UNIVERZITET U BEOGRADU
MATEMATIČKI FAKULTET

MASTER RAD

Razvoj aplikacije za korišćenje alata za poravnanje bioloških sekvenci

Zorana Stošić

Beograd, maj 2018.

Autor rada: Zorana Stošić
Mentor: dr Jovana Kovačević

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 koja je korišćena za uporednu analizu 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. Značajnost 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. Pod višestrukim poravnanjem sekvenci podrazumevamo poravnanje tri ili više bioloških sekvenci za koje se, u mnogim slučajevima, pretpostavlja da su evolutivno povezane. 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. Primer jednog višestrukog poravnanja:

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. *Match value* – vrednost koja se dodeljuje kada se karakteri poklapaju
2. *Mismatch value* – vrednost koja se dodeljuje kada se karakteri ne poklapaju.
3. *Gap penalty* – vrednost praznina (odnosno "-„ simbola)

Proces skorovanja višestrukog poravnanja zasnovan je na sumi rezultata svih mogućih parova sekvenci u višestrukom poravnanju na osnovu neke matrice skora (eng. *score matrix*).

Cilj višestrukog poravnanja je da se maksimizuje sledeća suma parova:

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

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

Primer 1.0.2 *Skor višestrukog poravnanja*

Sek_1: *G K N*

Sek_2: *T R N*

Sek_3: *S H E*

Suma: -1 + 1 + 6 = 6

Suma druge kolone, Kol_2 = *ocena(K, R) + ocena(R, H) + ocena(K, H)* = 2 + 0 + (-1) = 1

2. Vrste i načini izgradnje višestrukog poravnanja

Procedura višestrukog poravnanja je značajna iz više razloga. Jedna korisna primena je ta što se na osnovu višestrukog poravnanja može zaključiti homologija sekvenci. Homologija sekvenci ukazuje na zajedničko evolutivno poreklo između DNK, RNK ili proteinskih sekvenci. Na primer, dva segmenta DNK mogu deliti poreklo kao posledica 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.

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. Posmatrajmo dve sekvence X i Y, prikazane ispod. Poravnanje se dobija ubacivanjem praznina kako bi dužina sekvenci bila ista i kako bi se sekvence optimalno poklopile.

Primer 2.1.1 *Globalno poravnanje*
X: AC-GCTGAT
. | | | | |
Y: - CAGC - TAT

Jedan od algoritama koji koristi dinamičko programiranje za dobijanje globalnog poravnanja je *Needleman-Wunsch*-ov algoritam. 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. Ovaj algoritam pronalazi globalno poravnanje dve sekvence sa najboljom ocenom.

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. Na primer, neka su nam date sekvence: X = GGT**CTGAT**G i Y = AAA**CGAT**C. Razmatramo samo podebljane podsekvence i najbolje lokalno poravnanje koje se dobija je:

Primer 2.1.2 *Lokalno poravnanje*
CTGAT (u X)
| | | |
C-GAT (u Y)

Smith-Waterman-ov algoritam je algoritam lokalnog poravnanja. Profesori *Temple Ferris Smith* i *Michael Spencer Waterman* su 1981. godine objavili primenu dinamičkog programiranja za nalaženje optimalnog lokalnog poravnanja. Ovaj algoritam je sličan *Needleman-Wunsch*-ovom algoritmu, ali postoje male razlike u procesu bodovanja.

2.2. Načini izgradnje višestrukog poravavnanja

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 više od dve sekvence. Ipak, u praksi je suviše spor, jer vremenska i prostorna složenost postaju veoma velike. 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. Dva najpoznatija načina za izgradnju višestrukog poravnanja sekvenci su:

- *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 klasteruju zajedno, kako bi se predstavio njihov međusobni odnos, korišćenjem metoda klasterovanja kao što su *mBed* i *k*-sredina. Rezultati sličnosti se tada pretvaraju u ocene udaljenosti koje se koriste kako bi se konstruisalo *guide* stablo, pomoću metoda kao što su *NJ* (*Neighbour-Joining*) i *UPGMA* (*Unweighted Pair Group Method with Aritmetic Mean*). Kada se kreira *guide* stablo, višestruko poravnanje se konstruiše tako što se sekvence, jedna po jedna, ubacuju na osnovu *guide* 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 guide stablo, tako što se sekvence postavljaju kao listovi

5. Korišćenjem *guide* stabla 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 je osnovna procedura nekoliko popularnih algoritama poravnanja kao što su *ClustalW*, *ClustalOmega*, *MAFFT*, *Kalign*, *Probalign*, *MUSCLE*, *DIALIGN*, *PRANK*, *FSA*, *T-Coffee*, *ProbCons* i *MSAProbs*.

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. Najkorišćeniji algoritmi iterativnog poravnanja uključuju *PRRP*, *MUSCLE*, *Dialign*, *SAGA* 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 konstru-

isati 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. Ove metode 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 (tzv. motiv) u dugačkoj sekvenci. Jedan od najčešće korišćenih alata za traženje motiva je *PHI-Blast* i *Gapped Local Alignments of Motif* (*GLAM2*).

[1]Trodimenzionalna struktura proteina, definisana atomskim koordinatama

3. Alati višestrukog poravnanja

Broj algoritama višestrukog poravnanja raste skoro na mesečnom nivou. Računarska složenost i kvalitet poravnanja se konstantno poboljšava. *ClustalW* je najpopularniji algoritam višestrukog poravnanja, dok je nedavno razvijeni algoritam *ClustalOmega* najprecizniji i najnapredniji trenutno dostupan algoritam višestrukog poravnanja. U nastavku 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. 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 *guide* stablu. 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, uključujući „otežavanje“ sekvence, poziciono određene kazne za praznine i automatski izbor 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 trenutnim izdanjem *ClustalW*, korisnik može bolje proceniti poravnanja u *ClustalX*-u.
Sve verzije *Clustal*-a prate tri glavna koraka:

– Kreiranje dvostrukog poravnanja

– Kreiranje *guide* stabla [2]

– Upotreba stabla kako bi pronašlo višestruko poravnanje

3.1.1. ClustalW

ClustalW je predstavljen 1994. godine i brzo je postao najpopularnija metoda za stvaranje 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 uvedena shema „otežavanja" za preopterećene sekvencne grupe.

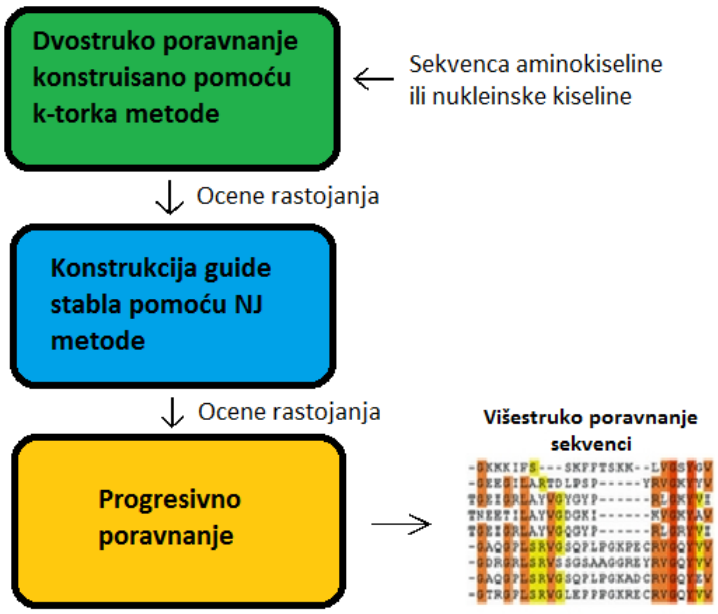
Algoritam

Prvo, 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. Rezultati sličnosti se pretvaraju u ocene rastojanja, a zatim algoritam koristi ove ocene kako bi napravio *guide* stablo, koristeći *NJ* metod za konstrukciju *guide* stabla. Poslednji korak algoritma je konstrukcija višestrukog poravnanja svih sekvenci. Višestruko poravnanje je konstruisano progresivnim poravnanjem najbližih, odnosno najsličnijih, sekvenci prema *guide* stablu.

Dvostruko poravnanje

Metoda *k*-torka, brza heuristička metoda, koristi se za dvostruko poravnanje svih mogućih parova sekvenci. Ova metoda ima posebno dobre perfomanse kada je broj sekvenci koje treba poravnati veliki. Ocena sličnosti se izračunava i zatim se kazne za praznine oduzimaju od te ocene. 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, stvaranjem optimalnog poravnanja koji je sličan *Needleman-Wunsch*-ovoj metodi.

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



Slika 3.1: Koraci ClustalW algoritma

Konstrukcija guide stabla

ClustalW kreira *guide* stablo u skladu sa metodom *NJ*, koja se često naziva metodom dekompozicije. *NJ* metoda ne prati tzv. „taksonome“ (taksonomska kategorija ili grupa, poput 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 odvojena samo jednom granom. Stablo proizvedeno *NJ* metodom 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. *Guide* stablo 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 sekvence 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 *ClustalW*-a koristi seriju dvostrukih poravnanja prateći redosled grananja u *guide* stablu. Postupak počinje od listova stabla i nastavlja prema korenu. U svakom koraku se koristi algoritam dinamičkog programiranja sa matricom težine (*BLOSUM*) i kaznama za otvarajuće i proširujuće praznine.

3.1.2. Clustal Omega

Clustal Omega, objavljen 2014. godine, je najnoviji algoritam višestrukog poravnanja iz *Clustal* familije. Ovaj algoritam 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 klasteruju pomoću *mBed* metode. Nakon toga se primenjuje metod klasterovanja, *k*-sredina. *Guide* stablo se konstruiše korišćenjem *UPGMA* metode. Na kraju se višestruko poravnanje dobija progresivnim poravnanjem korišćenjem *HHalign* paketa.

Dvostruko poravnanje

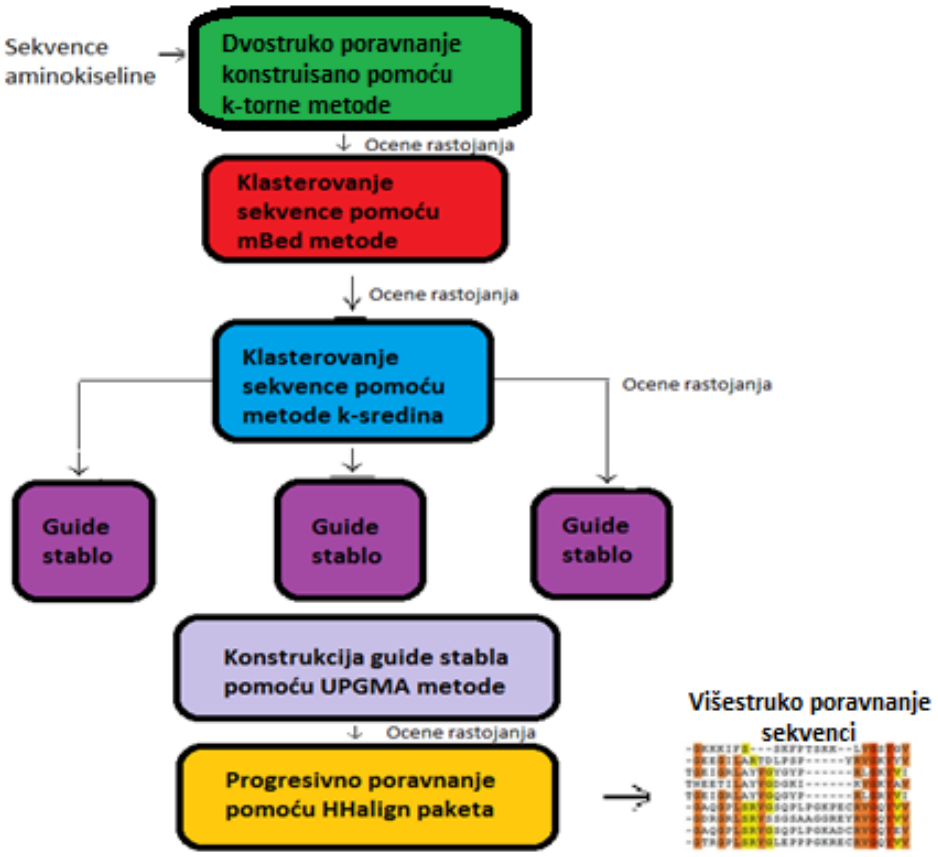
Dvostruko poravnanje se proizvodi pomoću *k*-torne metode, isto kao i kod *ClustalW*.

Klasterovanje

Nakon što su iz dvostrukog poravnanja određene ocene sličnosti, *Clustal Omega* koristi metod *mBed*, složenosti $O(N \cdot logN)$. *mBed* radi tako što „ugrađuje“ (eng. *emBedding*) svaku sekvencu u prostoru dimenzije *n*, gde je *n* proporcionalno *logN*. Svaka sekvenca se zatim zamenjuje vektorom od *n* elemenata, gde svaki element predstavlja rastojanje do jedne od *n* „referentnih sekvenci“. Ovi vektori se onda mogu brzo klasterovati pomoću metoda za klasterovanje, poput *k*-sredine ili *UPGMA*.

K-sredine

Clustal Omega pored *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.



Slika 3.2: Koraci *Clustal Omega* algoritma

Konstrukcija guide stabla

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

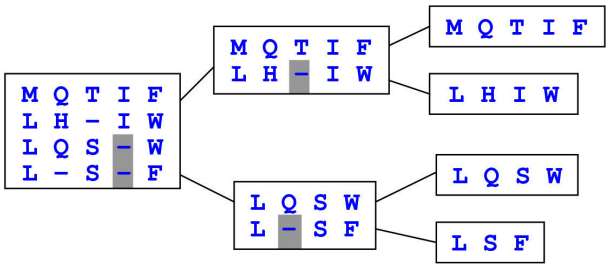
Progresivno poravnanje

Za progresivno poravnanje *Clustal Omega* koristi *HHalign* alat, koji generiše skriveni Markovljev model (eng. *Hidden Markov Model, HMM*) iz datih poravnanja i izračunava optimalno poravnanje i sva značajna nepreklapajuća podoptimalna poravnanja. *HHalign* upoređuje dva poravnanja (odnosno izvršava profil-profil poravnanje), prihvata samo cela poravnanja i ne može da automatski kreira poravnanja na osnovnu unetih sekvenci. Alat *Clustal Omega*

predstavlja trenutnu standardnu verziju *Clustal* programa, besplatan je i javno dostupan [3].

3.2. MUSCLE

MUSCLE (*Multiple Sequence Comparison by Log-Expectation*) je precizniji od *T-Coffee* i brži od *ClustalW*. Osnovna strategija koju koristi *MUSCLE* je slična onoj koju koriste *PRRP* i *MAFFT*, 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 3.3: 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 kreira 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, profil-profil (Slika 3.3) poravnanje 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] http://www.ebi.ac.uk/Tools/msa/clustalo/

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. Idealno, ali uglavnom nepoznato, je rastojanje mutacije, tj. broj mutacija koje su se desile na evolutivnom putu između dve sekvence. 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 guide stabla

Guide stablo se konstruiše metodama klasterovanja na osnovu matrice rastojanja. Dve metode se ovde koriste: *NJ* i *UPGMA*. *MUSCLE* implementira tri varijante *UPGMA*-e 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:

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

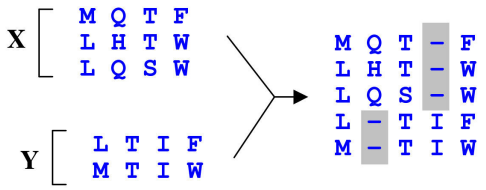
Ili čak kombinaciju minimalnog i prosečnog rastojanja:

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

gde je parameter *s* podrazumevano 0.1

3.2.5. Profil-profil 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 visoku ocenu parovima kolona koji sadrže slične aminokiseline.



Slika 3.4: Primer profil-profil 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*.

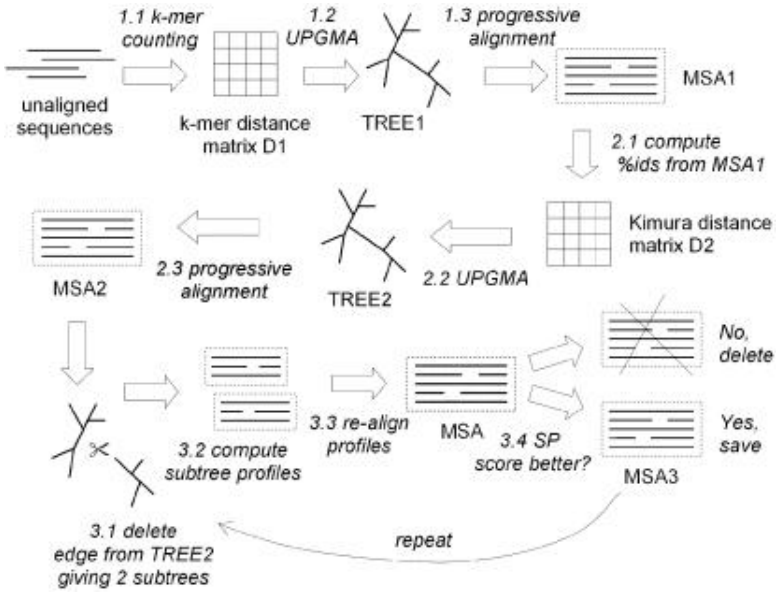
Faza 3, *Refinement*.

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 3.5: Tok MUSCLE algoritma[5]

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 skorovanja. Homologne oblasti se identifikuju brzom Furijeovom transformacijom (*FFT*), dok uvedeni pojednostavljeni sistem skorovanja 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 zatim se izvršava druga metoda, *FFT-NS-i*. *FFT-NS-i* je 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*. *Guide* stablo se konstruiše pomoću *UPGMA* ili *NJ* metode, a progresivno poravnanje se završava tako što se prati *guide* stablo. Nasuprot postojećim metodama, ono što čini ovaj algoritam drugačijim je korišćenje *Wu-Manber*-ovog algoritma 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 perfomanse. 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 FASTA formatu 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 skor.
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

Biopython-ova 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*, *transcribe*, *translate*). Alfabeti se nalaze u modulu *Bio.Alphabet* koji sadrži klase za određi-

vanje tipa *Seq* objekata:

- *Alphabet()*, generički alfabet
- *ProteinAlphabet()*, generički proteinski alfabet
- *NucleotideAlphabet()*, generički nukleotidni alfabet
- *DNAAlphabet()*, generički DNK alfabet
- *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. *Biopython*-ov *Bio.SeqIO* modul pruža jednostavan 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 FASTA datoteke. Datoteka sadrži podatke o 94 nukleotidne sekvence, svaka počinje simbolom “>” nakon koga sledi sekvenca.

```
>gi|2765658|emb|Z78533.1|CIZ78533 C.irapeanum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGATGAGACCGTGGAATAAACGATCGAGTG
AATCCGGAGGACCGGTGTACTCAGCTCACCGGGGGCATTGCTCCCGTGGTGACCCTGATTTGTTGTTGGG
CCGCCTCGGGAGCGTCCATGGCGGGTTTGAACCTCTAGCCCGGCGCAGTTTGGGCGCCAAGCCATATGAA
AGCATCACCGGCGAATGGCATTGTCTTCCCCAAAACCCGGAGCGGCGGCGTGCTGTCGCGTGCCCAATGA
ATTTTGATGACTCTCGCAAACGGGAATCTTGGCTCTTTGCATCGGATGGAAGGACGCAGCGAAATGCGAT
AAGTGGTGTGAATTGCAAGATCCCGTGAACCATCGAGTCTTTTGAACGCAAGTTGCGCCCGAGGCCATCA
GGCTAAGGGCACGCCTGCTTGGGCGTCGCGCTTCGTCTCTCTCCTGCCAATGCTTGCCCGGCATACAGCC
AGGCCGGCGTGGTGCGGATGTGAAAGATTGGCCCCTTGTGCCTAGGTGCGGCGGGTCCAAGAGCTGGTGT
TTTGATGGCCCGGAACCCGGCAAGAGGTGGACGGATGCTGGCAGCAGCTGCCGTGCGAATCCCCCATGTT
GTCGTGCTTGTCGGACAGGCAGGAGAACCCTTCCGAACCCCAATGGAGGGCGGTTGACCGCCATTCGGAT
GTGACCCCAGGTCAGGCGGGGGCACCCGCTGAGTTTACGC

>gi|2765657|emb|Z78532.1|CCZ78532 C.californicum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAACAGAATATATGATCGAGTG
AATCTGGAGGACCTGTGGTAACTCAGCTCGTCGTGGCACTGCTTTTGTCGTGACCCTGCTTTGTTGTTGG
GCCTCCTCAAGAGCTTTCATGGCAGGTTTGAACTTTAGTACGGTGCAGTTTGCGCCAAGTCATATAAAGC
ATCACTGATGAATGACATTATTGTCAGAAAAAATCAGAGGGGCAGTATGCTACTGAGCATGCCAGTGAAT
TTTTATGACTCTCGCAACGGATATCTTGGCTCTAACATCGATGAAGAACGCAGCTAAATGCGATAAGTGG
TGTGAATTGCAGAATCCCGTGAACCATCGAGTCTTTGAACGCAAGTTGCGCTCGAGGCCATCAGGCTAAG
GGCACGCCTGCCTGGGCGTCGTGTGTTGCGTCTCTCCTACCAATGCTTGCTTGGCATATCGCTAAGCTGG
CATTATACGGATGTGAATGATTGGCCCCTTGTGCCTAGGTGCGGTGGGTCTAAGGATTGTTGCTTTGATG
GGTAGGAATGTGGCACGAGGTGGAGAATGCTAACAGTCATAAGGCTGCTATTTGAATCCCCCATGTTGTT
GTATTTTTTCGAACCTACACAAGAACCTAATTGAACCCCAATGGAGCTAAAATAACCATTGGGCAGTTGA
TTTCCATTCAGATGCGACCCCAGGTCAGGCGGGGCCACCCGCTGAGTTGAGGC

>gi|2765656|emb|Z78531.1|CFZ78531 C.fasciculatum 5.8S rRNA gene and ITS1 and ITS2 DNA
CGTAACAAGGTTTCCGTAGGTGAACCTGCGGAAGGATCATTGTTGAGACAGCAGAACATACGATCGAGTG
AATCCGGAGGACCCGTGGTTACACGGCTCACCGTGGCTTTGCTCTCGTGGTGAACCCGGTTTGCGACCGG
GCCGCCTCGGGAACTTTCATGGCGGGTTTGAACGTCTAGCGCGGCGCAGTTTGCGCCAAGTCATATGGAG
CGTCACCGATGGATGGCATTTTTGTCAAGAAAAACTCGGAGGGGCGGCGTCTGTTGCGCGTGCCAATGAA
TTTATGACGACTCTCGGCAACGGGATATCTGGCTCTTGCATCGATGAAGAACGCAGCGAAATGCGATAAG
TGGTGTGAATTGCAGAATCCCGCGAACCATCGAGTCTTTGAACGCAAGTTGCGCCCGAGGCCATCAGGCT
AAGGGCACGCCTGCCTGGGCGTCGTGTGCTGCGTCTCTCCTGATAATGCTTGATTGGCATGCGGCTAGTC
TGTCATTGTGAGGACGTGAAAGATTGGCCCCTTGCGCCTAGGTGCGGCGGGTCTAAGCATCGGTGTTCTG
ATGGCCCGGAACTTGGCAGTAGGTGGAGGATGCTGGCAGCCGCAAGGCTGCCGTTCGAATCCCCCGTGTT
GTCGTACTCGTCAGGCCTACAGAAGAACCTGTTTGAACCCCCAGTGGACGCAAAACCGCCCTCGGGCGGT
GATTTCCATTCAGATGCGACCCCAGTCAGGCGGGCCACCCGTGAGTAA
```

Slika 4.2: Primer jedne FASTA datoteke (ls_orchid.fasta)

Na primer, nakon parsiranja ove datoteke:

```
from Bio import SeqIO
handle = open("ls_orchid.fasta")
for seq_record in SeqIO.parse(handle, "fasta") :
        print seq_record.id
        print repr(seq_record.seq)
        print len(seq_record)
handle.close()
```

dobijamo ovakav izlaz:

```
gi|2765658|emb|Z78533.1|CIZ78533 Seq('CGTAACAAGGTTTCCGTAGGTGA...CGC',
SingleLetterAlphabet()) 740
...
gi|2765564|emb|Z78439.1|PBZ78439 Seq('CATTGTTGAGATCACATAATAAT...GCC',
SingleLetterAlphabet()) 592
```

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 poravnanjima, 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 *Biopython*-u se višestruko poravnanje sekvenci posmatra kao kolekcija višestrukih sekvenci koje su poravnate, obično umetanjem karaktera "-", tako da su sve sekvencne niske 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 *Bio.AlignIO.write()* koji uzima tri argumenta: *MultipleSeqAlignment* objekte, ime datoteke i format sekvence.

Alati poravnanja

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.

– Pozove se alat komandne linije koji obrađuje ovu datoteku, obično preko *Biopython*-ovih omotača komandne linije.

– Pročita se izlaz korišćenog alata, tj. poravnanje, pomoću *Bio.AlignIO*.

Za potrebe aplikacije su korišćena dva popularna alata komandne linije za višestruko poravnanje: ClustalW i MUSCLE. Pored alata višestrukog poravnanja dodati su i alati dvostrukog poravnanja: *Biopython*-ov modul *pairwise2* i EMBOSS-ovi *water* (*Smith-Waterman* algoritam) i *needle* (*Needleman-Wunsch* algoritam) alati za lokalno i globalno poravnanje.
Alat ClustalW podrazumevano vraća poravnanje i datoteku sa opisom *guide* stabla, koju možemo parsirati pomoću *Bio.Phylo* kako bismo dobili pravi prikaz stabla. Na primer, za ulaznu datoteku "test.fasta" (slika 4.3), ClustalW će generisati dve datoteke: "test.aln" (slika 4.4) i "test.dnd" (slika 4.5). Nakon parsiranja "test.dnd" datoteke, dobijamo formatirano filogenetsko stablo (slika 4.6).

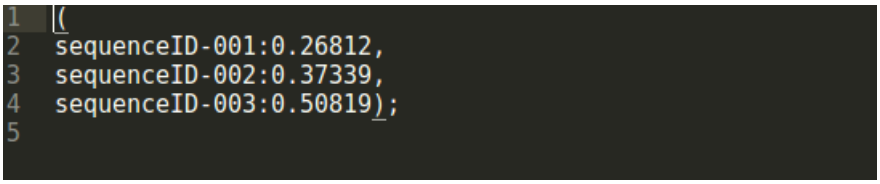
```
>sequenceID-001 description
AAGTAGGAATAATATCTTATCATTATAGATAAAAACCTTCTGAATTTGCTTAGTGTGTAT
ACGACTAGACATATATCAGCTCGCCGATTATTTGGATTATTCCCTG

>sequenceID-002 description
CAGTAAAGAGTGGATGTAAGAACCGTCCGATCTACCAGATGTGATAGAGGTTGCCAGTAC
AAAAATTGCATAATAATTGATTAATCCTTTAATATTGTTTAGAATATATCCGTCAGATAA
TCCTAAAAATAACGATATGATGGCGGAAATCGTC

>sequenceID-003 description
CTTCAATTACCCTGCTGACGCGAGATACCTTATGCATCGAAGGTAAAGCGATGAATTTAT
CCAAGGTTTTAATTTG
```

Slika 4.3: *test.fasta*

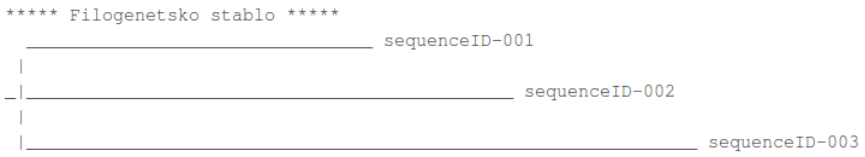
```
CLUSTAL 2.1 multiple sequence alignment


sequenceID-001      --------------------------------------------------AAGTAGGAA
sequenceID-002      CAGTAAAGAGTGGATGTAAGAACCGTCCGATCTACCAGATGTGATAGAGGTTGCCAGTAC
sequenceID-003      ------------------------------------------------------------


sequenceID-001      TAATATCTTATCATTATAGATAAAAACCTTCTGAATTTGCTTAGTGTGTATA-CGACTAG
sequenceID-002      AAAAATTGCATAATAATTGATTAA--TCCTTTAATATTGTTTAGAATATATC-CG-TCAG
sequenceID-003      -------------CTTCAATTAC--CCTGCTGACGCGAGATACCTTATGCATCGAAGGT
                                 *  ** *    *   * *       **   * *    **

sequenceID-001      ACATATATCAGCTCGCCGATTATTTGGATTATTCCCTG
sequenceID-002      ATAATCCTAAAAATAACGATATGATGGCGGAAATCGTC
sequenceID-003      AAAGCGATGAATTTATCCAAGGTTTTAATTTG------
                    * *    * *      * *     *
```

Slika 4.4: *test.aln*

MUSCLE je noviji alat i podrazumevano vraća FASTA datoteku koju možemo parsirati pomoću *Bio.AlignIO* modula kako bismo pročitali poravnanje (slika 4.7). *Biopython*-ov modul *Bio.Emboss.Applications* poseduje omotače za EMBOSS-ove *needle* i *water* alate, kao i omotače za EMBOSS verzije PHYLIP alata. Na primer, ako želimo da nađemo globalno



Slika 4.5: *test.dnd*



Slika 4.6: Filogenetsko stablo

poravnanje dve sekvence, potrebno je da pripremimo dve ulazne FASTA datoteke. 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 EMBOSS alata 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 skora za *matches*, *mismatches* i *gaps*. Prvim karakterom definišemo *match* skor, na primer, *x* podrazumeva da se *match* računa kao 1, dok *mismatches* nemaju cenu. Ako su nam potrebni drugačiji *match* skorovi, obično koristimo matrice skora, na primer za aminokiseline skorovi se nalaze u PAM i BLOSUM matricama. Drugi karakter predstavlja cenu za 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 GUI-ja. 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

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

Slika 4.7: Izlaz primene MUSCLE alata za *test.fasta*

negrafičkih programa, poput alata komandne linije i konzolnih servera. *Qt* dolazi sa sopstvenim skupom alata, poput *Qt Creator*-a 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 koga je razvila *The Qt Company* i predstavlja vezivanje *Python*-a na *Qt*, tj. API (*Application Programming Interface*) koji nam omogućava da koristimo *Qt* i sve njegove alate u *Python*-u.
Ceo proces korišćenja ove kombinacije je vrlo jednostavan. Potrebno je prvo instalirati *PySide* i *QtCreator*. U *QtCreator*-u kreiramo dizajn aplikacije koga 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 manipulisati komponentama aplikacije (dugmićima, labelama, dijalozima itd.) i praviti neke svoje događaje (tzv. *events*).

4.1.3. Modul *profile*

Python-ov modul *profile* pruža deterministično profilisanje *Python* programa. Predstavlja skup statistika koji opisuje koliko često i koliko dugo je trajalo izvršavanje nekog dela programa. Ovaj modul se pokazao korisnim u upoređivanju perfomansi alata višestrukog poravnanja. Na primer, kada funkciji *profile.run* prosledimo poziv funkcije koja predstavlja primenu MUSCLE alata, dobijamo rezultat prikazan na slici 4.8.
Prva linija nam govori da je izvršavanje MUSCLE alata trajalo 0.066 sekundi i u tom vremenskom periodu je bilo 2159 funkcijskih poziva, od kojih su 2119 bili primitivni, odnosno nisu rekurzivno pozvani. Prikazane kolone predstavljaju sledeće:

- **ncalls** – broj poziva date funkcije
- **tottime** – ukupno vreme izvršavanja date funkcije
- **percall** – tottime/ncalls
- **cumtime** – vreme izvršavanja date funkcije i svih podfunkcija (od trenutka poziva do kraja)

– **percall** – cumtime/primitive calls

– **filename:lineno(function)** – pruža odgovarajuće podatke o svakoj funkciji

```
zoka@zoka-Lenovo-G50-30:~/Desktop/BIpr/slozenosti$ python muscle.py
         2159 function calls (2119 primitive calls) in 0.066 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 :0(WEXITSTATUS)
        1    0.000    0.000    0.000    0.000 :0(WIFEXITED)
        1    0.000    0.000    0.000    0.000 :0(WIFSIGNALED)
      151    0.003    0.000    0.003    0.000 :0(add)
      168    0.002    0.000    0.002    0.000 :0(append)
        8    0.000    0.000    0.000    0.000 :0(close)
        1    0.000    0.000    0.000    0.000 :0(count)
        1    0.000    0.000    0.000    0.000 :0(disable)
        1    0.000    0.000    0.000    0.000 :0(enable)
        1    0.000    0.000    0.002    0.002 :0(extend)
       16    0.000    0.000    0.000    0.000 :0(fcntl)
        3    0.000    0.000    0.000    0.000 :0(fdopen)
        6    0.000    0.000    0.000    0.000 :0(fileno)
      127    0.001    0.000    0.001    0.000 :0(find)
        1    0.000    0.000    0.000    0.000 :0(flush)
        1    0.002    0.002    0.002    0.002 :0(fork)
        1    0.000    0.000    0.000    0.000 :0(hasattr)
        1    0.000    0.000    0.000    0.000 :0(isenabled)
      299    0.004    0.000    0.004    0.000 :0(isinstance)
        1    0.000    0.000    0.000    0.000 :0(items)
        1    0.000    0.000    0.000    0.000 :0(iter)
       11    0.000    0.000    0.000    0.000 :0(join)
    80/41    0.001    0.000    0.002    0.000 :0(len)
        2    0.000    0.000    0.000    0.000 :0(lower)
       75    0.002    0.000    0.002    0.000 :0(match)
      3/2    0.000    0.000    0.018    0.009 :0(next)
        4    0.000    0.000    0.000    0.000 :0(pipe)
       44    0.003    0.000    0.003    0.000 :0(poll)
        2    0.000    0.000    0.000    0.000 :0(pop)
       43    0.001    0.000    0.001    0.000 :0(read)
        2    0.000    0.000    0.000    0.000 :0(register)
        3    0.000    0.000    0.000    0.000 :0(remove)
       18    0.000    0.000    0.000    0.000 :0(replace)
      119    0.001    0.000    0.001    0.000 :0(rstrip)
       75    0.002    0.000    0.002    0.000 :0(setattr)
        1    0.001    0.001    0.001    0.001 :0(setprofile)
```

Slika 4.8: Vreme izvršavanja MUSCLE alata

4.1.4. Pohlepni algoritam višestrukog poravnanja

Pored ugrađenih alata višestrukog poravnanja, aplikaciji je dodata i implementacija pohlepnog algoritma [7]. 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 poravna-

nja se spajaju i kreiraju kontigu (eng. *contig*), koja predstavlja neprekidnu sekvencnu nisku. Algoritam kreće sa upoređivanjem svih parova sekvenci, računa njihov skor i kreira trougaonu matricu *M*. Svaki element matrice *M* predstavlja rezultat jednog para, na primer e_{ij} predstavlja skor 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 sledeći najveći skor. 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:

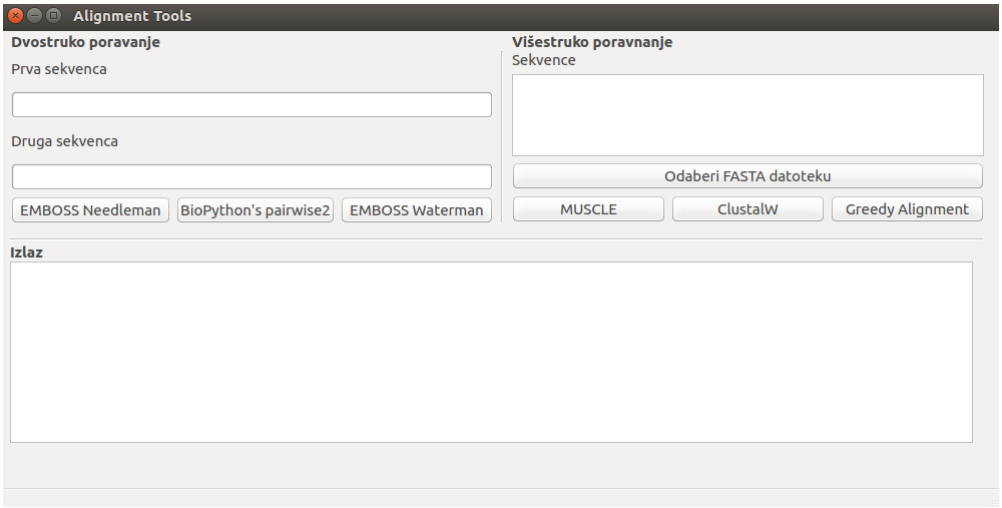
1. Ako se ni *sa* ni *sb* ne nalaze u nekoj već postojećoj kontigi, onda se od njih kreira nova kontiga
2. Ako *sa* već pripada nekoj kontigi, a *sb* ne pripada, onda se *sb* dodaje kontigi u kojoj se nalazi *sa*.
3. Ako *sa* pripada jednoj, a *sb* nekoj drugoj kontigi, onda se te dve kontige spajaju.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

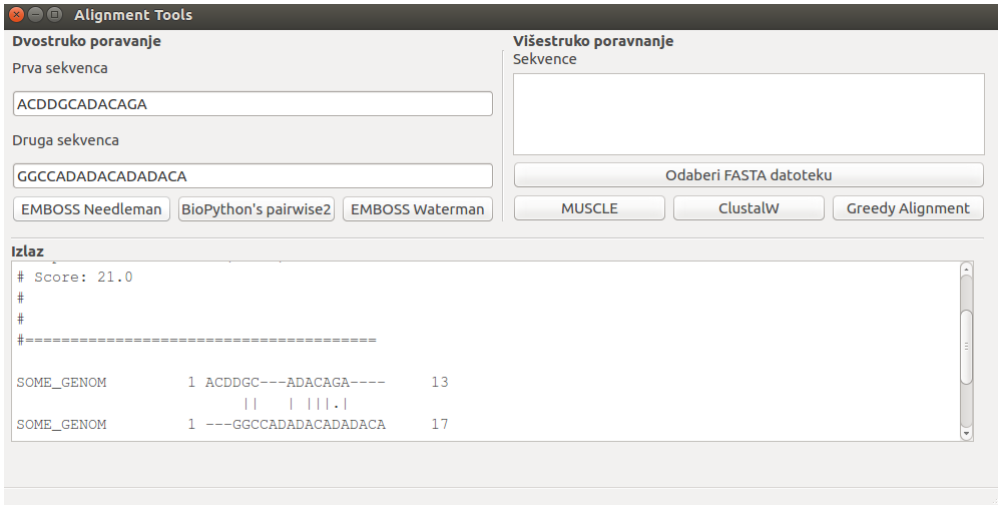
4.2. Izgled aplikacije i uputstvo za korišćenje

Inicijalni izgled aplikacije je prikazan na slici 4.9.



Slika 4.9: 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 bez ikakvih kazni za *matches*, *mismatches* i *gaps*. Na primer, na slici 4.10 se vidi izlaz jednog takvog alata, prikazan je skor dvostrukog poravnanja i finalno dvostruko poravnanje unetih sekvenci.



Slika 4.10: 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 FASTA datoteku 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 4.11: Izlaz ClustalW alata

Iako ugrađeni alati višestrukog poravnanja imaju svoj način izračunavanja skora (uglavnom koriste SP skor, koji će biti detaljnije objašnjen u narednom poglavlju), implementirano je i izračunavanje konsenzus skora na osnovu koga možemo uporediti alate za isti skup ulaznih sekvenci. Da bismo izračunali konsenzus skor 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 [6]:

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

Nakon toga, saberemo vrednosti kolone i dobijamo konsenzus skor. Što je konsenzus skor manji to imamo bolje poravnanje. Na primer, za iste ulazne sekvence konsenzus skor MUSCLE-ovog višestrukog poravnanja je 86.0, ClustalW-ovog je 74.5, dok pohlepni algoritam vraća poravnanje sa konsenzus skorom od 309.5. Na osnovu ovih skorova možemo zaključiti da je alat ClustalW vratio najbolje višestruko poravnanje.

5. Uporedna analiza alata

Pored razvoja aplikacije, zadatak ovog rada bio je i da uporedimo perfomanse 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 [8] 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[4] koji se javlja u oblasti sekvence koja se može poravnati sa instancom

[4]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[5] i pored referentnih poravnanja možemo preuzeti i *C* program tzv. *bali_score* koji upoređuje naša poravnanja sa referentnim i izračunava skor za kvalitet poravnanja (*Sum-Of-Pairs* skor, SPS i *Total-Column* skor, TCS) . Da bi se ovaj program pokrenuo potrebno je imati instaliran XML parser *expat*. Program se pokreće:

```
bali_score ref_aln test_aln
```

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

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

Dakle, program *bali_score* računa SP skor koji procenjuje tačnost poravnanja motiva. SP skor 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 skor. 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. Skor 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 skor 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 skor za S_i, i-te kolone u referentnom poravnanju.

5.2. BAliBASE Reference Set 10

BAliBASE Reference Set 10 [9] 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[6]. Referentna poravnanja možemo naći u *.msf ili *.xml datotekama, a sekvence u *.tfa datotekama. Pored ovih podataka, možemo preuzeti i *C* program: *bali_score* koji upoređuje naša poravnanja sa referentnim i izračunava skor za kvalitet poravnanja. Da bi se ovaj program pokrenuo potrebno je imati instaliran XML parser *expat*. Program *bali_score* se pokreće na sledeći način:

```
bali_score ref_aln test_aln
```

[6] http://www.lbgi.fr/balibase/BAliBASE_R10/

gde je *ref_aln* referentno poravnanje u xml/fasta formatu, a *test_aln* je test poravnanje u msf/fasta formatu.
Objasnićemo kako program *bali_score* nalazi skorove. 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 skor 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 skor poravnanja

U nastavku je prikazan samo deo podataka koji su sakupljeni prilikom uporedne analize alata. Prvo su iskorišćeni podaci iz *Reference Set 9* i u Tabeli 5.1 su predstavljeni SP skorovi poravnanja. Možemo primetiti da je rad alata ClustalW i MUSCLE daleko bolji od rada *Greedy Alignment* alata, 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. U Tabeli 5.2 je prikazano vreme izvršavanja svakog od alata gde možemo videti da ClustalW premašuje rad ostalih alata i pruža nam najbolje vreme izvršavanja bez obzira o kakvom se skupu sekvenci radi.

Nakon testiranja podataka iz drugog referentnog skupa, *Reference Set 10*, rezultati se nisu drastično promenili. Kao što je već iznad pomenuto, perfomanse alata *Greedy Alignment* u ovom slučaju nisu testirane, tako da su upoređivanje perfomanse ClustalW i MUSCLE alata. U Tabeli 5.3 je prikazan samo deo podataka, a rezultati su slični i za ostale podatke iz ovog

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 skorovi

Skup sekvenci (broj sekvenci)	**Greedy Alignment**	**ClustalW**	**MUSCLE**
BOX001 (5)	4.857	**0.060**	0.131
BOX032 (7)	14.731	**0.088**	0.291
BOX045 (9)	16.328	**0.077**	0.205
BOX050 (7)	9.689	**0.064**	0.129
BOX054 (5)	5.910	**0.075**	0.191
BOX075 (13)	*memory error*	**0.099**	0.333
BOX076 (7)	11.041	**0.072**	0.178
BOX121 (11)	generiše poravnanje sa 19 sekvenci	**0.083**	0.245
BOX154 (5)	7.677	**0.068**	0.255

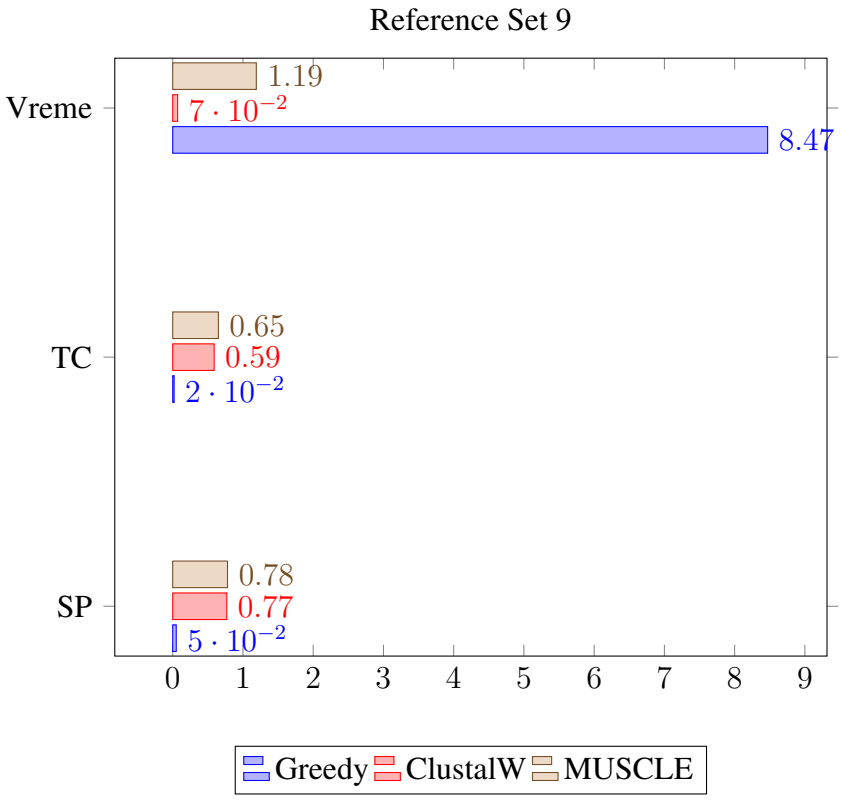
Tablica 5.2: Vreme izvršavanja u sekundama

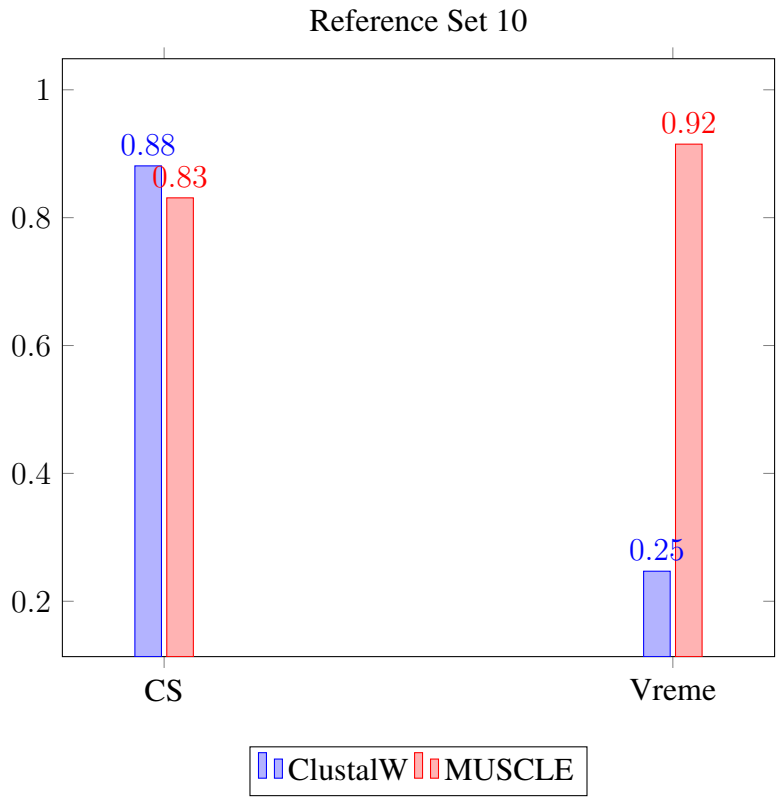
Skup sekvenci (broj sekvenci)	**ClustalW**	**MUSCLE**
BBA0011(102)	**0.900**	0.881
BBA0012 (69)	0.870	**0.979**
BBA0013 (19)	0.885	**0.886**
BBA0009 (59)	**0.905**	0.898
BBA0007 (41)	**0.984**	**0.984**
BBA0004 (248)	**0.737**	0.376
BBA0005 (44)	0.689	**0.709**

Tablica 5.3: CS skorovi

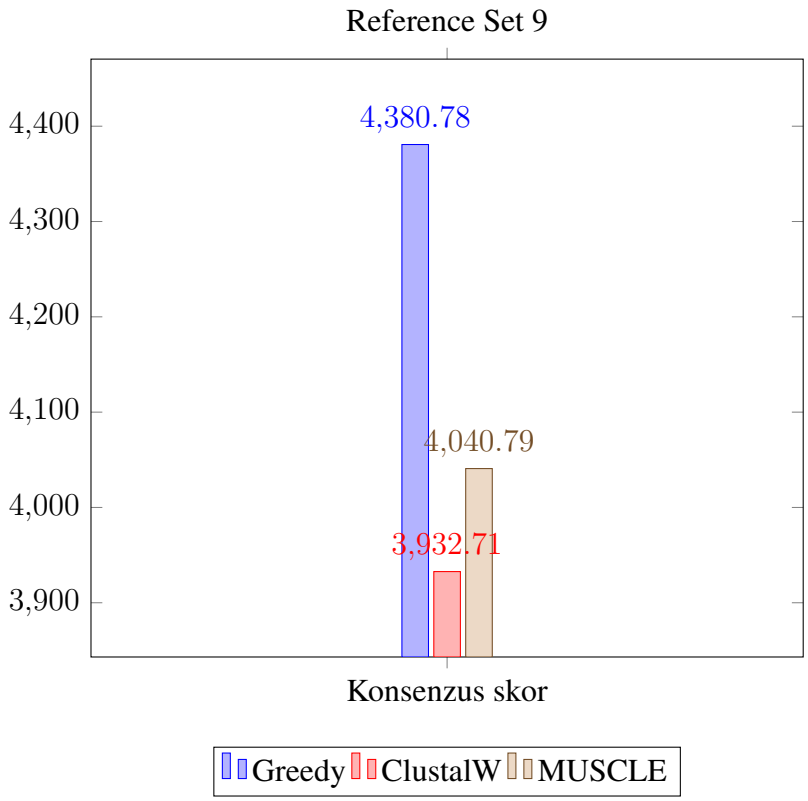
referentnog skupa. Može se opet zaključiti da postoji vrlo mala razlika između kvaliteta poravnanja ClustalW i MUSCLE alata i da ClustalW daje bolje optimalno poravnanje u slučaju velikog broja sekvenci. Što se tiče vremena izvršavanja, situacija je ista kao i kod prethodnog referentnog skupa, ClustalW daleko brže kreira poravnanja od MUSCLE.

Za potrebe testiranja u radu je iskorišćeno oko 1000 (iz *Reference Set 10*) i 40 (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, mogu videti u grafikonima ispod.





Kao što je već pomenuto u prethodnom poglavlju, pored ugrađenih skorova u radu je implementirano i izračunavanje konsenzus skora prema formuli (4.1). Zbog toga što konsenzus skor koristi drugačiju logiku za evaluaciju višestrukih poravnanja od ostalih predstavljenih skorova, u pojedinim slučajevima se može desiti da *Greedy Alignment* daje poravnanje sa najboljim skorom. Razlog tome leži u prirodi samog algoritma i odabrane formule za računanje konsenzus skora. Naime, prilikom upoređivanja dve sekvence kada *Greedy Algorithm* naiđe na slučaj kada se dva simbola ne poklapaju on će pre ubaciti *gap*, nego što će "žrtvovati" to nepoklapanje kako bi se možda ostatak bolje poravnao. Upravo zbog toga *Greedy Alignment* kreira višestruka poravnanja sa ogromnim brojem *gap*-ova. Sa druge strane, prilikom računanja konsenzus skora najveća kazna se dobija u slučaju kada se simboli ne poklapaju, dok se manja kazna dobija u slučaju poklapanja nekog simbola sa *gap*-om. Upravo iz ovih razloga se može desiti da *Greedy Alignment* nadmaši ClustalW i MUSCLE. Na primer, ako poravnanje koje je kreirao *Greedy Alignment* ima veliki broj *gap*-ova, a poravnanje koje je kreirao ClustalW ima nekoliko nepoklapajućih simbola, tada se može desiti da prvo poravnanje dobije manji konsenzus skor i da prema ovom skoru predstavlja optimalnije poravnanje. Međutim, prilikom testiranja ovakav specijalan slučaj se desio svega nekoliko puta i u većini slučajeva konsenzus skor *Greedy Alignment* poravnanja je bio najgori, što se može i zaključiti na sledećem grafikonu koji prikazuje prosečne vrednosti konsenzus skorova.


Reference Set 9
4,400
4,300
4,200
4,100
4,000
3,900
4,380.78
4,040.79
3,932.71
Konsenzus skor
Greedy
ClustalW
MUSCLE

6. Zaključak

Zbog značajnosti višestrukog poravnanja sekvenci, mnogi alati za višestruko poravnanje su razvijeni. 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 *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 skorova koje dobijamo nakon upoređivanja sa referentnim poravnanjima. Na primer, ako referentno poravnanje iz BAliBASE *Reference Set 10* uporedimo sa samim sobom dobićemo *CS* skor 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* skorova 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 ocene za *matches*, *mismatches* i *gaps* i izbor matrice skorova (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] Robert C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 2004.

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

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

[8] 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.

[9] 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/`.