Prebrojavanje približnih pojavljivanja

Pored pronalaženja pozicija na kojima se obrazac pojavljuje sa odstupanjima, često je potrebno izračunati i ukupan broj takvih pojavljivanja. Zbog toga se uvodi funkcija:

$$ApproximatePatternCount(Text, Pattern, d),$$

koja vraća broj podniski u tekstu čije je Hamingovo rastojanje od obrasca $Pattern$ manje ili jednako d.

Algoritam 15: Approximate Pattern Count

```
Funkcija ApproximatePatternCount(Text, Pattern, d)
    count ← 0
    for i ← 0 to |Text| − |Pattern| do
        Pattern′ ← Text[i..i + |Pattern| − 1]
        if HammingDistance(Pattern, Pattern′) ≤ d then
            count ← count + 1
        end
    end
    return count
```

Pseudokod je prikazan u Algoritmu 15 i veoma je sličan prethodnom, ali se umesto čuvanja svih pozicija računa samo brojač. Svaki put kada se pronađe podniska koja se razlikuje u najviše d pozicija, brojač se povećava za 1.

Na kraju, algoritam vraća ukupan broj približnih pojavljivanja, odnosno vrednost funkcije $ApproximatePatternCount$.

3.4.4 Vremenska složenost približnog poklapanja

Algoritam za približno poklapanje prolazi kroz sve podniske dužine $|Pattern|$, kojih ima $|Text| - |Pattern| + 1$. Za svaku od njih računa se Hamingovo rastojanje, što zahteva $O(|Pattern|)$ operacija.

Zbog toga ukupna vremenska složenost iznosi:

$$O(|Text| \cdot |Pattern|)$$

Ova složenost je linearna u odnosu na dužinu teksta, ali zavisi i od dužine obrasca. Za kraće obrasce algoritam je efikasan, dok za duže sekvence ili veliki broj poređenja može postati zahtevniji.

3.4.5 Problem čestih reči sa propustima

Približno poklapanje i približno prebrojavanje pojavljivanja obrazaca predstavljaju osnovu za definisanje složenijeg problema. Sada cilj nije pronaći poklapanje sa datim obrascem, već pronaći najčešće obrasce kada su dozvoljena odstupanja.

Drugim rečima, ako za dati obrazac možemo da izračunamo koliko puta se pojavljuje u tekstu sa najviše d razlika, tada možemo pokušati da pronađemo one obrasce koji maksimizuju ovu vrednost. Na taj način dolazimo do problema čestih reči sa propustima (eng. *Frequent Words with Mismatches*).

Ovaj problem se može definisati na sledeći način:

- **Ulaz:** niska $Text$, dužina k i maksimalan broj dozvoljenih odstupanja d
- **Izlaz:** svi k-grami koji maksimizuju vrednost funkcije $ApproximatePatternCount(Text, Pattern, d)$

Za razliku od standardnog problema čestih reči, ovde frekvencija obrasca ne zavisi od broja tačnih poklapanja, već od broja približnih poklapanja. To znači da obrazac sa propustima koji je najčešći ne mora nužno da se pojavljuje u tekstu

u svom originalnom obliku. Upravo to čini ovaj problem složenijim i istovremeno biološki značajnijim.

3.4.6 d-susedstvo obrasca

Kod problema tačnih poklapanja, prilikom razmatranje kandidata za motiv (eng. *pattern*), bilo je dovoljno posmatrati one k-grame koji se zaista pojavljuju u tekstu. Ovde to više nije slučaj. Razlog za to je što se k-gram koji ima najveći broj približnih pojavljivanja ne mora pojaviti u tekstu. Moguće je da se više podniski iz teksta razlikuje od nekog zajedničkog motiva u jednoj ili dve pozicije, i da upravo taj „centralni“ motiv ima najveću približnu frekvenciju, iako se sâm nikada ne pojavljuje u svom originalnom obliku. Zbog toga je potrebno razmotriti ne samo postojeće podniske, već i njihove varijacije.

Za dati obrazac $Pattern$, njegovo d-susedstvo, označeno sa $Neighbors(Pattern, d)$, predstavlja skup svih sekvenci iste dužine koje se razlikuju od $Pattern$ u najviše d pozicija.

Na primer, ako je dozvoljeno jedno odstupanje, tada susedstvo nekog obrasca sadrži sve sekvence koje se od njega razlikuju u najviše jednoj poziciji. Kako broj dozvoljenih odstupanja raste, susedstvo postaje sve veće.

Pojam susedstva je ključan zato što omogućava da se pretraga ograniči na skup kandidata koji su biološki smisleni, odnosno na sekvence koje su dovoljno bliske onima koje se zaista pojavljuju u tekstu.

Generisanje d-susedstva

Funkcija `Neighbors(Pattern, d)` može se implementirati rekurzivno. Osnovna ideja rekurzije zasniva se na tome da se prvo generišu „direktni” susedi sufiksa obrasca, odnosno oni na rastojanju 1, a zatim se u zavisnosti od njihovog Hamingovog rastojanja dodaje odgovarajući simbol na početak. Na ovaj način se korak po korak generišu sve sekvence koje zadovoljavaju uslov da se razlikuju od početnog obrasca u najviše d pozicija. Pseudokod prikazan je u Algoritmu 16.

Objašnjenje rada funkcije *Neighbors*

Ako je broj dozvoljenih odstupanja jednak nuli, jedini sused obrasca jeste sam obrazac. Ako je dužina obrasca jednaka 1, tada su svi simboli iz alfabeta mogući susedi.

Algoritam 16: Neighbors

```
Funkcija Neighbors(Pattern, d)
    if d = 0 then
        return {Pattern}
    end
    if |Pattern| = 1 then
        return {A, C, G, T}
    end
    Neighborhood ← prazan skup
    SuffixNeighbors ← Neighbors(Suffix(Pattern), d)
    foreach Text ∈ SuffixNeighbors do
        if HammingDistance(Suffix(Pattern), Text) < d then
            foreach x ∈ {A, C, G, T} do
                dodaj x · Text u Neighborhood
            end
        end
        else
            dodaj FirstSymbol(Pattern) · Text u Neighborhood
        end
    end
    return Neighborhood
```

U opštem slučaju najpre se rekurzivno generišu susedi maksimalnog sufiksa obrasca, koji se dobija uklanjanjem prvog karaktera. Zatim se za svaki takav sused proverava da li je njegovo Hamingovo rastojanje od originalnog sufiksa strogo manje od d. Ako jeste, na početak se može dodati bilo koji simbol, jer još uvek postoji prostor za dodatnu razliku. U suprotnom, mora se zadržati originalni prvi simbol, kako ukupan broj odstupanja ne bi prešao dozvoljenu vrednost.

3.4.7 Algoritam za česte reči sa propustima

Jedan od efikasnijih pristupa za rešavanje problema čestih reči sa propustima zasniva se na tome da se ne razmatra svih 4^k mogućih obrazaca, već samo oni koji pripadaju susedstvu nekog k-grama iz teksta.

U tu svrhu koriste se nizovi `Close` i `FrequencyArray`. Niz `Close` sadrži identifikatore kandidata, dok `FrequencyArray` čuva broj njihovih približnih pojavljivanja. Algoritam 17 prikazuje pseudokod ove funkcije *Frequent Words with Mismatches*.

Algoritam 17: Frequent Words with Mismatches

```
Funkcija FrequentWordsWithMismatches(Text, k, d)
    FrequentPatterns ← prazan skup
    for i ← 0 to 4^k − 1 do
        Close[i] ← 0
        FrequencyArray[i] ← 0
    end
    for i ← 0 to |Text| − k do
        Neighborhood ← Neighbors(Text[i..i + k − 1], d)
        foreach Pattern ∈ Neighborhood do
            index ← PatternToNumber(Pattern)
            Close[index] ← 1
        end
    end
    for i ← 0 to 4^k − 1 do
        if Close[i] = 1 then
            Pattern ← NumberToPattern(i, k)
            FrequencyArray[i] ←
              ApproximatePatternCount(Text, Pattern, d)
        end
    end
    maxCount ← maksimalna vrednost u FrequencyArray
    for i ← 0 to 4^k − 1 do
        if FrequencyArray[i] = maxCount then
            Pattern ← NumberToPattern(i, k)
            dodaj Pattern u FrequentPatterns
        end
    end
    return FrequentPatterns
```

Objašnjenje rada algoritma

U prvom delu algoritma inicijalizuju se pomoćni nizovi. Nakon toga prolazi se kroz sve k-grame u tekstu, i za svaki od njih generiše se odgovarajuće d-susedstvo. Svaki obrazac iz tog susedstva označava se kao potencijalni kandidat.

U drugom delu algoritma prolazi se kroz sve moguće indekse. Ako je neki obrazac označen kao kandidat, za njega se računa vrednost funkcije *ApproximatePatternCount*, odnosno broj približnih pojavljivanja u tekstu.

Na kraju se pronalazi maksimalna vrednost u nizu frekvencija i izdvajaju se svi obrasci koji postižu tu vrednost.

Važno je uočiti da se funkcija `ApproximatePatternCount` ne poziva za svih 4^k mogućih k-grama, već samo za one koji pripadaju susedstvu nekog obrasca iz teksta. Time se postiže značajna ušteda u odnosu na iscrpnu pretragu.

3.4.8 Uzimanje u obzir obrnutih komplementa

U analizi DNK sekvenci nije dovoljno posmatrati samo jednu orijentaciju obrasca. Sekvenca i njen obrnuti komplement mogu imati istu biološku funkciju zato što se proteini mogu vezivati za motive prisutne na oba komplementarna lanca.

Zbog toga, ovaj problem se može dodatno proširiti tako da se za svaki obrazac posmatra i njegov obrnuti komplement. U tom slučaju frekvencija obrasca računa se kao zbir:

$$ApproximatePatternCount(Text, Pattern, d)$$

$$+$$

$$ApproximatePatternCount(Text, ReverseComplement(Pattern), d)$$

Na taj način se objedinuju obe moguće orijentacije iste informacije zapisane u DNK.

Problem čestih reči sa propustima i obrnutim komplementima

Kada se u obzir uzmu i obrnuti komplementi, problem se može definisati na sledeći način:

- **Ulaz:** DNK sekvenca $Text$, dužina k i maksimalan broj odstupanja d
- **Izlaz:** svi k-grami koji maksimizuju zbir približnih frekvencija svog oblika i svog obrnutog komplementa

Ova formulacija predstavlja realističniji model biološkog problema, jer uzima u obzir i prirodnu varijabilnost sekvenci i njihovu komplementarnost.

3.4.9 Vremenska složenost

Najveći izazov ovog pristupa jeste generisanje d-susedstva, što zavisi od njegove dimenzije. Za dati obrazac dužine k, broj suseda raste sa povećanjem broja dozvoljenih odstupanja d. Taj broj se može precizno izraziti formulom:

$$\sum_{i=0}^{d} \binom{k}{i} \cdot 3^i,$$

jer se za svaku od i izmenjenih pozicija bira jedna od tri moguće alternativne baze.

Zbog toga, generisanje d-susedstva za sve k-grame u tekstu, kojih ima približno $|Text| - k + 1$, zahteva značajan broj operacija. Ukupan broj generisanih kandidata reda je veličine:

$$O(|Text| \cdot 3^d \cdot k^d)$$

Dodatni trošak nastaje u drugoj fazi algoritma, gde se za svakog kandidata računa vrednost funkcije `ApproximatePatternCount`. Neka je broj različitih kandidata označen sa C. Kako funkcija `ApproximatePatternCount` ima vremensku složenost $O(|Text| \cdot k)$, ukupni trošak ove faze iznosi:

$$O(C \cdot |Text| \cdot k),$$

gde važi:

$$C \leq \min(4^k,\ |Text| \cdot 3^d \cdot k^d).$$

Pored toga, algoritam prolazi kroz nizove `Close` i `FrequencyArray` dužine 4^k, pri čemu rekonstrukcija odgovarajućih obrazaca pomoću funkcije `NumberToPattern` zahteva ukupno:

$$O(4^k \cdot k).$$

Kombinovanjem svih koraka dobija se ukupna vremenska složenost:

$$O(|Text| \cdot 3^d \cdot k^d + C \cdot |Text| \cdot k + 4^k \cdot k).$$

Ovaj rezultat pokazuje da algoritam ima visoku vremensku složenost, koja brzo raste sa povećanjem parametara k i d. Iako pristup zasnovan na susedstvu značajno smanjuje broj kandidata u odnosu na potpuno iscrpnu pretragu, problem i dalje ostaje računarski zahtevan.

3.4.10 Optimizovana implementacija za česte reči sa propustima

Prethodno opisani pristup za rešavanje problema čestih reči sa propustima zasniva se na korišćenju nizova dimenzije 4^k, što može predstavljati značajan nedostatak u pogledu memorijske efikasnosti. Kako vrednost k raste, broj svih mogućih k-grama eksponencijalno se povećava, pa čuvanje informacija za svaki od njih postaje neprakticno.

Da bi se ovaj problem ublažio, može se primeniti optimizovani pristup zasnovan na korišćenju heš mape. Umesto da se unapred razmatraju svi mogući obrasci, u memoriji se čuvaju samo oni koji se pojavljuju u d-susedstvu nekog k-grama iz posmatranog teksta.

Ideja je da se prolazi kroz tekst i za svaki izdvojeni k-gram generiše njegovo d-susedstvo. Za svaki obrazac iz tog susedstva proverava se da li je već obrađen. Ako nije, izračunava se broj njegovih približnih pojavljivanja u tekstu i čuva u rečniku. Ključ u rečniku predstavlja obrazac, dok vrednost predstavlja broj njegovih približnih pojavljivanja.

Na ovaj način svaki kandidat se obrađuje najviše jednom, čime se izbegava nepotrebno ponavljanje izračunavanja, kao i čuvanje frekvencija za obrasce koji se uopšte ne pojavljuju kao relevantni kandidati.

Algoritam

Algoritam prolazi kroz tekst, generiše d-susedstvo za svaki k-gram i za svakog suseda računa približnu frekvenciju samo ako on već nije evidentiran u rečniku. Na kraju se iz rečnika izdvajaju svi obrasci koji imaju maksimalnu frekvenciju i oni predstavljaju najzastupljenije k-grame sa dozvoljenim odstupanjima [15]. Pseudokod je prikazan u Algoritmu 18.

Prednosti pristupa

Glavna prednost ovog pristupa je smanjenje memorijske složenosti, jer se izbegava korišćenje nizova dimenzije 4^k. Umesto toga, čuvaju se samo oni obrasci koji su zaista relevantni kandidati.

Pored toga, izbegava se i prolazak kroz sve moguće obrasce, pa se fokus stavlja isključivo na obrasce koji se javljaju u tekstu i njihove susede. Iako dominantni deo vremenske složenosti ostaje vezan za generisanje d-susedstva i računanje funkcije

Algoritam 18: Frequent Words with Mismatches (Dictionary)

```
Funkcija FrequentWordsWithMismatchesDictionary(Text, k, d)
    FrequentPatterns ← prazan skup
    FrequencyMap ← prazna heš mapa
    for i ← 0 to |Text| − k do
        Pattern ← Text[i..i + k − 1]
        Neighborhood ← Neighbors(Pattern, d)
        foreach Neighbor ∈ Neighborhood do
            if Neighbor nije ključ u FrequencyMap then
                FrequencyMap[Neighbor] ←
                 ApproximatePatternCount(Text, Neighbor, d)
            end
        end
    end
    maxCount ← maksimalna vrednost u FrequencyMap
    foreach Pattern u FrequencyMap do
        if FrequencyMap[Pattern] = maxCount then
            dodaj Pattern u FrequentPatterns
        end
    end
    return FrequentPatterns
```

ApproximatePatternCount za različite kandidate, ovaj pristup je u praksi memorijski efikasniji i jednostavniji za implementaciju.

3.5 Ograničenja i otvoreni problemi

U prethodnim poglavljima predstavljeni su algoritmi za analizu DNK sekvenci zasnovani na učestalosti podniski, njihovom lokalnom grupisanju i dozvoljenim odstupanjima. Iako ovi pristupi pružaju značajne uvide i omogućavaju identifikaciju potencijalno važnih obrazaca, njihova primena u realnim biološkim podacima otkriva niz ograničenja.

U nastavku ćemo razmotriti dodatne aspekte i otvorene probleme koji proizlaze iz ovih metoda.

3.5.1 Ograničenja frekvencijskih pristupa

Algoritmi poput *Frequent Words* i *Frequent Words with Mismatches* zasnivaju se na pretpostavci da su biološki značajni motivi ujedno i najčešće zastupljeni u

sekvenci. Međutim, u praksi ova pretpostavka nije uvek validna.

U dugim DNK sekvencama određeni k-grami mogu se pojavljivati često isključivo kao posledica slučajnosti. Sa druge strane, funkcionalno važni motivi mogu biti prisutni u manjem broju kopija ili sa varijacijama, zbog čega ih frekvencijski pristupi ne detektuju pouzdano.

Uvođenjem dozvoljenih odstupanja povećava se fleksibilnost algoritma, ali istovremeno dolazi do značajnog povećanja broja kandidata, što može dovesti do pojave velikog broja lažno pozitivnih rezultata.

Kombinatorna složenost i praktični izazovi

Jedan od glavnih izazova jeste velika dimenzija prostora pretrage, budući da broj mogućih k-grama iznosi 4^k. Ova kombinatorna eksplozija čini problem računarski zahtevnim, posebno za veće vrednosti parametara k i d, prilikom generisanja d-susedstva. Iako optimizacije, poput korišćenja niza frekvencija i ograničavanja na relevantne kandidate, smanjuju broj operacija, algoritmi i dalje mogu biti spori za velike genomske sekvence.

Statistički aspekti i verovatnoća obrazaca

Važno pitanje koje se prirodno nameće jeste kako razlikovati obrasce koji su biološki značajni od onih koji su nastali slučajno. U tu svrhu neophodno je posmatrati verovatnoću pojavljivanja određenih podniski u sekvenci.

Ukoliko se neki obrazac pojavljuje češće nego što bi se očekivalo na osnovu slučajnog modela, može se smatrati potencijalno značajnim. Međutim, precizno modeliranje ovih verovatnoća predstavlja dodatni izazov i zahteva statističke metode koje prevazilaze osnovne algoritamske pristupe.

3.5.2 Struktura DNK i dodatni biološki faktori

Analiza DNK sekvenci ne zavisi samo od frekvencije podniski, već i od njihove biološke interpretacije. Na primer, značajnu ulogu imaju:

- komplementarnost DNK lanaca i postojanje obrnutih komplementa,
- pravac (5' $\rightarrow$ 3') u kome se sekvence čitaju,
- preklapanje podniski, koje može uticati na statističku raspodelu obrazaca.

Ovi faktori dodatno komplikuju analizu i ukazuju na to da algoritamski rezultati moraju biti interpretirani u biološkom kontekstu.

Otvoreni problemi u pronalaženju motiva

Jedan od najvažnijih otvorenih problema jeste pronalaženje skrivenih motiva u DNK sekvencama. Za razliku od prethodnih problema, gde se analizirala jedna sekvenca, ovde se često posmatra više sekvenci i traži zajednički obrazac koji se u njima pojavljuje, ali ne nužno u identičnom obliku.

Ovaj problem je složen zato što:

- motivi mogu sadržati mutacije,
- ne pojavljuju se na istim pozicijama,
- mogu biti maskirani šumom u podacima,
- nisu nužno najčešći obrasci u pojedinačnoj sekvenci.

Zbog toga standardni pristupi zasnovani na frekvenciji nisu dovoljni, već je potrebno razvijati naprednije algoritme koji kombinuju različite kriterijume.

Primeri otvorenih problema

U okviru savremene bioinformatike izdvajaju se i specifični problemi vezani za različite organizme:

- identifikacija više početnih regiona replikacije u nekim bakterijama,
- pronalaženje *oriC* regiona kod arheja, gde struktura genoma odstupa od bakterijske,
- analiza složenih eukariotskih genoma, kao što je kvasac, gde su regulatorni mehanizmi znatno kompleksniji.

Ovi problemi ukazuju na to da univerzalno rešenje ne postoji i da metode moraju biti prilagođene konkretnom biološkom sistemu.

Iako predstavljeni algoritmi čine osnovu za analizu DNK sekvenci, njihova ograničenja ukazuju na potrebu za daljim razvojem metoda u bioinformatici. Kombinacija algoritamskih tehnika, statističkih modela i biološkog znanja predstavlja ključ za uspešno rešavanje ovih problema. Zbog toga se savremena istraživanja fokusiraju na

razvoj složenijih pristupa koji mogu pouzdanije identifikovati funkcionalno značajne motive u genomu.

Glava 4

Elektronska lekcija

U ovom poglavlju opisana je elektronska lekcija razvijena kao praktični deo rada. Lekcija je namenjena interaktivnom prikazu algoritama koji se koriste pri analizi DNK sekvenci i pronalaženju potencijalnog početnog regiona replikacije. Pored teorijskog objašnjenja pojmova, aplikacija omogućava korisniku da unese DNK sekvencu, pokrene odgovarajuće algoritme i prati njihove rezultate kroz vizuelni prikaz.

Elektronska lekcija se sastoji iz dve komponente: klijentske i serverske. Klijentska komponenta predstavlja korisnički interfejs aplikacije i implementirana je pomoću ***Next.js*** [3] veb okvira, uz korišćenje biblioteka ***React*** [6] i ***Node.js*** [4]. Serverska komponenta implementirana je u programskom jeziku ***Python*** [5], uz korišćenje ***FastAPI***[1] okvira. Izvorni kod aplikacije nalazi se u javnom *GitHub* [2] repozitorijumu na adresi:

`https://github.com/dunjaci/master-rad`

Aplikacija je javno dostupna preko sledeće adrese:

`https://master-rad-seven.vercel.app`

Serverska komponenta dostupna je preko adrese:

`https://master-rad-rynl.onrender.com/docs`

4.1 Arhitektura aplikacije

Aplikacija je organizovana kao sistem sa odvojenom klijentskom i serverskom komponentom. Klijentska komponenta je zadužena za prikaz stranica, unos podata-

ka, navigaciju kroz lekciju i vizuelizaciju rezultata. Serverska komponenta obrađuje zahteve koje šalje klijentska aplikacija i izvršava algoritamsku logiku.

Ovakva organizacija omogućava jasnu podelu odgovornosti. Korisnički interfejs se razvija nezavisno od serverskog dela, dok se algoritmi i obrada podataka centralizuju u serverskoj komponenti. Na taj način se izbegava dupliranje algoritamske logike u klijentskoj aplikaciji, a rezultati koji se prikazuju korisniku dobijaju se preko definisanih API ruta.

Komunikacija između klijentske i serverske komponente ostvaruje se slanjem HTTP zahteva. Na primer, kada korisnik unese DNK sekvencu i izabere određene parametre, klijentska aplikacija šalje zahtev serverskoj komponenti, koja izvršava odgovarajući algoritam i vraća rezultat u JSON formatu.

4.2 Korišćene tehnologije

Klijentski deo aplikacije implementiran je korišćenjem ***Next.js*** okvira. *Next.js* omogućava izradu *React* aplikacija sa organizovanom strukturom stranica, jednostavnim rutiranjem i podrškom za produkcioni *build*. Za stilizovanje korisničkog interfejsa korišćen je ***Tailwind CSS*** [8], što omogućava jednostavno kreiranje responzivnih i preglednih veb stranica.

Serverski deo aplikacije implementiran je pomoću ***FastAPI*** okvira. *FastAPI* je pogodan za izradu *REST API-ja* jer omogućava jednostavno definisanje ruta, validaciju ulaznih podataka i automatski generisanu dokumentaciju programskog interfejsa. U ovoj aplikaciji serverska komponenta sadrži implementacije algoritama i vraća rezultate klijentskoj komponenti.

Za kontrolu verzija i čuvanje izvornog koda korišćen je ***Git*** sistem, dok je repozitorijum postavljen na ***GitHub*** platformu. Za javno postavljanje aplikacije korišćene su platforme ***Render*** [7] i ***Vercel*** [9]. Serverska komponenta postavljena je na *Render*, dok je klijentska komponenta postavljena na *Vercel*.

4.3 Pokretanje aplikacije u lokalnom okruženju

Aplikacija se može pokrenuti lokalno pokretanjem serverske i klijentske komponente u odvojenim terminalima. Serverska komponenta se nalazi u direktorijumu `backend`, dok se klijentska komponenta nalazi u direktorijumu `frontend`.

4.3.1 Pokretanje serverske komponente

Za pokretanje serverske komponente potrebno je pozicionirati se u direktorijum *backend*. Nakon toga se instaliraju potrebne biblioteke iz datoteke `requirements.txt`. Serverska aplikacija se pokreće pomoću alata *uvicorn*.

```
cd backend
pip install -r requirements.txt
uvicorn main:app --reload --port 8000
```

Nakon pokretanja, serverska komponenta dostupna je na adresi:

```
http://localhost:8000
```

Ispravnost rada serverske komponente može se proveriti otvaranjem sledeće adrese:

```
http://localhost:8000/api/health
```

Ukoliko je server uspešno pokrenut, dobija se odgovor:

```
{"status":"ok"}
```

4.3.2 Pokretanje klijentske komponente

Za pokretanje klijentske komponente potrebno je pozicionirati se u direktorijum *frontend*. Zatim se instaliraju potrebni paketi i pokreće razvojni server.

```
cd frontend
npm install
npm run dev
```

Nakon pokretanja, klijentskoj aplikaciji se može pristupiti preko adrese:

```
http://localhost:3000
```

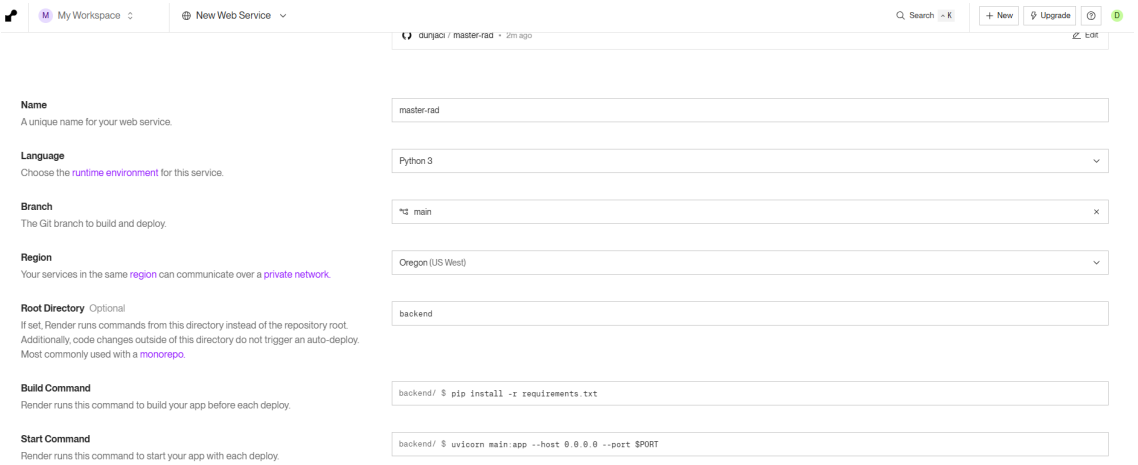
U lokalnom okruženju klijentska aplikacija podrazumevano šalje zahteve na adresu `http://localhost:8000`. Na taj način se lokalno pokrenuti frontend povezuje sa lokalno pokrenutim backend serverom.

4.4 Postavljanje aplikacije na javno dostupne servise

Da bi elektronska lekcija bila dostupna korisnicima preko javnog linka, aplikacija je postavljena na dve platforme. Serverska komponenta postavljena je na *Render*, dok je klijentska komponenta postavljena na *Vercel*. Ovakav pristup odgovara arhitekturi projekta jer su klijentski i serverski deo odvojeni.

4.4.1 Postavljanje serverske komponente na *Render*

Serverska komponenta postavljena je na platformu *Render* kao *Web Service*. *Render* je povezan sa *GitHub* repozitorijumom, čime je omogućeno da se nova verzija serverske komponente automatski postavi nakon izmena u repozitorijumu.



Slika 4.1: Podešavanja za postavljanje (eng. deploy) serverske komponente na Render platformu

Kao što je prikazano na Slici 4.1, prilikom kreiranja servisa podešeni su sledeći parametri:

```
Root Directory: backend
Runtime: Python
Build Command: pip install -r requirements.txt
Start Command: uvicorn main:app --host 0.0.0.0 --port $PORT
```

Parametar `Root Directory` označava da se serverska komponenta nalazi u direktorijumu `backend`. Komanda za pripremu (eng. *build command*) instalira sve zavi-

snosti navedene u datoteci `requirements.txt`, dok komanda za pokretanje startuje *FastAPI* aplikaciju.

Nakon uspešnog postavljanja, *Render* generiše javni URL serverske komponente:

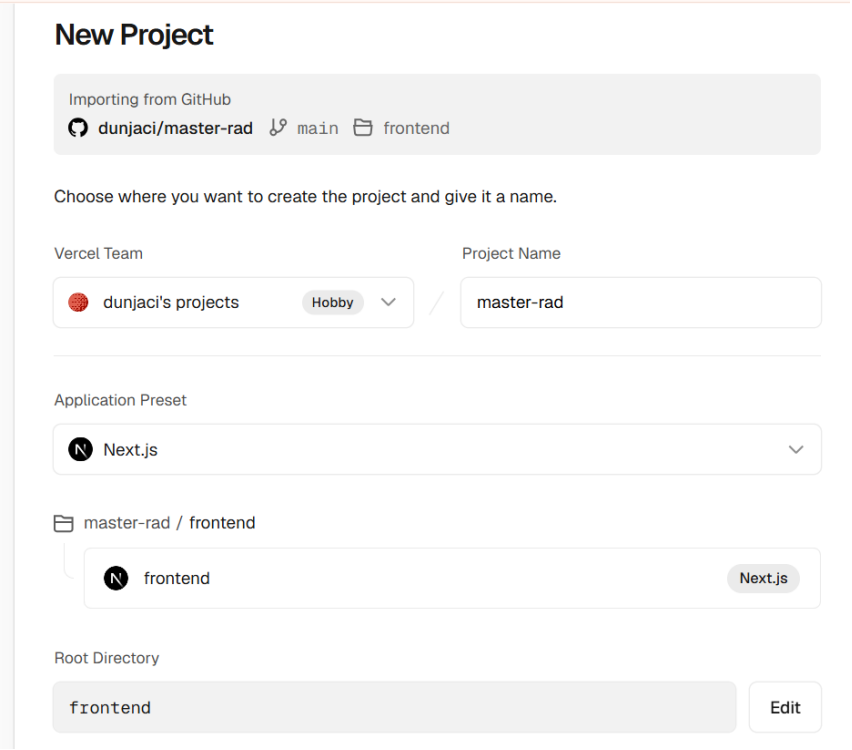
```
https://master-rad-rynl.onrender.com
```

4.4.2 Postavljanje klijentske komponente na *Vercel*

Klijentska komponenta postavljena je na platformu *Vercel*. *Vercel* je posebno pogodan za *Next.js* aplikacije jer automatski prepoznaje strukturu projekta i izvršava proces izgradnje aplikacije.

Slike 4.2 i 4.3 prikazuju podešavanja parametara prilikom kreiranja *Vercel* projekta:

```
Root Directory: frontend
Framework Preset: Next.js
Build Command: npm run build
Install Command: npm install
```



Slika 4.2: Podešavanja za postavljanje (eng. deploy) klijentske komponente na Vercel platformu

Pošto se klijentska aplikacija nalazi u direktorijumu `frontend`, taj direktorijum je označen kao korenski direktorijum projekta. Nakon uspešnog postavljanja, *Vercel* generiše javnu adresu aplikacije:

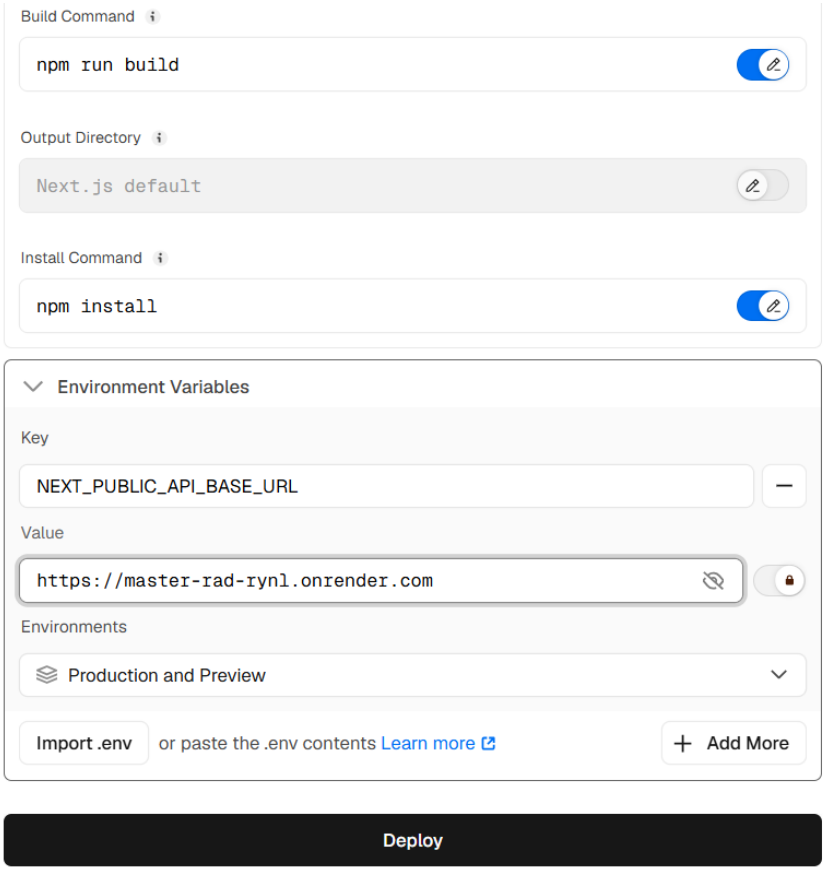
`https://master-rad-seven.vercel.app`

4.4.3 Povezivanje klijentske i serverske komponente

Povezivanje klijentske i serverske komponente ostvareno je pomoću promenljivih okruženja. Klijentska aplikacija koristi promenljivu `NEXT_PUBLIC_API_BASE_URL`, koja sadrži adresu serverske komponente:

```
NEXT_PUBLIC_API_BASE_URL=https://master-rad-rynl.onrender.com
```

Na osnovu ove promenljive, klijentska aplikacija zna na koju adresu treba da šalje API zahteve. U lokalnom okruženju, ukoliko promenljiva nije definisana, koristi se podrazumevana adresa `http://localhost:8000`.



Slika 4.3: Podešavanja za deploy klijentske komponente na Vercel platformu

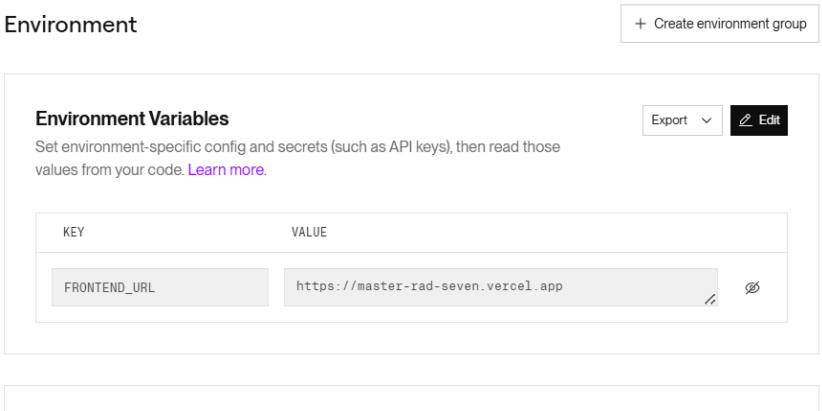
Sa druge strane, kao što je prikazano na Slici 4.4, serverska komponenta koristi promenljivu `FRONTEND_URL`, kojom se definiše sa kog domena je dozvoljeno slanje zahteva ka backend serveru:

```
FRONTEND_URL=https://master-rad-seven.vercel.app
```

Ova promenljiva koristi se za podešavanje *CORS* politike. Time se omogućava da produkciona verzija klijentske aplikacije komunicira sa serverskom komponentom, dok lokalna verzija i dalje može da koristi `http://localhost:3000`.

Nakon povezivanja komponenti, tok komunikacije može se predstaviti na sledeći način:

```
Vercel frontend -> Render backend API -> rezultat se vraća korisniku
```



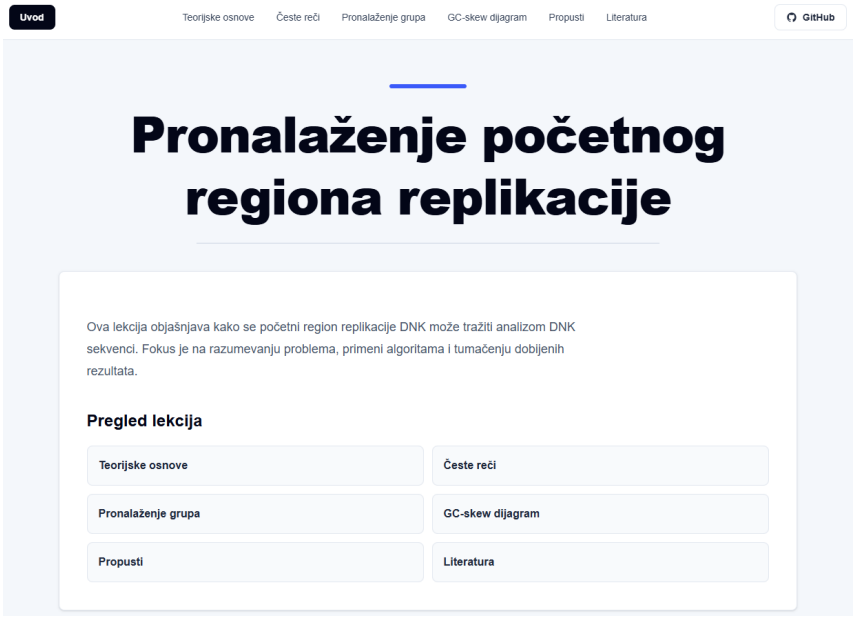
Slika 4.4: Podešavanja za povezivanje klijentske i serverske komponente

4.5 Funkcionalnosti elektronske lekcije

Elektronska lekcija se sastoji iz više stranica koje zajedno prikazuju teorijski uvod, algoritme i literaturu. Na vrhu aplikacije nalazi se navigacioni meni preko kog korisnik može da pristupi svim delovima lekcije. Navigacija sadrži stranice: Uvod, Teorijske osnove, Česte reči, Pronalaženje grupa, *GC-skew* dijagram, Propusti i Literatura. U navigaciji se nalazi i link ka GitHub repozitorijumu.

4.5.1 Početna stranica

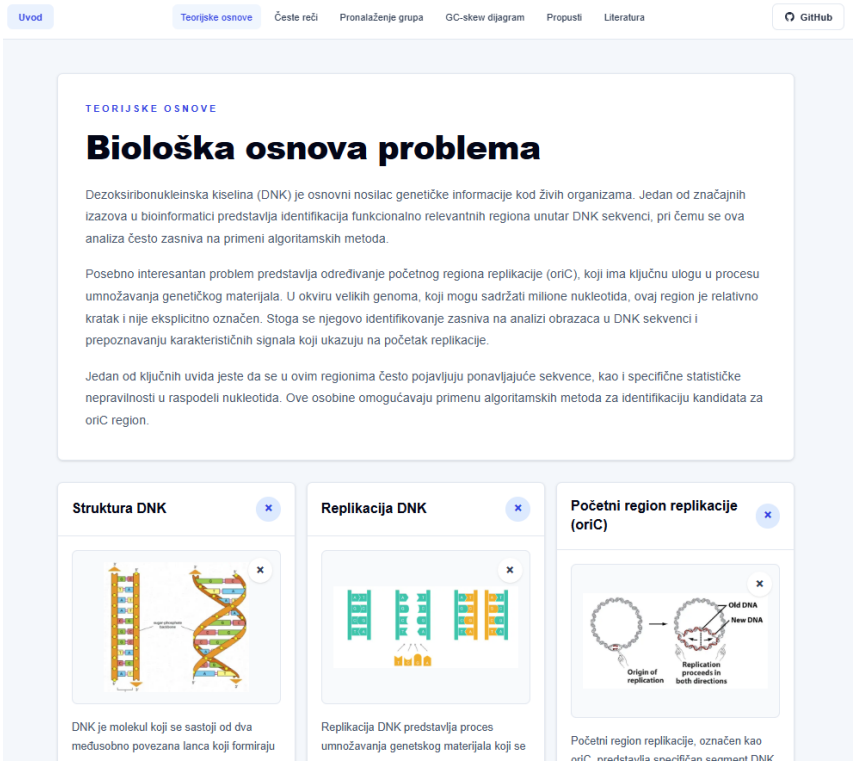
Početna stranica predstavlja ulaznu tačku aplikacije (Slika 4.5). Na njoj je prikazan naslov lekcije i kratak pregled dostupnih celina. Korisnik sa ove stranice može da pređe na teorijske osnove, prikaze algoritama ili literaturu.



Slika 4.5: Početna stranica elektronske lekcije

4.5.2 Teorijske osnove

Stranica Teorijske osnove, prikazana na Slici 4.6, uvodi biološke pojmove potrebne za razumevanje algoritama. Objašnjena je struktura DNK molekula, proces replikacije, početni region replikacije, DnkA boksovi i pojam obrnutog komplementa. Klikom na sekciju prikazuju se interaktivne kartice sa detaljnim objašnjenjima pojmova. Ovaj deo lekcije povezuje biološku motivaciju sa algoritamskim problemima koji se kasnije obrađuju.

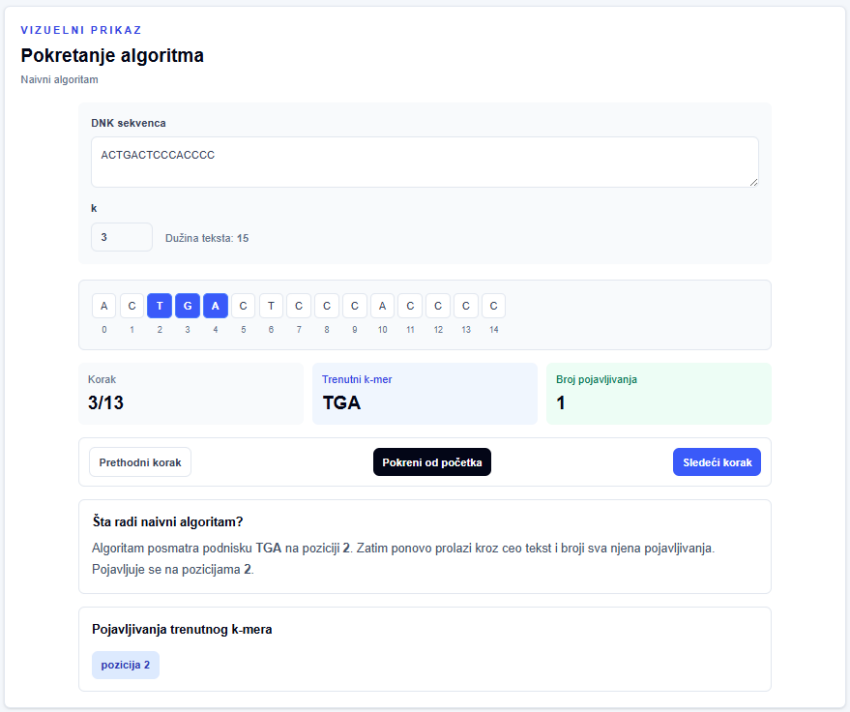


Slika 4.6: Stranica sa teorijskim osnovama

4.5.3 Česte reči

Stranica Česte reči prikazuje algoritamski problem pronalaženja najčešćih k-grama u zadatoj DNK sekvenci. Korisnik može da unese sekvencu i vrednost parametra `k`. Aplikacija zatim prikazuje rezultat i omogućava praćenje rada različitih pristupa.

Na ovoj stranici predstavljeni su naivni algoritam, optimizacija nizom frekvencija, optimizacija heš mapom i pristup sortiranjem. Za svaki pristup prikazana je vremenska složenost, objašnjenje i programski kod u jeziku *Python*. Vizuelni prikaz omogućava korisniku da prati trenutni k-gram, njegovu poziciju i način na koji algoritam obrađuje ulaznu sekvencu (Slika 4.7).

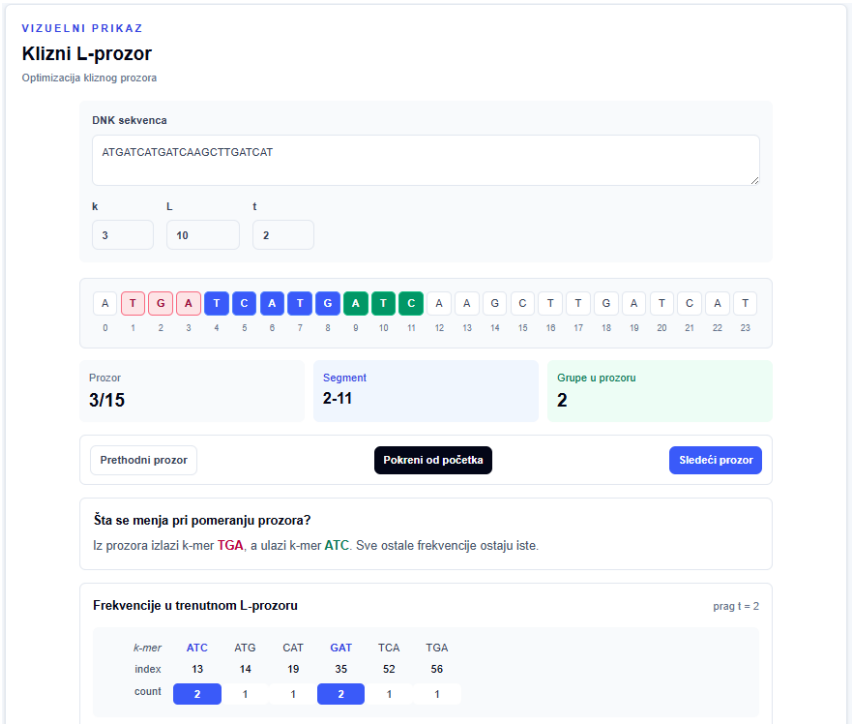


Slika 4.7: Vizuelizacija naivnog algoritma za problem čestih reči

4.5.4 Pronalaženje grupa

Stranica Pronalaženje grupa prikazuje traženje k-grama koji formiraju (L, t)-grupe. K-gram formira grupu ukoliko postoji segment genoma dužine L u kojem se taj k-gram pojavljuje najmanje t puta.

Korisnik unosi DNK sekvencu i parametre `k`, `L` i `t`. Aplikacija prikazuje prozore dužine L, frekvencije k-grama u trenutnom prozoru i k-grame koji zadovoljavaju zadati prag. Pored osnovnog pristupa, prikazana je i optimizacija pomoću kliznog prozora, gde se pri svakom pomeranju prozora ažuriraju samo vrednosti za k-grame na granicama posmatranog segmenta (odnosno vrednosti za k-gram koji postaje obuhvaćen prozorom / prestaje da bude obuhvaćen prozorom). Vizuelizacija ovog pristupa prikazana je na Slici 4.8.

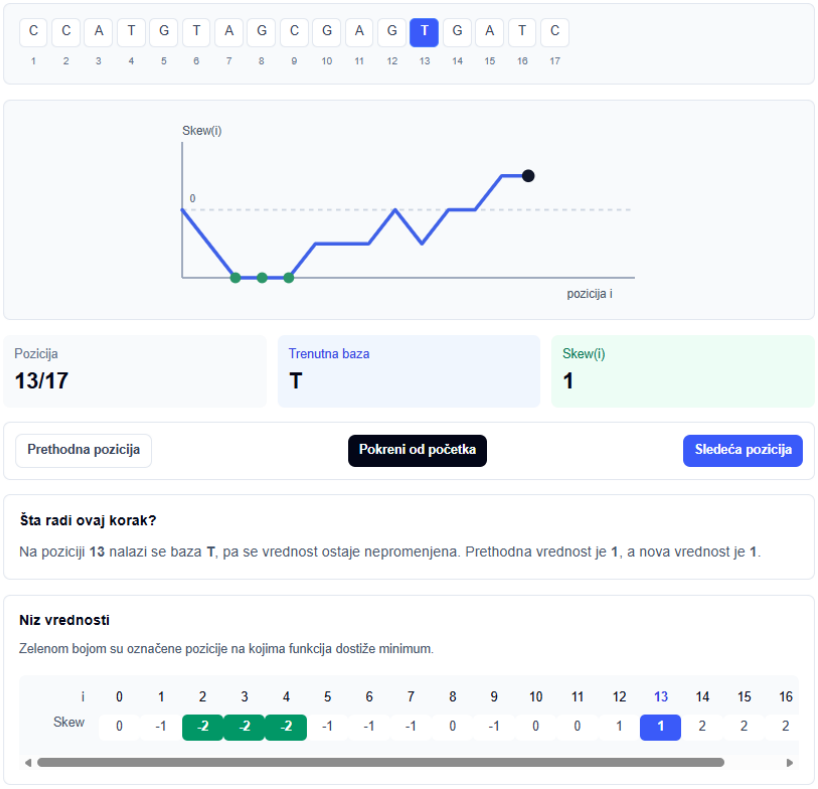


Slika 4.8: Vizuelizacija optimizovanog pristupa za pronalaženje grupa pomoću kliznog prozora

4.5.5 *GC-skew* dijagram

Stranica *GC-skew* dijagram prikazuje algoritam za izračunavanje *GC-skew* vrednosti duž DNK sekvence. Ova vrednost prati razliku između broja baza G i C od početka sekvence do tekuće pozicije. Pozicije na kojima GC-skew funkcija dostiže minimum mogu predstavljati kandidate za početni region replikacije.

Korisnik unosi DNK sekvencu, a aplikacija prikazuje konstrukciju niza vrednosti korak po korak. Vizuelni prikaz obuhvata trenutnu nukleotidnu bazu, trenutnu vrednost funkcije, grafički prikaz *GC-skew* dijagrama i označene pozicije minimuma (Slika 4.9).

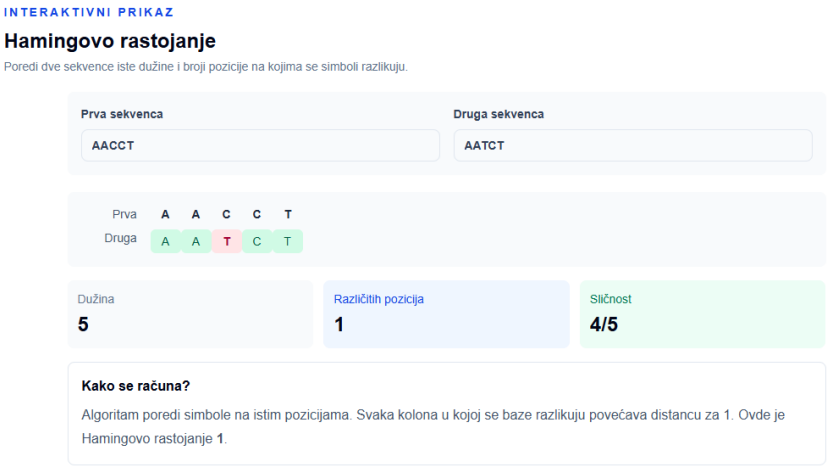


Slika 4.9: Vizuelizacija kontrukcije GC-skew dijagrama

4.5.6 Približno poklapanje i česte reči sa propustima

Stranica Propusti obuhvata više povezanih algoritamskih pojmova: Hamingovo rastojanje, približno poklapanje (pod)sekvenci, Hamingovo susedstvo sekvence/k-grama, česte reči sa propustima i obrnute komplemente DNK (pod)sekvenci/k-grama. Ovi pojmovi su važni jer se biološki značajni motivi ne moraju uvek pojaviti u potpuno istom obliku. Mutacije mogu promeniti pojedinačne nukleotide, dok motiv i dalje može zadržati svoju funkciju.

Kao što je prikazano na Slici 4.10, za Hamingovo rastojanje korisnik unosi dve sekvence iste dužine, a aplikacija prikazuje broj pozicija na kojima se one razlikuju.

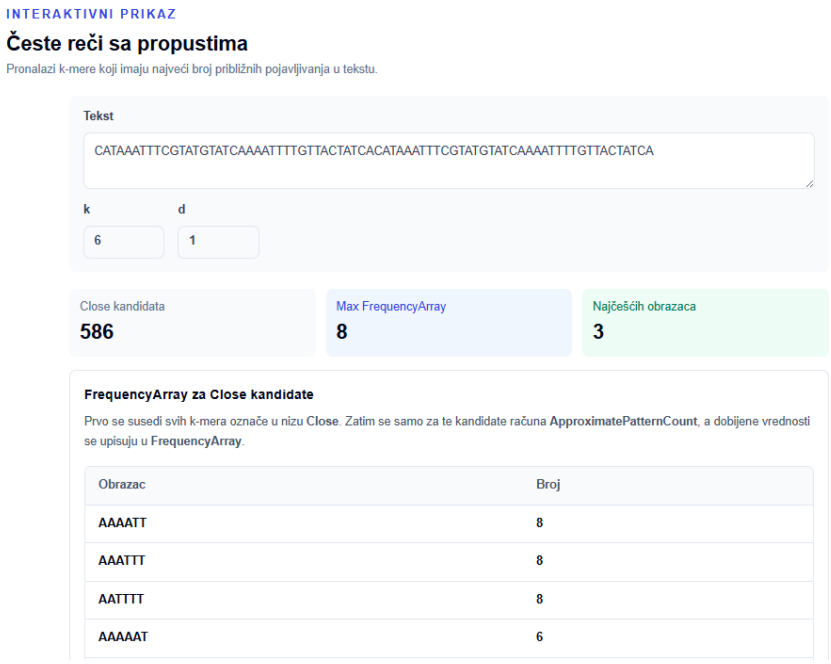


Slika 4.10: Interaktivni prikaz računanja Hamingovog rastojanja za dve sekvence

Kod algoritma sa približnim poklapanjima korisnik unosi tekst, obrazac i dozvoljeni broj odstupanja, nakon čega se prikazuju pozicije na kojima se obrazac pojavljuje uz najviše d razlika. Deo za d-susedstvo prikazuje skup svih sekvenci koje su od zadatog obrasca udaljene najviše d pozicija (Slika 4.11).

Slika 4.11: Prikaz d-susedstva niske AAAAA, za d=1

Na istoj stranici obrađuju se i česte reči sa propustima, gde se traže obrasci koji imaju najveći broj približnih pojavljivanja u tekstu, kao i efikasniji pristup pomoću heš mape. Dodatno, prikazana je i varijanta algoritma koja uzima u obzir obrnute komplemente, što je značajno pri analizi DNK sekvenci jer se motivi mogu posmatrati u obe orijentacije. Na Slici 4.12 vidimo interaktivni prikaz za česte reči sa propustima.



Slika 4.12: Interaktivni prikaz algoritma za česte reči sa propustima

4.5.7 Literatura

Stranica Literatura sadrži izvore korišćene za izradu teorijskog i algoritamskog dela lekcije. Pored literature iz oblasti molekularne biologije i bioinformatike, na ovoj stranici nalazi se i link ka *GitHub* repozitorijumu u kojem se nalaze izvorni kod aplikacije, prateći projektni fajlovi i tekstualna verzija rada.

4.6 Upotreba alata zasnovanih na veštačkoj inteligenciji

Tokom izrade ovog rada korišćeni su *Microsoft Copilot* i *ChatGPT* kao pomoćni alati. *ChatGPT* korišćen je prvenstveno tokom razvoja programske implementacije, za generisanje i unapređenje delova programskog koda, kao i za pomoć pri otklanjanju grešaka i pronalaženju odgovarajućih implementacionih rešenja. *Microsoft Copilot* je korišćen za doradu, preformulisanje i ujednačavanje stila pojedinih delova teksta.

Sadržaj i predlozi dobijeni korišćenjem alata zasnovanih na veštačkoj inteligenciji nisu preuzimani bez dodatne provere. Ispravnost generisanog i izmenjenog programskog koda proveravana je testiranjem implementiranih funkcionalnosti i poređenjem

dobijenih rezultata sa očekivanim rezultatima. Teorijske tvrdnje i objašnjenja proveravani su na osnovu stručne literature navedene u bibliografiji, dok su svi sadržaji uključeni u rad dodatno provereni, korigovani i prilagođeni temi master rada.

Glava 5

Zaključak

U ovom radu analizirani su algoritamski pristupi za identifikaciju početnog regiona replikacije u DNK sekvencama. Kroz prikaz različitih algoritama pokazano je na koji način se analiza učestalosti podniski, njihove lokalne raspodele i statističkih karakteristika genoma može iskoristiti za pronalaženje kandidata za *oriC* region.

Za svaki obrađeni algoritam analizirana je osnovna ideja algoritma, složenost, kao i njegove prednosti i ograničenja. Prikazane optimizacije pokazale su kako se odgovarajućim izborom struktura podataka i algoritamskih tehnika može značajno unaprediti efikasnost obrade DNK sekvenci.

Praktični doprinos ovog rada predstavlja razvoj elektronske lekcije koja objedinjuje teorijska objašnjenja obrađenih algoritama i njihovu interaktivnu primenu kroz veb aplikaciju. Implementirana aplikacija omogućava korisnicima da pokreću algoritme nad proizvoljnim ulaznim podacima, prate njihovo izvršavanje korak po korak i analiziraju dobijene rezultate, čime se olakšava razumevanje njihovog rada i primene u bioinformatici.

Iako opisani algoritmi predstavljaju značajnu osnovu za identifikaciju početnog regiona replikacije, u praksi se oni često kombinuju sa drugim bioinformatičkim metodama i biološkim informacijama kako bi se postigla veća pouzdanost rezultata. Budući razvoj može obuhvatiti proširenje elektronske lekcije novim algoritmima i interaktivnim primerima, kao i primenu opisanih metoda na većim skupovima realnih genomskih podataka, čime bi se dodatno unapredila njena obrazovna i praktična vrednost.

Bibliografija

[1] Fastapi dokumentacija, 2026. on-line at: https://fastapi.tiangolo.com/.

[2] Github repozitorijum sa izvornim kodom elektronske lekcije, 2026. on-line at: https://github.com/dunjaci/master-rad.

[3] Next.js dokumentacija, 2026. on-line at: https://nextjs.org/docs.

[4] Node.js dokumentacija, 2026. on-line at: https://nodejs.org/en/docs.

[5] Python 3.12 dokumentacija, 2026. on-line at: https://docs.python.org/3.12/index.html.

[6] React dokumentacija, 2026. on-line at: https://react.dev/.

[7] Render dokumentacija, 2026. on-line at: https://render.com/docs.

[8] Tailwind css dokumentacija, 2026. on-line at: https://tailwindcss.com/docs.

[9] Vercel dokumentacija, 2026. on-line at: https://vercel.com/docs.

[10] B. Alberts, A. Johnson, J. Lewis, M. Raff, K. Roberts, and P. Walter. *Molecular Biology of the Cell*. W. W. Norton & Company, 7th edition, 2022.

[11] Vinod Chugani. Hamming Distance Explained: The Theory and Applications, 2025. on-line at: https://www.datacamp.com/tutorial/hamming-distance.

[12] Jovana Kovačević. Materijal sa predavanja „Uvod u bioinformatiku", 2022. on-line at: https://www.bioinformatika.matf.bg.ac.rs/predavanja/Chapter_1.pdf.

[13] Adam R. Leman and Eishi Noguchi. The replication fork: Understanding the eukaryotic replication machinery and the challenges to genome duplication, 2013. Genes, 4(1), 1–32. on-line at: https://doi.org/10.3390/genes4010001.

[14] Pavel Pevzner Philip Compeau. Bioinformatics Algorithms: An Active Learning Approach Vol. I, Chapter 1: Where in the Genome Does DNA Replication Begin?, 2015. on-line at: `https://www.bioinformaticsalgorithms.org/`.

[15] Aleksandar Veljković. Uvod u bioinformatiku: Materijali sa časova vežbi - Pronalaženje početka replikacije, 2022. on-line at: `https://github.com/aleksandar-veljkovic/MATF-Uvod-u-bioinformatiku/tree/master/2022`.

Biografija autora

Dunja Čitlučanin rođena je 22. septembra 1998. godine u Beogradu. Smer Računarstvo i informatika na Matematičkom fakultetu Univerziteta u Beogradu upisala je 2017. godine, a završila 2022. godine. Nakon toga je upisala master studije na istom smeru. Od 2023. do novembra 2024. godine je zaposlena na poziciji *Junior Software engineer* u firmi **Telekom Srbija**, a potom od novembra 2024. godine do marta 2026. je zaposlena na poziciji *Software developer* u firmi **Development and Testing Center**. Sada je zaposlena kao *Data scientist* u firmi **NCR Atleos**.