UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

MASTER RAD

Razvoj aplikacije za korišćenje alata za poravnanje bioloških sekvenci

Zorana Stošić

Beograd, jun 2018.

Mentor:

dr Jovana Kovačević
Univerzitet u Beogradu, Matematički fakultet

Članovi komisije:

dr Gordana Pavlović-Lažetić
Univerzitet u Beogradu, Matematički fakultet

dr Saša Malkov
Univerzitet u Beogradu, Matematički fakultet

Datum odbrane: _____

Razvoj aplikacije za korišćenje alata za poravnanje bioloških sekvenci

Sažetak

U ovom radu opisano je nekoliko alata višestrukog poravnanja i razvijena aplikacija koja pruža zajednički interfejs za njihovo korišćenje. Rad se sastoji iz teorijskog uvoda o višestrukom poravnanju sekvenci, detaljnog opisa najpopularnijih alata, opisa kreirane aplikacije i opisa BAliBASE (*Benchmark Alignment dataBASE*) *benchmark*-a koji je korišćen za poređenje performansi alata predstavljenih u ovom radu.

Ključne reči: Višestruko poravnanje sekvenci, *Biopython*, ClustalW, MUSCLE, BAliBASE *benchmark*

SADRŽAJ

1. Uvod

Višestruko poravnanje sekvenci (eng. *Multiple Sequence Alignment, MSA*) je danas jedna od najzastupljenijih procedura za analizu bioloških sekvenci u molekularnoj biologiji. Značaj ove procedure se ogleda u tome što uz pomoć višestrukih poravnanja sekvenci biolozi mogu da utvrde delove sekvenci koji nisu menjani kroz evoluciju, kao i veze između predaka različitih organizama. Alati višestrukog poravnanja predstavljaju centralni deo ovog rada i oni će biti detaljno opisani u glavi 3. Međutim, radi lakšeg shvatanja svih pojmova koji će se pominjati u radu, u nastavku sledi detaljniji opis procesa višestrukog poravnanja.

Pod višestrukim poravnanjem sekvenci podrazumevamo poravnanje tri ili više bioloških sekvenci. Poravnanje uključuje umetanje praznina (simbol "-„) na odgovarajuće pozicije tako da se što veći broj karaktera iz svake sekvence na istim pozicijama poklapa. Svako umetanje (tzv. insercija) i brisanje (tzv. delecija) karaktera nazivamo indelom. Jedno višestruko poravnanje sekvenci se može videti u primeru 1.0.1.

Primer 1.0.1 *Sekvence i njihovo višestruko poravnanje*

Sekvence

x_1 : G C T G A T A T A G C T

x_2 : G G G T G A T T A G C T

x_3 : G C T A T C G C

x_4 : A G C G G A A C A C C T

Poravnanje sekvenci

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

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

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

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

Prilikom ocenjivanja višestrukog poravnanja treba razmatrati tri osnovna aspekta:

1. *Vrednost za poklapanje (eng. match value)* – vrednost koja se dodeljuje kada se karakteri poklapaju

2. *Vrednost za nepoklapanje (eng. mismatch value)* – vrednost koja se dodeljuje kada se karakteri ne poklapaju.

3. *Kazna za praznine (eng. gap penalty)* – vrednost praznina (odnosno "-„ simbola)

U glavi 5 će biti reči o načinu ocenjivanja višestrukih poravnanja koji će biti od velikog značaja za upoređivanje kvaliteta optimalnih poravnanja. Ocena višestrukog poravnanja zasnovana je na sumi rezultata svih mogućih parova sekvenci u višestrukom poravnanju na osnovu neke matrice ocena (eng. *score matrix*). Cilj višestrukog poravnanja je da se maksimizuje sledeća suma:

$$OcenaVisestrukogPoravnanja = \sum ocena(A, B) \tag{1.1}$$

gde je *ocena(A,B)* = ocena dvostrukog poravnanja sekvenci *A* i *B*.

Primer 1.0.2 *Ocena višestrukog poravnanja*
Vrednost za poklapanje je 1 i vrednost za nepoklapanje je 0
Sek_1: *A C G*
Sek_2: *A C A*
Sek_3: *A G G*

Kol_1 = *ocena(A, A) + ocena (A, A) + ocena(A, A) = 1 + 1 + 1 = 3*
Kol_2 = *ocena(C, C) + ocena(C, G) + ocena(C, G) = 1 + 0 + 0 = 1*
Kol_3 = *ocena(G, A) + ocena(A, G) + ocena(G, G) = 0 + 0 + 1 = 1*

Ocena poravnanja = Kol_1 + Kol_2 + Kol_3 = *3 + 1 + 1 = 5*

Kao što je već pomenuto ranije, procedura višestrukog poravnanja je značajna iz više razloga. Jedna korisna primena je ta što se na osnovu višestrukog poravnanja može zaključiti homologija sekvenci, gde homologija sekvenci ukazuje na zajedničko evolutivno poreklo između DNK, RNK ili proteinskih sekvenci. Na primer, dva segmenta DNK mogu deliti poreklo usled jednog od tri fenomena:

- *Specijacija*, evolutivni proces kojim populacija evoluira do odvojene vrste
- *Dupliranje gena* (ili dupliranje hromozoma ili amplifikacija gena), glavni mehanizam kroz koji se generiše novi genetski materijal tokom molekularne evolucije
- *Horizontalni prenos gena* (HGT) ili *lateralni prenos gena* (LGT), kretanje genetskog materijala između jednoćelijskih i/ili višećelijskih organizama

Na osnovu višestrukog poravnanja se takođe može izvršiti filogenetska analiza, kojom razmatramo razlike u istim proteinskim sekvencama kod različitih organizama kako bi dobili

informacije o evolutivnim odnosima organizama.

U poslednjih 25 godina razvijeni su mnogi alati za dvostruko i višestruko poravnanje i poznato je da postoji oko 47 alata za dvostruko i oko 52 alata za višestruko poravnanje sekvenci. Međutim danas još uvek ne postoji alat koji zadovoljava sve kriterijume i koji predstavlja tzv. savršen alat. Napredak u oblasti genomike u poslednjih dvadeset godina doveo je do smanjenja troškova sekvenciranja genoma, čime je povećan obim skoro svih projekata istraživanja genoma, poput *Projekta ljudskog genoma* (eng. *Human Genome Project*, *HGP*)[8]. Sa druge strane, jedan od problema alata višestrukog poravnanja je što povećanje broja sekvenci koje treba poravnati usporava kreiranje poravnanja. Zbog toga je ova oblast bioinformatike veoma aktivna, posebno u pravcu razvoja metoda koje će moći u optimalnom vremenskom roku da kreiraju optimalno poravnanje od velikog broja dugih sekvenci.

Nakon uvodnog dela o višestrukom poravnanju u nastavku će se rad, u glavi 2, baviti vrstama i načinima izgradnje višestrukog poravnanja. Nekoliko popularnih alata višestrukog poravnanja kao i način njihovog rada je prikazan u glavi 3, nakon čega je, u glavi 4, prikazana razvijena aplikacija i opisani su svi pomoćni alati koji su doprineli izgledu i funkcionalnosti aplikacije. Nakon opisa aplikacije, rad se bavi upoređivanjem performansi alata i u glavi 5 je opisan korišćeni *benchmark* i prikazani su rezultati rada alata predstavljenih u aplikaciji.

2. Vrste i načini izgradnje višestrukog poravnanja

2.1. Vrste poravnanja

Razlikujemo dve vrste poravnanja: globalno i lokalno poravnanje sekvenci [1]. Globalni pristup uzima u obzir celu dužinu sekvence i vrši poravnanje od početka do kraja. Kada se globalno poravnanje konstruiše, obično se koriste dodatne metode kako bi se prepoznali konzervisani (evolutivno sačuvani) ili pouzdani delovi unutar poravnanja.
Jedan od algoritama koji koristi dinamičko programiranje za dobijanje globalnog poravnanja je algoritam *Needleman-Wunsch*. Ovaj algoritam su objavili genetičar *Saul B. Needleman* i doktor medicine *Christian D. Wunsch* 1970. godine za poravnanje dve proteinske sekvence i to je bila prva primena dinamičkog programiranja u analizi bioloških sekvenci. Osnovna ideja algoritma je da se poravnanje postepeno gradi upotrebom poravnanja manjih podsekvenci, odnosno razmatraju se svi mogući parovi karaktera iz dve sekvence. Algoritam koristi matricu ocena i matricu povratka (eng. *traceback matrix*) i sastoji se iz tri koraka: inicijalizacija matrica, izračunavanje ocena i popunjavanje matrice povratka i određivanje poravnanja na osnovu popunjene matrice povratka. Konstrukcija ovih matrica i detaljan opis koraka algoritma su opisani u primeru 2.1.1.

Primer 2.1.1 *Globalno poravnanje*
Posmatrajmo dve sekvence ***ACGC*** *i* ***GGC*** *za koje ćemo primenom Needleman-Wunsch-ovog algoritma pronaći globalno poravnanje. Pre inicijalizacije matrica, potrebno je ustanoviti*

		A	C	G	C
	0	-1	-2	-3	-4
G	-1				
G	-2				
C	-3				

		A	C	G	C
	kraj	levo	levo	levo	levo
G	gore				
G	gore				
C	gore				

Tablica 2.1: Inicijalizacija matrica

		A	C	G	C
	0	-1	-2	-3	-4
G	-1	0			
G	-2				
C	-3				

		A	C	G	C
	kraj	levo	levo	levo	levo
G	gore	dijag			
G	gore				
C	gore				

Tablica 2.2: Popunjavanje ćelije M(2,2)

		A	C	G	C
	0	-1	-2	-3	-4
G	-1	0	-1	-1	-2
G	-2	-1	0	0	-1
C	-3	-2	0	0	1

		A	C	G	C
	kraj	levo	levo	levo	levo
G	gore	dijag	levo	dijag	levo
G	gore	gore	dijag	dijag	dijag
C	gore	gore	dijag	dijag	dijag

Tablica 2.3: Putanja

shemu bodovanja. U ovom primeru je korišćena najjednostavnija shema, a to je da je vrednost za poklapanja 1, vrednost za nepoklapanja je 0 i kazna za praznine je -1.
Nakon koraka inicijalizacije, nastavljamo dalje sa popunjavanjem matrica, tako što krećemo od ćelije M(2,2) i tražimo maksimum sledećih vrednosti:

q_{dijag} = *M(i-1, j-1) + S(i,j)*
q_{gore} = *M(i-1, j) + g*
q_{levo} = *M(i, j-1) + g*

gde je S(i, j) ocena za poklapanje ili nepoklapanje karaktera, a g je kazna za praznine.

Vrednost ćelije M(2,2) se dobija na sledeći način:
q_{dijag} = *M(1, 1) + S(A, G) = 0 + 0 = 0*
q_{gore} = *M(1, 2) + (-1) = -1 - 1 = -2*
q_{levo} = *M(2, 1) + (-1) = -1 -1 = -2*
odakle se zaključuje da je M(2,2) = 0 = q_{dijag}

Nakon popunjavanja matrica, poravnanje pronalazimo tako što pronalazimo putanju u matrici povratka koja nas iz ćelije u donjem desnom uglu vodi do ćelije u gornjem levom uglu, koju smo ovde nazvali "kraj". U ovom primeru je dobijena putanja "dijag, dijag, levo, dijag, kraj" i poravnanje se dobija tako što pratimo sledeća pravila:

1. ***dijag*** *- poravnati karaktere iz date dve sekvence*

dijag	**C** **C**
dijag	**G**C **G**C
levo	**C**GC **-** GC
dijag	**A**CGC **G**- GC

2. ***levo*** *- uvodi se praznina u levoj sekvenci*

3. ***gore*** *- uvodi se praznina u gornjoj sekvenci*

Sa druge strane, lokalni pristup identifikuje podsekvence koje su slične. Nepouzdani ili manje slični regioni se ili isključuju iz poravnanja ili se posebno označavaju (mala/velika slova), kako bi se mogli razlikovati od ostatka poravnanja.
Algoritam lokalnog poravnanja koji je sličan *Needleman-Wunsch*-ovom algoritmu, sa malim razlikama u procesu bodovanja, je *Smith-Waterman*-ov algoritam. Profesori *Temple Ferris Smith* i *Michael Spencer Waterman* su 1981. godine objavili ovaj algoritam i predstavili primenu dinamičkog programiranja za nalaženje optimalnog lokalnog poravnanja. Koraci ovog algoritma su isti kao kod prethodno opisanog *Needleman-Wunsch*-ovog algoritma i njihov detaljan opis je prikazan u primeru 2.1.2.

Primer 2.1.2 *Lokalno poravnanje*
Može se reći da Smith-Waterman-ov algoritam predstavlja modifikaciju Needleman-Wunsch-ovog algoritma jer algoritam prati iste korake, s tim što je proces popunjavanja matrica i izračunavanja ocena malo promenjen:

1. *Elementi prve kolone i prve vrste matrice ocena su postavljeni na 0*

2. *U poslednjoj vrsti i koloni matrice ocena se ne dodaje kazna za praznine na vrednost prethodnog elementa poslednje vrste i kolone*

3. *Ako je vrednost nekog elementa matrice manja od 0, postavlja se 0*

4. *Putanja, na osnovu koje ćemo dobiti poravnanje, ne kreće od donjeg desnog elementa već od maksimalnog elementa i ne ide do gornjeg levog elementa već do prvog elementa koji je jednak nuli*

		C	A	C	G	T	A	T
	0	0	0	0	0	0	0	0
C	0							
G	0							
C	0							
A	0							

		C	A	C	G	T	A	T
	kraj	levo	levo	levo	levo	levo	levo	levo
C	gore							
G	gore							
C	gore							
A	gore							

Tablica 2.4: Inicijalizacija matrica

		C	A	C	G	T	A	T
	0	0	0	0	0	0	0	0
C	0	1	0	1	0	0	0	0
G	0	0	1	0	2	0	0	0
C	0	1	0	2	1	2	1	0
A	0	1	2	2	2	2	3	3

Tablica 2.5: Popunjena matrica ocena

		C	A	C	G	T	A	T
	kraj	levo	levo	levo	levo	levo	levo	levo
C	gore	dijag	d/j/g	dijag	d/j/g	d/j/g	d/j/g	d/j/g
G	gore	d/j/g	dijag	d/j/g	dijag	d/j/g	d/j/g	d/j/g
C	gore	dijag	d/j/g	dijag	l/g	dijag	levo	d/j/g
A	gore	gore	dijag	l/g	dijag	l/g	dijag	levo

Tablica 2.6: Popunjena matrica povratka i putanja

Na osnovu putanje dobijamo sledeće lokalno poravnanje:

CACGTAT
- -CGCA -

2.2. Načini izgradnje višestrukog poravnanja

Ručno upoređivanje tri ili više sekvenci, koje mogu biti velikih dužina, može biti veoma teško i vremenski zahtevno. Zbog toga se razni algoritmi koriste za automatsko konstruisanje i analizu višestrukih poravnanja, uključujući spore, ali egzaktne algoritme koje koriste metode poput dinamičkog programiranja i brze, ali manje precizne koje koriste heurističke ili probabilističke metode. Danas većina programa višestrukog poravnanja koristi heurističke metode umesto egzaktnih metoda globalne optimizacije, jer je prepoznavanje optimalnog poravnanja računski skupo. Međutim, treba napomenuti da nije uvek bilo ovako tako da su se koristile i druge metode, koje i dalje možemo sresti u pojedinim programima.
Dinamično programiranje (DP) je matematička i računarska metoda koja pojednostavljuje složeni problem tako što ga deli na manje i jednostavnije komponente. Tehnika dinamičkog programiranja je primenjena na rešavanje problema globalnog poravnanja kroz *Needleman-Wunsch*-ov algoritam kao i na rešavanje lokalnog poravnanja kroz *Smith-Waterman*-ov algoritam. Do sredine 1980-ih, tradicionalni algoritmi višestrukog poravnanja su bili prikladni za upoređivanje dve sekvence i kada se javila potreba za upoređivanje tri ili više sekvenci, otkriveno je da je ručno poravnanje brže od korišćenja tradicionalnih algoritama dinamičkog programiranja. Algoritmi dinamičkog programiranja se koriste za izračunavanje dvostrukih poravnanja, vremenske složenosti $O(L \cdot n)$, gde je L dužina sekvence, a n je ukupan broj sekvenci. U teoriji, ovaj metod se može proširiti tako da vrši poravnanje za više od dve sekvence, ali je u praksi suviše spor, stoga stvaranje višestrukog poravnanja zahteva korišćenje metoda sofisticiranijih od onih koji se koriste za dvostruko poravnanje. Pronalaženje optimalnog višestrukog poravnanja skupa sekvenci može se uopšteno definisati kao složeni problem optimizacije. Ovaj problem pripada skupu NP kompletnih problema i rešava se heurističkim metodama.
U nastavku rada su opisana dva najpoznatija načina za izgradnju višestrukog poravnanja sekvenci:

- *Progresivno višestruko poravnanje*, počinje sa jednom sekvencom i progresivno (postepeno) poravnava ostale
- *Iterativno višestruko poravnanje*, tokom procesa se u svakoj iteraciji ponovo poravnavaju sekvence

2.2.1. Progresivno višestruko poravnanje

Najpoznatiju korišćenu heuristiku, od koje se većina višestrukih poravnanja generiše, razvili su američki biohemičar *Russell Doolittle* i profesor *Da-Fei Feng*, koju su nazvali *progresivno poravnanje*. Progresivno poravnanje radi tako što se postepeno gradi višestruko poravnanje. Prvo se nalaze dvostruka poravnanja korišćenjem metoda poput *Needleman-Wunsch*-ovog, *Smith-Waterman*-ovog, *k*-torka ili *k*-mer algoritma. Potom se sekvence grupišu, kako bi se predstavio njihov međusobni odnos, korišćenjem metoda klasterovanja kao što su *mBed* i *k*-sredina. Izračunavaju se ocene sličnosti koje se zatim pretvaraju u ocene udaljenosti koje se koriste kako bi se konstruisalo stablo navođenja, pomoću metoda kao što su *NJ* (*Neighbour-Joining*) i *UPGMA* (*Unweighted Pair Group Method with Aritmetic Mean*). Kada se kreira stablo navođenja, višestruko poravnanje se konstruiše tako što se sekvence, jedna po jedna, ubacuju na osnovu tog stabla. Tačnije, najsličnije sekvence se prve ubacuju, a zatim se postepeno ubacuju udaljenije, odnosno manje slične sekvence. Detaljniji opis progresivnog poravnanja uključuje sledeće korake:

1. Počni sa najsličnijim sekvencama
2. Izvrši poravnanje nove sekvence u odnosu na prethodne sekvence
3. Kreiraj matricu udaljenosti za svaki par sekvenci
4. Na osnovu matrice konstruiši filogenetsko stablo navođenja, tako što se sekvence postavljaju kao listovi
5. Korišćenjem stabla navođenja odredi sledeću sekvencu koja će se dodati poravnanju
6. Sačuvaj praznine
7. Vrati se na korak 1.

Progresivno poravnanje je jedan od najbržih pristupa višestrukog poravnanja i značajno je brži od procesa prilagođavanja dvostrukih poravnanja višestrukom poravnanju. Nažalost, ova heuristika je pohlepne prirode. U jednom trenutku uzima u obzir samo dve sekvence i ignoriše preostale podatke i samim tim ne može garantovati optimalno rešenje. Takođe, ako se u početnim fazama poravnanja naprave greške, one se ne mogu popraviti u kasnijim fazama, a greška će nastaviti da postoji tokom procesa poravnanja i problem će se pogoršavati s povećanjem broja sekvenci. Progresivno poravnanje se koristi kao osnovni postupak u nekoliko algoritama za poravnanje sekvenci kao što su algoritmi *Clustal W*[18],*Clustal Omega*[7], *MAFFT*[11], *Kalign*[12], *Probalign*[17], *MUSCLE*[6], *DIALIGN*[13], *T-Coffee*[15], *ProbCons*[5] i *MSAProbs*[20].

2.2.2. Iterativno višestruko poravnanje

Unapređena verzija progresivnog poravnanja razvijena je pod nazivom iterativno poravnanje. Algoritmi ovog poravnanja rade na sličan način kao i kod progresivnog poravnanja. Jedina razlika je ta što ovaj pristup više puta primenjuje dinamičko programiranje za poravnanje početnih sekvenci kako bi se poboljšao kvalitet poravnanja i istovremeno dodaje nove sekvence rastućem višestrukom poravnanju.
Razlikujemo dva tipa iterativnog poravnanja: stohastičko i nestohastičko iterativno poravnanje. Stohastički algoritmi koriste neki nivo slučajnosti kako bi došli do rešenja, kreiraju dobra poravnanja, ali su previše spori za ogromne skupove podataka. Sa druge strane, raniji nestohastički algoritmi su vršili ponovna poravnanja sekvenci u višestrukom poravnanju, sve dok je kvalitet višestrukog poravnanja mogao da se popravi. Noviji nestohastički algoritmi koriste strategiju duple iteracije, tačnije koriste dve petlje. Unutrašnja petlja optimizuje SP (*sum-of-pairs*) ocenu, a spoljašnja optimizuje težine u filogenetskom stablu. Prednost iterativnog konstruisanja višestrukog poravnanja je ta što ispravlja bilo kakve početne greške, čime se poboljšava ukupni kvalitet poravnanja. Iterativno poravnanje predstavlja metodu optimizacije i može da koristi pristupe mašinskog učenja, poput genetskih algoritama i skrivenih Markovljevih modela. Jedina mana je što su iterativne metode ograničene, jer mogu raditi sa skupovima od samo nekoliko stotina sekvenci. Najpopularniji algoritmi koji koriste iterativni način poravnanja su *MUSCLE*, *Dialign*, *SAGA*[14] i *T-Coffee*.

Trebalo bi napomenuti da, pored prethodno opisana dva načina izgradnje višestrukog poravnanja, postoje i drugi. Na primer, višestruka poravnanja se takođe mogu konstruisati korišćenjem već postojeće informacije o strukturi proteina. Proces strukturnog poravnanja uspostavlja homologiju između dve ili više polimerne strukture i obično se primenjuje nad tercijarnom strukturom proteina[1], ali se može koristiti i za velike RNK molekule. Ovaj pristup se može primeniti samo u slučajevima kada su nam poznate informacije o strukturi. Smatra se da se, zahvaljujući informacijama o strukturi, finalni kvalitet višestrukog poravnanja može povećati. Bolji kvalitet se ne dobija zbog boljeg algoritma, već zbog stabilnosti strukturne evolucije, tj. zbog činjenice da strukture evoluiraju sporije od sekvenci.
Algoritmi otkrivanja motiva su još jedna vrsta algoritama višestrukog poravnanja koji se koriste. Motiv je deo (podsekvenca) proteina ili DNK sekvence koja ima posebnu strukturu i ima ili se pretpostavlja da ima biološki značaj. Na primer, kod proteina strukturalni motiv je motiv koji se sastoji od trodimenzionalnog rasporeda aminokiselina. Metode otkrivanja motiva se koriste za pronalaženje motiva u dugačkim sekvencama. Ovaj proces se posmatra kao problem „traženja igle u plastu sena“, zbog činjenice da algoritam traži kratak niz aminokiselina (motiv) u dugačkoj sekvenci. Jedan od najčešće korišćenih alata za traženje

[1] Trodimenzionalna struktura proteina, definisana atomskim koordinatama

motiva je *PHI-Blast* i *Gapped Local Alignments of Motif* (*GLAM2*).

3. Alati višestrukog poravnanja

Kao što je već pomenuto u uvodnom delu rada, oblast bioinformatike koja se bavi postupkom višestrukog poravnanja sekvenci je veoma aktivna i već danas imamo mnogobrojne algoritme višestrukog poravnanja koji se konstantno unapređuju. Jedan od najpoznatijih algoritama višestrukog poravnanja je algoritam *ClustalW*, dok je nedavno razvijeni algoritam *Clustal Omega* trenutno najprecizniji i najnapredniji algoritam višestrukog poravnanja. U nastavku ove glave je dat detaljan opis nekih od najpoznatijih algoritama višestrukog poravnanja [4].

3.1. Clustal

Clustal serija programa je široko korišćena u molekularnoj biologiji za višestruko poravnanje nukleinskih kiselina i proteinskih sekvenci kao i za pripremanje filogenetskih stabala. Popularnost ovih programa zavisi od raznih faktora, uključujući ne samo kvalitet rezultata već i robusnost i prenosivost i upravo najčešće korišćeni programi za globalno višestruko poravnanje su iz serije *Clustal* programa. Prvi *Clustal* program je napisao *Des Higgins* 1988. godine i bio je posebno dizajniran da efikasno radi na personalnim računarima, koji su u to vreme imali slabu računsku moć prema današnjim standardima. Predstavljao je kombinaciju memorijski efikasnog algoritma dinamičkog programiranja i strategiju progresivnog poravnanja koju su razvili *Da-Fei Feng* i *Russell Doolittle* i *Willie Taylor*. Višestruko poravnanje je izgrađeno progresivno serijom dvostrukih poravnanja, prateći redosled grananja u stablu navođenja. Godine 1992. napravljeno je novo izdanje pod nazivom *ClustalV*, koji je uključio profilna poravnanja (poravnanja postojećih poravnanja). Treća generacija serije *ClustalW*, objavljena 1994. godine, donela je niz poboljšanja algoritma poravnanja, poput pridruživanja određenih težina sekvencama, poziciono određenih kazni za praznine i automatskog izbora odgovarajuće matrice upoređivanja u svakoj fazi višestrukog poravnanja. *ClustalW* je podstakao razvoj ostalih *Clustal* programa, uključujući i *ClustalX* (verzija sa grafičkim korisničkim interfejsom). Iako su dobijena poravnanja ista kao i ona koja su dobijena upotrebom alata *ClustalW*, korisnik može bolje proceniti poravnanja u *ClustalX*-u.
Sve verzije *Clustal*-a prate tri glavna koraka:

– Kreiranje dvostrukog poravnanja

– Kreiranje stabla navođenja [2]

– Upotreba stabla kako bi pronašlo višestruko poravnanje

3.1.1. ClustalW

ClustalW je predstavljen 1994. godine i brzo je postao najpopularnija metoda za kreiranje višestrukih poravnanja, jer je zapaženo drastično povećanje kvaliteta i brzine poravnanja u poređenju sa drugim algoritmima. U nazivu *ClustalW*, "*W*" predstavlja težine (eng. *weights*), pošto je u ovoj verziji *Clustal* programa uvedeno dodeljivanje težine sekvencama.

Algoritam

Na samom početku algoritam vrši dvostruko poravnanje svih sekvenci pomoću metode *k*-torka, poznate i kao *word* metoda ili pomoću *Needleman-Wunsch*-ove metode. Ove metode izračunavaju matricu sličnosti svakog para sekvenci. Ocene sličnosti se pretvaraju u ocene rastojanja, a zatim algoritam koristi ove ocene kako bi napravio stablo navođenja, koristeći metod *NJ* za konstrukciju stabla. Poslednji korak algoritma je konstrukcija višestrukog poravnanja svih sekvenci koje se konstruiše progresivnim poravnanjem najbližih, odnosno najsličnijih, sekvenci prema stablu navođenja.

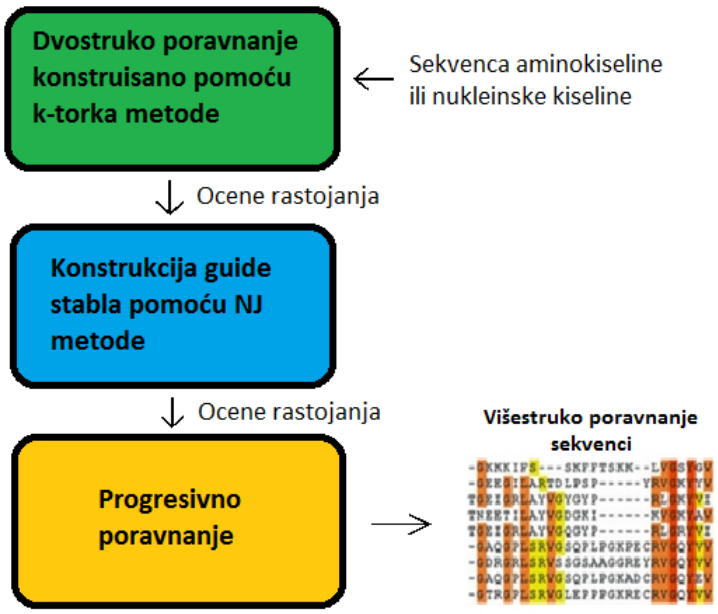
Dvostruko poravnanje

Dvostruka poravnanja svih mogućih parova sekvenci se kreiraju korišćenjem metode *k*-torka, brze heurističke metode koja ima posebno dobre performanse kada je broj sekvenci koje treba poravnati veliki. Prvo se ocene sličnosti izračunavaju i kazne za praznine se oduzimaju od tih ocena. Ocene sličnosti se kasnije pretvaraju u ocene udaljenosti tako što se ocena sličnosti deli sa 100 i oduzima od 1.0. Sve *k*-torke između dve sekvence se nalaze korišćenjem heš tabele. Poslednja faza *k*-torne metode je pronalaženje potpunog rasporeda svih *k*-tornih pogodaka (poklapanja), stvaranjem optimalnog poravnanja koji je sličan *Needleman-Wunsch*-ovoj metodi.

Konstrukcija stabla navođenja

ClustalW kreira stablo navođenja u skladu sa metodom *NJ*, koja se često naziva metodom dekompozicije. Metoda *NJ* ne prati tzv. „taksonome" (taksonomska kategorija ili grupa, po-

[2] Sve progresivne metode poravnanja zahtevaju dve etape: prva u kojoj su veze između sekvenci predstavljene u obliku stabla koji se sa naziva stablo navođenja (eng. *quide tree*) i druga, u kojoj se višestruko poravnanje gradi sekvencijalnim dodavanje sekvenci na osnovu ovog stabla. Inicijalno stablo navođenja je određeno metodom klasterovanja, kao što je NJ ili UPGMA



Slika 3.1: Koraci algoritma ClustalW

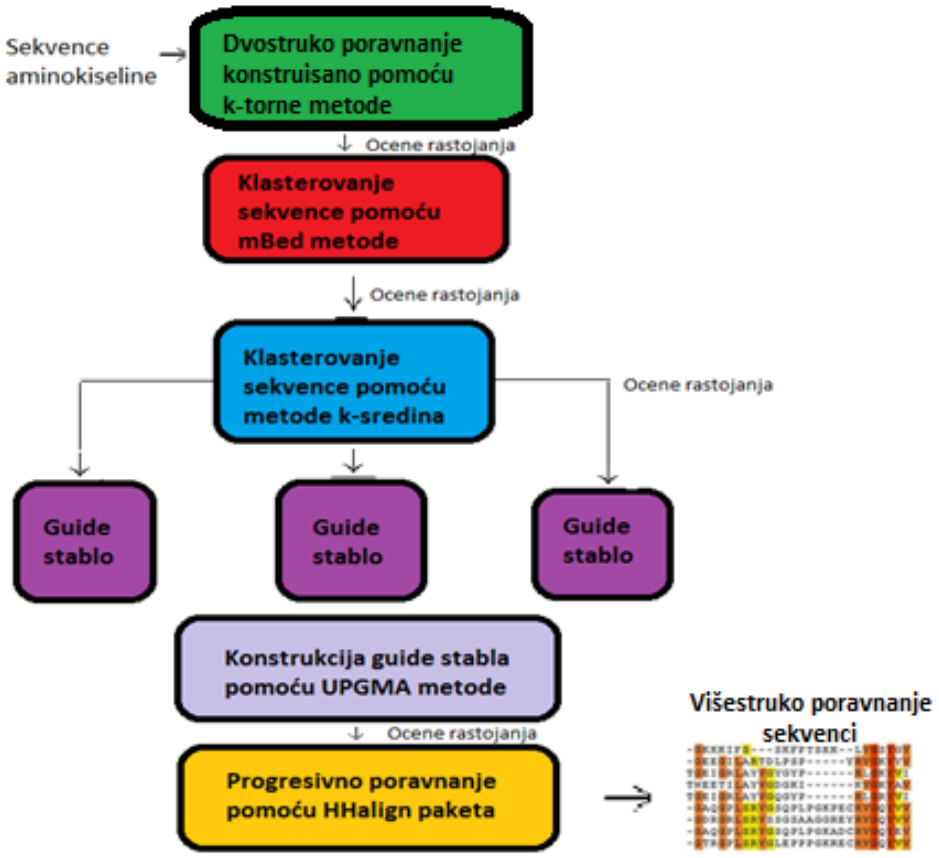
put reda, familije, roda ili vrste) ili klastere „taksonoma“, već prati čvorove u stablu. Ocene sličnosti se uzimaju iz prethodne *k*-torne metode i čuvaju se u matrici, a potom se kreira modifikovana matrica rastojanja u kojoj je rastojanje između svakog para čvorova dobijeno izračunavanjem prosečne vrednosti odstupanja u odnosu na ostale čvorove. Konstrukcija stabla počinje povezivanjem čvorova koji su najmanje udaljeni. Kada su dva čvora povezana, njihov zajednički čvor predak se dodaje stablu i listovi sa odgovarajućim granama se uklanjaju iz stabla. U svakoj fazi procesa, dva lista su zamenjena jednim čvorom. Proces se završava kada dva čvora ostanu povezana samo jednom granom. Stablo proizvedeno metodom *NJ* je bez korena i dužine njegovih grana su proporcionalne razlici duž svake grane. Koren se postavlja tamo gde se mogu dobiti dva podstabla jednakih dužina. Stablo navođenja se zatim koristi za izračunavanje težine svake sekvence, gde težina zavisi od udaljenosti grane od korena. Ako sekvenca deli granu sa nekom drugom sekvencom, onda će dve ili više sekvenci deliti težinu izračunatu na osnovu te zajedničke grane i dužina sekvence će se sumirati i podeliti sa brojem sekvenci koje dele istu granu.

Progresivno poravnanje

Progresivno poravnanje koristi seriju dvostrukih poravnanja prateći redosled grananja u stablu navođenja. Postupak počinje od listova stabla i nastavlja prema korenu. U svakom koraku se koristi algoritam dinamičkog programiranja sa matricom težine (*BLOSUM*) i vrednostima za otvarajuće i proširujuće praznine.

3.1.2. Clustal Omega

Clustal Omega[3], objavljen 2014. godine, je najnoviji algoritam višestrukog poravnanja iz *Clustal* familije i trenutno se koristi samo za poravnanje proteinskih sekvenci. Na velikom skupu *Clustal Omega* nadmašuje ostale algoritme višestrukog poravnanja, ako posmatramo vreme izvršavanja i ukupan kvalitet poravnanja. Na primer, *Clustal Omega* je sposoban da konstruiše višestruko poravnanje 190 000 sekvenci na jednom procesoru za nekoliko sati. Ovaj algoritam kreira višestruko poravnanje tako što prvo proizvodi dvostruka poravnanja koristeći *k*-tornu metodu. Zatim se sekvence grupišu pomoću metode *mBed*, nakon čega se primenjuje metoda *k*-sredina. Nakon konstrukcije stabla navođenja, pomoću metode *UPGMA*, višestruko poravnanje se dobija progresivnim poravnanjem korišćenjem paketa *HHalign*.



Slika 3.2: Koraci *Clustal Omega* algoritma

[3]http://www.ebi.ac.uk/Tools/msa/clustalo/

Dvostruko poravnanje

Dvostruko poravnanje se dobija pomoću metode *k*-torka, isto kao i kod *ClustalW*.

Klasterovanje

Nakon određivanja dvostrukih poravnanja, izračunate su ocene sličnosti, odnosno određena je matrica sličnosti. Sledeći korak je da na osnovu matrice sličnosti izračunamo matricu rastojanja. Međutim, izračunavanje cele matrice rastojanja za *N* sekvenci ima vremensku i prostornu složenost $O(N^2)$, što predstavlja problem kada je *N* veliko. Zbog toga *Clustal Omega* koristi algoritam *mBed* složenosti $O(N \cdot logN)$. Algoritam *mBed* matricu rastojanja, dimenzije $N \times N$, redukuje na matricu dimenzije $N \times logN$, tako što iz ukupnog skupa sekvenci slučajnim odabirom uzima $M = logN$ sekvenci (tzv. *seed* sekvence). Nakon toga se izračunavaju rastojanja svih *N* sekvenci u odnosu na prethodno slučajno odabrane sekvence. Ova rastojanja se zatim mogu klasterovati pomoću metoda za klasterovanje, poput *k*-sredine ili *UPGMA*.

K-sredine

Clustal Omega pored metoda *k*-sredina koristi i algoritam *k*-sredine++ (eng. *k-means++*) koji vrši odabir početnih vrednosti za *k*-sredine. *K*-sredine je jednostavna, brza i široko korišćena tehnika klasterovanja koja minimizuje prosečno kvadratno rastojanje između tačaka u istom klasteru. Korišćenjem *k*-sredine++ uspešno se prevazilaze problemi definisanja početnih klasterskih centara i poboljšava se brzina i preciznost metode *k*-sredine.

Konstrukcija stabla navođenja

Clustal Omega za konstrukciju stabla navođenja koristi metodu *UPGMA*, koja je jednostavna metoda za konstrukciju stabla koja koristi algoritam sekvencijalnog klasterovanja u kojem je lokalna homologija između operativnih taksonomskih jedinica (tzv. OTJ) identifikovana po redosledu sličnosti. Stablo se gradi postepeno tako što se prvo određuju parovi OTJ-ova koji su najsličniji i onda se tretiraju kao pojedinačni OTJ. Posle toga, iz nove grupe OTJ-ova, pronalazi se i grupiše par sa najvećom sličnošću. Ovaj proces se nastavlja sve dok ne ostanu samo dva OTJ.

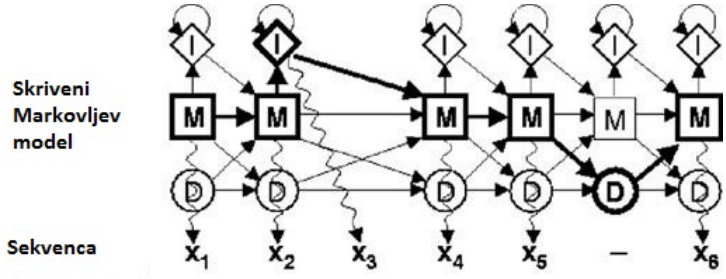
Progresivno poravnanje

Kod profilnog poravnanja sekvence su poravnate na osnovu redosleda grananja u stablu navođenja gde se u svakom koraku vrši poravnanje dva poravnanja. U prvom koraku ovo su pojedinačne sekvence, ali se one povećavaju tako što se dodaju nove sekvence obilaskom

stabla. *Clustal Omega* koristi alat *HHalign* koji se zasniva na skrivenim Markovljevim modelima. Alat *HHalign* generiše profil skrivenog Markovljevog modela (eng. *Hidden Markov Model, HMM*) iz datih poravnanja i izračunava optimalno poravnanje i sva značajna nepreklapajuća podoptimalna poravnanja. Profil skrivenih Markovljevih modela je konačni automat koji se sastoji iz niza čvorova, gde svaki čvor odgovara poziciji (koloni) u poravnanju. Ako zanemarimo praznine, svaki čvor odgovara tačno jednoj koloni u poravnanju i svaki čvor može imati samo jedno stanje, stanje poklapanja, što znači da će se sve pozicije iz modela poklopiti sa svim pozicijama sekvence. Pored stanja, profili sadrže i nekoliko tipova verovatnoće, kao što su:

1. *Verovatnoća tranzicije* - verovatnoća da se iz jednog stanja pređe u drugo
2. *Verovatnoća prenosa* - verovatnoća da se dati karakter nalazi na određenoj poziciji u poravnanju

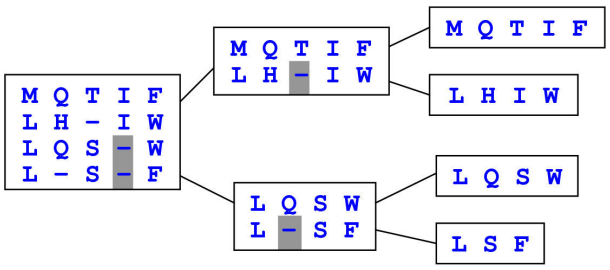
Na početku smo zanemarili praznine, međutim u praksi ćemo ih uvek imati, pa tako razlikujemo dva tipa praznine koje se ovde mogu javiti. Prvi tip praznine se javlja kada sekvenca sadrži deo koji se ne nalazi u modelu (insercija), a drugi tip se javlja kada postoji deo u modelu koji se ne nalazi u sekvenci (delecija). Zbog toga su uvedena još dva stanja tako da svaki čvor profila mora da ima tri stanja: stanje poklapanja *M*, stanje insercije *I* i stanje delecije *D*.
Poravnanje sekvence i profila se izvršava algoritmom dinamičkog programiranja koji u modelu pronalazi putanju sa najvećom verovatnoćom. Na primer, ako je sekvenca jednaka konsenzusu originalnog poravnanja, putanja u modelu će se sastojati samo od čvorova koji su u stanju poklapanja i biće linearna. Ako sekvenca sadrži deleciju, putanja će proći kroz jedno ili više stanja delecije pre nego što pređe na sledeće stanje poklapanja, a ako sekvenca sadrži inserciju, putanja će proći kroz stanje insercije između dva stanja poklapanja.



Slika 3.3: Primer poravnanja sekvence sa skrivenim Markovljevim modelom

3.2. MUSCLE

MUSCLE (*Multiple Sequence Comparison by Log-Expectation*) je precizniji od *T-Coffee* (jedan od najpopularnijih alata za višestruko poravnanje pomenut u potpoglavlju 2.2.1) i brži od *ClustalW*. Osnovna strategija koju koristi *MUSCLE* je slična onoj koju koristi *MAFFT* (koji će biti opisan u poglavlju 3.3), tj. konstruiše se progresivno poravnanje na koje se zatim primenjuje horizontalno prečišćavanje. Radi boljeg razumevanja, u nastavku su prvo opisani elementi koje *MUSCLE* koristi u svom algoritmu, a kasnije i sam algoritam.



Slika 3.4: Primer progresivnog poravnanja

3.2.1. Progresivno poravnanje

Progresivno poravnanje se može predstaviti binarnim stablom u kom je svaka sekvenca dodeljena listu, a u unutrašnjim čvorovima se nalaze poravnanja direktnih potomaka. Stablo se konstruiše klasterovanjem trougaone matrice koja sadrži mere udaljenosti za svaki par sekvenci. Redosled obilaska stabla je *postorder* (tj. deca se posećuju pre roditelja). Na svakom unutrašnjem čvoru, profilno poravnanje (Slika 3.3) se koristi za poravnanje postojećih poravnanja podstabla i dobijeno poravnanje se dodeljuje tom unutrašnjem čvoru. Na kraju, višestruko poravnanje svih ulaznih sekvenci se dobija u korenu.

3.2.2. Mere sličnosti

MUSCLE koristi dve mere sličnosti: frakcioni identitet D koji se dobija iz globalnog poravnanja dve sekvence i k-mer rastojanje. K-mer je neprekidna podsekvenca dužine k, takođe poznata kao reč ili k-torka. Značajna je jer srodne sekvence teže ka tome da imaju više zajedničkih k-mera, pod uslovom da k nije preveliko

3.2.3. Mere rastojanja

Na osnovu date vrednosti sličnosti, možemo proceniti meru rastojanja. Mera rastojanja računa *d(A, B)*, udaljenost između dve sekvence *A* i *B*:

$$d(A, B) = d(A, C) + d(C, B) \tag{3.1}$$

gde je *C* bilo koja treća sekvenca, pod pretprostavkom da su *A*, *B* i *C* povezane.
Idealna mera rastojanja bi bilo rastojanje mutacije koje predstavlja broj mutacija do kojih je došlo između dve sekvence tokom evolucije, ali je ovo rastojanje najčešće nepoznato. Frakcioni identitet *D* često se koristi kao mera sličnosti i za blisko povezane sekvence *1 – D* je dobra aproksimacija rastojanja mutacije. Kako sekvence divergiraju, postoji velika verovatnoća višestrukih mutacija na jednoj lokaciji. Da bismo ovo ispravili koristi se sledeća procena udaljenosti, tzv. *Kimura* rastojanje:

$$d_{Kimura} = -log_e(1 - D - D^2/5) \tag{3.2}$$

3.2.4. Konstrukcija stabla navođenja

Stablo navođenja se konstruiše metodama klasterovanja na osnovu matrice rastojanja i ovde se koriste dve metode: *NJ* i *UPGMA*. *MUSCLE* implementira tri varijante metode *UPGMA* koje se razlikuju u dodeljivanju rastojanja novom klasteru. Na primer, posmatrajmo dva klastera (podstabla) *L* i *R* koji će biti spojeni u jedan novi klaster *P*, koji nakon spajanja postaje njihov roditelj u binarnom stablu. Novom klasteru možemo dodeliti prosečno rastojanje:

$$d_{PC}^{Avg} = (d_{LC} + d_{RC})/2 \tag{3.3}$$

Možemo dodeliti minimalno umesto prosečnog rastojanja:

$$d_{PC}^{Min} = min[d_{LC}, dRC] \tag{3.4}$$

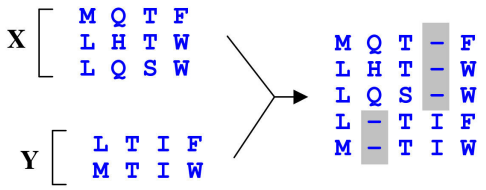
Ili čak kombinaciju minimalnog i prosečnog rastojanja:

$$d_{PC}^{Mix} = (1 - s) \cdot d_{PC}^{Min} + s \cdot d_{PC}^{Avg} \tag{3.5}$$

gde je parameter *s* podrazumevano 0.1

3.2.5. Profilno poravnanje

Dva profila (odnosno dva višestruka poravnanja) *X* i *Y* su poravnata tako da su kolone iz *X* i *Y* sačuvane u rezultujućem poravnanju. Kolone indela (Slika 3.4, siva pozadina) se ubacuju po potrebi kako bi se kolone mogle pravilno poravnati. Rezultat za poravnanje para kolona se određuje profilnom funkcijom, koja dodeljuje veću ocenu parovima kolona koji sadrže slične aminokiseline.



Slika 3.5: Primer profilnog poravnanja

3.2.6. Algoritam

Postoje tri osnovne faze algoritma: faza 1 (*draft progressive*), faza 2 (*improved progressive*) i faza 3 (*refinement*). Višestruko poravnanje je dostupno na kraju svake faze.

Faza 1, *Draft progressive*. Cilj prve faze je da se napravi višestruko poravnanje.

1.1 *K*-mer rastojanje je izračunato za svaki par ulaznih sekvenci; dobijamo matricu rastojanja *D1*.

1.2 Matrica *D1* je klasterovana pomoću *UPGMA* metode; dobijamo binarno stablo *TREE1*

1.3 Progresivno poravnanje se konstruiše tako što se prati redosled grananja u *TREE1*. Na svakom listu, profil je konstruisan iz ulazne sekvence. Čvorovi u stablu se obilaze u dubinu (postorder). Na svakom unutrašnjem čvoru dvostruko poravnanje dva profila-deteta je konstruisano, čime dobijamo novi profil koji je dodeljen tom čvoru. Na kraju dobijamo višestruko poravnanje svih ulaznih sekvenci, *MSA1*, u korenu.

Faza 2, *Improved progressive*. Glavni izvor greške u prvoj fazi je približno *k*-mer rastojanje, zbog čega dobijamo polu-optimalno stablo. *MUSCLE* stoga ponovo procenjuje stablo koristeći Kimura rastojanje, koje je preciznije, ali zahteva poravnanje.

2.1 Iz *MSA1* se izračunava Kimura rastojanje za svaki par ulaznih sekvenci; na kraju dobijamo matricu rastojanja *D2*.

2.2 Matrica *D2* je klasterovana pomoću *UPGMA* metode; dobijamo binarno stablo *TREE2*

2.3 Progresivno poravnanje se dobija tako što se prati *TREE2*, gde na kraju dobijamo višestruko poravnanje *MSA2*. Ovo se optimizuje izračunavanje poravnanja samo sa podstabla čiji se redosled grananja promenio u odnosu na *TREE1*.

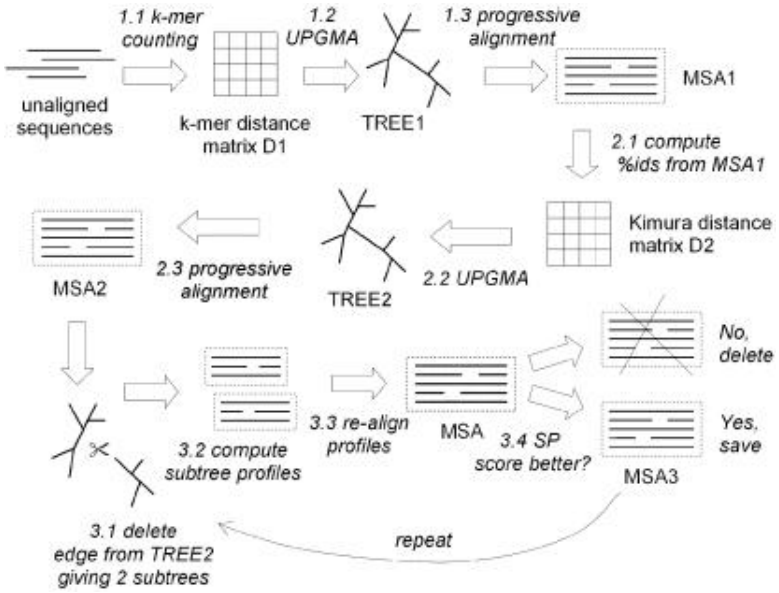
Faza 3, *Refinement*.

3.1 Grana je odabrana iz *TREE2*.

3.2 *TREE2* je podeljen na dva podstabla, brisanjem grane. Profil višestrukog poravnanja u svakom podstablu se izračunava.

3.3 Novo višestruko poravnanje se dobija ponovnim poravnanjem dva profila.

3.4 Ako je SP ocena poboljšana, novo poravnanje se zadržava, inače se odbacuje. Koraci 3.1-3.4 se ponavljaju do konvergencije ili dok se ne dostigne korisnički definisano ograničenje



Slika 3.6: Tok MUSCLE algoritma[6]

3.3. MAFFT

Još jedan kvalitetan algoritam višestrukog poravnanja je algoritam koji se naziva *MAFFT* (*Multiple Alignment using Fast Fourier Transform*). MAFFT koristi dve nove tehnike, *FFT* i pojednostavljeni sistem bodovanja. Homologne oblasti se identifikuju brzom Furijeovom transformacijom (*FFT*), dok uvedeni pojednostavljeni sistem bodovanja smanjuje procesorsko vreme i povećava kvalitet poravnanja. MAFFT takođe koristi heuristiku dvostrukog ciklusa, progresivni metod (*FFT-NS-2*) i metod iterativnog prečišćavanja (*FFT-NS-i*). U *FFT-NS-2* metodi, niskokvalitetne, dvostruke udaljenosti se brzo izračunavaju, konstruiše se privremeno višestruko poravnanje, zatim se prečišćena rastojanja izračunavaju iz višestrukog poravnanja i onda se izvršava druga metoda, *FFT-NS-i*, progresivna metoda jednog ciklusa koja je brža, ali i manje precizna od *FFT-NS-2*.

3.4. Kalign

Kalign je još jedan visokokvalitetni algoritam višestrukog poravnanja. Algoritam prati strategiju koja je vrlo slična standardnim progresivnim metodama, poput dvostrukog rastojanja koji se izračunava najpre pomoću *k*-torne metode usvojene iz *ClustalW*. Stablo navođenja se konstruiše pomoću *UPGMA* ili *NJ* metode, a progresivno poravnanje se izvršava obilaskom stabla. Nasuprot postojećim metodama, ono što čini ovaj algoritam drugačijim je korišćenje algoritma *Wu-Manber* za približno uparivanje niski, gde se rastojanje između dve niske meri pomoću Levenštajnovog rastojanja.

3.5. EMBOSS

Paket EMBOSS (*European Molecular Biology Open Software Suite*) sadrži niz aplikacija za poravnanje sekvenci, brzu pretragu baze podataka, identifikaciju proteinskih motiva (uključujući i analizu domena) i još mnogo toga. EMBOSS paket uključuje tzv. *water* i *needle* alate za *Smith-Waterman*-ov algoritam lokalnog poravnanja i *Needleman-Wunsch*-ov algoritam globalnog poravnanja. Alati dele isti stil interfejsa, tako da je prebacivanje između njih trivijalno.

4. Opis aplikacije

U ovom radu razvijena je aplikacija čiji je cilj da pruži zajednički interfejs za nekoliko popularnijih alata višestrukog poravnanja i da uporedi njihove performanse. Napisana je u programskom jeziku *Python*, koristi mnoge njegove pakete i razvijena je na operativnom sistemu *Ubuntu 16.04 LTS*. Korisnik zadaje sekvence učitavanjem datoteka u formatu FASTA[4] ili direktno navođenjem sekvenci u polje za unos teksta. Nakon toga, aplikacija prikazuje višestruka poravnanja unetih sekvenci dobijena odabranim alatima kao i njihove ocene. Pored ocena koje su već ugrađene i podržane od strane alata, za svako poravnanje se računa i konsenzus ocena.
Pre samog opisa aplikacije, sledi detaljan opis svih pomoćnih komponenti koje su doprinele razvoju ove aplikacije.

4.1. Pomoćni alati

4.1.1. *Biopython*

Python je programski jezik visokog nivoa koji je široko rasprostranjen i komercijalno i akademski. Njegova sintaksa je laka za učenje, poseduje objektno-orijentisane osobine programiranja i širok spektar biblioteka. *Biopython* [3] je skup alata napisanih u jeziku *Python* koji se može koristiti za različita biološka izračunavanja, simulacije i analize. Od uvođenja 1999. godine, *Biopython* je prerastao u veliku kolekciju modula, namenjenu upotrebi u računarskoj biologiji. Pored toga što sadrži klase kojima se mogu predstaviti biološke sekvence, *Biopython* omogućava i pristup raznim *online* biološkim bazama podatka, poput NCBI (*National Center for Biotechnology Information*).

Rad sa sekvencama

U biblioteci *Biopython* osnovna reprezentacija sekvence je predstavljena *Seq* objektom. *Seq* objekat je dosta sličan *Python* niski, s tim što su dodati alfabet (za deklarisanje tipa sekvence, npr. DNK ili protein) i neke ključne biološke metode (*complement*, *reverse_complement*,

[4]Tekstualni format za predstavljanje nukleinskih ili peptidnih sekvenci

transcribe, *translate*). Alfabeti se nalaze u modulu *Bio.Alphabet* koji sadrži klase za određivanje tipa *Seq* objekata:

- *Alphabet()*, generički alfabet, klasa koja se koristi kao bazna klasa ostalih tipova alfabeta
- S*ingleLetterAlphabet*, jednoslovni generički alfabet
- *ProteinAlphabet()*, jednoslovni generički proteinski alfabet
- *NucleotideAlphabet()*, jednoslovni generički nukleotidni alfabet
- *DNAAlphabet()*, jednoslovni generički alfabet za DNK
- *RNAAlphabet*, jednoslovni generički alfabet za RNK
- *SecondaryStructure*, alfabet koji opisuje sekundarnu strukturu i dopušta slova: '*H*', '*S*', '*T*', '*C*'
- *ThreeLetterProtein()*, troslovni proteinski alfabet koji dopušta sledeća "slova": "*Ala*", "*Asx*", "*Cys*", "*Asp*", "*Glu*", "*Phe*", "*Gly*", "*His*", "*Ile*", "*Lys*", "*Leu*", "*Met*", "*Asn*", "*Pro*", "*Gln*", "*Arg*", "*Ser*", "*Thr*", "*Sec*", "*Val*", "*Trp*", "*Xaa*", "*Tyr*", "*Glx*"

Na Slici 4.1 je prikazan *Seq* objekat sa generičkim alfabetom – što znači da nismo naveli da li je u pitanju DNK ili proteinska sekvenca i prikazano je korišćenje ugrađenih bioloških metoda *Seq* objekta.

```
>>> from Bio.Seq import Seq
>>> my_seq = Seq("ACGTACACTGGT")
>>> my_seq
Seq('ACGTACACTGGT', Alphabet())
>>> print my_seq
ACGTACACTGGT
>>> my_seq.alphabet
Alphabet()
>>> my_seq.complement()
Seq('TGCATGTGACCA', Alphabet())
>>> my_seq.reverse_complement()
Seq('ACCAGTGTACGT', Alphabet())
>>> my_seq.translate()
Seq('TYTG', ExtendedIUPACProtein())
>>> my_seq.transcribe()
Seq('ACGUACACUGGU', RNAAlphabet())
```

Slika 4.1: Primer afabeta i metoda *Seq* objekta

Parsiranje datoteta

Rad sa različitim formatima datoteka koje čuvaju biološke podatke je nezaobilazan deo bioinformatičkih analiza i u biblioteci *Biopython* se mogu naći razni moduli koji podržavaju rad sa različitim formatima datoteka. Na primer, modul *Bio.SeqIO* pruža jednostavan

```
>gi|6273291|gb|AF191665.1|AF191665 Opuntia marenae rpl16 gene;
chloroplast gene for chloroplast product, partial intron sequence
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAAATGAATCTAAATGATA
TAGGATTCCACTATGTAAGGTCTTTGAATCATATCATAAAAGACAATGTAATAAAGCATGAATACAGATT
CACACATAATTATCTGATATGAATCTATTCATAGAAAAAAGAAAAAAGTAAGAGCCTCCGGCCAATAAAG
ACTAAGAGGGTTGGCTCAAGAACAAAGTTCATTAAGAGCTCCATTGTAGAATTCAGACCTAATCATTAAT
CAAGAAGCGATGGGAACGATGTAATCCATGAATACAGAAGATTCAATTGAAAAAGATCCTATGNTCATTG
GAAGGATGGCGGAACGAACCAGAGACCAATTCATCTATTCTGAAAAGTGATAAACTAATCCTATAAAACT
AAAATAGATATTGAAAGAGTAAATATTCGCCCGCGAAAATTCCTTTTTTATTAAATTGCTCATATTTTCT
TTTAGCAATGCAATCTAATAAAATATATCTATACAAAAAAACATAGACAAACTATATATATATATATATA
TAATATATTTCAAATTCCCTTATATATCCAAATATAAAAATATCTAATAAATTAGATGAATATCAAAGAA
TCTATTGATTTAGTGTATTATTAAATGTATATATTAATTCAATATTATTATTCTATTCATTTTTATTCAT
TTTCAAATTTATAATATATTAATCTATATATTAATTTAGAATTCTATTCTAATTCGAATTCAATTTTTAA
ATATTCATATTCAATTAAAATTGAAATTTTTTCATTCGCGAGGAGCCGGATGAGAAGAAACTCTCATGTC
CGGTTCTGTAGTAGAGATGGAATTAAGAAAAAACCATCAACTATAACCCCAAAAGAACCAGA

>gi|6273290|gb|AF191664.1|AF191664 Opuntia clavata rpl16 gene;
chloroplast gene for chloroplast product, partial intron sequence
TATACATTAAAGGAGGGGGATGCGGATAAATGGAAAGGCGAAAGAAAGAAAAAAATGAATCTAAATGATA
TAGGATTCCACTATGTAAGGTCTTTGAATCATATCATAAAAGACAATGTAATAAAGCATGAATACAGATT
CACACATAATTATCTGATATGAATCTATTCATAGAAAAAAGAAAAAAGTAAGAGCCTCCGGCCAATAAAG
ACTAAGAGGGTTGGCTCAAGAACAAAGTTCATTAAGAGCTCCATTGTAGAATTCAGACCTAATCATTAAT
CAAGAAGCGATGGGAACGATGTAATCCATGAATACAGAAGATTCAATTGAAAAAGATCCTAATGNTNCAT
TGGGAAGGATGGCGGAACGAACCAGAGACCAATTCATCTATTCTGAAAAGTGATAAACTAATCCTATAAA
ACTAAAATAGATATTGAAAGAGTAAATATTCGCCCGCGAAAATTCCTTTTTTATTAAATTGCTCATATTT
TCTTTTAGCAATGCAATCTAATAAAATATATCTATACAAAAAAACATAGACAAACTATATATATATATAA
TATATTTCAAATTCCCTTATATATCCAAATATAAAAATATCTAATAAATTAGATGAATATCAAAGAATCT
ATTGATTTAGTGTATTATTAAATGTATATATTAATTCAATATTATTATTCTATTCATTTTTATTCATTTT
CAAATTTATAATATATTAATCTATATATTAATTTAGAATTCTATTCTAATTCGAATTCAATTTTTAAATA
TTCATATTCAATTAAAATTGAAATTTTTTCATTCGCGAGGAGCCGGATGAGAAGAAACTCTCATGTCCGG
TTCTGTAGTAGAGATGGAATTAAGAAAAAACCATCAACTATAACCCCAAAAGAACCAGA
```

Slika 4.2: Primer jedne datoteke u FASTA formatu ("*opuntia.fasta*")

interfejs za rad sa biološkim datotekama i to u raznim formatima, poput FASTA, GenBank, EMBL, Swiss-Prot, ClustalW, PHYLIP, NEXUS i Stockholm. Bez obzira o kakvom se formatu radi, informacije se čuvaju kao *SeqRecord* objekti.

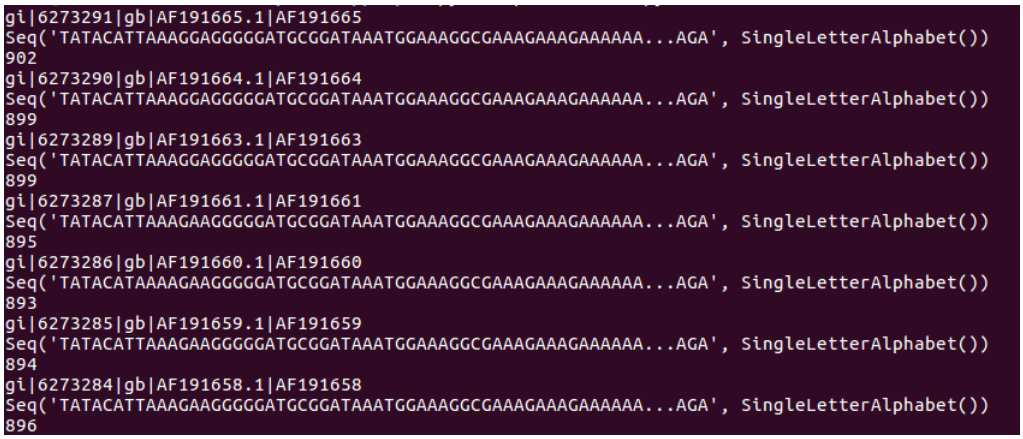
Slika 4.2 prikazuje izgled jedne datoteke u FASTA formatu. Datoteka sadrži podatke o nukleotidnim sekvencama, svaka počinje simbolom "**>**" nakon koga sledi sekvenca.

Na primer, nakon parsiranja datoteke u FASTA formatu (Slika 4.3), dobićemo prikaz sekvenci i dodatne informacije koje smo odabrali prilikom parsiranja (Slika 4.4).

```
1  from Bio import SeqIO
2
3  handle = open("opuntia.fasta")
4  for seq_record in SeqIO.parse(handle, "fasta"):
5      print seq_record.id          # id sekvence
6      print repr(seq_record.seq) # sekvenca
7      print len(seq_record)       # duzina sekvence
8  handle.close()
```

Slika 4.3: Parsiranje datoteke u formatu FASTA

Kada je reč o datotekama koje sadrže rezultat višestrukog poravnanja, *Bio.SeqIO* ih intrepretira kao kolekcije sekvenci istih dužina, dok modul *Bio.AlignIO* direktno radi sa pora-



Slika 4.4: Rezultat parsiranja datoteke "*opuntia.fasta*"

vnanjima, uključujući i datoteke koje sadrže više od jednog poravnanja. *Biopython* takođe podržava alate za izvršavanje operacija nad sekvencama, poput translacije, transkripcije i izračunavanja težine. Sadrži i module za mašinsko učenje, kao što su Bajesova metoda, Markovljevi modeli i klasterovanje. Navešćemo još nekoliko korisnih modula:

- Modul *Bio.Phylo* pruža alate za rad i prikaz filogenetskih stabala
- Modul *GenomeDiagram* pruža metode za prikaz sekvenci, gde sekvence mogu biti prikazane u linearnoj ili cirkularnoj formi.
- *Bio.PDB* modul može da učita strukture iz PDB i mmCIF datoteka. *Structure* objekat je centralni objekat ovog modula, organizuje makromolekularnu strukturu u obliku hijerarhije. Pomoću ovog modula se možemo kretati kroz individualne komponente makromolekularne strukturne datoteke, kao što je ispitivanje svakog atoma u proteinu.
- *Bio.PopGen* modul podržava *Genepop*, softverski paket za statističku analizu populacione genetike.
- Mnogi moduli *Biopython*-a sadrže tzv. omotače komandne linije, poput BLAST, Clustal, PhyML, EMBOSS i SAMtools.

Objekti višestrukog poravnanja sekvenci

U biblioteci *Biopython* se višestruko poravnanje sekvenci posmatra kao kolekcija višestrukih sekvenci koje su poravnate, obično umetanjem karaktera "-", tako da su sve sekvence istih dužina. *MultipleSeqAlignment* objekat je taj koji čuva ove podatke i koristi se u kombinaciji sa *Bio.AlignIO* modulom koji se koristi za njihovo čitanje i pisanje u raznim formatima. Razlikujemo dve funkcije za čitanje poravnanja: *Bio.AlignIO.read()*, za čitanje datoteka sa

dvostrukim poravnanjem i *Bio.AlignIO.parse()*, za čitanje datoteka sa višestrukim poravnanjem. Obe funkcije očekuju dva argumenta:

1. Ime datoteke ili otvorena datoteka
2. Niska koja precizira format poravnanja

Za pisanje se koristi funkcija *Bio.AlignIO.write()* koja uzima tri argumenta: *MultipleSeqAlignment* objekte, ime datoteke i format sekvence.

Alati za dvostruko i višestruko poravnanje sekvenci

Biblioteka *Biopython* se može koristiti za pozivanje alata za poravnanje iz komandne linije i to na sledeći način:

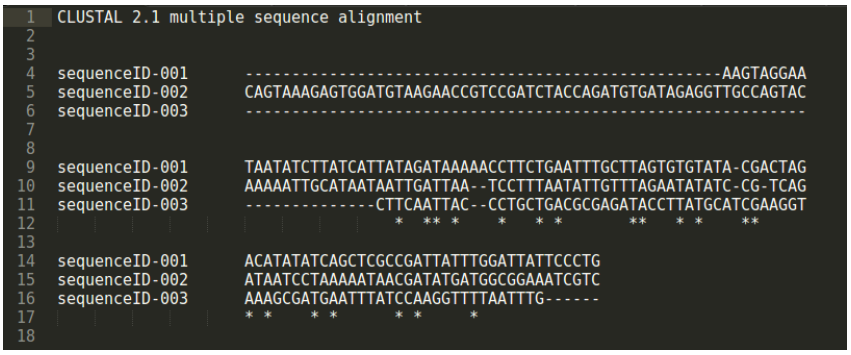
- Pripremi se ulazna datoteka koja sadrži sekvence za koje želimo izvršiti poravnanje
- Iz komandne linije se pozove alat koji obrađuje ovu datoteku, obično preko omotača biblioteke *Biopython*
- Pročita se izlaz korišćenog alata, tj. poravnanje, pomoću *Bio.AlignIO*.

```
1  >sequenceID-001 description
2  AAGTAGGAATAATATCTTATCATTATAGATAAAAACCTTCTGAATTTGCTTAGTGTGTAT
3  ACGACTAGACATATATCAGCTCGCCGATTATTTGGATTATTCCCTG
4
5  >sequenceID-002 description
6  CAGTAAAGAGTGGATGTAAGAACCGTCCGATCTACCAGATGTGATAGAGGTTGCCAGTAC
7  AAAAATTGCATAATAATTGATTAATCCTTTAATATTGTTTAGAATATATCCGTCAGATAA
8  TCCTAAAAATAACGATATGATGGCGGAAATCGTC
9
10 >sequenceID-003 description
11 CTTCAATTACCCTGCTGACGCGAGATACCTTATGCATCGAAGGTAAAGCGATGAATTTAT
12 CCAAGGTTTTAATTTG
```

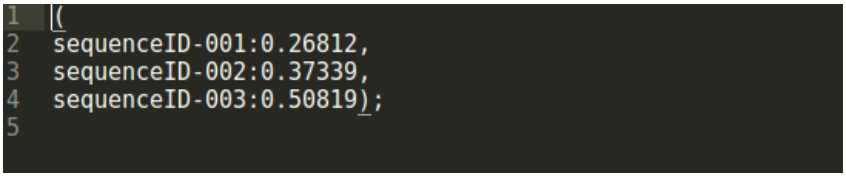
Slika 4.5: *Sadržaj datoteke "test.fasta"*

Za potrebe aplikacije su korišćena dva popularna alata za višestruko poravnanje: ClustalW i MUSCLE. Pored alata višestrukog poravnanja dodati su i alati dvostrukog poravnanja: modul biblioteke *Biopython pairwise2* i alati *water* (*Smith-Waterman* algoritam) i *needle* (*Needleman-Wunsch* algoritam) za lokalno i globalno poravnanje, koji su deo EMBOSS paketa.

Alat ClustalW podrazumevano vraća poravnanje i datoteku sa opisom stabla navođenja, koju možemo parsirati pomoću *Bio.Phylo* kako bismo dobili formatirani prikaz stabla. Na primer, za ulaznu datoteku "*test.fasta*" (Slika 4.3) u kojoj se nalaze sekvence koje treba poravnati, ClustalW će generisati dve datoteke: "*test.aln*" (Slika 4.4) u kojoj se nalazi višestruko



Slika 4.6: *Sadržaj datoteke "test.aln"*

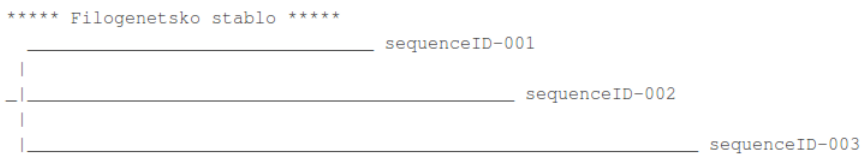


Slika 4.7: Sadržaj datoteke "*test.dnd*"

poravnanje i "*test.dnd*" (Slika 4.5) koja sadrži informacije o filogenetskom stablu. Nakon parsiranja "*test.dnd*" datoteke, dobijamo formatirano filogenetsko stablo (Slika 4.6).

Alat MUSCLE podrazumevano vraća datoteku u formatu FASTA koju možemo parsirati pomoću *Bio.AlignIO* modula kako bismo pročitali poravnanje (Slika 4.7). Modul *Bio.Emboss.Applications* poseduje omotače za alate paketa EMBOSS, *needle* i *water*, kao i omotače za EMBOSS verzije PHYLIP alata. Na primer, ako želimo da nađemo globalno poravnanje dve sekvence, potrebno je da pripremimo dve ulazne datoteke u formatu FASTA. Kreiramo *needle* objekat komandne linije, pokrenemo komandu i na kraju dobijamo izlaznu datoteku koju kasnije parsiramo kao i obično, pomoću *Bio.AlignIO* modula. Ono što je zanimljivo kod alata paketa EMBOSS je to što druga ulazna datoteka može imati više sekvenci (npr. tri) i alat će u tom slučaju napraviti tri dvostruka poravnanja.

Za dvostruka globalna i lokalna poravnanja *Biopython* ima *Bio.pairwise2* modul, koji je dopunjen funkcijama napisanim u programskom jeziku C radi poboljšanja efikasnosti. Podrazumevana funkcija globalnog poravnanja je *align.globalxx* (analogno *align.localxx* za lokalno poravnanje). U nazivu funkcije su nam posebno interesantna poslednja dva karaktera, odnosno "**xx**", koja se koriste za definisanje vrednosti za poklapanja, nepoklapanja i praznine. Prvim karakterom definišemo vrednost za poklapanje, na primer, *x* podrazumeva da se ta vrednost računa kao 1, dok nepoklapanja nemaju cenu. Ako su nam potrebne drugačije vrednosti za poklapanja, obično koristimo matrice ocena, na primer za aminokiseline ocene se nalaze u PAM i BLOSUM matricama. Drugi karakter predstavlja kaznu za



Slika 4.8: Filogenetsko stablo

```
SingleLetterAlphabet() alignment with 3 rows and 159 columns
----CAGTAAAGAGTGGATGTAAGAACCGTCCGATCTACCAGAT...GTC sequenceID-002
CTTCAATTACCCTGCTGACGCGAGA-----------TACCTTAT...--- sequenceID-003
-----------------AAGTAGGAATAA-------TATCTTAT...CTG sequenceID-001
```

Slika 4.9: Izlaz primene MUSCLE alata za *test.fasta*

praznine, gde x podrazumeva da praznine nemaju cenu. Takođe, ako umesto x ubacimo s, možemo dodati kazne za tzv. otvarajuće i proširujuće praznine.

4.1.2. Qt i PySide

U razvoju aplikacije, jedan od izazova je bilo preusmeravanje izlaza prethodno opisanih alata i implementacija jednostavnog korisničkog grafičkog interfejsa. Kako *Python* sam sa svojim paketima ne pruža nikakav grafički prikaz, korišćena je kombinacija dva programa: *QtCreator* i *PySide*. *Qt* je višeplatformski aplikacioni interfejs koji se koristi za razvoj grafički orijentisanih programa i negrafičkih programa, poput alata komandne linije i konzolnih servera. *Qt* dolazi sa sopstvenim skupom alata, poput alata *Qt Creator* koji predstavlja višeplatformsko integrisano razvojno okruženje (eng. *Integrated Development Environment, IDE*) za *C++* i *QML*, *Qt Designer* interfejsa, *Qt Assistant* pomoćnog pretraživača, *Qt Linguist* alata prevođenja, itd.

PySide je slobodni softver koji je razvila *The Qt Company* i predstavlja vezivanje jezik a*Python* na *Qt*, tj. API (*Application Programming Interface*) koji nam omogućava da koristimo *Qt* i sve njegove alate u jeziku *Python*.

Ceo proces korišćenja ove kombinacije je vrlo jednostavan. Potrebno je prvo instalirati *PySide* i *QtCreator*. U *QtCreator*-u kreiramo dizajn aplikacije koji sačuvamo kao npr. "ime_datoteke.ui". Prevodimo ovu datoteku u *Python* datoteku na sledeći način:

```
pyside-uic ime_datoteke.ui -o ime_datoteke.py
```

Datoteku koju smo izgenerisali dodamo u zaglavlju glavnog *Python* programa, nakon čega možemo raditi sa komponentama aplikacije (dugmićima, labelama, dijalozima itd.) i praviti neke svoje događaje (tzv. *events*).

4.1.3. Pohlepni algoritam višestrukog poravnanja

Pored ugrađenih alata višestrukog poravnanja, aplikaciji je dodata i implementacija pohlepnog algoritma [10]. Čak i bez dodatnih testiranja korisnik može videti koliko su ugrađeni alati brži od običnog pohlepnog algoritma koji ne koristi nikakvu heuristiku.
Osnovna ideja pohlepnog algoritma je da razmatra sve moguće parove poravnanja i da postepeno kreira sklop (eng. *assemble*) na osnovu najboljeg takvog para. Najbolja poravnanja se spajaju i kreiraju kontigu (eng. *contig*), koja predstavlja neprekidnu nisku sekvence. Algoritam kreće sa upoređivanjem svih parova sekvenci, računa njihovu ocenu i kreira trougaonu matricu *M*. Svaki element matrice *M* predstavlja rezultat jednog para, na primer e_{ij} predstavlja ocenu poravnanja sekvence *i* i sekvence *j*. U svakoj iteraciji iz matrice *M* biramo najveći element (pošto on predstavlja trenutno najbolje poravnanje), obrađujemo taj par sekvenci i zatim taj element iz *M* zamenjujemo nulom, kako bismo u sledećoj iteraciji našli sledeću najveću ocenu. Najbitniji korak algoritma je kreiranje sklopa. Sklop možemo shvatiti kao listu kontiga i cilj algoritma je da na kraju ostanemo samo sa jednim elementom te liste, odnosno sa jednom kontigom koja će predstavljati najbolje višestruko poravnanje. Kreće se sa praznim sklopom koji postepeno gradimo. Kada naiđemo na *sa* i *sb*, dve poravnate sekvence, obrađujemo ih na jedan od četiri načina:

1. Ako se ni *sa* ni *sb* ne nalaze u nekoj već postojećoj kontigi, onda se od njih kreira nova kontiga
2. Ako *sa* već pripada nekoj kontigi, a *sb* ne pripada, onda se *sb* dodaje kontigi u kojoj se nalazi *sa*.
3. Ako *sa* pripada jednoj, a *sb* nekoj drugoj kontigi, onda se te dve kontige spajaju.
4. Ako *sa* i *sb* pripadaju istoj kontigi, ništa se ne menja.

Osnovne funkcije implementiranog algoritma su sledeće:

- **ChopSeq** – funkcija koja kao ulazni argument prima sekvencu, željeni broj segmenata i dužinu segmenata; vraća slučajno generisane segmente.
- **NewContig** – funkcija koja dva segmenta spaja u jednu zajedničku kontigu; koristi se kada ulazni segmenti ne pripadaju nijednoj drugoj kontigi, pa je potrebno kreirati novu.
- **ShowContig/ShowContig2** – funkcije za formatirano ispisivanje kontige.
- **Finder** – pomoćna funkcija koja vraća pozicije tražene sekvence u kontigi
- **Add2Contig** – funkcija koja modifikuje već postojeću kontigu, tako što joj dodaje novu sekvencu i ostale sekvence pomera za određeni broj mesta kako bi cela kontiga sadržavala lepo poravnate sekvence

– **JoinContigs** – funkcija koja spaja dve već postojeće kontige

– **Assemble** – najbitnija funkcija koja u petlji gradi finalni sklop; iz petlje se izlazi kada matrica *M* više ne sadrži vrednosti koje ispunjavaju uslov (vrednosti veće od nekog zadatog praga *gamma*); na kraju se proverava da li su sve sekvence deo finalnog sklopa, ako neka od njih nije, od nje se formira kontiga i kao takva se dodaje sklopu

Primer 4.1.1 *Primer rada pohlepnog algoritma*

Skup ulaznih sekvenci:

s0 MENREGIIEF

s1 GGGVGGFYTIV

s2 IYRRAIDIDK

s3 YPIYRRAIDI

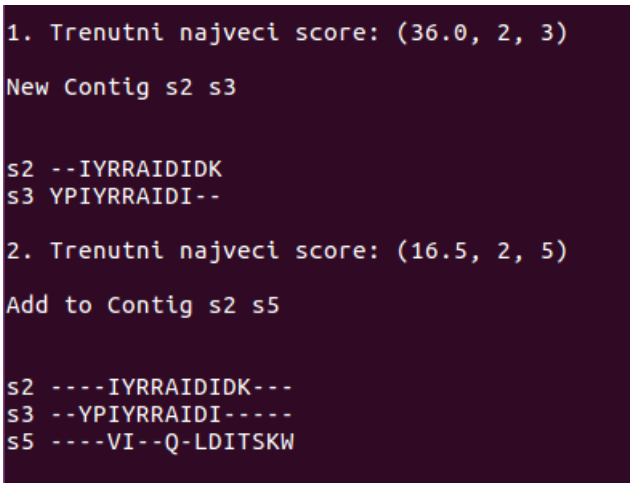
s4 YLESKLLKGP

s5 VIQLDITSKW

s6 LFDFVGEPRY

s7 EFYENKKYRI

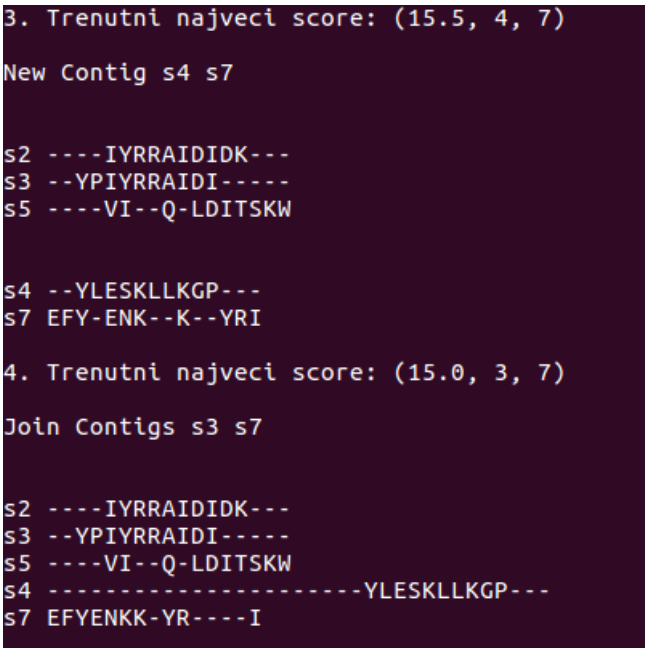
U svakom koraku u matrici M pronalazimo trenutnu najveću ocenu i obrađujemo par sekvenci koji odgovara toj oceni.



Slika 4.10: Kreiranje nove kontige i pridruživanje nove sekvence već postojećoj kontigi

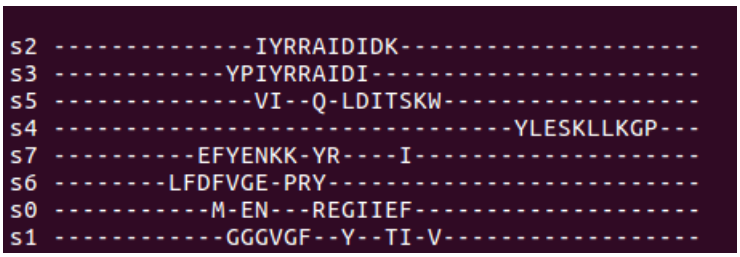
Na samom početku najveća ocena je pronađena za sekvence s2 i s3. U ovom slučaju, kako ni s2 ni s3 ne pripadaju nijednoj kontigi, kreiramo novu kontigu. Sledeća najveća ocena je pronađena za sekvence s2 i s5. Kako sekvenca s2 već pripada jednoj kontigi, sekvencu s5 dodajemo toj istoj kontigi (Slika 4.9).

Nastavljamo dalje i najveću ocenu pronalazimo za sekvence s4 i s7. Pošto ove sekvence ne pripadaju nijednoj kontigi, od njih pravimo novu. Sledeći zanimljiv korak je spajanje kontiga. Pronađena je najveća ocena koja odgovara sekvencama s3 i s7. Sekvenca s3 pripada jednoj kontigi (koja se sastoji od s2, s3 i s5), a sekvenca s7 pripada drugoj kontigi (koja se sastoji od s4 i s7) i u ovom slučaju spajamo ove dve kontige (Slika 4.10).



Slika 4.11: Kreiranje nove kontige i spajanje dve kontige u jednu

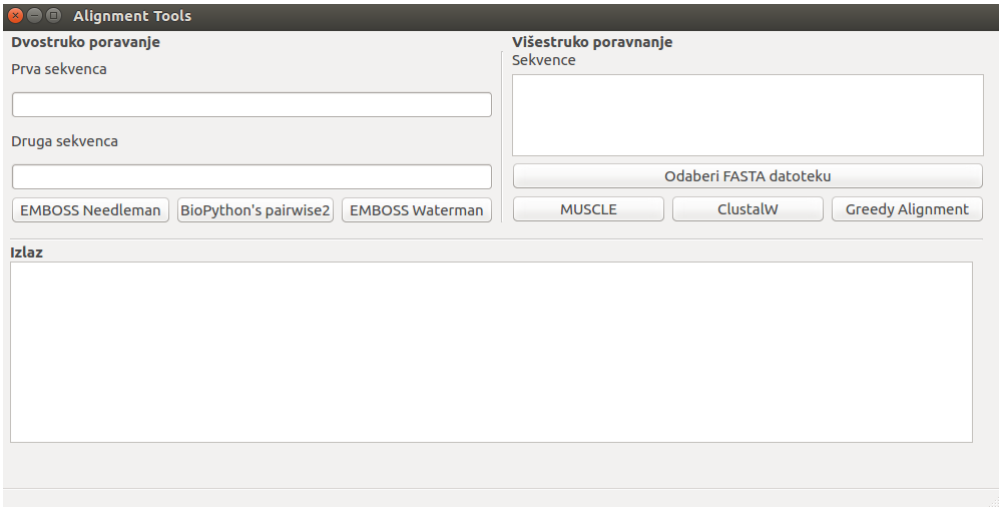
Koraci se dalje nastavljaju po istom principu. Negde spajamo dve kontige u jednu, negde kreiramo novu, a negde pridružujemo sekvence već postojećim kontigama. U ovom primeru smo imali tačno 16 koraka i finalni sklop, odnosno poravnanje, koji na kraju dobijamo je prikazan na Slici 4.11.



Slika 4.12: Finalno poravnanje

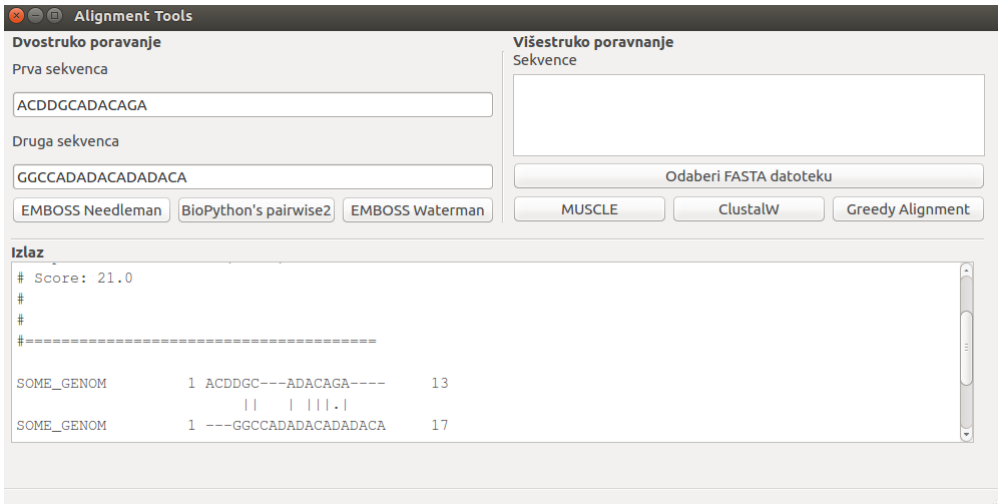
4.2. Izgled aplikacije i uputstvo za korišćenje

Inicijalni izgled aplikacije je prikazan na Slici 4.9.



Slika 4.13: Inicijalni izgled aplikacije

Leva sekcija prozora se koristi za dvostruko poravnanje sekvenci. Korisnik unosi sekvence za koje želi izvršiti dvostruko poravnanje i onda bira jedan od tri alata: *EMBOSS Needleman* ako želi globalno poravnanje, *EMBOSS Waterman* ako želi lokalno poravnanje i *pairwise2* ako želi podrazumevano obično poravnanje sa podrazumevanom shemom bodovanja (vrednost ya poklapanje je 1, za nepoklapanje je 0 i kazne za praznine ne postoje). Na primer, na Slici 4.10 se vidi izlaz jednog takvog alata, prikazana je ocena dvostrukog poravnanja i finalno dvostruko poravnanje unetih sekvenci.



Slika 4.14: Izlaz EMBOSS-ovog *needle* alata

Kod višestrukog poravnanja korisnik može sam uneti sekvence razdvojene prelaskom u novi red. Program će učitati tako unete sekvence i smestiti ih u odgovarajuću datoteku koja će se prosleđivati alatima. Pored ove mogućnosti ručnog unošenja sekvenci, korisnik može odabrati i datoteku u FASTA formatu koja sadrži sekvence za koje se treba izvršiti višestruko poravnanje. U tom slučaju program će ih ispisati u deo koji je prethodno bio predviđen za ručno unošenje. Nakon toga, korisnik bira jedan od tri alata kojim će se izvršiti višestruko poravnanje: MUSCLE, ClustalW ili *Greedy Alignment*. Na Slici 4.11 vidimo izlaz ClustalW alata, gde pored višestrukog poravnanja imamo i prikaz filogenetskog stabla.

Slika 4.15: Izlaz ClustalW alata

Iako ugrađeni alati višestrukog poravnanja imaju svoj način izračunavanja ocena (uglavnom koriste SP ocenu, koja će biti detaljnije objašnjena u narednom poglavlju), implementirano je i izračunavanje konsenzus ocene na osnovu koje možemo uporediti alate za isti skup ulaznih sekvenci. Da bismo izračunali konsenzus ocenu višestrukog poravnanja potrebno je prvo naći konsenzus višestrukog poravnanja koji predstavlja sekvencu najčešćih karaktera u svakoj koloni poravnanja. Zatim se kolone konsenzusa i sekvenci upoređuju i računamo vrednost kolone na osnovu sledeće funkcije [9]:

$$d(x,y) = \begin{cases} 2\, za\, x \neq y \\ 1.5\, za\, x = - \, ili\, y = - \, ali\, (x,y) \neq (-,-) \\ 0,\, inace \end{cases} \tag{4.1}$$

Nakon toga, saberemo vrednosti kolone i dobijamo konsenzus ocenu. Što je konsenzus ocena manja to imamo bolje poravnanje. Na primer, za iste ulazne sekvence konsenzus ocena alata MUSCLE je 86.0, alata ClustalW je 74.5, dok pohlepni algoritam vraća poravnanje sa

ocenom od od 309.5. Na osnovu ovih ocena možemo zaključiti da je alat ClustalW vratio najbolje višestruko poravnanje.

5. Poređenje performansi alata

Pored razvoja aplikacije, zadatak ovog rada bio je i da uporedimo performanse predstavljenih alata višestrukog poravnanja. U ovom radu alati su upoređivani na osnovu njihove brzine izvršavanja i na osnovu kvaliteta višestrukog poravnanja koji ovi alati kreiraju. Međutim, da bismo mogli da uporedimo kvalitet potreban nam je neki standard ili skup test podataka za koji već znamo da predstavlja kvalitetna višestruka poravnanja. Za ove referentne skupove podataka široko je rasprostranjen termin *benchmark*.
Većina radova koja se bavi upoređivanjem peformansi alata višestrukog poravnanja koristi neki od poznatih *benchmark*-ova [2], pa je tako i u ovom radu korišćen jedan od njih, i to poznati BAliBASE *benchmark*. BAliBASE je dizajniran tako da služi kao resurs za evaluaciju kako bi se rešili problemi sa kojima se susrećemo prilikom poravnanja sekvenci i tako uključuje nekoliko tzv. referentnih skupova. Prva verzija je imala pet referentnih skupova koji su posebno dizajnirani da predstavljaju probleme sa kojima se susrećemo prilikom višestrukog poravnanja globularnih proteina i slično, a druga verzija uključuje tri referentna skupa koja su posvećena ponavljanjima, cirkularnim permutacijama i transmembranskim proteinima. U ovom radu su korišćena dva referentna skupa: *BAliBASE Reference Set 9* i *BAliBASE Reference Set 10*.

5.1. BAliBASE Reference Set 9

BAliBASE Reference Set 9 [16] je organizovan u četiri podskupa referentnih poravnanja koji sadrže proteinske familije sa linearnim motivima (*LM*), koji su dizajnirani da procene tačnost algoritama višestrukog poravnanja u različitim uslovima. LM-ovi uključuju važne funkcionalne regione proteina koji se često nalaze u neuređenim delovima za koje je teško izvršiti poravnanje klasičnim metodama višestrukog poravnanja. U skupu razlikujemo sekvence sa *true positive*, *false positive* i *false negative* motivima. *True positive* motiv je motiv koji se poklapa sa ELM-om[5] koji se javlja u oblasti sekvence koja se može poravnati sa instancom

[5]Eukariotski linearni motiv, računski biološki resurs, razvijen u Evropskoj laboratoriji za molekularnu biologiju (EMBL) za istraživanje kratkih linearnih motiva (eng. *short linear motifs, SLiMs*) u eukariotskim proteinima

motiva. *False positive* motiv je motiv koji odgovara ELM-u, ali koji je pronađen u delu sekvence koji se ne može poravnati sa instancom motiva. Suprotno, *false negative* motiv se definiše kao deo sekvence koji se može poravnati sa instancom motiva, ali se ne poklapa sa ELM-om.

Podskup 1 sadrži samo sekvence sa validnim LM-ovima (*true positive* ili *false negative* motivi). Podskup je dalje organizovan u tri različite grupe na osnovu varijabilnosti sekvence:

- RV911 - <20% identity
- RV912 - 20-40% identity
- RV913 - 40-80% identity

Podskup 2 sadrži sekvence sa mogućim greškama (to su loše predviđene sekvence, fragmenti, varijante spajanja). Ove sekvence dele neke homologije sa referentnom sekvencom, ali ne sadrže ELM motiv:

- RV921 – sekvence sa *true positive* motivima
- RV922 – sekvence sa *true positive* motivima i sekvence sa greškama

Podskup 3 sadrži *true positive* sekvence poravnate sa sekvencama koje imaju *false positive* motive, tj. sekvence sa *false positive* poklapanjima, koja se nalaze na drugim pozicijama u poravnanju i koja ne odgovaraju instanci motiva:

- RV931 – sekvence sa *true positive* motivima
- RV932 – sekvence sa *true positive* motivima i *false positive* motivima

Podskup 4 sadrži *true positive* sekvence poravnate sa sekvencama koje ne sadrže nikakav primer motiva:

- RV941 – sekvence sa *true positive* motivima
- RV942 – sekvence sa *true positive* motivima i *true negative* motivima

Za svako referentno poravnanje postoje tri odgovarajuće datoteke:

- *.in_tfa – neporavnate sekvence
- *.msf – poravnanje u GCG MAF formatu
- *.xml – poravnanje u XML formatu sa oznakama za motiv

BAliBASE *Reference Set 9* je javno dostupan[6] i pored referentnih poravnanja možemo preuzeti i *C* program tzv. *bali_score* koji upoređuje naša poravnanja sa referentnim i izračunava ocenu za kvalitet poravnanja (*Sum-Of-Pairs* ocena, SP i *Total-Column* ocena, TC) . Da bi se ovaj program pokrenuo potrebno je imati instaliran XML parser *expat*. Program se pokreće:

```
bali_score ref_aln test_aln
```

[6] `http://www.lbgi.fr/balibase/BAliBASE_R9/`

gde je *ref_aln* referentno poravnanje u formatu xml/msf, a *test_aln* je test poravnanje u formatu msf.

Dakle, program *bali_score* računa SP ocenu koja procenjuje tačnost poravnanja motiva. SP ocena odgovara procentu ispravno poravnatih parova karaktera u poravnanju. Uzimaju se u obzir samo sekvence sa *true positive* ili *false negative* motivima, dok poravnanje ostalih sekvenci nema uticaja na SP ocenu. Na primer, pretpostavimo da imamo test poravnanje i *N* sekvenci tog poravnanja sadrži *true positive* ili *false negative* motive. Pretpostavimo da referentni motiv odgovara *M* kolonama u poravnanju, i *i*-tu kolonu u poravnanju motiva označavamo sa A_{i1}, A_{i2},...,A_{iN}. Za svaki par karaktera A_{ij} i A_{ik}' se definiše p_{ijk}, tako da je p_{ijk} = 1 ako su A_{ij} i A_{ik} međusobno poravnati u referentnom poravnanju, a inače je p_{ijk} = 0. Ocena *i*-te kolone se definiše kao:

$$S_i = \sum_{j=1, j\neq k}^{N} \sum_{k=1}^{N} p_{ijk} \tag{5.1}$$

SP ocena poravnanja je zatim:

$$SPS = \frac{\sum_{i=1}^{M} S_i}{\sum_{i=1}^{M_r} S_{ri}} \tag{5.2}$$

gde je M_r broj kolona koje odgovaraju referentnom motivu, a S_{ri} je ocena za S_i, *i*-te kolone u referentnom poravnanju.

5.2. BAliBASE Reference Set 10

BAliBASE Reference Set 10 [19] je referentni skup koji se sastoji od 218 referentnih poravnanja i 17892 proteinskih sekvenci. Skup se fokusira na: podfamilije određenih osobina, motive u neuređenim regionima i efekat pogrešnih sekvenci na kvalitet višestrukog poravnanja. Takođe, ovaj referentni skup za svako referentno poravnanje identifikuje lokalno konzervirane regione, tzv. blokove. Rezultujuća *benchmark* poravnanja pokrivaju neke od problema specifičnih za poravnanje velikih skupova kompleksnih sekvenci proteina. Poravnanja takođe sadrže i veliki broj sekvenci sa odstupanjem (odnosno nepoklapanjem), npr sekvence sa neočekivanim proširenjima, insercijama ili delecijama.

Kao i prethodni referentni skup i BAliBASE *Reference Set 10* je javno dostupan[7]. Referentna poravnanja možemo naći u datotekama koje su u formatu *.msf ili *.xml, a sekvence u datotekama u formatu *.tfa. Kao i kod BAliBASE *Reference Set 9* i ovde pored ovih podataka, možemo preuzeti i *C* program: *bali_score* koji upoređuje naša poravnanja sa referentnim i izračunava ocene za kvalitet poravnanja. Ovaj program *bali_score* se pokreće na sličan način kao i prethodni (pomenut u poglavlju 5.1), s tim što zahteva različit format ulaznih podataka:

[7] `http://www.lbgi.fr/balibase/BAliBASE_R10/`

```
bali_score ref_aln test_aln
```

gde je *ref_aln* referentno poravnanje u formatu xml/fasta , a *test_aln* je test poravnanje u formatu msf/fasta.
Objasnićemo kako program *bali_score* nalazi ocene. Pretpostavimo da imamo test poravnanje od *N* sekvenci i *M* blokova, svaki blok *b* se sastoji od n_b sekvenci i m_b kolona i *i*-toj koloni bloka se dodeljuje ocena C_{bi} = 1 ako su svi karakteri u koloni ispravno poravnati, a inače se dodeljuje C_{bi} = 0. Ukupan kvalitet poravnanja, tzv. *Column Score* (CS) se izračunava na sledeći način:

$$CS = \frac{\sum_{b=1}^{M} \frac{n_b \sum_{i=1}^{m_b} C_{bi}}{m_b}}{\sum_{b=1}^{M} n_b} \tag{5.3}$$

5.3. Rezultati

Kako bi se uporedila poravnanja alata predstavljenih u aplikaciji sa refrentnim poravnanjima, prvo su iskorišćeni podaci iz *Reference Set 10*. Međutim, izvršavanje alata *Greedy Alignment* je trajalo veoma dugo na kućnom računaru prosečne konfiguracije (čak ni nakon 40 minuta nisu dobijena rešenja), na osnovu čega možemo zaključiti da ovaj alat radi isuviše sporo sa velikim brojem sekvenci. Zbog toga je za evaluaciju ovog alata iskorišćen *Reference Set 9* koji sadrži poravnanja sa značajno manjih brojem sekvenci u odnosu na *Reference Set 10*.
Proces evaluacije alata u oba slučaja je isti:

1. Kreiramo višestruko poravnanje, tzv. test poravnanje
2. Odredimo vreme izvršavanja alata
3. Uporedimo test poravnanje sa odgovarajućim referentnim poravananjem i izračunamo ocenu poravnanja

U nastavku je prikazan samo deo podataka koji su sakupljeni prilikom upoređivanja perfomansi alata. Prvo su iskorišćeni podaci iz *Reference Set 9* i u Tabeli 5.1 su predstavljene SP ocene poravnanja. Možemo primetiti da je rad alata ClustalW i MUSCLE daleko bolji od rada lata *Greedy Alignment*, ako posmatramo kvalitet optimalnog poravnanja i brzinu izvršavanja. ClustalW i MUSCLE daju poravnanja koja su približno istog kvaliteta i očigledno je da ClustalW daje za nijansu bolja poravnanja u slučaju većeg broja sekvenci.
Iako postoje paralelne verzije alata višestrukog poravnanja (poput *ClustalW-MPI*, paralelna verzija alata ClustalW), u ovom radu su oba alata ClustalW i MUSCLE sekvencijalna. Pošto se alati ClustalW, MUSCLE i implementirani *Greedy Alignment* izvršavaju na istom broju jezgara (na jednom), moguće je njihovo međusobno upoređivanje vremena izvršavanja, gde je vreme izvršavanja u ovom slučaju dobijeno korišćenjem obične funkcije za merenje vremena t*ime()* istoimenog modula. U Tabeli 5.2 je prikazano vreme izvršavanja svakog od

Skup sekvenci (broj sekvenci)	**Greedy Alignment**	**ClustalW**	**MUSCLE**
BOX001 (5)	0.039	0.683	**0.754**
BOX032 (7)	0.025	**0.848**	0.831
BOX045 (9)	0.011	0.685	**0.831**
BOX050 (7)	0.026	**0.757**	0.732
BOX054 (5)	0.004	**0.757**	0.685
BOX075 (13)	*memory error*	**0.813**	0.774
BOX076 (7)	0.018	0.821	**0.868**
BOX121 (11)	generiše poravnanje sa 19 sekvenci	**0.894**	0.889
BOX154 (5)	0.042	0.635	**0.753**

Tablica 5.1: SP ocene

alata gde možemo videti da alat MUSCLE premašuje rad ostalih alata i u većini slučajeva nam pruža najbolje vreme izvršavanja.

Nakon testiranja podataka iz drugog referentnog skupa, *Reference Set 10*, rezultati se nisu drastično promenili. Kao što je već iznad pomenuto, performanse alata *Greedy Alignment* u ovom slučaju nisu testirane, tako da su upoređivanje performanse ClustalW i MUSCLE alata. U Tabeli 5.3 je prikazan samo deo podataka, a rezultati su slični i za ostale podatke iz ovog referentnog skupa. Može se opet zaključiti da postoji vrlo mala razlika između kvaliteta poravnanja alata ClustalW i MUSCLE i da ClustalW daje bolje optimalno poravnanje u slučaju velikog broja sekvenci. Što se tiče vremena izvršavanja, situacija je u ovom slučaju drugačija, alat ClustalW daleko brže kreira poravnanja od alata MUSCLE, što se može i zaključiti na osnovu podataka iz Tabele 5.4.

Za potrebe testiranja u radu je iskorišćeno oko 1000 (iz *Reference Set 10*) i 45 (iz *Reference Set 9*) referentnih poravnanja, pa zbog velikog obima podataka u radu nisu predstavljeni svi rezultati testiranja, ali se prosečne vrednosti svih atributa, koji su korišćeni u uporednoj analizi alata (*SP*, *TC*, *CS* i vreme izvršavanja), mogu videti u grafikonima ispod.

Skup sekvenci (broj sekvenci)	**Greedy Alignment**	**ClustalW**	**MUSCLE**
BOX032 (7)	6.529	**1.574**	1.689
BOX045 (9)	2.052	0.477	**0.293**
BOX050 (7)	1.030	0.183	**0.142**
BOX054 (5)	2.021	0.819	**0.626**
BOX175 (13)	2.224	0.604	**0.441**
BOX122 (22)	*memory error*	**2.350**	3.163
BOX154 (5)	3.214	**0.959**	0.998
BOX032 (6)	3.055	**0.882**	0.985
BOX121 (13)	3.790	0.551	**0.410**

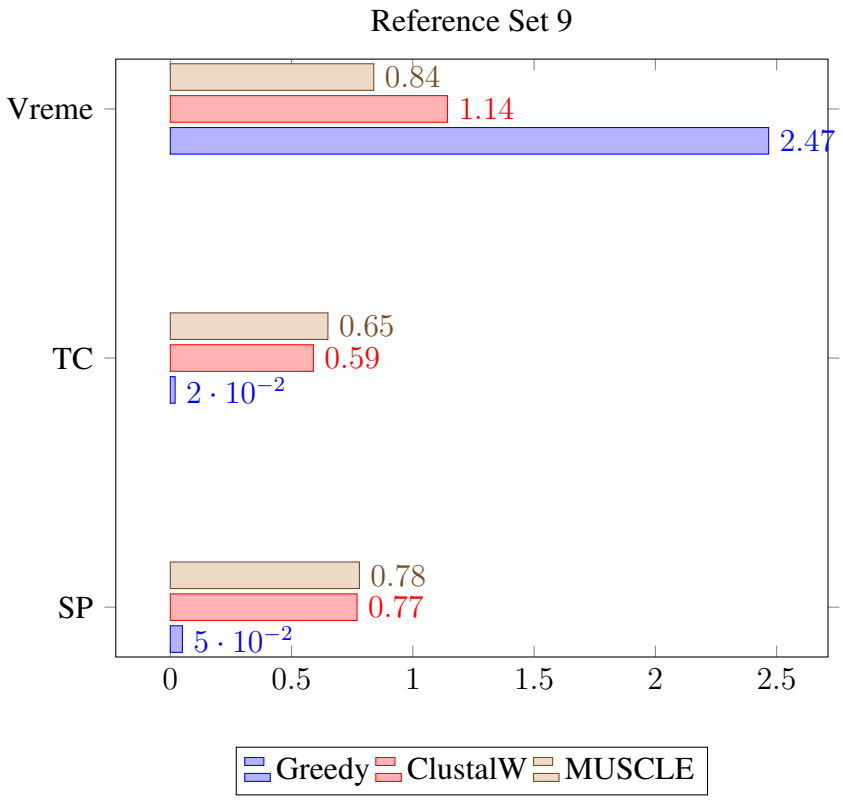
Tablica 5.2: Vreme izvršavanja u sekundama(*Reference Set 9*)

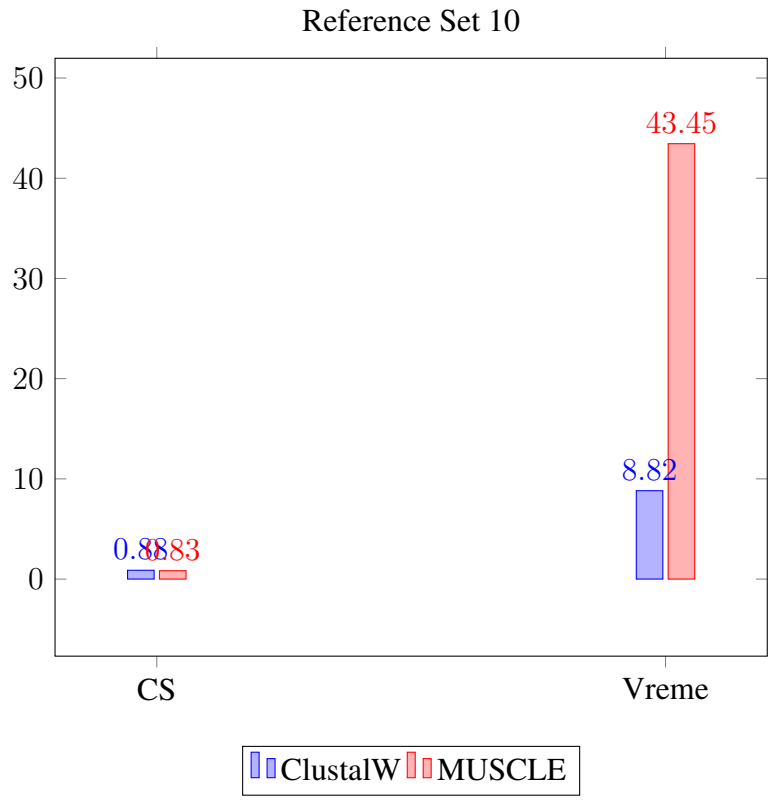
Skup sekvenci (broj sekvenci)	**ClustalW**	**MUSCLE**
BBA0011(102)	**0.900**	0.881
BBA0012 (69)	0.870	**0.979**
BBA0013 (19)	0.885	**0.886**
BBA0009 (59)	**0.905**	0.898
BBA0007 (41)	**0.984**	**0.984**
BBA0004 (248)	**0.737**	0.376
BBA0005 (44)	0.689	**0.709**

Tablica 5.3: CS ocene

Skup sekvenci (broj sekvenci)	**ClustalW**	**MUSCLE**
BBA0011(102)	4.417	**3.385**
BBA0012 (69)	**1.840**	7.187
BBA0013 (19)	**0.344**	0.359
BBA0015 (122)	**7.222**	73.772
BBA0019 (83)	2.173	**1.521**
BBA0038 (48)	53.214	**32.014**
BBA0005 (807)	**1904.889**	<7200

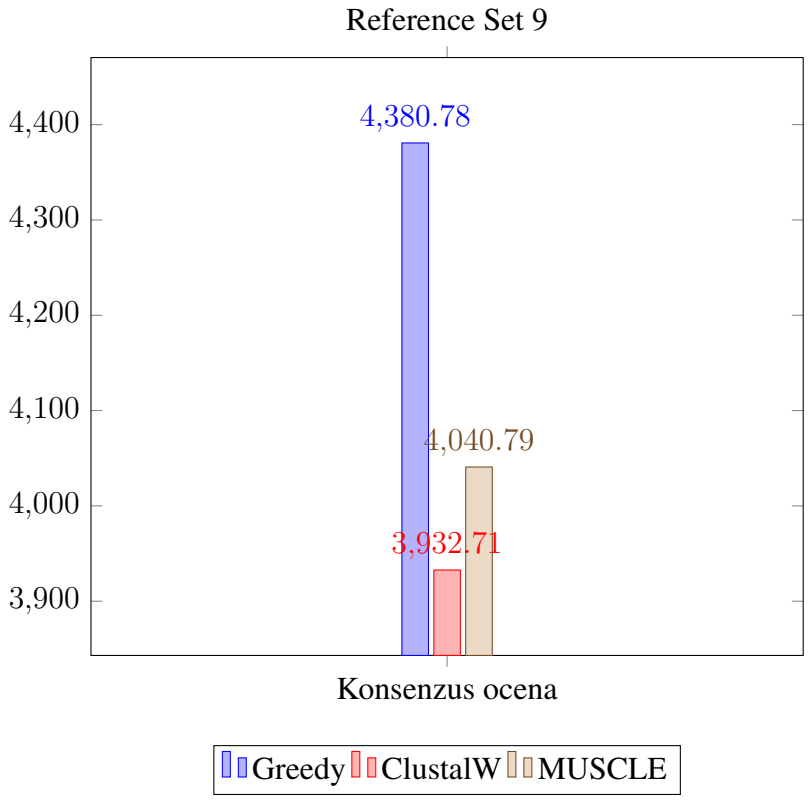
Tablica 5.4: Vreme izvršavanja u sekundama (*Reference Set 10*)





Kao što je već pomenuto u prethodnom poglavlju, pored ugrađenih ocena u radu je implementirano i izračunavanje konsenzus ocene prema formuli (4.1). Zbog toga što konsenzus ocena koristi drugačiju logiku za evaluaciju višestrukih poravnanja od ostalih predstavljenih ocena, u pojedinim slučajevima se može desiti da alat *Greedy Alignment* daje pora-

vnanje sa najboljom ocenom. Razlog tome leži u prirodi samog algoritma i odabrane formule za računanje konsenzus ocene. Naime, prilikom upoređivanja dve sekvence kada pohlepni algoritam naiđe na slučaj kada se dva simbola ne poklapaju on će pre ubaciti simbol za prazninu, nego što će "žrtvovati" to nepoklapanje kako bi se možda ostatak bolje poravnao. Upravo zbog toga *Greedy Alignment* kreira višestruka poravnanja sa ogromnim brojem praznina. Sa druge strane, prilikom računanja konsenzus ocene najveća kazna se dobija u slučaju kada se simboli ne poklapaju, dok se manja kazna dobija u slučaju poklapanja nekog simbola sa prazninom. Upravo iz ovih razloga se može desiti da *Greedy Alignment* nadmaši alate ClustalW i MUSCLE. Na primer, ako poravnanje koje je kreirao *Greedy Alignment* ima veliki broj praznina, a poravnanje koje je kreirao ClustalW ima nekoliko nepoklapajućih simbola, tada se može desiti da prvo poravnanje dobije manju konsenzus ocenu i da prema ovoj oceni predstavlja optimalnije poravnanje. Međutim, prilikom testiranja ovakav specijalan slučaj se desio svega nekoliko puta i u većini slučajeva konsenzus ocena *Greedy Alignment* poravnanja je bila najgora, što se može i zaključiti na sledećem grafikonu koji prikazuje prosečne vrednosti konsenzus ocena.



6. Zaključak

Značaj problema višestrukog poravnanja sekvenci za rešavanje bioinformatičkih problema doveo je do razvoja velikog broja alata za višestruko poravnanje. Iako nijedan od trenutnih alata ne kreira savršeno optimalno poravnanje, studije su pokazale da su u poslednjih deset godina alati pokazali značajan napredak u poboljšanju kvaliteta poravnanja i brzine izvršavanja. Ako uporedimo poravnanja koje je kreirao alat *Greedy Alignment* sa poravnanjima ostalih alata koji su predstavljeni u radu, možemo videti koliko su alati zaista napredovali po pitanju brzine i kvaliteta poravnanja.

U mnogim stručnim radovima koji se bave alatima višestrukog poravnanja, programi iz *Clustal* serije (*ClustalW* i *Clustal Omega*), MUSCLE i T-Coffee se izdvajaju kao jedni od trenutno najboljih alata za višestruko poravnanje sekvenci. I u ovom radu se može primetiti kvalitet njihovih poravnanja na osnovu ocena koje dobijamo nakon upoređivanja sa referentnim poravnanjima. Na primer, ako referentno poravnanje iz BAliBASE *Reference Set 10* uporedimo sa samim sobom dobićemo *CS* ocenu od 1.0, što znači da je 1.0 najveća ocena koju neko poravnanje može dobiti. Ako pogledamo poravnanja koje je ClustalW kreirao, videćemo da je prosečna vrednost njihovih *CS* ocena 0.88, što predstavlja relativno dobar rezultat.

Aplikacija predstavljena u radu prikazuje samo tri alata višestrukog poravnanja i sledeći korak u razvoju aplikacije bi bio dodavanje još nekoliko trenutno popularnih alata (T-Coffee, Dialign, Mafft i Probcons). Dalji razvoj aplikacije bi uključio i mogućnost ručne promene nekih parametara, kao što su vrednosti za poklapanja i nepoklapanja, kazne za praznine i izbor matrice ocena (BLOSUM ili PAM).

LITERATURA

[1] Felix Autenrieth, Barry Isralewitz, Zaida Luthey-Schulten, Anurag Sethi, i Taras Pogorelov. *Bioinformatics and Sequence Alignment*. University of Illinois at Urbana-Champaign, 2005.

[2] Anne Bahr, Julie D. Thompson, J.-C. Thierry, i Olivier Poch. BAliBASE (Benchmark Alignment dataBASE): enhancements for repeats, transmembrane sequences and circular permuatations. *Oxford University Press*, 2001.

[3] Jeff Chang, Brad Chapman, Iddo Friedberg, Thomas Hamelryck, Michiel de Hoon, Peter Cock, Tiago Antao, Eric Talevich, i Bartek Wilczyński. Biopython tutorial and cookbook, 2018. URL `http://biopython.org/DIST/docs/tutorial/Tutorial.html`.

[4] Jurate Daugelaite, Aisling O' Driscoll, i Roy D. Sleator. An Overview of Multiple Sequence Alignments and Cloud Computing in Bioinformatics. 2013. URL `https://www.hindawi.com/journals/isrn/2013/615630/`.

[5] Chuong B. Do, Mahathi S.P. Mahabhashyam, Michael Brudno, i Serafim Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Nucleic Acids Research*, 2005.

[6] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 2004.

[7] Sievers F i Higgins DG. Clustal Omega, accurate alignment of very large numbers of sequences. *Nucleic Acids Research*, 2014.

[8] Judith L. Fridovich-Keil. Human Genome Project. 2018. URL `https://www.britannica.com/event/Human-Genome-Project`.

[9] Sepp Hochreiter. *Bioinformatics I Sequence Analysis and Phylogenetics* . Institute of Bioinformatics, Johannes Kepler University Linz, 2013.

[10] D.Sc. Jason M. Kinser. *Computational Methods for Bioinformatics in Python 3.4*. George Mason University, 2017.

[11] Kazutaka Katoh, Kazuharu Misawa, Kei ichi Kuma, i Takashi Miyataa. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, 2002.

[12] Timo Lassmann i Erik LL Sonnhammer. Kalign – an accurate and fast multiple sequence alignment algorithm. *Nucleic Acids Research*, 2005.

[13] Burkhard Morgenstern. DIALIGN: multiple DNA and protein sequence alignment at BiBiServ. *Nucleic Acids Research*, 2004.

[14] C Notredame i D G Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 1996.

[15] Cedric Notredame, Desmond G. Higgins, i Jaap Heringa. T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment. *Academic Press*, 2002.

[16] Emmanuel Perrodou, Claudia Chica, Olivier Poch, Toby J Gibson, i Julie D Thompson. A new protein linear motif benchmark for multiple sequence alignment software. 2008.

[17] Usman Roshan i Dennis R. Livesay. Probalign: multiple sequence alignment using partition function posterior probabilities. *Nucleic Acids Research*, 2006.

[18] J D Thompson, D G Higgins, i T J Gibson. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 1994.

[19] Julie D. Thompson, Benjamin Linard, Odile Lecompte, i Olivier Poch. A Comprehensive Benchmark Study of Multiple Sequence Alignment Methods: Current Challenges and Future Perspectives. 2011. URL `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3069049/`.

[20] Liu Y1, Schmidt B, i Maskell DL. MSAProbs: multiple sequence alignment based on pair hidden Markov models and partition function posterior probabilities. *Nucleic Acids Research*, 2010.