SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Bionformatika 2 - projekt za 100 bodova

Poravnanje parova sljedova korištenjem HMM

Matija Pintarić i Jakov Krčadinac

Zagreb, siječanj 2025.

SADRŽAJ

1. Uvod

1.1. Poravnanje nukleotida

Poravnavanje nukleotida je proces uspoređivanja sekvenci nukleinskih kiselina (DNA ili RNA) kako bi se identificirale sličnosti i razlike među njima. Obično se radi na molekularnoj razini kako bi se razumjele genetske veze, evolucijska povijest i funkcionalni odnosi među organizmima ili genima. Postoji nekoliko vrsta porananja:

- **Globalno poravnavanje:** Uspoređuje cijele sekvence od početka do kraja, čak i ako su različitih duljina. Koristi se kada su sekvence slične i podjednake duljine.
    - **Algoritam:** Needleman-Wunsch.
- **Lokalno poravnavanje:** Traži slične dijelove unutar sekvenci, ignorirajući dijelove koji se ne podudaraju. Pogodno za sekvence koje dijele samo određene regije sličnosti.
    - **Algoritam:** Smith-Waterman.
- **Višestruko poravnavanje:** Istovremeno uspoređuje više sekvenci kako bi se pronašli zajednički motivi ili konzervirani dijelovi.

Neke od svrha poravnavanja nukleotida uključuju proučavanje filogenetskih odnosa i rekonstrukciju evolucijskog stabla, a u komparativnoj genomici usporedbom sekvenci među organizmima moguće je identificirati konzervirane regije koje ukazuju na esencijalne biološke funkcije. Poravnavanje također omogućava detekciju točkastih mutacija, brisanja, umetanja i drugih genetskih promjena, dok se u medicini može koristiti za otkrivanje potencijalnih meta za terapije ili rezistencije na lijekove. U ovom smo radu implementirali poravnavanje sljedova nukleotida korištenjem skrivenog Markovljevog modela, koje se temelji na Viterbijevom algoritmu. Istestirali smo naš algoritam s nekoliko različitih skupova parametara modela. Prvo smo pokrenuli Viterbijev algoritam na slučajnim vrijednostima parametara, zatim na statistički estimiranim vrijednostima parametara, te smo naposljetku procjenili parametre Baum-Welchovim algoritmom, prvo mu dajući slučajne parametre kao početne, te onda estimirane, pa smo s istreniranim parametrima ponovno pokrenuli Viterbijev algoritam. Za kraj smo koristeći isti kriterij bodovanja usporedili uspješnost poravnanja naše implemen-

tacije Viterbijevog algoritma sa svim mogućim načinima postavljanja parametara modela i gotove implementacije Needleman-Wunschovog algoritma.

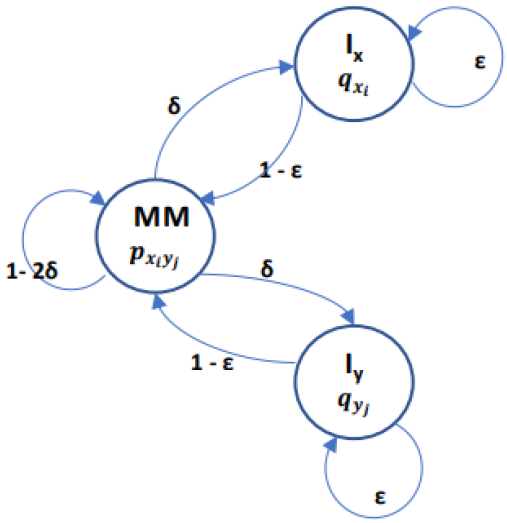
1.2. Skriveni Markovljevi modeli (HMM)

Skriveni Markovljevi modeli (HMM, Hidden Markov Models) su statistički modeli koji se koriste za analizu i modeliranje sekvenci podataka u kojima su stanja sustava skrivena, dok su opažanja (promatrani podaci) dostupna. U kontekstu poravnanja sekvenci, skriveni Markovljevi modeli koriste se za analizu bioloških sekvenci (DNA, RNA, proteini), odnosno u našem slučaju virusa HIV-a [1], kako bi identificirali homologije, evolucijske odnose i konzervirane regije. HMM omogućuje modeliranje sekvenci kao stohastičkih procesa s nizom stanja, pri čemu svako stanje odgovara različitom dijelu poravnanja. Budući da je HMM stohastičke prirode, provođenje ovog algoritma neće nužno dati najbolje moguće poravnanje. Prednost korištenja ovog modela je njegova složenost zbog koje ga se isplati koristiti za velike sljedove nukleotida kada nam je dovoljno približno optimalno poravnanje. Model kojim poravnavamo sljedove nukleotida može se nalaziti u sljedećim stanjima:

- **Match/mismatch (MM)**: Riječ je o paru nukleotida iz oba slijeda koji su isti (npr. AA) ili se razlikuju (npr. AG). Dolazi do emisije oba nukleotida.
- **Nukleotid i praznina (Ix)**: Imamo par u kojem se nalazi nukleotid iz prvog slijeda, a u drugom slijedu je na tom indexu praznina, odnosno znak '-' (npr. A-). Emitira se samo nukleotid.
- **Praznina i nukleotid (Iy)**: Imamo par u kojem se nalazi praznina iz prvog slijeda, a u drugom slijedu nukleotid (npr. -A). I ovdje se emitira samo nukleotid.

Parametri HMM-a:

- **pi**: Početne vjerojatnosti, odnosno vjerojatnosti da model započne u svakom od 3 moguća stanja.
- **A**: Vjerojatnosti prijelaza iz trenutnog stanja u sljedeće. Budući da imamo 3 stanja, ovaj parametar je matrica 3x3.
- **E**: Vjerojatnosti emisije pojedinih simbola. Za stanje MM postoji 16 mogućih emisija (emitiraju se 2 nukleotida) i svaka od njih ima određenu vjerojatnost. Stanja Ix i Iy imaju po 4 moguće emisije (svaki od nukleotida sa prazninom), stoga se u parametru E nalazi 8 vjerojatnosti za ta dva stanja.



Slika 1.1: moguća stanja i vjerojatnosti prijelaza u modelu bez početnog (START) i konačnog (END), Durbin et al. 2002, Ch. 4.

2. Procesiranje podataka, procjena početnih parametara

Projekt smo započeli preuzimanjem podataka u .fasta formatu, odnosno višestruko poravnatih sljedova nukleotida virusa HIV-a. Preuzete sekvence nalaze se u mapi data u datoteci *HIV1_ALL_2022_genome_DNA.fasta*.

2.1. Priprema sljedova

Za procesiranje, odnosno pripremu preuzetih korištena je skripta `processing.py` iz mape `python`. Ta Python skripta sadrži nekoliko bitnih funkcija:

- **load_data** – učitava sve sljedove iz `.fasta` datoteke i odvaja ih u 3 skupa sljedova: skup za početne estimacije parametara, veličine 20 sekvenci, od kojih ćemo dobiti 190 parova, skup za testiranje i skup za treniranje, oba dva veličine 7 sekvenci, od kojih dobivamo 21 par.
- **make_pairs** – prima skup (listu) koji sadrži sekvence i uparuje ih
- **MSA_TO_PSA** – poravnava sekvence iz „multiple sequence alignmenta“ u „pairwise sequence alignment“ na način da eliminira prazninu kada se nalazi na istom indexu u obje sekvence (npr. iz `C-AGG` i `A-AGG` nastaje `CAGG` i `AAGG`)
- **remove_all_dashes** – eliminira sve praznine (znakove −) iz para sekvenci, funkcija korištena kao pomoćna unutar `MSA_TO_PSA`
- **write_data** – zapisuje parove sekvenci u proizvoljne datoteke

2.2. Procjena početnih parametara

Nakon pripreme podataka, moguće je izračunati procjene početnih parametara π, A i E. Za to se koristi skripta `estimate.py` koja se također nalazi u mapi `python`. Za procjenu parametara uzimaju se parovi iz mape `estimate` koja se nalazi u mapi `train_data`. U nastavku slijedi opis najbitnijih funkcija iz `estimate.py`:

- **initialize_A**, **initialize_pi** i **initialize_e** – funkcije za inicijaliziranje potrebnih podatkovnih struktura za svaki parametar. Parametar π koristi riječnik, dok A i E koriste rječnike čije su vrijednosti rječnici.
- **update_pi** – bilježi frekvenciju početnog stanja gledajući sve parove sljedova
- **update_A** – bilježi frekvencije prijelaza iz jednog stanja u drugo iz svih parova sljedova
- **update_E** – bilježi frekvencije svih opaženih parova nukleotida (`'AA'`, `'-C'`, `'GC'` i slično) pri prolaženju kroz svaki od parova sljedova
- **calculate_A**, **calculate_pi**, **calculate_E** – pretvaraju vrijednosti u rječnicima u vjerojatnosti na način da ih dijele s ukupnim brojem vrijednosti (n)
- **write_probs** – služi za zapisivanje dobivenih vjerojatnosti u odgovarajuće `.txt` datoteke koje se nalaze u mapi `estimate` koja je unutar mape `transmission_values`

Nakon spremanja parametara, odnosno odgovarajućih vjerojatnosti u odgovarajuće `.txt` datoteke, parametri se statistički estimiraju prebrojavanjem prijelaza stanja i njihovih izlaza. Za estimaciju parametara koristili smo 190 parova sljedova nukleotida iz mape `estimate` koja je u mapi `train_data`. Isto tako, u mapi `transmission_values` dodajemo i mapu `random`, u kojoj za svaku kombinaciju početnog stanja, trenutnog stanja i izlaza iz tog stanja dodjeljujemo jednaku vjerojatnost.

3. Optimizacija parametara (Baum-Welch)

Baum-Welch algoritam je iterativna metoda koja se koristi za treniranje skrivenih Markovljevih modela (HMM) procjenom parametara modela (vjerojatnosti početnog stanja, prijelaza i emisije). To je poseban slučaj Expectation-Maximization (EM) algoritma.

Ključne vjerojatnosti ovog algoritma:

- $\alpha_t(i)$: Vjerojatnost da je model u vremenskom trenutku t u stanju i ($i \in S, 0 \leq t \leq T$)
- $\beta_t(i)$: Vjerojatnost da je model u vremenskom trenutku t u stanju i te da generira preostali opaženi slijed.
- $\gamma_t(i)$: Vjerojatnost da je model u vremenskom trenutku t u stanju i, s obzirom na opaženi niz opažanja $observations$ i trenutne parametre modela.
- $\xi_t(i, j)$: Vjerojatnost da je model u vremenskim trenucima t i $t + 1$ u stanjima i i j, s obzirom na niz opažanja $observations$ i trenutne parametre modela.

Opis implementacije Baum-Welch algoritma

U nastavku slijedi opis naše implementacije Baum-Welch algoritma podijeljen na glavne korake.

Inicijalizacija

U ovisnosti o početnim vrijednostima parametara, vrijednosti parametara modela se učitavaju iz mape `transmission_values/estimate` ili `transmission_values/random`. Vjerojatnosti početnog stanja učitavamo u vektor, a vjerojatnosti prijelaza i vjerojatnosti emisije u matricu (vektor vektora). Budući da matrica emisije nije prava matrica, odnosno nema konzistentne dimenzije $a \times b$, moramo ju nadopuniti nulama kako ne bismo imali pogrešku kada u algoritmu iteriramo kroz nju. Mogućih emisija ima 24 (16 za stanje MM i po 4 za

stanja Ix i Iy), stoga nam je ažurirana matrica E dimenzija 24×3, sa nulama za emisije koje sigurno neće biti opažene u tom stanju (npr. u stanju MM nikada neće postojati emisija A-, stoga na to mjesto u matrici dolazi 0.0).

Odlučili smo raditi s matricama, a ne rječnicima, stoga prilikom iterativnog učitavanja parova sljedova moramo sve opservacije pretvoriti u integere. Funkcija `readAndPairSequences` učitava iz zadanog `path`-a par sekvenci, prolazi kroz njih i vraća vektor uparenih nukleotida na koji su na istom indexu. U sljedećem koraku taj vektor uparenih nukleotida pretvaramo u vektor integera pomoću vektora `possibleObservations`, gdje su sve moguće opservacije poredane po istom redu kao i moguće vjerojatnosti emisije u matrici E. Sada kad su svi potrebni podaci ispravno učitani, mogu se uvrstiti u funkciju `baumWelch` koja provodi algoritam.

Provođenje Baum-Welch algoritma

Kada smo inicijalno napravili implementaciju ovog algoritma, slijedili smo pseudokod (odnosno formule) iz prezentacija koje su vezane za ovu tematiku. Prilikom testiranja na vrlo kratkom nizu opservacija (`testNuclPairs`) algoritam je radio dobro, međutim kada smo počeli učitavati prave parove sekvenci koje zapravo koristimo za učenje, svi su se parametri modela nakon učenja ispunili NaN vrijednostima.

Razlog je bio što algoritam u nekoliko koraka dijeli i množi učitane vjerojatnosti, a kada se to provodi preko 9000 puta (svaki put za jednu opservaciju u nizu) dobijemo brojeve koji su iznimno mali. Taj problem je očekivan za duge nizove, te zato vjerojatnosti i način izračuna moramo prebaciti u logaritamski prostor. Ovaj smo problem riješili koristeći logaritmirane vjerojatnosti koje su se potom zbrajale, odnosno, ovisno po potrebi formule, uvrštavale u `logSumExp` funkciju iz `helper.cpp` datoteke. Ta funkcija nad zadanim nizom logaritmiranih vjerojatnosti `logValues` provodi sljedeći izračun:

$$\text{logSumExp} = \text{maxLog} + \log\left(\sum e^{(\text{logValues}_i - \text{maxLog})}\right)$$

U prvom koraku provodi se iterativni izračun vrijednosti matrice α koristeći forward algoritam, a u drugom koraku se računaju vrijednosti matrice β koristeći backward algoritam. Nakon toga se prolaženjem kroz svih T opservacija računaju elementi γ i ξ koristeći već izračunate α i β. U zadnjem koraku se pomoću izračunatih γ i ξ ažuriraju parametri modela (vjerojatnosti početnih stanja π, prijelaza A i emisije E).

U izračunu novih parametara A i E dolazi do dijeljenja logSumExp vrijednosti elemenata γ za svako stanje, ali postoji opasnost da je dobiveni nazivnik -INF vrijednost što bi poremetilo parametre A i E. Zbog toga moramo napraviti provjeru korištenjem funkcije `isfinite` iz biblioteke `math.h`.

U funkciji `main` postavili smo da Baum-Welch algoritam ažurira vrijednosti parametara za svaki par sljedova nukleotida iz mape `baum_welch_train`. Kroz sve sljedove prolazi se `MAXITER` puta, čime reguliramo trajanje i stupanj učenja iz svih sljedova.

Normalizacija Parametara

Nakon svakog ažuriranja Baum-Welch algoritmom, primijetili smo da vjerojatnosti u parametru E nisu normalizirane, odnosno da suma svih vjerojatnosti emisija za stanje MM, suma svih vjerojatnosti za Ix i suma svih vjerojatnosti za MM nisu jednake 1. Zbog toga smo uveli novi korak (`normalize_E`) u kojem se sve ove vjerojatnosti normaliziraju tako da im je, poštujući proporcije, suma jednaka točno 1. U originalnoj implementaciji Baum-Welcha ovaj korak ne postoji, ali primijetili smo da nam on nikako ne narušava kvalitetu rezultata, već povećava, stoga smo ga odlučili zadržati.

4. Viterbijev algoritam

Viterbijev algoritam za poravnanje sekvenci koristi tri matrice za praćenje vjerojatnosti (Vmm, Vx i Vy), gdje svaka matrica predstavlja Viterbijevu matricu za određeno stanje (MM, Ix, Iy). Koriste se i odgovarajuće "tracker" matrice za praćenje najboljeg puta do svakog stanja. Ovaj prošireni algoritam, koji je vrlo sličan klasičnom Viterbijevom algoritmu, koristi logaritamske vrijednosti za izbjegavanje numeričkog podljeva pri radu s malim vjerojatnostima. Radi jednostavnosti, stanje MM predstavljamo s integerom 0, stanje Ix sa 1, a stanje Iy sa 2.

$$\log V^{M}(i,j) = \log p_{x_i y_j} + \max\begin{cases} \log(1-2\delta) + \log V^{M}(i-1,j-1) \\ \log(1-\varepsilon) + \log V^{Ix}(i-1,j-1) \\ \log(1-\varepsilon) + \log V^{Iy}(i-1,j-1) \end{cases}$$

$$\log V^{Ix}(i,j) = \log q_{x_i} + \max\begin{cases} \log\delta + \log V^{M}(i-1,j) \\ \log\varepsilon + \log V^{Ix}(i-1,j) \end{cases}$$

$$\log V^{Iy}(i,j) = \log q_{y_j} + \max\begin{cases} \log\delta + V^{M}(i,j-1) \\ \log\varepsilon + V^{Iy}(i,j-1) \end{cases}$$

Slika 4.1: Pseudokod popunjavanja matrica V za Viterbijev algoritam

Iz priloženoga pseudokoda se može primjetiti da nam je za izračun novih vrijednosti matrica V potreban samo trenutačni (indeks i) i prijašnji (indeks i-1) redak matrice, pa radi sačuvanja memorije, ne stvaramo cijele matrice, nego samo vektor za trenutačni i prijašnji red. Prije pokretanja algoritma, provodimo inicijalizaciju po priloženoj slici. Pošto smo Viterbijeve i tracker matrice implementirali pomoću vektora već zadanih veličina, ne inicijalizirane pozicije punimo s različitim vrijednostima, koje ne doprinose izračunu, jer će ih algoritam promijeniti ako je potrebno, da izbjegnemo stvaranje "garbage" vrijednosti.

$$V^{M}(0,0) = 0,\ V^{Ix}(0,0) = -\infty,\ V^{Iy}(0,0) = -\infty$$

Slika 4.2: Inicijalizacija Viterbijevih matrica

Vrijednosti Vmm, Vx, Vy računamo tako što prolazimo kroz sve moguće parove (`i, j`)

ulaznih neporavnatih sekvenci, te po pseudokodu mijenjamo pripadne vrijednosti. Isto tako, u ovisnosti koja je max vrijednost koraka za svaku od V matrica, dodajemo vrijednost 0, 1 ili 2, ovisno o tome koje je stanje dalo najveću vjerojatnost, na (`i, j`) indeks pripadne tracker matrice.

Primjer: u koraku za promjenu Vmm vrijednosti, najveća vrijednost dolazi iz Vx matrice. U tracker od matrice Vmm dodajemo stanje 1 (Ix). Isto tako, u izrazi za Vy, najveća vjerojatnost dolazi iz Vmm matrice, pa u tracker matricu stanja Iy dodajemo vrijednost 0 (MM) na pripadni indeks.

Nakon što prođemo kroz sve vrijednosti (`i`) i (`j`) para sekvenci, mijenjajući vrijednosti Viterbijevih matrica po pseudokodu, dobivamo posljednji redak Viterbijeve matrice za svako stanje, te tri ispunjene tracker matrice. Iz svakog retka uzimamo zadnju vrijednost, te gledamo za koje je stanje najveće. U ovisnosti o najvećem stanju, uzimamo njegovu tracker matricu. Zatim, krećući od kraja sekvenci u našem paru, postavljamo (`i = duljina prve sekvence`) i (`j = duljina druge sekvence`). Sljedeći korak je gledanje vrijednosti tracker matrice na indeksu (`i, j`) . Ako je vrijednost 0, to znači da smo došli iz stanja MM (match mismatch), te na varijable rezultata, u kojima spremamo poravnanja, dodajemo nukleotide iz obe sekvence i smanjujemo varijable (`i`) i (`j`) za 1. Zatim prebacujemo pointer na tracker matricu stanja 0. Sličan proces radimo i kada su vrijednosti tracker matrice 1 ili 2, gdje u ovisnosti o stanju smanjujemo vrijednost ili (`i`) ili (`j`), te dodajemo nukleotid u jednu varijablu poravnanja, a prazninu u drugu. Kada prođemo kroz sve nukleotide, obrnemo varijable poravnjanja, zbog toga što smo ih generirali od kraja sekvenci, te dobivamo globalno poravnanje.

Primjer:

Ulazne sekvence: AAT, CATG. i = 3, j = 4

Varijable poravnjanja: align1 = "", align2 = ""

Maksimalna vrijednost posljednjih redova matrica Vmm, Vx, Vy na posljednjem indeksu dobiva se za Vy. Uzimamo tracker matricu stanja Iy. Vrijednost na (`i, j`) je 2. Radimo (`j-=1`) i dodajemo G na align2, te - na align1. Sljedeća tracker matrica je opet od Iy, jer je stanje bilo 2. No, na novom indeksu će biti stanje 0. Radimo (`i-=1, j-=1`) te dodajemo T u align2, i T u align1. Nakon što provedemo algoritam do kraja, dobivamo align1 = -TAA, align2= GTAC, te obrnemo rezultate, i dobijemo konačno poravnanje:

AAT-

CATG

5. Analiza rezultata

Konačna poravnanja nalaze se unutar nadmape `data`. Kako bi smo testirali performanse algoritma, generirali smo i poravnanja koristeći Needleman-Wunsch algoritam, pomoću `minineedle` python biblioteke, s kojima spo uspoređivali naša poravnanja.

- **poravnanja generirana s slučajnim vrijednostima parametara modela** - `alignments_hmm_random` folder
- **poravnanja generirana s procijenjenim vrijednostima parametara modela** - `alignments_hmm_estim` folder
- **poravnanja generirana s slučajnim vrijednostima parametara modela optimiranim Baum-Welch algoritmom** - `alignments_hmm_rand_train` folder
- **poravnanja generirana s procijenjenim vrijednostima parametara modela optimiranim Baum-welch algoritmom** - `alignments_hmm_estim_train` folder
- **poravnanja generirana s Needleman-Wunschom** - `alignments_needle` folder

Kako bi smo procjenili poravnanja, koristili smo dva načina bodovanja. Prvi je EDNA-Full matrica, u kojoj smo dodali kaznu za procjep. Drugi način je jednostavno bodovanje, u kojem se rezultatu dodaje +1 za match, ili -1 za indel i mismatch. Generiranje rezultata obavlja se pomoću `python/scoring.py` datoteke.

	A	C	G	T	-
A	5	-4	-4	-4	-4
C	-4	5	-4	-4	-4
T	-4	-4	-4	5	-4
G	-4	-4	5	-4	-4
-	-4	-4	-4	-4	-4

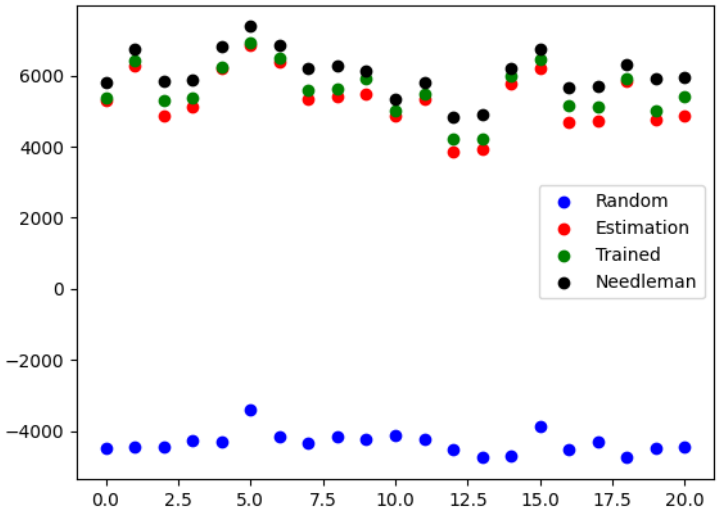
Tablica 5.1: EDNA matrica bodovanja

Za svaki od pristupa koji je računao parametre s Baum-Welchom, odrađeno je 10 iteracija tog algoritma.

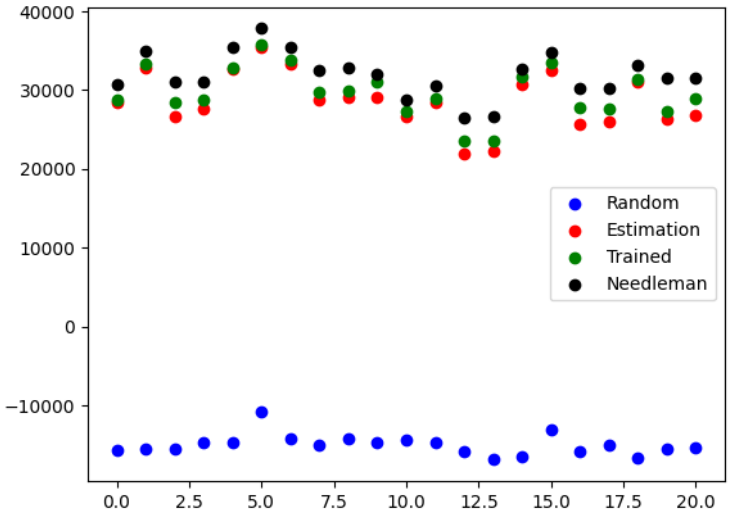
Algoritam	Viterbi	Viterbi	Viterbi	Viterbi	Needleman
Parametri	Random	Procijenjeni	Random=>BW	Procjena=>BW	
Bodovi (+1,-1,-1)	-4327.52	5335.809	5582.90	5582.90	6064.57
EDNA bodovi	-14950.71	28676.19	29704.52	29704.52	31924.4

Tablica 5.2: Tablica koja prikazuje prosječne bodove poravnanja sekvenci s Viterbijevim algoritmom, u ovisnosti o načinu odabira parametara. (Baum-Welch = BW)

Iz tablice možemo iščitati rezultate, koji su očekivani. Viterbijev algoritam sa slučajnim parametrima daje jako loša poravnanja. Znatno bolji od njega je Viterbijev algoritam sa procijenjenim parametrima. Blago poboljšanje je vidljivo s parametrima dobivenim Baum-Welchom. Tu je prisutno iznenađenje, gdje vidimo da smo dobili iste rezultate kada u Baum-Welch damo već procijenjene parametre, i slučajne. Mogući razlog toga je što imamo 10 iteracija, a zbog limitacija hardvera imamo samo 21 trening primjer, pa s Baum-Welchom dostižemo isti optimum neovisno o početnoj matrici. Provjerom istreniranih parametara u `transimission_values` možemo vidjeti da su iste vjerojatnosti u oba dva slučaja. Isto tako možemo ustvrditi da iako su poravnanja jako dobra, još uvijek nisu jednako dobra kao poravnanja dobivena Needleman-Wunschom. Grafički smo prikazali bodove za svaki par, da vidimo pobjeđuju li HMM poravnanja Needlemana za pojedinačne parove.



Slika 5.1: Prikaz bodova (1,-1,-1) sheme za pojedinačne parove u test skupu



Slika 5.2: Prikaz EDNA bodova za pojedinačne parove u test skupu

Iz grafova možemo zaključiti da Needleman daje najbolje rezultate i pojedinačno. Možemo vidjeti da je za neke sekvence razlika između performansi algoritama manja, dok je za neke dosta veća.

6. Zaključak

U ovome projektu implementirali smo algoritme Baum-Welch i Viterbi u svrhu poravnanja parova sekvenci. Ustvrdili smo da i s malim brojem podataka za treniranje možemo napraviti model koji će davati dobra poravnanja. Tema budućeg rada bi mogla biti pronalaženje načina procesiranja podataka i inicijalizacije početnih matrica u Baum-Welchu za svrhu dobivanja parametara koji mogu dati bolja poravnanja nego Needleman-Wunsch algoritam.

7. Literatura

[1] https://www.hiv.lanl.gov/content/sequence/HIV/mainpage.html
[2] Richard Durbin, Sean R. Eddy, Anders Krogh, Graeme Mitchison; Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, 2002, Ch. 4