UNIVERZITET U BEOGRADU

MATEMATIČKI FAKULTET


Ivana M. Cvetkoski

# ALGORITMI ISTRAŽIVANJA TEKSTA ZA PRONALAŽENJE MOTIVA U BIOLOŠKIM SEKVENCAMA

master rad

Beograd, 2024.

Mentor:

**prof. dr Nenad Mitić**
*Matematički fakultet*
*Univerzitet u Beogradu*

Članovi komisije:

**prof. dr Saša Malkov**
*Matematički fakultet*
*Univerzitet u Beogradu*

**prof. dr Jelena Graovac**
*Matematički fakultet*
*Univerzitet u Beogradu*

Datum odbrane:

---

*Hvala porodici na podršci.*
*Hvala profesoru Nenadu Mitiću.*
*Hvala Bojani i Milošu.*

# Sadržaj

**Naslov master rada:** Algoritmi istraživanja teksta za pronalaženje motiva u biološkim sekvencama

**Rezime:** Istraživanje teksta je proces izvođenja visokokvalitetnih informacija iz tekstualnih podataka. Algoritmi istraživanja teksta se primenjuju na nestrukturiran tekst radi prevođenja u strukturiran i/ili numerički oblik. Pri tome tekst može da se posmatra kao jedna jako dugačka niska karaktera u kojoj se traže korisne informacije. Željene informacije mogu da se dobiju na različite načine, na primer uspostavljanjem veza između podniski ili pronalaženjem određenih motiva.

Tehnologija istraživanja teksta imaju široku primenu u naučnim i poslovnim oblastima: otkrivanju bezbednosnih pretnji, biomedicini, softverskim aplikacijama (na primer prepoznavanju aktivnosti vezanih za terorizam), filtriranju materijala za elektronske medije, analizi sentimenata (raspoloženja), pretraživanju naučne i stručne literature, itd.

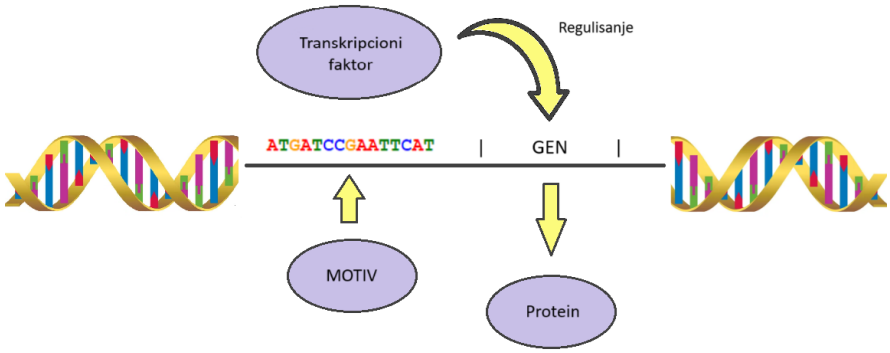
Istraživanje teksta u biomedicini predstavlja vrlo značajan alat za određivanje mogućeg horizontalnog i vertikalnog prenosa naslednog materijala između različitih organizama, kao i otkrivanje motiva (biomarkera) koji mogu da pomognu u proučavanju različitih bolesti. Cilj rada je prikaz najznačajnihih algoritama za obradu teksta i ilustracija njihove primene pronalaženjem zajedničkih motiva u genomima agresivnih koronavirusa.

**Ključne reči:** Traženje motiva, biomarkeri, Risotto, MITRA, Winnower, PMS5, GA, Voting, EM

# 1 Uvod

## 1.1 Biološka pozadina pronalaženja motiva

DNK u sebi sadži sve potrebne informacije koje služe za normalno funkcionisanje ćelija. U njoj se nalaze i geni koji predstavljaju osnovnu funkcionalnu jedinicu, sadrže genetske informacije i učestvuju u proizvodnji različitih proteina. Regulacija genske ekspresije obuhvata procese koje ćelije i virusi koriste da regulišu način na koji se informacije gena pretvaraju u genske produkte. Važan deo regulacije gena je vezan za određenu vrstu proteina koji se zovu transkripcioni faktori. Proces transkripcije proteina započinje tako što se protein, transkripcioni faktor, vezuje za regulatorno mestu u DNK, posle koga se nalazi određen gen. Mesto u DNK koje ovi proteini prepoznaju i za koje se vezuju se naziva motiv. Nakon vezivanja, proteini utiču na aktivaciju gena koji se nalazi nakon mesta vezivanja. Motivi su kratke niske koje se ponavljaju na više mesta u DNK i njihov pronalazak je jedan od fundamentalih problema u molekularnoj biologiji. *Ukoliko se neki šablon često javlja u DNK, to nam govori da je važan i da ima značaja [1].* Zbog toga se ovaj problem svodi na traženje motiva u skupu DNK sekvenci, tj. traženje niske koja se često ponavlja u datim DNK sekvencama. Kako je dužina motiva u većini slučajeva mala, od 5 do 20 nukleotida, dok je dužina DNK koja se pretražuje veoma velika, od nekoliko stotina pa sve do nekoliko hiljada istih, ovaj zadatak nije trivijalan. Ono što dodatno otežava opisani problem jeste mogućnost mutacija nukleotida, što znači da motivi koji se ponavljaju u sekvencama ne moraju u svakoj biti potpuno isti [2, 3].




Slika 1: Transkripcioni faktor se vezuje za motiv i vrši regulisanje gena

## 1.2 Detaljan opis problema i načini rešavanja

Postoje 3 kategorije pronalaženja motiva:

1. Jednostavna pretraga motiva - gde je zadatak pronaći sve motive u ulaznim sekvencama koji su date dužine $l$ i koji se ponavljaju u svim sekvencama.

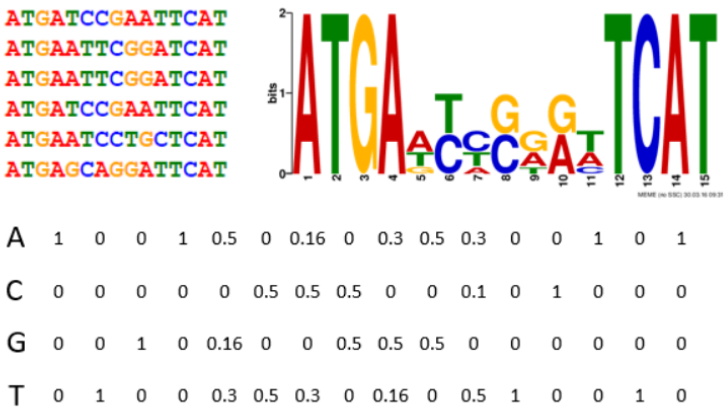
2. Izmenjena pretraga motiva - poenta ovog zadatka je poravnanje bioloških sekvenci kako bi se uočile slične regije koje mogu biti od strukturnog ili funkcionalnog značaja između sekvenci.

3. Pretraga usađenog motiva - kako algoritmi opisani u ovom radu rešavaju ovaj problem, njega ćemo opisati detaljnije u sledećem pasusu [3].

Problem usađenog motiva su opisali Pevzner i Sze (2000): Na ulazu imamo $n$ ulaznih sekvenci (dužine svih sekvenci je ista je iznosi $m$) i dva cela broja $l$ i $d$. Zadatak je pronaći sve niske $M$ dužine $l$ koje se ponavljaju u svakoj od $n$ ulaznih sekvenci sa najviše $d$ razlika. Uslovi na ulazu koji moraju biti ispunjeni su sledeći: mora da postoji niska $M_i$ dužine $l$ u sekvenci $i$, za svako $i$ ($1 \leq i \leq n$), takva da se razlikuje od niske $M$ na manje ili tačno $d$ karaktera. Kao mera broja karaktera za kojih se dve niske razlikuju koristi se Hamingovo rastojanje. Niska $M$ predstavlja motiv.

Npr. ukoliko imamo tri sekvence GCGCGAT, CAGGTGA, i CGATGCC, i zadate vrednosti $l$=3 i $d$=1, dva primera (3, 1)-motiva su GAT i GTG. Ovaj zadatak je poznat u bioinformatici i pokazano je da je NP-kompletan. Složenost ovog zadatka dolazi iz činjenice da ne može da se svede na jednostavno traženje podniske u listi sekvenci, zbog slučajnih mutacija koje komplikuju proces pretrage. Ono što dodatno otežava pronalaženje motiva je mala dužina motiva u odnosu na veliku dužinu sekvenci na ulazu. U radu [4] istaknuto je da za 20 ulaznih sekvenci od po 600 karaktera može da se očekuje barem jedan (9,2) slučajan motiv.

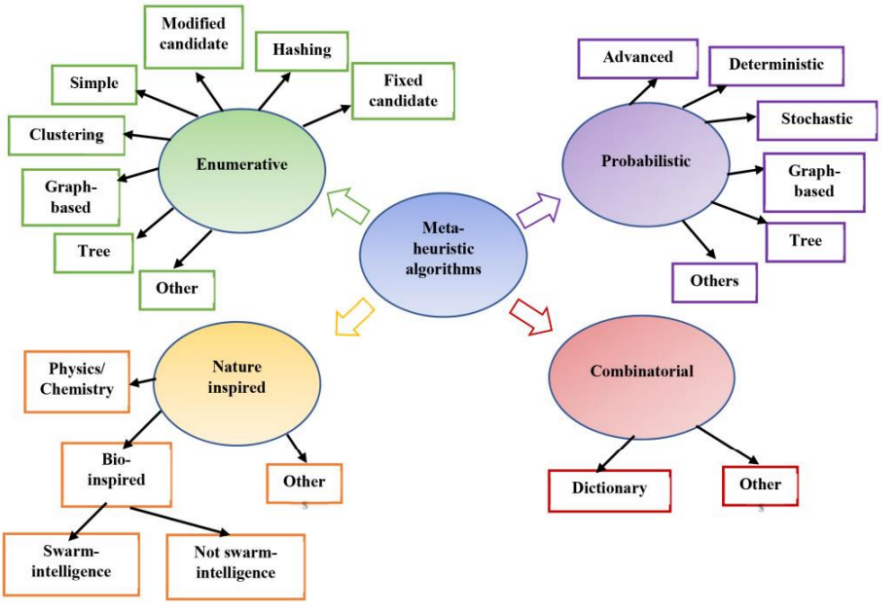
Za datu listu niski, konsenzus niska predstavlja sekvencu nukleotida koja na svakoj poziciji ima onaj nukleotid koji se najčešće ponavlja u datim niskama. Sledeći primer je vizuelni prikaz frekvencije svakog nukleotida po indeksima. Visina slova nam govori koliko često se određen karakter pojavljuje na datom mestu.




Slika 2: Primer frekvencije nukleotida

Postoje četiri glavne podele algoritama koji rešavaju ovaj problem, a to su enumerativni, probabilistički, evolucioni i kombinatorni algoritmi. Za enumerativne važi da će uvek pronaći tačno rešenje, pod uslovom da takvo postoji, dok probabilistički ne mogu to da garantuju. Enumerativni algoritmi, kao što im ime kaže, koriste enumerativni princip pretrage za niskom. Traženje motiva se vrši tako što se prolazi kroz ceo prostor pretrage kako bi se utvrdilo koje niske se ponavljaju sa dozvoljenim mutacijama upoređivajući ih. Zbog ovakvog pristupa, jasno je zašto ovi egzaktni algoritmi uvek dolaze do globalnog optimuma, ali po visoku cenu vremena i memorije. Potrebno je eksponencijalno vreme za rešavanje problema na ovaj način i za velike ulazne sekvence i dužine motiva, ovi algoritmi su vrlo neefikasni. Druga grupa algoritama koriste probabilistički pristup rešavanja zadatka. Ovi algoritmi često koriste matricu težina pozicija, koja predstavlja verovatnoću da se na određenoj poziciji motiva nalazi određeni nukleotid. U većini slučajeva aprokimativni su brži od egzaktnih algoritama. Iako većina algoritama upada u prve dve grupe, postoje i drugi načini rešavanja ovog problema koji su se dobro pokazali. U trećoj grupi su algoritmi koji su inspirisani prirodom, i obično simuliraju ponašanje insekata ili drugih životinja za rešavanje problema. Evolucioni algoritmi mogu da prevaziđu nedostatke lokalnog pretraživanja i da ujedine lokalnu i globalnu pretragu. Za kraj, poslednja kategorija predstavlja kombinatorne algoritme koji spajaju više algoritama kako bi došli do optimalnog

rešenja [5].




Slika 3: Na slici su prikazane četiri glavne kategorije algoritama i njihove podkategorije. Slika je preuzeta iz rada [5].

U ovom radu su opisani i upoređeni rezultati osam algoritama, od kojih su šest iz grupe enumerativnih i po jedan iz grupa probabilističkih i evolutivnih algoritama. Implementirani su tako da rade i za ulazne sekvence različitih dužina i podrazumevana azbuka sa kojom radi je **A**, **C**, **G**, **T**, ali je moguće svakom proslediti datoteku sa novom specifičnom azbukom, tako da može da rešava pronalaženje motiva i za druge nukleotide i amino kiseline (u opisima algoritama koji slede, korišćena je podrazumevana azbuka od 4 nukleotida). Ni za jedan od navedenih algoritama nije pronađena implementacija na internetu. Implementacija priložena u ovom master radu je rađena prema opisu algoritama iz referentnih radova. Izvorni kodovi implementiranih algoritama su priloženi u dodatku master rada u elektronskom obliku.

# 2 Algoritam Risotto

Sufiksno stablo predstavlja strukturu podataka koja je pogodna za rešavanje problema pretrage teksta i pronalaženje šablona. Kao i obično stablo, sastoji se iz čvorova i grana, ali tako da svaka grana predstavlja jedan karakter i ovakvo stablo sadrži sve sufikse datog teksta. Omogućeno je njegovo efikasno obilaženje pa se zbog toga ova struktura često koristi i u rešavanju problema u bioinformatici. Dubina stabla, u zavisnosti od problema kojeg rešavamo, je jednaka dužini najduže reči na ulazu ili dužini šablona kog tražimo. Vreme i prostor za konstruisanje sufiksnog stabla je linearna u odnosu na dužinu ulaza.

## 2.1 Opis osnovne ideja algoritma

Algoritam Risotto se zasniva na konstrukciji i obilaženju sufiksnog stabla u kome će biti predstavljene sve kombinacije slova iz naše azbuke. Kako broj grana i čvorova, samim tim i dubina stabla, može dostići ogroman broj upotrebljena je optimizacija odsecanjem grana ukoliko je to moguće.

Algoritam na ulazu dobija listu od $N$ sekvenci, stopu greške $e$, koja predstavlja broj mesta na koliko reč može da se razlikuje od motiva da bismo je smatrali mutiranim motivom, i kvorum $q$ koji označava minimalan broj sekvenci u kome je neophodno da se reč nalazi kako bi se smatrala potencijalnim motivom ($q \leq N$). Reč koja ispunjava ova dva uslova, nalazi se u $q$ ili više sekvenci i razlikuje se na $e$ ili manje mesta od motiva se naziva ispravnom rečju. Pored toga, na ulazu se dobijaju i još dve vrednosti koje predstavljaju donju i gornju granicu za dužinu motiva (*kmin, kmax*). Izlaz algoritma je skup svih reči iz date azbuke koje ispunjavaju ove uslove i one predstavalju tražene motive. Algoritam je egzaktan i ukoliko postoji motiv odgovarajuće dužine na ulazu, sigurno će biti pronađen [6].

## 2.2 Algoritam bez optimizacije

Početni algoritam se zasniva na pravljenju sufiksnog stabla počevši od prazne reči koja predstavlja koren stabla, iz kojeg idu četiri grane, koje prestavljaju četiri nukleotida. Slično, svi čvorovi, osim listova, će imati četiri deteta, a dubina stabla će dostići *kmax*. Obilazak stabla predstavlja postepeno dodavanje slova iz grana na pređenom putu. Slova dodajemo, tj. obilazimo

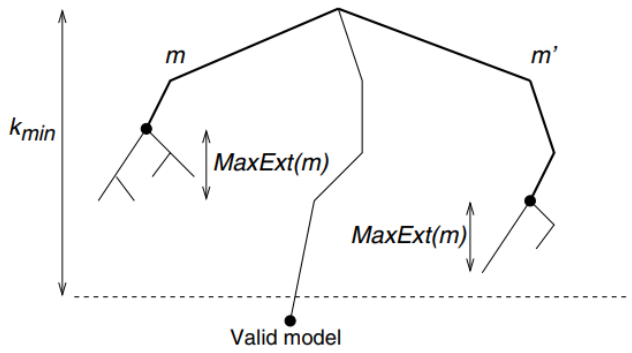
stablo u dubinu, sve dok je kvorum zadovoljen i još nismo došli do ispravne reči koja ima maksimalnu dozvoljenu dužinu ili sve dok kvorum više nije zadovoljen. U prvom slučaju došli smo do motiva, dok u drugom moramo da proverimo da li je uslov za dužinu reči ispunjen, tj. da li dužina reči pre dodavanje poslednjeg slova upada u unapred određen opseg *(kmin, kmax)*. U oba slučaja krećemo da tražimo novi motiv vraćanjem unazad i ispitivanjem nove grane. Postupak se ponavlja sve dok se ne obiđe celo stablo, kako bi svi slučajevi bili pokriveni.

Složenost ovakvog algoritma u najgorem slučaju, pod pretpostavkom da je kmin=kmax=k je: O(N*nk*v(e, k)), gde je nk broj čvorova u dubini k, dok v(e,k) predstavlja broj reči koje se razlikuju od motiva na e ili manje mesta. Složenost je linearna po veličini ulaza, ali može biti i eksponencijalna po broju mutacija. Modifikacijom ovog algoritma možemo dobiti manju složenost.

## 2.3 Algoritam sa optimizacijom - Risotto

U cilju ubrzanja ovog algoritma, osmišljena je optimizacija čija ideja je čuvanje vrednosti *maksimalne rastegljivosti* svakog čvora stabla. Pomoću ove informacije vrši se filtriranje puteva u stablu koji sigurno ne bi doveli do motiva. Vrednost *maksimalne rastegljivosti*, *MaxExt*, predstavlja broj karaktera koji mogu da se dodaju kao sufiks na već postojaću ispravnu reč tako da se uslov ispravnosti ne naruši. Čuvanje ove vrednosti nam može pomoći u slučajevima kada nam nova reč koju istražujemo sadrži drugu, već istraženu reč, kao sufiks. U ovom slučaju naša reč se može maksimalno proširiti sa isto karaktera koliko i sufiksna reč, a da je pri tome uslov ispravnosti i dalje zadovoljen. U slučaju reči m1 i m2, gde je m2 sufiks od m1, kada je zbir dužina trenutne reči m1 i MaxExt(m2), manja od donje granice za dužinu motiva (kmin) znamo da nije moguće reč m1 dovoljno proširiti tako da ispuni uslove dužina i ispravnosti, i zato prekidamo njeno proširivanje.

Ukoliko prethodni uslov za dužinu nije ispunjen ili ne postoji sufiks koji je već ispitan (ovo je moguće jer se reči proveravaju leksikografskim redosledom, zato m2 može doći na red posle m1), proveravamo ispravnost naše reči. Ako je reč ne ispunjava, proveravamo dužinu reči i razlikujemo dva slučaja: prvi je ukoliko je dužina reći manja od donje granice, u tom slučaju postavljamo MaxExt na 0. Razlog za ovo je taj da ne postoji karakter kojim se reč može




Slika 4: Primer gde je moguće odsecanje grane m', čiji je sufiks m, jer je |m'| + MaxExt(m) < kmin. Slika je preuzeta iz knjige [6].

proširiti takav da nova reč koja ima ovaj prefiks bude ispravna.

Drugi slučaj je ako reč m1 ima dužinu koja ulazi u granice dozvoljenih motiva (kmin, kmax). Kako je reč ipak neispravna, postavljamo vrednost MaxExt za njen prefiks dužine (kmin - 1) na razliku dužine motiva i vrednosti (kmin - 1). Ova vrednost se postavlja samo ako je trenutna veličina MaxExt za taj prefiks manja od nove, jer tražimo maksimalnu vrednost.
U suprotnom, ako je reč ispravna, proveravamo da li ispunjava zahteve za dužinu. Ukoliko je veća od minimalne dužine došli smo do potencijalnog motiva koji ispunjava sve zahteve i njega čuvamo. Ukoliko je manja od maksimalne dužine rekurzivno pozivamo algoritam za proširivanje reči. Takođe, moramo obraditi još jedan slučaj, ako dužina reči prekorači zadatu gornju granicu, tada vrednost MaxExt postavljamo na $+\infty$ njenom prefiksu. Razlog za ovo je taj da bilo koja reč koja ima ovu reč kao sufiks može biti proširena tako da ispunjava sve uslove. U trenutku kada završimo ispitivanje poslednjeg slova u azbuci i vratimo se nazad, za vrednost MaxExt trenutne reči stavljamo broj za jedan veći od maksimuma MaxExt njegove dece. Inicijalna vrednost MaxExt se postavlja na -1 za sve čvorove. Ono što možemo da primetimo iz prethodnog objašnjenja postavljanje MaxExt vrednosti jeste da za svaku reč m važi da je MaxExt(m) $\leq$ MaxExt(w), gde je w sufiks od m. Možemo primetiti da postavljenjem manje vrednosti za MaxExt na kraće prefikse ima više uticaja na broj izvršenih odsecanja u odnosu na duže prefikse, jer kraći prefiksi utiču na odsecanje više grana.

Iako složenost ovako optimizovanog algoritma ostaje ista kao i bez optimizacije u najgorem slučaju, izvršavanje prosečnog slučaja je dosta brže po ceni korišćenja više prostora zbog informacija o maksimalnoj rastegljivost za svaki čvor.

# 3 Glasački Algoritam

Hash mapa je struktura podataka u kojoj se podaci sastoje iz dva dela, prvi deo je ključ, a drugi je vrednost. Glavna karakteristika ove strukture je to da su svi ključevi u njoj jedinstveni. Korišćenje hash mapa za rešavanje problema koji podrazumevaju velike ulaze je dobra ideja, jer su hash mape osmišljene tako da operacije pristupanja, dodavanja, modifikovanja i brisanja budu brze, upravo zbog pristupa pomoću jedinstvenog ključa. Iz ovog razloga ova struktura je primenjena u rešavanju našeg problema u sledećem algoritmu.

## 3.1 Opis algoritma

Glasačku algoritam se zasniva na korišćenju hash mape radi prebrojavanja ponavljanja potencijalnih motiva. Ključevi hash mape su k-meri, dok su vrednosti brojači. Zamisao ovog algoritma je da se pronađu svi k-meri od svih ulaznih sekvenci i da se za svakog od njih pronađu svi njegovi susedi, gde sused predstavlja k-mer koji se razliku na do d mesta od početnog k-mera. Ove susede ubacujemo u hash mapu i dodeljujemo im brojače (inicijalna vrednost brojača je jedan), ukoliko već postoji neki k-mer u hash mapi onda njemu povećavamo brojač za jedan.

Ovaj način traženja motiva je vrlo intuitivan, jer se svodi na to da će svi mutirani motivi koji se nalaze u svakoj od ulaznih sekvenci povećati brojač ubačenom traženom motivu, jer se po pravilu od njega razlikuju za do d karaktera. Iz tog razloga traženi motiv bi bio onaj k-mer čija je vrednost brojača ista kao i broj ulaznih sekvenci. Ovo nas dovodi do toga da nakon ažuriranja svih brojača u hash mapi, nam samo preostaje da prođemo jednom kroz sve vrednosti iz nje i nađemo onu koja ima vrednost jednaku broju ulaznih sekvenci [7].

Iako ideja nije komplikovana i za implementaciju algoritma se ne koriste nikakve kompleksne strukture podataka, do traženog motiva ćemo sigurno doći. Problem koji postoji sa ovim pristupom je kompleksnost, vremenska i prostorna. Od dužine motiva i broja dozvoljenih mutacija zavisi koliko varijacija neki k-mer ima, tj. koliki će biti skup njegovih suseda. Na konkretnom primeru možemo videti koliko je taj broj veliki, naime za reč dužine 15 i d dužine 4, broj suseda te jedne reči iznosi 110 565. Broj se eksponencijalno

povećava u odnosu na veličinu d. Hash mape, iako vrlo efikasno mogu da manipulišu sa podacima, preveliki broj ipak može da predstavlja problem.

## 3.2 Modifikovan algoritam

Malom promenom uslova možemo da smanjimo kako vremensku tako i prostornu složenost na sledeći način. Prolaskom kroz prvu sekvencu, izdvajamo sve moguće k-mere i za svaki od njih računamo sve moguće kombinacije koji se razlikuju na tačno d mesta od njega. Ovim pokrivamo ceo skup njegovih suseda, zatim svaki od njih dodajemo u hash mapu i dodeljujemo brojač. Obrađivanje svake sledeće ulazne sekvence se vrši ponavljanjem koraka za računanje svih suseda za svaki njen k-mer, zatim za svaki od njih proveravamo da li se nalazi u hash mapi, ukoliko se nalazi povećavamo brojač, ukoliko se ne nalazi takvog suseda preskačemo, umesto da dodajemo u mapu. Ova modifikacija dovodi do toga da se veličina hash mape ne povećava nakon obrade prve sekvence, već se samo brojači ažuriraju ukoliko je to potrebno. Pored vođenja računa o pokrivanju svih suseda, moramo da vodimo računa i da se brojači povećavaju samo u slučaju ponavljanja u različitim ulaznim sekvencama. U slučaju da u jednoj sekvenci imamo dva k-mera koji nam daju istog suseda, što se ne dešava retko, pogotovo sa velikim ulazom, povećavanje brojača dva puta može da dovede do toga da nam uslov za vrednost brojača ne bude dovoljan i da dođemo do netačnog rezultata u krajnjem prebrojavanju ponavljanja tog potencijalnog motiva. Iz ovog razloga dodajemo još jednu hash mapu u kojoj čuvamo iz koje sekvence je dobijen glas i sa kojom regulišemo povećavanje brojača.

Vremenska složenost algoritma iznosi O(n*t*$(3l)^d$), dok je prostorna složenost O(n*t*$(3l)^d$ + n*t). Ovaj algoritam se pokazao bolje nego algoritam grube sile, ali još uvek ima mesta za optimizacije. Jedan od načina optimizacije je da se koristi u kombinaciji sa algoritmom slučajne projekcije, tako da ne koristimo ceo k-mer kao ključ naše hash mape, već samo podskup njegovih indeksa, uspešnost ovog pristupa zavisi od veličine podskupa.

# 4 Winnower

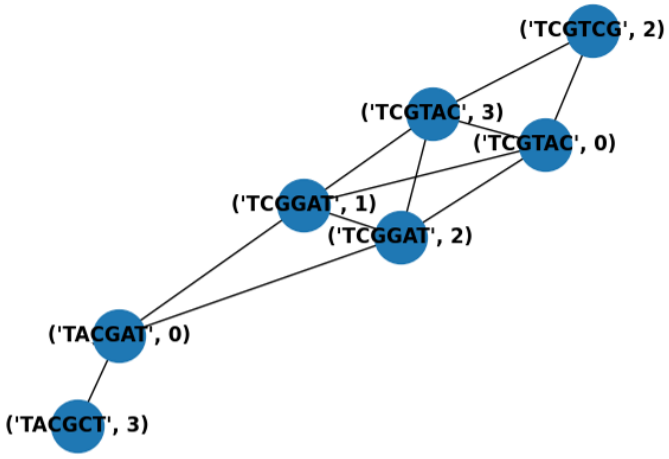
Graf je struktura podataka koju čine čvorovi koji su međusobno povezani granama i zato se može predstaviti preko skupova čvorova (V) i grana (E) - G(V,E). Ova nelinearna struktura može biti veoma moćna u predstavljanju komplikovanih povezanosti između instanci i zbog toga se primenjuje u rešavanju mnogih problema iz različitih oblasti. Drugi važan pojam za ovaj algoritam jesu klike. Klike u grafu predstavljaju podskup čvorova za koje važi da su svi međusobno povezani granama. Grafovi i klike su dve fundamentalne stvari za sledeći algoritam za traženje motiva.

## 4.1 Opis algoritma

Winnower se zasniva na traženju motiva pomoću prethodno kreiranog grafa. Kako bismo našli motiv koji se ponavlja u svim sekvencama (q je broj sekvenci na ulazu) moramo najpre konstruisati graf čiji čvorovi će predstavljati sve k-mere iz svih ulaznih sekvenci. Ovo znači da će svaka podsekvenca dužine k, svih ulaznih sekvenci, predstavljati čvor u grafu. Povezanost čvorova se reguliše dozvoljenom razlikom između potencijalno mutiranih motiva (d). Naime, dva čvora su povezana ako je njihovo Hamingovo rastojanje manje ili jednako 2d. Povezanost čvorova čije vrednosti dolaze iz iste sekvence nije dozvoljeno. Primećujemo da do traženog motiva možemo doći ako pronađemo kliku veličine q u ovako konstruisanom grafu.

Ono što može da se izračuna na osnovu ulaza je broj čvorova u grafu, jer svaki k-mer predstavlja jedan čvor. Ono što ne znamo jeste broj grana kojima su ovi čvorovi povezani, ali možemo da očekujemo da je broj veliki.

Kako je pronalaženje klike u grafu NP kompletan problem, pokušaj pronalaska iste na grafu sa velikim brojem čvorova bi trajalo predugo. Zbog toga u ovom algoritmu akcenat je na odsecanju što većeg broja grana kako bismo olakšali pronalazak klike. Možemo da očekujemo da u grafu postoji mnogo povezanih čvorova čije nam konekcije nisu značajne, tj. njihovi čvorovi se ne nalaze ni u jednog kliki veličine q. Iz tog razloga uvodimo odsecanje irelevantnih grana [8].




Slika 5: Primer grafa za 6-mere dužine sa dozvoljenom razlikom 1. Cifre pored niski predstavljaju indekse sekvence odgovarajućeg 6-mera.

## 4.2 Uslovi odsecanja grana

Postoje uslovi različitih jačina kojima se može određivati da li grane imaju potencijal da se nađu u kliki veličine q ili ne. Ukoliko postoje čvorovi čiji je stepen manji od (q-1) jasno je da ti čvorovi ne mogu biti deo q-klike, jer svaki čvor koji se nalazi u kliki veličine q mora imati barem (q-1) suseda. Svi čvorovi koji ne ispunjuju ovaj uslov se uklanjaju iz grafa. Takođe, moramo imati u vidu da ukoliko neki čvor zadovoljava uslov da mu je stepen veći ili jednak od q-1, to ne znači da je on povezan sa svim delovima našeg multiparcijalnog grafa. Po definiciji povezivanja čvorova nije isključeno, čak se često dešava, pogotovo na većim ulazima, da je jedan čvor povezan sa više čvorova iz iste sekvence. Ovaj uslov odsecanja je definisan kao **k=1** uslov. Možemo primetiti da nije dovoljno dobar uslov da izbaci značajan broj grana i time olakša traženje klike u grafu, iz tog razloga uvodimo jače uslove koji će sklanjati grane iz grafa.

Za svaka dva susedna proveravamo da li skup njihovih zajedničkih suseda ima najmanje (q-2) čvorova. Logika je slična kao za k=1, samo što se umesto čvorova sada izbacuju grane. Po definiciji klike očekivano je da za svaka dva fiksirana čvora iz klike q, mogu da se pronađu (q-2) zajednička suseda, tako

da ih ukupno ima q. Grane koje ne ispunjuju ovaj uslov se uklanjaju, ovaj uslov je poznat kao **k=2** uslov. I kao takav dosta je jači, u smislu broja odsečenih grana, od prethodnog.

Ovaj uslov na sličan način mozemo pojačati na to da umesto trouglova u grafu mora postojati (q-3) 4-klike za svaka susedna tri čvora. Za svaki iz prethodno pomenutog skupa suseda proveravamo da li pravi 4-kliku sa dva početna fiksirana čvora i ostalim susedima iz skupa. Ovaj uslov je **k=3**. Očekivano je da na kraju imamo više od dozvoljenog broja čvorova koje prave 4-klika. Cilj nam je da obrišemo granu ako imamo manje od (q-3) 4-klike. Brisanjem grana koje ne ispunjuju date uslove dolazimo do grafa kod koga je lakše proveriti postojanje q-klike. Brisanja se rade iterativno jer prvi krug filtriranja može da dovede do novih grana koje ispunjavaju uslov iako u prošlom krugu nisu. Zbog jačine uslova i zbog većeg broja iteracija potrebnih za k=2, složenost za k=2 i k=3 su slične [9].

Ono što možemo da zaključimo iz implementacije jeste da je potrebno dosta vremena za konstruisanje grafa i prostora za čuvanje istog. A zatim i za prolazak kroz svaka dva (ili tri u slučaju k=3) povezana čvora i provera uslova. Takođe, ovaj algoritam ne daje prednost k-merima koji su sličniji, niti uzima u obzir broj razlike kada pravi grane, tako da će k-meri koji su isti i oni koji se razlikuju na maksimalnih 2d mesta biti tretirani isto. Postoji optimizacija ovog algoritma koja se zove cWinnower, kod koje se pored uslova Hamingovog rastojanja, uvodi još jedan uslov koji sprečava pravljenje grana koji ga ne ispunjavaju. Optimizacija je opisana u radu [9] i radi samo nad manjim ulazima, dok na velikim nema uticaja i zbog toga nije obrađena detaljnije.

# 5 MITRA

Kao što smo ranije istakli, sufiksna stabla su efikasan način predstavljanja ulaza koji je u obliku teksta i za rešavanje problema nad njim. U sledećem algoritmu koristi se modifikacija ove strukture koja je nastala baš za rešavanje ovog problema.

## 5.1 Opis strukture i algoritma

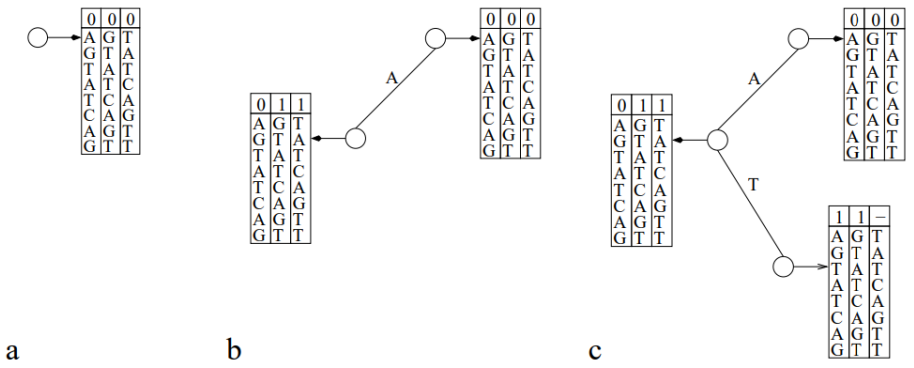
MITRA (*eng. MIsmatch TRee Algorithm*) je dobio ime po specijalnoj strukturi podataka koju koristi, *Mismatch Tree Data Structure*. Koristi sufiksno stablo, ali za razliku od njega putevi od korena do čvora ne predstavljaju samo sve podniske sa tim prefiksom već i njihove susede.

Algoritam se zasniva na deljenju prostora pretrage na delove koji se ne preklapaju (pomoću jedinstvenih prefiksa) i smanjenju prostora pretrage kad god je to moguće. Smanjenje prostora pretrage odsecanjem grana vrši se kada zaključimo da je potprostor *slab*. Potprostor je slab ukoliko su sve instance u njemu slabe. Instancu nazivamo slabom ako je broj njenih suseda manji od unapred zadate granice k, koja je jedan od dodatnih parametara ovog algoritma.

Na samom početku čvor koji predstavlja koren stabla pokazuje na sve k-mere koji postoje u ulaznim sekvencama. Iz svakog čvora ovog drveta, pa tako i korena, izlaze 4 grane, svaka ima oznaku jednog od slova azbuke (jednog od nukleotida A, C, G ili T). Obilaskom stabla u dubinu delimo prostor na potprostore tako što fiksiramo prefiks. Svaki unutrašnji čvor čuva sve instance iz početnih sekvenci kojima odgovara ovaj prefiks ili koji se razlikuju od prefiksa za najviše dozvoljenih d karaktera.

Prvi potprostor čine instance koje imaju prefiks A ili se razlikuju za manje ili jednako od d karaktera od prefiksa A, sledeći je potprostor C koga čine instance koje imaju prefiks C ili se razlikuju za manje ili jednako od d karaktera od prefiksa C itd. Ovo znači da grane predstavljaju nukleotide, isto kao i u običnom sufiksnom stablu, ali za razliku od njega, za svaki k-mer u čvorovima čuvamo brojač koji nam govori na koliko mesta se taj k-mer razlikuje od prefiksa. Ukoliko neka instanca u čvoru ima više od dozvoljenih razlika, tj. njen brojač premaši d, ta instanca nam više nije relevantna.

Postoje dva moguća slučaja pri dodavanju novog karaktera u prefiks, prvi je da se taj karakter poklapa sa karakterom na odgovarajućem mestu u posmatranoj instanci, u tom slučaju instanca je bila i ostaje ispravna. U drugom slučaju, kada se karakter ne poklapa, brojač koji svaka instanca ima i koji broji razlike se povećava. U tom slučaju imamo novo grananje: ukoliko je brojač još uvek u dozvoljenoj granici, manji ili jednak od d, instanca jeste ispravna i kao takva ostaje u pretrazi, i slučaj ako je brojač postao veći od d, instanca se sklanja iz ovog čvora. Ovaj proces ponavljamo proširivanjem prefiksa i izbacivanjem instanci koje ne zadovoljavaju uslov. Ukoliko izbacivanjem instanci dođemo do slabog potprostora, vraćamo se nazad kroz stablo, jer nastavkom obilaska te grana nećemo doći do motiva. U suprotnom, ako ne možemo da zaključimo da je potprostor slab, proširićemo prefiks sa četiri različita slova i svaki potprostor istražiti posebno.




Slika 6: Primer strukture podataka. Ako tri kolone predstavljaju sve k-mere na ulazu, a broj dozvoljenih razlika je 1. Slika a) predstavlja koren ovog stabla koji pokazuje na sve k-mere. Na slici b) vidimo potprostor kada dodamo granu A iz korena (čvor desno), i kako su se brojači ažurirali u skladu sa ovim prefiksom i prvim karakterom k-mera. Na slici c) je prikazano kada se slučaj b) proširi dodavanjem nove grane T. Ažuriranjem brojača vidimo da se brojač poslednjeg k-mer-a u tabeli prešao dozvoljenu granicu i zbog toga se neće više prosleđivati niz stablo. Slika je preuzeta iz rada [10].

Ideja je da ćemo u većini slučajeva brzo doći do slabog potprostora i tako ćemo uštedeti vreme zaustavljanjem proširivanja tog prefiksa i presecanjem te grane za pretragu. Ono što možemo da primetimo jeste da do dubine d neće biti nikakvih odsecanja, jer je d razlika dozvoljeno. Maksimalna dubina stabla je dužina motiva i ako dođemo do te dubine, put koji prave nukleotidi od korena do lista predstavlja potencijalan motiv. Razlog zašto nismo sigurni da je ovaj motiv pravi je taj što koran našeg stabla pokazuje na sve k-mere iz svih sekvenci i nema podatak kojoj sekvenci pripada koji k-mer. Zbog toga, može se desiti da veliki broj ponavljanja imaju neki slučajni k-meri koji se zapravo ne nalaze u svim sekvencama, već u određenim više puta, a u nekima nijednom. Zato radimo ocenjivanje ovog motiva, da bismo filtrirali one koji su prošli dosadašnje provere, ali ipak nisu motivi. Ocenjivanje može da se vrši na različite načine, ali mi smo koristili jednostavan uslov: da li se motiv nalazi u svakoj od početnih sekvenci sa ulaza. Može da bude očekivano da se nalazi u određenom broju sekvenci a ne svim, takođe može za ocenjivanje da se uzme udaljenost od konsenzusa ili ocenjivanje bazirano na entropiji. Kao i u slučaju slabog potprostora, u slučaju da dođemo do potencijalnog motiva, sledeći korak je vraćanje unazad kroz stablo.

Kako bismo uštedeli memorijski prostor, ne moramo imati ceo graf u memoriji i stvarno raditi obilazak nad njim, već možemo virtuelno obilaziti graf, jer znamo da čvorovi zavise od informacija u roditeljskom čvoru i grane sa kojom su povezani sa njim. Sa ovim pristupom obezbeđujemo da će najveći broj čvorova koji se nalaze u memoriji u isto vreme biti k, jer je k maksimalna dubina stabla [10].

# 6 Algoritam PMS5

PMS5 (*eng. Planted motif search 5*) je egzaktan algoritam i jedan je od više algoritama iz PMS familije. On predstavlja kombinaciju PMS1 i PMSPrune algoritma i koristi obilazak stabla suseda k-mera kao i linearno programiranje kako bi sprečio pretragu podstabla koje nema potencijal da sadrži motiv u sebi. Osnovna ideja ovog algoritma je grupisanje po tri k-mera iz različitih sekvenci i traženje zajedničkih suseda. Kombinovanjem računanje preseka i unije skupova suseda za svaka tri k-mera dolazimo do motiva.

## 6.1 Skupovi suseda

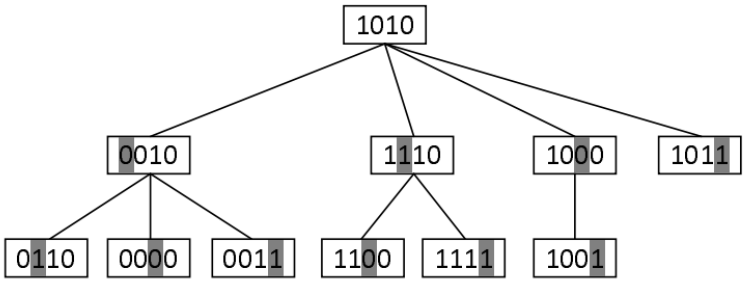
Ukoliko podelimo ulazne sekvence na dva disjunktna skupa, motiv za ulazne sekvence će se naći u preseku skupova motiva oba skupa. Možemo da primetimo da skup motiva za jedan k-mer predstavlja skup svih njegovih suseda koji se razlikuju za d karaktera od njega. Ove dve činjenice predstavljaju suštinu ideje PMS5 algoritma. Izvršavanje se započinje tako što se prva sekvenca izdvaja, dok se sve ostale uparuju u skupove po dve. Ukoliko je ukupan broj sekvenci paran, poslednju sekvencu ćemo duplirati kako bismo osigurali da svaka sekvenca ima para, kada se prva izuzme. Ovo neće uticati na traženje motiva, jer je podrazumevano da se motiv nalazi u svakoj sekvenci pa tako i u poslednjoj koja će biti kopirana. Zatim fiksiramo k-mer iz prve sekvence $S_1$ i fiksiramo par sekvenci $(S_2, S_3)$ iz skupa parova. Za svaka dva k-mera iz fiksiranih uparenih sekvenci računamo presek njihovih suseda sa susedima fiksiranog k-mera iz prve sekvence. Sve ove susede čuvamo u jednom skupu pomoću unije i ovaj skup označavamo sa $Q$. Ovaj postupak nastavljamo da radimo sa istim k-merom iz prve sekvence, ali iteriramo po parovima ostalih sekvenci ($(S_4, S_5)$, $(S_6, S_7)$...). Svaki put kada završimo računanje za jedan par sekvenci, nad unijom njegovih suseda $Q$ radimo presek sa unijom preseka prethodnog para sekvenci.

Motiv ovoga je sledeći, ako se u prvoj sekvenci fiksira motiv, u preseku njegovih suseda i suseda iz bilo kog para sekvenci (suseda svih k-mera iz druge dve sekvence) će biti sadržan motiv. Što znači da rađenjem preseka sa prethodnim skupom ne rizikuje izbacivanje motiva, već svih ostalih slučajnih zajedničkih suseda. Ono što smo ovde pokazali jeste da traženje motiva u skupu sekvenci može da se svede na računanje preseka suseda tri k-mera i odgovarajućom kombinacijom operacija nad skupovima. Računanje preseka

za svaka tri k-mera nije jeftino i ako je presek skupova $Q$ dovoljno mali, isplativije je da prođemo kroz sve potencijalne motive u njemu i za svaki testiramo da li je motiv, nego da prolazimo kroz sve parove ulaznih sekvenci. Testiranje da li je motiv se vrši prolaskom kroz sve sekvence i računanjem najmanjeg Hamingovog rastojanja u svakoj, ukoliko je najveći broj od njih manji od d, jeste motiv, ukoliko nije prelazi se na sledeći potencijalni motiv i na kraju na sledeću iteraciju.

## 6.2 Stablo pretrage

Ideja računanja preseka svih skupova deluje intuitivno, ali kao što se može pretpostaviti postoji preveliki broj kombinacija, samim tim vreme izvršavanja bi bilo preveliko. Zbog toga PMS5 algoritam dovodi ideju iz PMSPruna, kako bi se smanjio prostor pretrage. Pre svega, za svaki k-mer iz prve sekvence se konstruiše stablo sa svim njegovim susedima, gde ovaj k-mer predstavlja koren stabla. Stablo se gradi sistematično i pored k-mera svaki čvor ima i broj p, koji predstavlja indeks promene u toj nisci u odnosu na roditelja. Dubina stabla je d, tj. dozvoljenog broja mutacija, takođe na kojoj se dubini čvor nalazi govori o tome na koliko se karaktera razliku od k-mera u korenu. Ukoliko je čvor (t1,p1) dete čvora (t,p) važi da je p1 je veće od p, za 1 i da se svi karakteri osim onog na indeksu p1 poklapaju u t i t1. Koren stabla ima vrednost p=0.




Slika 7: Primer stabla suseda za 1010, d=2. Slika je preuzeta iz rada [11].

## 6.3 Linearno programiranje

Kada smo konstruisali stablo za k-mer iz prve sekvence, sledeći korak jeste obilazak istog i proveravanje da li se neki od čvorova nalazi u preseku suseda

k-mera iz druge dve sekvence. Provera da li se nalazi se vrši proverom Hamingovog rastojanja, ali pored toga želimo da uštedimo obilazak celog stabla ukoliko znamo da neki čvor nema potomka koji je potencijalan zajednički sused. U ovom trenutku su nam dostupne sledeće informacije: x, y, z i t(čvor iz stabla suseda čvora x do kog smo stigli u obilasku). Pošto znamo da je t potomak od x, a želimo da proverimo da li postoji njegov potomak koji može biti motiv, moramo da primetimo da su karakteri od indeksa p do kraja k-mera x i t isti, zbog pravila koja važe za stablo. Radi lakšeg označavanja, podelićemo sve sekvence na x1 i x2, y1 i y2.. gde je x1 je od početka do indeksa p, a x2 je od p+1 do kraja niske. Potomak t1 će zbog prethodno nabrojanih osobina imati isti prefiks (veličine p) kao t. Kako bismo utvrdili postojanje potomka koji može biti u preseku suseda uvodimo pomoćne promenljive n1, n2, n3, n4 i n5. Ovi brojevi predstavljaju brojače za kombinacije mogućih razlike (sličnosti) karaktera u tri k-mera koja posmatramo na konkretnom indeksu. Prvi slučaj je kada su sva tri karaktera jednaka, drugi je kada se karakter na indeksu i poklapa u niskama x2 i y2, ali se razlikuje u z2, treći slučaj je kada se poklapaju x2 i z2, a y2 se razlikuje, četvrti kada se y2 i z2 poklapaju a x2 razlikuje, i na kraju kada se sva tri karaktera na datom indeksu razlikuju. Promenljive broje ove slučaje ovim redom.

Kako znamo broj razlika koje imamo do indeksa p, sada proveravamo drugi deo niske i tražimo potencijalno w (dužine k-p) čiji zbir rastojanja od y i z upada u dozvoljeni opseg d. Za ovu proveru potrebne su nam i dodatne promenljive:

- $N_1, a$ - broj pozicija i Tipa 1 tako da w[i] = x2[i]
- $N_2, a$ - broj pozicija i Tipa 2 tako da w[i] = x2[i]
- $N_2, b$ - broj pozicija i Tipa 2 tako da w[i] = z2[i]
- $N_3, a$ - broj pozicija i Tipa 3 tako da w[i] = x2[i]
- $N_3, b$ - broj pozicija i Tipa 3 tako da w[i] = y2[i]
- $N_4, a$ - broj pozicija i Tipa 4 tako da w[i] = y2[i]
- $N_4, b$ - broj pozicija i Tipa 4 tako da w[i] = x2[i]
- $N_5, a$ - broj pozicija i Tipa 5 tako da w[i] = x2[i]
- $N_5, b$ - broj pozicija i Tipa 5 tako da w[i] = y2[i]
- $N_5, c$ - broj pozicija i Tipa 5 tako da w[i] = z2[i]

Pomoću ovih vrednosti možemo linearnom kombinacijom da odredimo $d_H(x2, w)$.

1. (n1-$N_1, a$)+(n2-$N_2, a$)+(n3-$N_3, a$)+(n4-$N_4, a$)+(n5-$N_5, a$)$\leq d_H(x1, t1)$
2. (n1-$N_1, a$)+(n2-$N_2, a$)+(n3-$N_3, a$)+(n4-$N_4, a$)+(n5-$N_5, a$)$\leq d_H(y1, t1)$
3. (n1-$N_1, a$)+(n2-$N_2, a$)+(n3-$N_3, a$)+(n4-$N_4, a$)+(n5-$N_5, a$)$\leq d_H(z1, t1)$
4. $N_1, a \leq$ n1
5. $N_2, a + N_2, b \leq$ n2
6. $N_3, a + N_3, b \leq$ n3
7. $N_4, a + N_4, b \leq$ n4
8. $N_5, a + N_5, b + N_5, c \leq$ n5
9. Sve vrednosti su celu brojevi

Pomoću informacija koje imamo od x, y, z i t možemo izračunati udaljenosti i tipove razlika, i pomoću ovih ograničenja možemo korišćenjem linearnog programiranja da utvrdimo da li ovakvo rešenje postoji, ako postoji nastavljamo da istražujemo decu čvora t, ukoliko ne postoji prestajemo i vraćamo se na čvor iznad i odatle nastavljamo obilaženje stabla. Većina čvorova će ovim biti odsečeno, što je naš cilj. Možemo primetiti da su vrednosti koje koristimo ovde ograničene, n-ovi sa vrednosti dužine motiva, a udaljenosti vrednosti sa vrednosti razlika. Iz ovog razloga moguće je unapred napraviti tabelu pre pokretanja samog algoritma i tako uštedeti vreme. Tabela će biti 8-dimenziona sa $(l+1)^5(d+1)^3$ [11].

Kombinovanjem ideja za traženje zajedničkih suseda preko obilaska stabla i odsecanja grana za koje smo sigurni da ne dovode do mogućeg motiva, kao i korišćenje preseka skupova i u slučaju male veličine skupa potencijalnih motiva prekidanje iteracija i provera tih motiva, dobijamo dve optimizacije algoritma koje nam mogu pomoći u ubrzavanju algoritma pogotovo ako je ulaz veliki.

# 7 Genetski algoritam za traženje motiva

Genetski algoritmi su aproksimativni i koriste se za rešavanje optimizacionih problema. Opšta ideja koju koriste jeste započinjanje sa slučajno izabranom populacijom instanci i iterativno menjanje istih različitim operacijama i njihovo ocenjivanje. Nakon završene poslednje iteracije vraća se instanca koja ima najveću ocenu. U nastavku je opisano kako je ovaj algoritam je prilagođen za rešavanje našeg problema.

## 7.1 Opis algoritma

Kao što je opisano ranije, za početak genomskog algoritma je potrebno odrediti kako se inicijalizuje populacija. U ovom algoritmu početna populacija se ne inicijalizuje slučajnim vrednostima, već sadrži sve kombinacije slova iz azbuke ['A', 'C', 'G', 'T'] dužine tri, što u našem slučaju iznosi 64 početnih reči. Ova inicijalizacija se pokazala kao vrlo dobra, jer pomoću nje pokrivamo sve moguće prefikse motiva, u suprotnom da smo inicijalizovali populaciju sa slučajnim vrednostima moglo bi da dođe do toga da favorizujemo neki početak motiva (ponavljanjem sličnih prefiksa) dok drugi kažnjavamo, što nam svakako nije cilj.

Drugi korak je određivanje fitnes funkcije pomoću koje ćemo vršiti poređenje i ocenjivanje jedinki. Radi lakšeg razumevanja najpre je potrebno da definišemo određene oznake. $S$ je skup ulaznih sekvenci, dok je $M$ njihov broj, $m$ u $S_m$ predstavlja indeks sekvence, $k$ predstavlja poziciju u sekvenci $S_m$ i oznaka se koristi u gornjem desnom uglu $S_m^k$ i za kraj $S_m^{ki}$ predstavlja karakter na indeksu $i$ podniske niske $S_m$ koja počinje na poziciji $k$. $P$ predstavlja skup motiva i $n$ se koristi za njihovo indeksiranje. $P_n^i$ je karakter na indeksu $i$ motiva $P_n$. L je dužina motiva. Fitnes ocena se računa po sledećoj formuli:

$$FS(P_n, S_m^k) = \left( \sum_{i=1}^{L} \text{poklapanje}(P_n^i, S_m^{ki}) \right) / L$$

$$\text{poklapanje}(P_n^i, S_m^{ki}) = \begin{cases} 1, & \text{ako } P_n^i = S_m^{ki} \\ 0, & \text{ako } P_n^i \neq S_m^{ki} \end{cases}$$

Iz ove formule možemo da primetimo da se fitnes funkcija za poređenje dve niske svodi na sumiranja mesta na kojima se one poklapaju i zatim de-

ljenjem sa dužinom motiva. Sledeći korak je pronalaženje fitnes ocene jednog motiva nad celom jednom sekvencom, i za tu vrednost se uzima maksimalna vrednost prethodnog poređenja tog motiva sa svim podniskama dužine L date sekvence.

$$FS(P_n, S_m) = \max_{k=1}^{K}\{FS(P_n, S_m^k)\}$$

Na kraju, kako bismo dobili fitnes ocenu za jedan motiv za sve ulazne sekvence sabiramo sve najbolje rezultate poklapanja iz pojedinačnih sekvenci i delimo ga sa njihovim brojem.

$$TFS(P_n, S) = \frac{1}{M}\sum_{m=1}^{M} FS(P_n, S_m)$$

Operacije koje se koriste za menjanje trenutnih jedinki su sledeće:

1. Mutacija - na slučajan način se izabere pozicija i karakter sa kojom će se promeniti već postojeći na tom mestu. Ono što je bitno da se istakne jeste da se prva tri karaktera, koja su inicijalno postavljena, ne menjaju, kako bismo održali raznovrsnost jedinki i tako sprečili upadanje u lokalni optimum.

2. Dodavanje - jedan, na slučajan način izabran nukleotid na početak jedinke, i jedan na kraj, zatim poredimo njihove ocene koje vraća fitnes funkcija i preživljava jedinka sa većom ocenom i takva se vraća u populaciju.

3. Brisanje - vrši se jednostavnim odsecanjem poslednjeg nukleotida.

Na ulazu algoritma dobijamo listu sekvenci, standardnu dužinu motiva, maksimalni broj iteracija i broj koji predstavlja verovatnoću da se desi mutacija.

Kao što smo istakli ranije, prvi korak jeste inicijalizacija populacije i zatim dokle god ne dođemo do tražene dužine motiva u iteracijama dodajemo nukleotide i vršimo operacije sa određenom verovatnoćom. Onda sledi računanje i pamćenje fitnes ocene. Na kraju algoritam vraća najbolji pronađeni motiv, ali ne može da garantuje da će to biti traženi, usađeni, motiv. Možemo da primetimo da se ovaj algoritam bazira na slučajnoj pretrazi i da se može optimizovati paralelnim izvršavanjem [12].

# 8 Slučajna projekcija i maksimizacija očekivane verodostojnosti

Algoritmi lokalne pretrage se mogu koristiti kod traženja optimalnog rešenja nekog problema. Osnovna ideja ove heuristike je sledeća: prelaskom sa jednog rešenja na drugo, u prostoru pretrage, pokušavamo da nađemo ono koje maksimizuje unapred zadati kriterijum. Iako su ovi algoritmi jednostavni i imaju široku primenu u rešavanju različitih vrsta problema, oni takođe imaju i mane zbog kojih se nekada ne dolazi do najoptimalnijeg rešenja. Jedna od njih jeste zaglavljivanje u lokalnom optimumu. Ovo se dešava ukoliko algoritam dođe do rešenja koje je bolje od svih njegovih suseda, ali ne i od najoptimalnijeg rešenja u celom skupu pretrage. Drugi problem ovog pristupa jeste da uspešnost konvergiranja zavisi od inicijalnog rešenja. U najčešćem slučaju, inicijalno rešenje se bira na slučajan način, iako je očekivano da će nakon dovoljno velikog broja iteracija rešenje iskonvergirati do globalnog optimuma, ovo ne mora da bude tačno. Možemo da povežemo ove mane i da primetno da ukoliko inicijalno rešenje bude neki lokalni optimum ili blizu njega, algoritam će se tu zaustaviti. Kako bismo smanjili verovatnoću za ove slučajeve možemo ove algoritme unaprediti sa drugima koji rešavaju ove probleme. U ovom radu obrađen je algoritam za maksimizaciju očekivane verodostojnosti (*eng. expectation maximization - EM*) i njegova optimizacija slučajnom projekcijom.

## 8.1 Slučajna projekcija

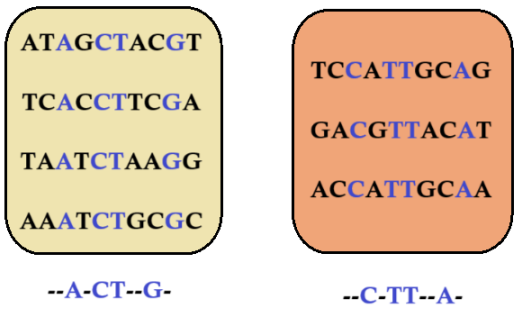
Algoritam slučajne projekcije je jedna od nekoliko verzija algoritma projekcija. Pored ove, postoje i njene optimizovane verzije uniformna projekcija i niska disperzija. Ovaj algoritam se ne koristi samostalno za otkrivanje motiva, već je osmišljen da bude prvi korak drugim algoritmima, tj. da se njegov izlaz koristi umesto slučajne početne inicijalizacije drugih algoritama koji se koriste za lokalu pretragu. Kao što je objašnjeno ranije, na taj način dobijamo veću šansu da dođemo do globalnog optimuma.

### 8.1.1 Opis algoritma

Ovi algoritmi se zasnivaju na grupisanju k-mera koji imaju iste karaktere na unapred određenim pozicijama. Važno je istaći da se ne proveravaju samo

uzastopna mesta motiva, već slučajno izabrane pozicije, zato što se mutacije u motivu mogu nalaziti na bilo kom mestu. Inicijalni korak algoritma je slučajan izbor $m$ brojeva u intervalu od 0 do dužine k (dužina traženog motiva) i ovi brojevi predstavljaju našu projekciju tj. indekse u nisci koji će biti fiksirani. Brojeve biramo u nadi da ćemo izabrati one pozicije na kojima u motivima u ulazim sekvencama nisu mutirani karakteri, već koji su zajednički za većinu k-mera koji jesu motivi. U idealnom slučaju, naša projekcija će izabrati indekse iz motiva koji nisu mutirani ni u jednoj ulaznoj sekvenci i tako će se svi motivi naći u istom skupu nakon projektovanja. Naravno, ovaj slučaj ima vrlo malu verovatnoću u praksi.

Nakon izabrane projekcije, prolazimo kroz sve podsekvence dužine k, iz svih ulaznih sekvenci i vršimo projekcije k-mera u odgovarajuće korpe. Svaka korpa će čuvati jedinstveni skup k-mera koje odgovaraju jednoj projekciji. Kako nam broj k-mera u jednoj korpi može biti dobar pokazatelj uspešnosti u otkrivanju motiva, sledeći korak je filtriranje onih korpi čiji broj k-mera u njoj prelazi unapred postavljenu granicu $s$.




Slika 8: Primer projekcije indeksa (2,4,5,8) u dve korpe

Ono što možemo da primetimo je da rezultat izvršavanja ovog algoritma potpuno zavisi kako od slučajno izabranih indeksa u inicijalnom koraku, kao i od vrednosti parametara $m$ i $s$. U slučaju da izaberemo indekse za koje se ispostavi da sadrže dosta mutiranih karaktera, naša pretraga može otići na pogrešnu stranu i time ćemo izgubiti šansu da dođemo do pravog motiva. Kako bismo povećali šansu da se ovo ne desi možemo više puta ponoviti

slučajno biranje indeksa i porediti rešenja iz svake iteracije. Važno je dobro izabrati i parametar $m$ koji predstavlja na koliko mesta naša projekcija pravi šablon, kako bismo dobili veću šansu da će šablon da se poklopi sa barem $s$ k-mera koji na tim mestima nisu mutirali. Parametar $m$ bi trebalo da bude manji od razlike k i d, jer bismo u suprotnom osigurali fiksiranje indeksa koji je mutiran. Naravno, ne odgovara nam da broj bude i previše mali, jer ćemo u tom slučaju imati i veliki broj slučajnih k-mera koji nisu motivi u korpi sa motivima. Očekivano je da u rezultujućoj korpi ne upadnu svi pravi motivi, i isto tako, da se u njoj nađu neki slučajni k-meri koji to nisu, ali sve dok prvih nema premalo, a drugih nema previše, rešenje ima potencijal. Granica za vrednost parametra $s$ bi trebalo da nam omogući da iz korpi koje imaju barem $s$ k-mera možemo doći do traženog motiva. Ovaj broj ne bi trebalo da bude previše veliki, jer to stvara mogućnost da odbacimo korpu sa motivima ukoliko projektovanje nije bilo najbolje [4].

Dobre strane ovog algoritma jesu te što ne povećava značajno vreme izvršavanja algoritma, jer njegovo izvršavanje ne zahteva da se proveravaju i porede svi mogući slučajevi motiva kao u nekim algoritmima. Takođe, prednost je što algoritam prelazi preko celog skupa sekvenci i vraća potencijalne motive, tako da može dođe do potencijalnog globalnog najboljeg rešenja. Ovo neutrališe manu algoritama lokalne pretrage, zaglavljivanje u lokalnom optimumu. Mane ovog algoritma su te što zavisi od parametara i od slučajno izabranih indeksa za projekciju.

Optimizacija ovog algoritma se svodi na biranje projekcije na sistematičniji način kako bi prostor bio pokriven uniformno, pa samim tim verovatnoća za pronalaženje dobrih šablona povećana. U radu [13] je istaknuto da je posledica pažljivijeg biranja šablona smanjenje broja korpi za 15-50%, što dovodi i do smanjenja vremena izvršavanja algoritma.

## 8.2 Maksimizacija očekivane verodostojnosti

Algoritam maksimizacije očekivane verodostojnosti je probabilistički algoritam koji se koristi za rešavanje problema kod kojih nedostaju informacije. Postupak je iterativno rešavanje niza problema u kojima se informacije koje nedostaju zamenjuju sa očekivanim vrednostima, zatim se očekivane vrednosti ažuriraju. Ova dva koraka koja se ponavljaju u svakoj iteraciji se označavaju ako E-korak (*eng. expectation*) i M-korak (*eng. maximization*).

### 8.2.1 Opis algoritma

Inicijalizacija je prvi korak i ovog algoritma i razlikuje se u slučaju ako ga implementiramo sa prethodnom optimizacijom ili ne. U slučaju da ovaj algoritam implementiramo bez slučajne projekcije, ili bilo koje druge optimizacije, prvi korak je da se iz svake sekvence iz ulazne liste na slučajan način biraju početni potencijalni motivi. Zatim se za njih pravi profilna matrica. U slučaju da algoritam radimo sa optimizacijom, motive ne tražimo slučajno već koristimo one koje dobijamo u korpama koje su prošle uslov za filtriranje (imaju više od $s$ k-mera). Za svaku korpu izvršavamo EM algoritam posebno i u ovom slučaju je sledeći korak računanje profilne matrice.

Ova matrica verovatnoća se računa na osnovu datih motiva tako što jednostavno prebrojimo za svaku poziciju k-mera koliko se koji nukleotid pojavljuje i podelimo sa ukupnim brojem nukleotida na tom mestu. Kako bismo izbegli da neka vrednost u ovoj matrici bude 0, dodajemo na sva mesta u matrici vrednost verovatnoće odgovarajućeg nukleotida u okolini. Iako u praksi zavisi od toga čiji DNK posmatramo, ovde se pretpostavlja da svi nukleotidi imaju jednaku verovatnoću u okolini. Sledeći korak predstavlja ažuriranje potencijalnih motiva korišćenjem profilne matrice, iz prethodnog koraka. Prolaskom kroz sekvence računamo verovatnoće svakog k-mer-a i onaj koji ima najveći odnos verovatnoće predstavlja novi potencijalni motiv iz te sekvence [14]. Odnos verovatnoće (*eng. likelihood ratio*) se računa po sledećoj formuli:

$$LR(W^*) = \frac{\Pr(T \mid W^*, P)}{\Pr(T \mid P)} \tag{1}$$

Gde $W^*$ predstavlja profilnu matricu, $P$ predstavlja verovatnoću nukleotida u okolini (količnik broja pojavljivanja svakog nukleotida u svim sekvencama i ukupan broj nukleotida u svim sekvencama), dok je $T$ oznaka za sve ulazne sekvence.

Računanja profilne matrica i odnosa verovatnoće se ponavlja sve dok se ne dostigne unapred postavljen broj iteracija ili dok je Euklidsko rastojanje dve uzastopne profilne matrice manje od unapred date granice. Rezultat ovih iteracija će biti lista motiva, iz svake sekvence po jedan. Kao što je već pomenuto, u slučaju optimizacije sa projekcijom, algoritam EM se radi za

svaku korpu i nad svim k-merima u njima. Slično, na kraju ćemo kao rezultat dobiti skup k-mera, po jedan iz svake ulazne sekvence, koji predstavljaju potencijalne motive.

Preostaje nam još jedan korak kako bismo dobili rešenje u obliku jednog motiva. Za njega prvo računamo profilnu matricu, kao i u inicijalnom koraku, od motiva koje smo prethodno dobili. Zatim definišemo ocenu odnosa verovatnoće na sledeći način:

$$LR(S_b) = \prod_{x \in S_b} \frac{\Pr[x \mid W_{S_b}]}{\Pr[x \mid P]} \tag{2}$$

Gde je $S_b$ skup motiva iz korpe b, $W_{S_b}$ profilna matrica izračunata pomoću motiva korpe b, i $P$ predstavlja verovatnoću nukleotida u okolini [4].

Krajnje rešenje izvršavanja ovog algoritma će biti konsenzus skupa motiva iz one korpe koja ima najveću ocenu odnosa verovatnoće iz svih iteracija. Iako je probabilistički, uz dovoljno veliki broj iteracija, i dobro postavljene argumente za projekciju možemo da očekujemo da će algoritam doći do traženog motiva.

# 9 Algoritam grube sile

Za kraj, obradićemo rešavanje ovog problema algoritmom grube sile, radi poređenje rezultata sa ostalim. Algoritmi grube sile imaju jednostavan pristup da ispituju sva moguća rešenja datog problema kako bi došli do optimalnog. Oni zasigurno dolaze do tačnog rešenja i zbog toga se često koriste, ali samo nad malim ulazima. Velika mana ovakvih algoritama je to što je za ispitivanje svih mogućih rešenja nekog problema potrebno mnogo resursa i zbog toga se ne koriste za rešavanje problema koji imaju velike ulaze.

## 9.1 Opis algoritma

Algoritam grube sile za traženje motiva je trivijalan, izvršavanje započinje generisanjem svih kombinacija niski dužine $l$ nad azbukom A, C, G i T. Broj ovih kombinacija zavisi od $l$ i azbuke, ali je često prevelik. Za nisku dužine 15 postoji $4^{15}$ kombinacija, što je jednako 1 073 741 824.

Sledeći korak je prolazak kroz sve izgenerisane niske i za svaku poziciju u svakoj ulaznoj sekvenci proveravamo Hamingovu udaljenost od naše niske. Ukoliko u svakoj sekvenci nađemo suseda koji se razlikuje od naše niske na najviše $d$ mesta, pronašli smo motiv. Kao što smo videli broj svih kombinacija niski je ogroman i zato ne možemo očekivati da se algoritam grube sile završi u prihvatljivom vremenu za velike ulaze.

# 10 Rezultati

U dodatku master rada priložene su implementacije opisanih algoritama, test primeri ulaznih sekvenci (za koje su prikazani rezultati) i Python skripta koja služi za pravljenje datoteke sa rezultatima linearnog programiranja za algoritam PMS5. U nastavku teksta ćemo uporediti rezultate koji se odnose na uspešnost pronalaska motiva i vreme izvršavanja algoritama.

## 10.1 Enumerativni

Pre upoređivanja rezultata izvršavanja opisanih programa, opisaćemo skup podataka koji je korišćen za njihovo testiranje. Ulazne sekvence su generisane na slučajan način tako što su zadate vrednosti broja sekvenci - t, njihovih dužina - m, dužina motiva - l i broja dozvoljenih mutacija - d. Sve sekvence su iste dužine u testiranim primerima, iako algoritmi mogu da se koriste i kada ovaj uslov nije ispunjen. Kada smo dobili ulazne parametre, možemo da izaberemo (m-l) nukleotida na slučajan uniforman način, tako da svi imaju istu verovatnoću. Zatim, na slučajan način generišemo nisku dužine l i u svaku sekvencu ubacujemo na slučajno mesto nisku koja sadrži do d razlika od početne. U ovom radu testirani parovi dužine motiva i dozvoljenih razlika su (8, 1), (11, 2), (13, 3) i (15, 4). Broj ulaznih sekvenci je 20, a njihove dužine su bile 300 i 600 karaktera. Računar na kome je vreme mereno ima procesor Intel od 2.10GHz i radnu memoriju od 32GB, dok je veličina diska malo manja od 1TB.

| (l, d) - m | Risotto | Glasački | Winnower k=2 | Winnower k=3 | Winnower k=4 | MITRA | PMS5 | Gruba sila |
|---|---|---|---|---|---|---|---|---|
| (8, 1) - 300 | 0.147 | 0.274 | 3.841 | 3.71 | 4.296 | 6.87 | 24.567 | 22.91 |
| (8, 1) - 600 | 0.92 | 0.311 | 7.168 | 7.87 | 7.73 | 26.90 | 223.82 | 52.85 |
| (11, 2) - 300 | 1.99 | 1.01 | 2.15 | 2.36 | 2.20 | 110.63 | 30.17 | 1647.24 |
| (11, 2) - 600 | 13.12 | 1.69 | 9.10 | 12.94 | 12.94 | 538.41 | 278.22 | – |
| (13, 3) - 300 | 28.55 | 11.56 | 6.63 | 25.92 | 29.37 | 1345.84 | 45.61 | – |
| (13, 3) - 600 | 157.65 | 25.15 | 801.09 | 668.73 | 671.94 | 6093.29 | 604.22 | – |
| (15, 4) - 300 | 355.25 | 258.86 | 867.73 | 592.92 | 584.19 | – | 125.91 | – |
| (15, 4) - 600 | 2518.28 | 559.50 | – | – | – | – | – | – |

Tabela 1: Vremena u sekundama potrebna za izvršavanje obrađenih algoritama

Ono što bi nas možda najviše iznenadilo gledajući tabelu sa vremenima, jeste to da vreme izvršavanja algoritma grube sile nije uvek najgore. Kao što smo opisali, gruba sila proverava sve moguće niske i zato deluje da ništa ne

može biti sporije od toga, ali zapravo nekada optimizovani algoritmi koriste kompleksne strukture ili imaju složene implementacije zbog čega se algoritam grube sile više isplati. Ovo je slučaj samo kada su ulazi manji, i kao što možemo da vidimo prednost koju on ima nestaje već u trećem redu, i to sa velikom razlikom. Takođe, neki algoritmi se nisu izvršili do kraja ni posle nekoliko sati, zbog toga su neka mesta u tabeli ostala nepopunjena.

Sledeća stvar koja se ističe u tabeli je to što se kod nekih algoritama vreme izvršavanja postepeno povećava kako se pomeramo po redovima. Primer za ovo je algoritam Risotto. Sa druge strane, neki algoritmi imaju 'stepenice', tj. vreme za istu dužinu motiva i isti broj mutacija se znatno razlikuje u zavisnosti od dužina ulaznih sekvenci. PMS5 je primer za ovaj slučaj. Možemo da zaključimo da na vreme izvršavanja prvih algoritama dužina niske i broj mutacija imaju veći uticaj od dužine sekvenci, za razliku od druge grupe algoritama.

Za algoritam Risotto možemo da primetimo da je vreme nekoliko puta veće za duplo duže sekvence, a istu dužinu motiva i broja razlika. Razlog ovome je jednostavan, algoritam neće moći da toliko često radi odsecanje grana i time smanjuje prostor pretraga kao pre, jer se sa duplo većim ulazom povećava šansa da je niska ispravna, tj. da je kvorum ispunjen. Ipak, algoritam je uspeo da se izvrši za sve date ulaze.

Ako posmatramo celokupnu uspešnost nad ovim testiranim podacima, modifikovan glasački algoritam se pokazao najbolje. Ovo se pogotovo vidi nad manjim dužinama motiva i malim brojem mutacija, razlog ovome je to što generisanje svih mogućih suseda potencijalnog motiva nije toliko iscrpna kao u dužim niskama i većim brojem promena.

Algoritam Winnower je testiran sa tri vrednosti uslova odsecanja (2, 3 i 4). Na manjim ulazima se dobro pokazao u sva tri slučaja, ali ono što nas može začuditi jeste kako su sva tri uslova dovela do istog vremena izvršavanja. Objašnjenje leži u broju iteracija, naime kada je korišćen uslov k=2 broj iteracija uklanjanja grana je bio 4, za k=3 je bio 2, dok je za k=4 kroz graf uklanjanje grana urađeno samo jednom. Ono zbog čega ovaj algoritam nije najefikasniji i ono što mu dosta vremena oduzima jeste konstrukcija grafa, pogotovo nad velikim ulazima, gde ima mnogo čvorova i još više slučajnih grana.

Algoritam MITRA se nije najbolje pokazao u ovom testiranju. Njegova najveća mana je glavna suština ovog algoritma, a to je čuvanje tabele u svakom čvoru stabla. Ovo je samo po sebi skupo memorijski i kada bismo pomislili da će odsecanje grana da nam pomogne setimo se da se odsecanje radi tek kada je prekoračen broj razlika, tj. vrednost d. Ovo nam govori da u prvih d koraka obilaska stabla u dubinu svih mogućih kombinacija nukleotida ni jedna grana neće biti odsečena.

Poslednji algoritam u ovoj grupi je PMS5, algoritam koji kombinuje obilazak stabla i linearno programiranje se nije pokazao najbolje nad datim test primerima. Ono što se može primetiti jeste velika razlika između sekvenci dužine 300 i 600. Ovo ima smisla jer se iteracije rade po svim k-merima u početnim sekvencama, i to u trostrukoj petlji. Razlog većeg vremena od većine ostalih algoritama je upravo njegova kompleksnost, složenost njegovih struktura i provera koja nije isplativa za manje ulaze.

Pored ovih ulaza, na algoritmu Risotto i Winnower (k=3) je testiran još jedan skup ulaznih sekvenci koje u sebi imaju 6 nukleotida, pored A, C, G i T imaju i R i Y. Broj ulaznih sekvenci je takođe 20, dužine su 300 i 600, a testirana dužina motiva je 11 sa 2 mutacija. U tabeli su predstavljena vremena izvršavanja, kao što možemo da primetimo vreme za prvi algoritam se povećalo dok se za drugi smanjilo. Razlog ovih promena je u njihovim implementacijama. Kako Risotto konstruiše i obilazi stablo sa svim mogućim karakterima iz azbuke, dodavanjem novih, stablo i prostor za obilazak je veći pa se i vreme povećalo. Sa druge strane, dodavanjem novih karaktera u azbuku smanjuje broj slučajnih grana u grafu u algoritmu Winnower, pa se samim tim i algoritam ubrzava.

| (l, d) - m | Risotto | Winnower k=3 |
|---|---|---|
| (11, 2) - 300 | 3.59 | 1.41 |
| (11, 2) - 600 | 29.25 | 5.46 |

Tabela 2: Vremena u sekundama za izvršavanje nad azbukom (A, C, G, T, R, Y)

## 10.2 Genomski i aproksimativni

Preostala dva algoritma nisu egzaktna pa je poređenje njihovih rezultata malo teže. Genomski algoritam se vrlo dobro pokazao pri pokretanju. Za broj iteracija uzimali smo vrednost od 100 do 500, a verovatnoća mutacija je bila 0,2. Evolucioni algoritmi, kao i probabilistički, mnogo zavise od inicijalizacije, i u slučaju loše, vrlo lako mogu da se zaglave u lokalnom optimumu. Rešenje koje je primenjeno ovde jeste inicijalizacija populacija na sve kombinacije dužine tri. Na taj način naš inicijalni skup sigurno pokriva i motiv i odatle krećemo sa iterativnim traženjem najboljeg rešenja. Ono što se često dešavalo jeste da umesto motiva, algoritam nađe deo njega bez prvog ili zadnjeg dela, tj. pomeren motiv.

Na kraju, kao poslednji algoritam je kombinacija slučajne projekcije i maksimizacije očekivane verodostojnosti. Kao što je u radu već istaknuto, algoritmi lokalne pretrage imaju tendenciju ka zaustavljanju u lokalnom maksimumu. Opisaćemo primer koji je inspirisan primerom iz rada [4] i na kome je lepo pokazano kako pozicije mutacija motiva utiču na uspešnost pronalaska istog algoritmom slučajne pretrage. Ukoliko imamo dva skupa motiva koji imaju istu konsenzus nisku, u ovom slučaju je to niska ACAGTCG. Oba skupa sadrže po 5 motiva dužine 7 i svaki od njih se razlikuje na dva mesta od konsenzus niske, tako da su ovi skupovi isti po broju mutacija.

| A | B |
|---|---|
| ACAtcCG | AgAGTaG |
| ACAacCG | ACAGgCt |
| ACAcaCG | ACccTCG |
| ACAtaCG | tgAGTCG |
| ACAccCG | gCAGTCc |

Tabela 3: Dva skupa motiva koji imaju isti konsenzus ACAGTCG

Ono što ih razlikuje jeste informacija koju imamo o delu koji nije podlegao mutaciji, o takozvanoj pozadini motiva. Lako može da se uoči da se mutacije u svim motivima u skupu A nalaze na istim indeksima, zbog čega je pozadina motiva potpuno očuvana. Za razliku od njega, u skupu B je informacija o pozadini slabo očuvana, jer se mutacije pojavljuju na različitim mestima. Kako algoritmi lokalne pretrage započinju slučajnim izborom rešenja i zatim pokušavaju da nađu bolje koje liči na početno, mutacije u

pozadini motiva mogu da povuku pretragu na pogrešnu stranu i zbog toga da dovedu zaustavljanje na lokalnom optimumu.

Zbog ovoga algoritam EM ne daje dobre rezultate kada se izvršava sam tj. sa slučajnom inicijalizacijom, ali isto tako dobro biranje slučajnih projekcija, koje se slikaju u većinu nemutiranih karaktera, može da bude znatno otežano. Uslov za filtriranje korpi sa projekcijama se može isplatiti ako se parametri pažljivo izaberu i to uz izvršavanje dovoljnog broja iteracija. Upravo zbog gornjih otežavajućih okolnosti, vrednost s, koja se koristi za filtriranje korpi, ne bi trebalo da bude prevelika, kako ne bismo preskočili korpu koja dobro projektuje motiv.

Ovaj algoritam se bolje pokazao nad manjim ulazima, jer kao što se može pretpostaviti, veći ulaz dovodi do punijih korpi sa niskama koje slučajno upadaju u njih. One mogu 'uprljati' našu matricu verovatnoće u sledećem koraku i odvesti pretragu na pogrešnu stranu.

# 11 Zaključak

Problem traženja usađenog motiva je poznat i za njegovo rešavanje postoje više stotina algoritama. Prikazan skup je samo mali deo različitih pristupa koji bi trebalo da čitaocima približi raznovrsnost i kreativnost rešavanja ovog problema. Iako postoji veliki broj algoritama i njihovih optimizovanih verzija, ni jedan se ne može izdvojiti kao najuspešniji. Razlog je taj što svi imaju jače i slabije strane, ali svuda jače dolaze po određenoj ceni koja često nije mala. Uspešnost pronalaska motiva i potrebno vremene za to zavisi od konkretnih podataka na ulazu. Neki algoritmi su uspešniji u traženju kraćih motiva, ali imaju poteškoća u traženju dužih. Postoje i oni na koje dužina motiva ne utiče na uspešnost, ali utiče na vreme izvršavanja. Takođe, na određene algoritme veličina ulaznih sekvenci ima veliki uticaj, dok na druge nema. Iz ovog razloga je važno da imamo na umu koliko je bitno za konkretan slučaj da pronađemo tačne motive i po koju cenu vremenskih i prostornih resursa.

Većina algoritama koji rešavaju ovaj problem se ne mogu koristiti u praksi, jer se ne može garantovati tačan broj mutacija u motivima. Međutim, proučavanje različitih metoda i njihovih kombinacija može biti korisno i može nas dovesti do boljih ideja. Zbog toga se objavljuje stotine radova godišnje u kojima se opisuju novi pristupi rešavanja ovog zadatka, iako ovaj problem nije nov.

# Literatura


[1] Jason T. L. Wang, Mohammed J. Zaki, Hannu Toivonen, and Dennis Shasha. *Data mining in bioinformatics*. Springer, 2005.

[2] Nasih Qader Nooruldeen and Khafaji Hussein. Motif discovery and data mining in bioinformatics. *International journal of computers & technology*, vol. 13(1):4082–4095, 2014.

[3] Mohanty Satarupa, Kumar Pattnaik Prasant, Abdulhakim Al-Absi Ahmed, and Kang Dae-Ki. A review on planted (l, d) motif discovery algorithms for medical diagnose. *Sensors*, vol. 22(3):1– 27, 2022.

[4] Buhler Jeremy and Martin Tompa. Finding motifs using random projections. *Journal of computational biology*, (9):225–242, 2002.

[5] Fatma A. Hashim, Mai S. Mabrouk, and Al-Atabany Walid. Review of different sequence motif finding algorithms. *Avicenna Journal of Medical Biotechnology*, (11):130–148, 2019.

[6] Jose R. Correa, Alejandro Hevia, and Marcos Kiwi. *LATIN 2006: Rheoretical Informatics*, pages 757–768. 2006.

[7] Francis Y.L. Chin and Henry C.M. Leung. Voting algorithms for discovering long motifs. page 261–271, 2005. Proceedings of the 3rd Asia-Pacific Bioinformatics Conference.

[8] Pavel A. Pevzner and Sing-Hoi Sze. Combinatorial approaches to finding subtle signals in dna sequences. vol. 8:269–78, 2000. Proc. Int. Conf. IntelligentSystems For Molecular Biology.

[9] Liang Shoudan, Pratim Samanta Manoj, and Bryan A. Biegel. cwinnower algorithm for finding fuzzy dna motifs. *Journal of Bioinformatics and Computational Biology*, (2):260–265, 2003. Proc IEEE Comput Soc Bioinform Conf.

[10] Price Alkes, Ramabhadran Sriram, and Pavel A. Pevzner. Finding subtle motifs by branching from sample strings. *Bioinformatics*, (19):149–155, 2003.

[11] Dinh Hieu, Rajasekaran Sanguthevar, and Vamsi K. Kundeti. PMS5: an efficient exact algorithm for the (l, d)- motif finding problem. *Bioinformatics*, vol. 12:260–265, 2011.

[12] Fana Yetian, Wua Wei, Liua Rongrong, and Yang Wenyu. An iterative algorithm for motif discovery. *Procedia Computer Science*, (24):25–29, 2013.

[13] Wang Xun, Miao Ying, and Cheng Minquan. Finding motifs in dna sequences using low-dispersion sequences. *Journal of computational biology*, vol. 21(4):320–329, 2014.

[14] Charles E. Lawrence and Andrew A. Reilly. An expectation maximization (EM) algorithm for the identification and characterization of common sites in unaligned biopolymer sequences. *Proteins Struct. Funct. Bioinform*, (7):41–51.

[15] Huo Hongwei, Zhao Zhenhua, Stojkovic Vojislav, and Liu Lifang. Optimizing genetic algorithm for motif discovery. *Mathematical and Computer Modelling*, (52):2011– 2020, 2010.

[16] Eskin Eleazar and Pavel A. Pevzner. Finding composite regulatory patterns in dna sequences. *Bioinformatics (Oxford, England)*, vol. 18(1):354–363, 2002.

[17] Liu Xiaowen, Ma Bin, and Wang Lusheng. Voting algorithms for the motif finding problem. page 37–47, 2008. Proceedings of the 3rd Asia-Pacific Bioinformatics Conference.