SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Ivan Vujević

Alat za brzo pretraživanje baza proteinskih sljedova

Zagreb, 2016.

Ovaj rad je izrađen u Laboratoriju za bioinformatiku i računalnu biologiju pod vodstvom Prof. dr. sc. Mile Šikića i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2015./2016.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Značajka suvremenih bioloških istraživanja je velika količina podataka.[2] Porastom količine podataka raste i potreba za računalnim metodama koje bi te podatke obrađivali. Jedan od prvih zadataka bio je spremiti i rukovati zapanjujućom količinom podataka čime nastaju biološke baze podataka.

Tema ovog rada bavi se upravo tom problematikom, cilj je napraviti alat koji će omogućiti efikasno pretraživanje baza podataka bioloških sljedova. Efikasnošću smatramo što manje potrošenih računalnih resursa uz što veću preciznost dobivenih rezultata.

Poglavlje 2 opisuje problem kojim ćemo se baviti u ovom radu te načine na koji se dani problem može riješiti. Uz mogućnosti različitih pristupa rješavanju problema navedeni su i najpoznatiji alati koji koriste pojedini pristup rješenja.

Poglavlje 3 govori o načinu zapisa baza bioloških sljedova koji se koriste u ovom radu.

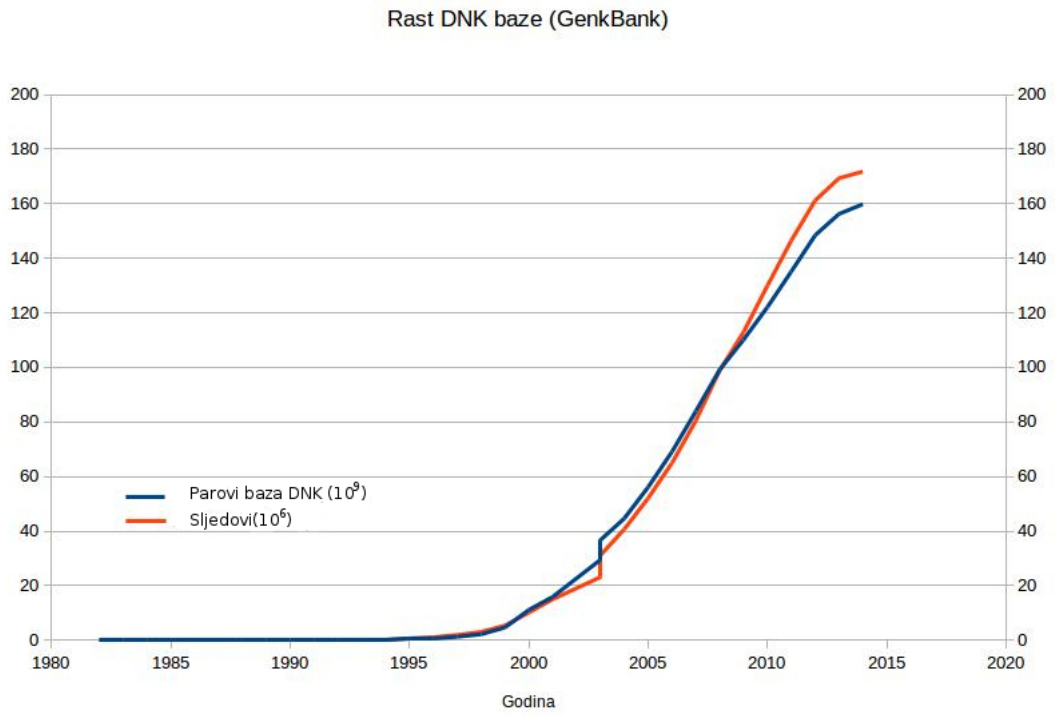
Poglavlje 4 govori o našem rješenju problema. Detaljnije je opisana problematika s kojom se susrećemo prilikom izrade alata te način na koji je dani problem riješen. Algoritam koji koristimo u radu je opisan po koracima.

Poglavlje 5 govori o načinu na koji je ostvareno programsko rješenje alata. Prikazan je dijagram razreda koji se koriste u implementaciji sustava

Poglavlje 6 govori o rezultatima dobivenim ispitivanjem alata. Prikazana je i usporedba s drugim alatima.

2. Pretraživanje baza bioloških sljedova

Kao što je već spomenuto, jedna od temeljnih uloga bioinformatike je omogućiti rukovanje velikom količinom podataka. Pod velikom količinom podataka podrazumijevamo baze podataka koje sadrže milijune zapisa. Što se tiče vrste podataka, to mogu biti baze sastavljene od proteinskih ili nukleotidnih sljedova. Kako se radi o velikoj količini podataka pretraživanje tih baza zahtjeva i uporabu većih računalnih resursa. Razvojem suvremene tehnologije povećavaju se i baze podataka kao i broj parova u njima.



Slika 2.1: Eksponencijalni rast baze podataka kroz godine

Charles Darwin je nakon povratka s otočja Galápagos predpostavio da postoji veza između sličnih, ali i različitih vrsta. Kasnije to iznosi u, danas svima dobro poznatom, djelu *Podrijetlo vrsta*. U tada iznesenim činjenicama leži razlog današnje primjene bioinformatike u pretraživanju velikih baza nukleotidnih ili proteinskih sljedova. Cilj je pronaći slične sljedove u drugim organizmima koji upućuju na zajedničko podrijetlo vrsta (*homologija*). Ako pogledamo sličnost između dvaju gena, odnosno ako postoje dva slijeda koji predstavljaju gene dvaju organizama, tada možemo zaključiti da oba gena potječu od zajedničkog pretka. Homolognost dvaju ili više sljedova možemo usporediti s razvojem jezika. Kao što znamo, vremenom jedan jezik preuzima riječi iz drugog jezika čime korijen riječi u jednom jeziku svoju osnovu vuče iz nekog drugog jezika, iz druge riječi. Pogledamo li u tablici 2.1 riječ *psihologija* prevedenu na engleski,njemački i francuski vidimo da postoji zajednički korijen riječi iz čega se može zaključiti da sve četiri riječi potječu iz istog jezika.

Tablica 2.1: Zajednički korijen riječi u različitim jezicima. Kao što postoje međusobni homologni sljedovi nukleotida tako možemo prikazati homolognost riječi u različitim jezicima. Homolognost sljedova označava postojanje zajedničkog pretka, a kod jezika govorimo o postojanju zajedničkog korijena iz nekog druog jezika.

Jezik	**Riječ**	**Poredak**
hrvatski	PSIHOLOGIJA	PS I- HOLOG IJA
engleski	PSYCHOLOGY	PSYCHOLOG I- -
njemački	PSYCHOLOGIE	PSYCHOLOGYE -
francuski	PSICOLOGIA	PS IC - OLOG IA

Kao što vidimo iz slike 2.1 povećanjem sljedova u bazi podataka istovremeno se povećava i broj parova koji se međusobno podudaraju. Povećanjem baza raste i potreba za novim alatima koji bi olakšali pretraživanje tih baza. Stvarajući algoritme za pretragu baza bioloških sljedova potrebno je poštivati određene kriterije. Prvi kriterij, kriterij **osjetljivosti**, zahtjeva pronalazak što je više moguće ispravnih pogodaka u bazi. Mjeri se opsegom uključenosti ispravno pronađenih sljedova koji pripadaju zajedničkoj *obitelji*[1]. Drugi kriterij je **selektivnost** koja označava sposobnost isključivanja pogrešnih pogodaka. Treći kriterij je **brzina**, koja predstavlja vrijeme potrebno za pronalaženje rezultata u bazi. Četvrti kriterij se odnosi na **memoriju**. Kriterij memorije od nas zahtijeva da trošimo što je moguće manje memorije. Idealan algoritam obuhvatio bi sva ta četiri kriterija, no u stvarnosti je to teško ostvariti. U odnosu na

[1] Obitelj - skup sljedova koji upućuju na zajedničku povezanost, zajedničko podrijetlo.

zahtjeve koji ispunjavanju, algoritme možemo razvrstati u dvije skupine. To su deterministički algoritmi koji nastoje ispuniti prva dva kriterija, te heuristički algoritmi koji teže povećanju brzine „zanemarujući" prva dva kriterija. Danas se uglavnom razvijaju alati temeljeni na heurističkom pristupu rješavanja problema koji povećava brzinu pretraživanja, ali ne garantira optimalnost. Korištenjem algoritama heurističkog pristupa imamo vrlo malu vjerojatnost za dobivanjem svih sljedova iz jedne obitelji. Iz tog razloga moramo naći kompromis između brzine i točnosti. Danas dva najpopularnija alata koji koriste heuristički pristup pretrage bioloških sljedova su **BLAST** i **FASTA** koji donose značajno vremensko poboljšanje u odnosu na metode koje koriste potpuno poravnanje za usporedbu (npr. Smith-Waterman algoritam) [2][11].
U nastavku rada opisana su dva suprotna pristupa rješavanja problema pretraživanja baza bioloških sljedova. Kao predstavnik heurističkog algoritma opisan je alat BLAST, a kao predstavnik deterministčkih algoritama opisan je algoritam Smith-Waterman.[11]

2.1. Smith - Waterman

Deterministčki algoritmi se vrlo rijetko koriste kod pretraga velikih baza podataka. Razlog tome je taj što ti algoritmi troše previše vremena i potrebni su ogromni računalni resursi. Heuristički algoritmi poput BLAST- a i FAST-a razvijeni su za brzu pretragu baza, no njihova mana je to što ne daju optimalno rješenje. Ispitivanja su pokazala da BLAST algoritam u prosjeku daje 30% manje izlaznih sljedova u odnosu na stvaran broj podudaranja u bazi.[6] Smith - Waterman jedan je od najpoznatijih algoritama u području bioinformatike, iako dosta spor, u slučaju dvaju sljedova duljine m, n složenost je $O(m \cdot n)$, algoritam se koristi jer jamči pronalazak optimalnog rezultata.

Smith - Waterman algoritam je matematički model koji osigurava najbolje lokalno poravnanje između dva slijeda i stoga se očekuje da će biti najpouzdanija metoda pretrage baza bioloških sljedova. Mana korištenja Smith - Waterman algoritma je u tome što je 50 - 100 puta sporiji od algoritama koji koriste heuristički pristup.

Cilj Smith - Waterman algoritma je pronaći najbolje lokalno poravnanje dvaju sljedova; rezultat poravnanja se ne mora nužno sastojati od svih elemenata iz oba skupa. Smith - Waterman algoritam nastaje kao nadogradnja na Needleman - Wunsch algoritam koji rješava problem globalnog poravnanja ulaznih sljedova. Algoritam koristi dinamičko programiranje i prvi je algoritam koji koristi dinamičko programiranje za

[2]Deterministički algoritam lokalnog poravnanja, jedan od najvažnijih u području bioinformatike.

usporedbu bioloških sljedova. Objavili su ga 1970. godine Saul Needleman i Christian Wunsch.[3]

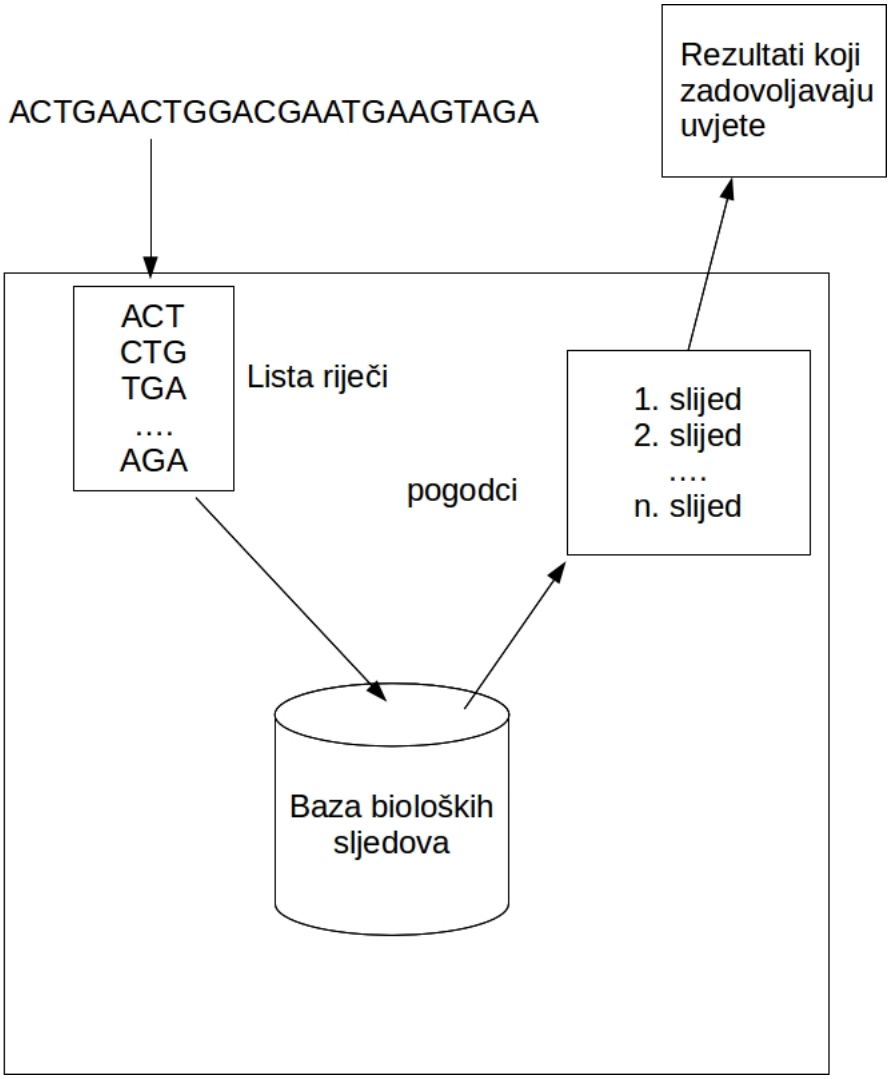
Poravnanjem dvaju sljedova cilj je postići da se što više elemenata (aminokiselina kod proteinskih i nukleotida kod nukleotidnih sljedova) iz jednog slijeda poklopi s elementima iz drugog slijeda. Kod lokalnog poravnanja važno je pronaći regije čije će poravnanje imati najveći rezultat. Kada promatramo sličnost dvaju sljedova ta vrijednost pada i raste kako se krećemo kroz oba slijeda. Smith-Waterman algoritam lokalnog poravnanja temelji se na sljedećoj formuli:

$$H(i,j) = max \begin{cases} 0 & i = 0 \; ili \; j = 0 \\ max \begin{cases} 0 \\ H(i-1, j-1) + w(s_i, t+j) \\ H(i-1, j) + d \\ H(i, j-1) + d \end{cases} & i \neq 0 \; i \; j \neq 0 \end{cases}$$

gdje je $w(a,b)$ cijena zamjene dva znaka, d je cijena umetanja ili brisanja, a H je maksimalna vrijednost između podniza $a[1...i]$ i podniza $b[1...j]$

2.2. BLAST

BLAST *(engl. Basic Local Alignment Search Tool)* je algoritam koji povećava brzinu poravnanja smanjujući prostor pretraživanja. Umjesto usporedbe sekvence iz upita sa svakom sekvencom iz baze, BLAST koristi kratke podnizove koji se nazivaju *riječi (engl.words)* te služe kao *sjeme (engl. seed)* za poravnanje.



Slika 2.2: Prikaz rada blast alata.

Kao što je prikazano na slici 2.2 BLAST na početku iz upitnog slijeda stvori listu riječi određene duljine; listu definira korisnik. Potom se te riječi koriste kao sjeme za usporedbu sa sljedovima iz baze. Kao rezultat BLAST vraća listu rezultata. Koraci BLAST alata (osnovni BLAST) prikazani su algoritmom 1.

Algoritam 1: BLAST

1. Iz upitnog slijeda odrediti listu riječi **L**.
2. $\forall w \in$ **L** pronaći slične riječi, izgraditi listu riječi **L'** čije poravnanje s w daje rezultat koji se nalazi unutar određenih granica.
3. U bazi se traže sljedovi koji sadrže riječi iz **L'**. Svaki pronađeni podniz iz liste L' nazivamo **pogotkom**.
4. Proširiti poravnanje oko pogotka u lijevo i desno, sve dok rezultat poravnanja ne počne padati ispod zadanog praga. Dobivena područja nazivaju se područja s visokim rezultatom poravnanja.[a]
5. Od prethodno dobivenih područja odabrati ona s najvišim rezultatom i odrediti njihov statistički značaj (E vrijednost).[b]

[a] engl. **HSP** - High-scoring Segment Pair

[b] **E vrijednost** - statistička mjera koja prikazuje relevantnost rezultata. Detaljnije objašnjena u nastavku rada.

Kod određivanja duljine riječi u obzir je potrebno uzeti radi li su o slijedu nukleotida ili aminokiselina. U slučaju slijeda aminokiselina, duljina riječi je manja nego u slučaju nukleotida jer je vjerojatnost pojavljivanja jednog nukleotida puno veća od vjerojatnosti pojavljivanja aminokiseline. Obično se kod nukleotida uzimaju riječi duljine 3, dok se za proteinske sljedove uzimaju riječi duljine 11. Lista riječi uključuje sve podnizove duljine k koji se mogu dobiti iz ulaznog slijeda.

3. Podaci

Koristi se baza bioloških sljedova zapisana u FASTA formatu. FASTA format sastoji se od dva dijela, opisa i slijeda. Redak u kojem je opis slijeda počinje znakom „>” nakon čega slijedi identifikator slijeda pa nakon toga opis. Ne smije postojati razmak između znaka „>” i identifikatora. Nakon opisa slijedi jedan ili više redaka u kojima je zapisan biološki slijed. Znakovi kojima je zapisan biološki slijed ovise o tome radi li se o slijedu nukleotida ili proteina. Ako se radi o slijedu nukleotida tada su znakovi :

A - Adenin	**C** - Citozin	**G** - Gvanin	**T** - Timin
U - Uracil	**R** - A ili G	**Y** - C, T ili U	**K** - G, T ili U
M - A ili C	**S** - C ili G	**W** - A, T ili U	**B** - ne A
D - ne C	**H** - ne G	**V** - niti T niti U	**N** - A, C, G,T, U
X - maskiranje	**–** - procjep neodređene duljine		

Ukoliko se radi o slijedu aminokiselina tada je slijed u FASTA formatu zapisan sljedećim znakovima:

A - Alanin	**C** - Cistein	**D** - Asparaginska kiselina	**E** - glutaminska kiselina
F - fenilanin	**G** - glicin	**H** - histidin	**I** - izoleucin
K - lizin	**L** - leucin	**M** - metionin	**N** - asparagin
P - prolin	**Q** - glutamin	**R** - arginin	**S** - serin
T - treonin	**V** - valin	**W** - triptofan	**Y** - tirozin
X - bilo što	* - STOP	**–** - procjep	**B** - D ili N
J - I ili L	**O** - pirolizin	**U** - selenocistein	

U datoteci može biti više zapisa, a svaki zapis se sastoji od linije opisa i jedne ili više linija u kojim je zapisan slijed. Preporučeno je da duljina redaka unutar datoteke FASTA formata ne bude duža od 80 znakova.

Primjer datoteke u FASTA formatu:

>Opis1
GAGGVMLLISTS
>Opis2
AMMLSTG

Format datoteke u kojoj su zapisani upiti koji se pretražuju u bazi također mora biti u FASTA formatu.

4. Metoda

U poglavlju 2.2 spomenuto je da alat BLAST koristi listu riječi kao sjeme na temelju kojeg počinje pretragu baza nukleotidnih ili proteinskih sljedova. Sličan pristup koristit ćemo i u ovom alatu, s razlikom što riječima ne pretražujemo cijelu bazu već riječi služe kao indeksi kojima su određeni sljedovi. Cilj je svaki slijed iz baze predstaviti određenim skupom riječi duljine k koji će omogućiti bržu pretragu baze. Tražeći skup riječi koji će odrediti pojedini slijed, potrebno je težiti ispunjenju kriterija pretrage baza bioloških sljedova koji su spomenuti u poglavlju 2.

Kao predstavnike pojedinog slijeda izabrat ćemo one riječi koje se najčešće pojavljuje u bazi sljedova[7]. Razlog biranja riječi koje se pojavljuju najčešće u bazi je taj što želimo dobiti što više sljedova za ulazni slijed, naravno, treba poštivati i kriterij da dobijemo što manje pogrešnih pogodaka u bazi. Uz riječi visoke frekvencije biramo i skup riječi niske frekvencije. Dakle, one riječi koje se pojavljuju najmanji broj puta.

Kako bi spriječili dominiranje „nepotrebnih" riječi prije određivanja predstavnika pojedinog slijeda potrebno je ukloniti sve regije koje imaju neuobičajen poredak nukleotida ili aminokiselina. Kod proteinskih sljedova koristi se alat Seg, a kod nukleotidnih sljedova alat DustMasker.[16] Također, kod brojanja frekvencija pojavljivanja potrebno je paziti da ne brojimo preklapajuće riječi.

Metoda koja se koristi u ovom radu sastoji se od dva koraka koji služe za pronalaženje sljedova i predprocesiranja kojim gradimo indekse iz velike baze. U prvom koraku pronalaze se potencijalni sljedovi iz baze koji mogu biti rezultat upita, a u drugom koraku se ti sljedovi obrađuju koristeći jedan od ponuđenih algoritama za poravnanje.

Kao ulazne podatke, algoritam prihvaća dva različita skupa podataka, lance proteina i lance DNK-a. Ako je na ulazu lanac proteina onda se odmah kreće na prvi korak algoritma i procedura ide dalje kako je objašnjeno u nastavku. Ako smo na ulazu dobili lanac DNK-a onda je taj lanac potrebno pretvoriti u lanac proteina. Pretvaranje obavljamo tako da prvo od jednog lanca proteina stvaramo 6 novih lanaca. Svaki od tih lanaca dobivamo kao pomak od oba kraja. Prvi lanac je lanac koji je na ulazu, drugi dobijemo izbacivanjem prve baze, a treći lanac dobijemo izbacivanjem prve dvije baze.

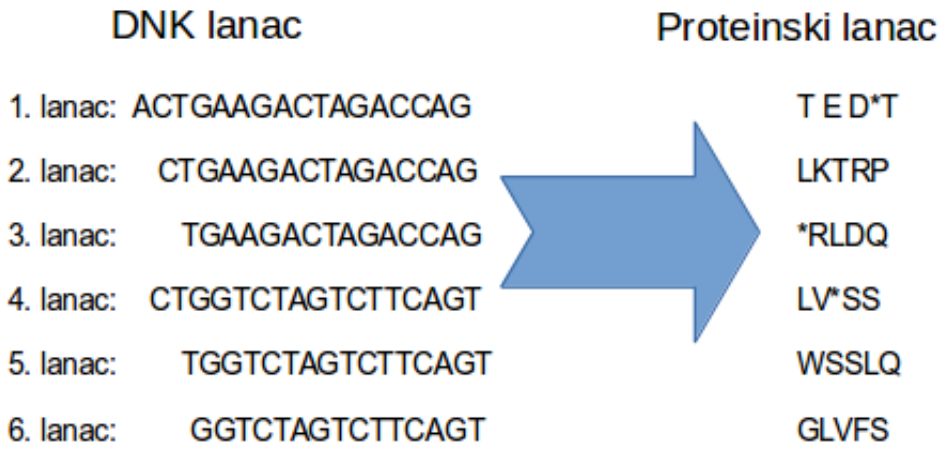
Za sljedeća tri lanca ponovimo isti postupak, ali ovaj put na početku imamo reverzni komplement ulaznog lanca. Razlog uzimanja 6 lanaca je taj što ne znamo kako je očitan lanac tako da se prilikom pretvaranje u proteine može dogodili bilo koji od tih slučajeva, ovisno o tome gdje se pojavio start kodon.

Nakon što smo dobili 6 lanaca, krećemo od početka svakog lanca i uzimamo po tri znaka. Tri znaka nam određuju kodon. Kodon je triplet baza nukleotida i kodira jednu aminokiselinu. U sljedećoj tablici prikazani su tripleti i njima odgovarajuće aminokiseline.

Tablica 4.1: Tablica kodona.

Prvi	T	C	A	G	Zadnji
T	Phe / F	Ser / S	Tyr / Y	Cys /C	T
	Phe / F	Ser / S	Tyr / Y	Cys /C	C
	Leu / L	Ser / S	Stop	Stop	A
	Leu / L	Ser / S	Stop	Trp / W	G
C	Leu / L	Pro / P	His / H	Arg / R	T
	Leu / L	Pro / P	His / H	Arg / R	C
	Leu / L	Pro / P	Gln / Q	Arg / R	A
	Leu / L	Pro / P	Gln / Q	Arg / R	G
A	Ile / I	Thr / T	Asn / N	Ser / S	T
	Ile / I	Thr / T	Asn / N	Ser / S	C
	Ile / I	Thr / T	Lys / K	Arg / R	A
	Met / M	Thr / T	Lys / K	Arg / R	G
G	Val / V	Ala / A	Asp / D	Gly / G	T
	Val / V	Ala / A	Asp / D	Gly / G	C
	Val / V	Ala / A	Glu / E	Gly / G	A
	Val / V	Ala / A	Glu / E	Gly / G	G

Na sljedećoj slici vidimo primjer pretvaranja jednog niza DNK u proteinski lanac.



Slika 4.1: Pretvaranje DNK niza u niz aminokiselina

4.1. Prvi korak

Prvim korakom algoritma cilj je pronaći što više potencijalnih sljedova koji će biti izlaz nakon drugog koraka. Idealan bi slučaj bio da u prvom koraku dobijemo sve moguće rezultate i da te rezultate „potvrdimo" drugim korakom. Kao što je poznato, algoritam koji se razvija u ovom radu spada u vrstu heurističkih algoritama te je stoga nemoguće očekivati idealan slučaj. Potrebno je pronaći kompromis između točnosti prvog koraka i potrebe za dobivanjem što većeg broja sljedova iz baze. U ovom koraku pretražujemo reduciranu bazu. Redukcijom baze na pojedine riječi vezali smo sekvence (redni broj u bazi) kojima je ta riječ jedan od indeksa. Na taj način, efikasnom pretragom dobivamo sve sekvence kojima je ta riječ jedan od indeksa. Dobivene sekvence prolaze kroz tri filtra. U prvom filtru odbacujemo one sekvence koje nisu pogođene barem tri puta s riječima visoke frekvencije ili barem 2 puta s riječima niske frekvencije. U drugom filtru potrebno je provjeriti jesu li riječi iz indeksnog skupa pojedinog slijeda u istom poretku kao i riječi u upitnom slijedu. Dalje idu samo oni sljedovi koji imaju isti raspored riječi kao i upitni slijed. U trećem koraku računamo dijagonalu na kojoj se riječi nalaze. Ako je upit duljine M, a potencijalna sekvenca duljine N, onda je dijagonala definirana kao NxM matrica. Sljedeća formula određuje dijagonalu na kojoj se pogodak nalazi:

$$D = N - 5 - (P_{cilja} - P_{upit})$$

gdje je D indeks dijagonale trenutne riječi, P_{cilja} je pozicija na kojoj se ta riječ u potencijalnom slijedu, a P_{upit} je pozicija na kojoj se ta riječ nalazi u upitu. Kada je

dijagonala izračunata povećavamo brojač za tu dijagonalu. Nakon što smo izračunali sve dijagonale tražimo dijagonalu koja ima najveću vrijednost brojača. Ako je maksimalna vrijednost manja od 2 odbacujemo taj slijed. Dakle, dalje propuštamo samo one sljedove koji imaju barem 2 pogotka na jednoj dijagonali.

Algoritam 2: Prvi korak algoritma

Data: $m_{high} = 3$,$m_{low} = 2$; $c_{high} = 0$,$c_{low} = 0$

Za svaku riječ w duljine k iz ulaznog slijed pronađi sve sljedove kojima je ta riječ jedan od predstavnika.

Ako je riječ predstavnik skupa riječi visoke frekvencije, c_{high} povećaj za 1

Ako je riječ predstavnik skupa riječi niske frekvencije, c_{low} povećaj za 1

Odbaci one sljedove kojima je $c_{high} < 3$ i $c_{low} < 2$

Odbaci sljedove kojima poredak riječi u upitu i potencijalnom nizu nije isti

Odbaci one sljedove čiji indeksi s upitom nemaju barem dvije riječi na jednoj od dijagonala

4.2. Drugi korak

Završetkom prvog koraka, kojim su se dobili sljedovi koji bi potencijalno mogli odgovarati upitu, prelazi se na drugi korak algoritma. Zadaća drugog koraka je uzeti one sljedove koji se s upitnim podudaraju s određenom sličnošću. Sličnost dvaju sljedova ovim alatom možemo izračunati na četiri načina. Na izbor korisniku nudi se mogućnost globalog poravnanja, lokalnog poravnanja i dvije mogućnosti polu-globalnog poravnanja. Za poravnanje se koristi biblioteka Opal [14] .

Poravnanje bioloških sljedova najčešći je postupak kojim se utvrđuje sličnost između dva niza[3]. Problem je prvi definirao Richard Hamming 1950. godine i to u području teorije informacije. Definirao je mjeru udaljenosti između dva niza znakova jednake duljine. Ako gledamo dva niza znakova s i t tada se Hammingova udaljenost definira kao broj pozicija na kojima se ti nizovi razlikuju. Budući je to mjera definirana samo za nizove iste duljine kao takva nam u bioinformatičkoj primjeni ne koristi. Potaknut time, godine 1965. Vladimir Levenshtein je poopćio ovaj problem za dva niza

proizvoljnih duljina. Mjeru udaljenosti između znakova s i t definirao je kao minimalan broj potrebnih modifikacija nad jednim znakom, potrebnih za pretvaranje niza s u t i nazvao ju je udaljenost uređivanja (engl. *"edit distance"*). Tri dozvoljene operacije nad jednim znakom su:

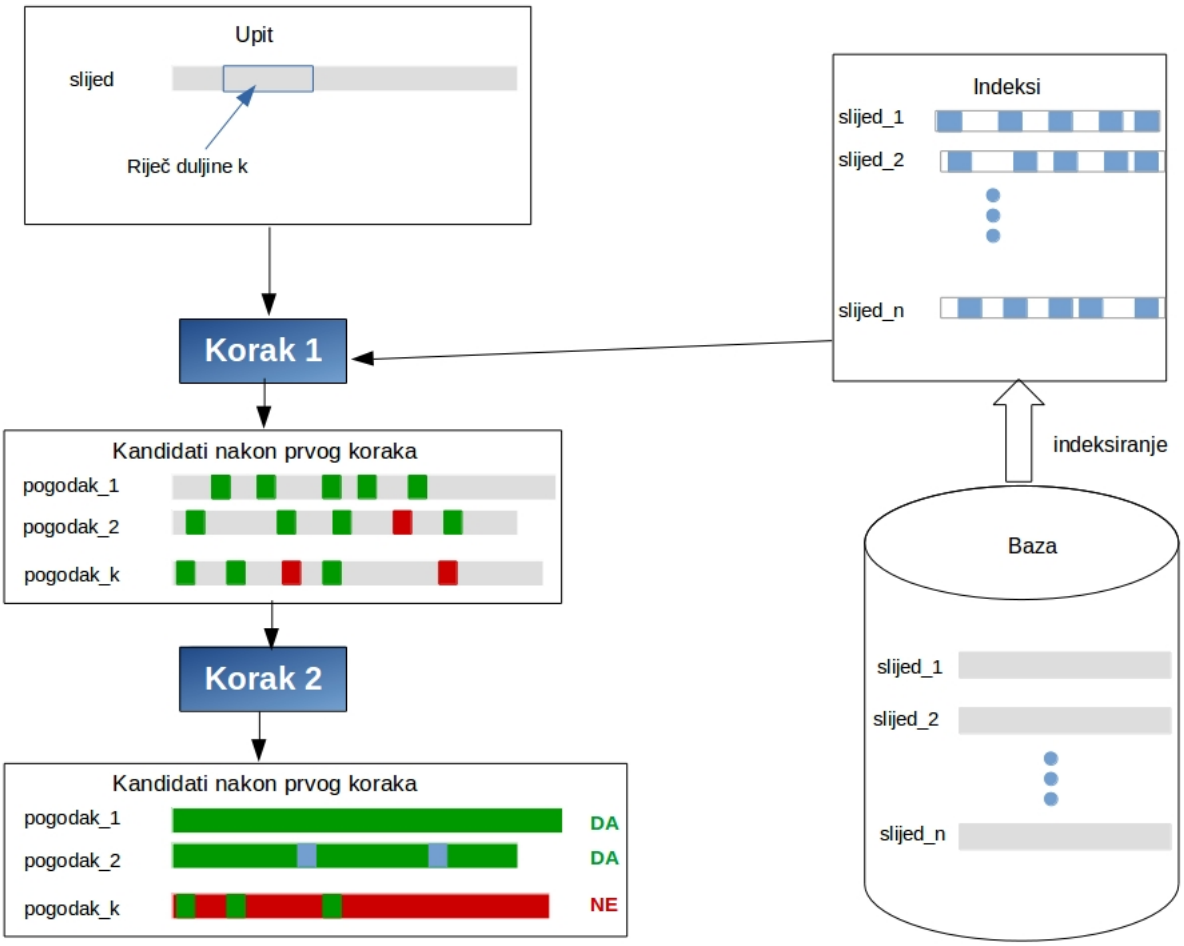
– Zamjena - promjena jednog znaka iz niza s u znak s odgovarajuće pozicije iz niza t

– Umetanje - umetanje jednog znaka u niz s u cilju da odgovara nizu t

– Brisanje - brisanje jednog znaka niza s u cilju da odgovara t

Nakon dobivenog poravnanja potrebno je izračunati njegovu dobrotu. Dobrotu poravnanja definiramo kao očekivanu vrijednost [1] na temelju vjerojatnosti da se to poravnanje pojavi u bazi. Način računanja očekivane vrijednosti objašnjen je u 4.2.1

Na slici 4.2 prikazan je rad algoritma. Na desnoj strani slike vidljiva je početna baza bioloških sljedova iz koje se, postupkom indeksiranja, dobivaju predstavnici pojedinog slijeda koji se pretražuju u prvom koraku algoritma. Na lijevoj strani slike vidljiv je ulazni slijed koji ulazi u prvi korak algoritma i pretragom indeksa dobiva određeni broj rezultata. Na slici su prikazana tri slijeda koji se dobivaju kao rezultat te koji zadovoljavaju uvjet od minimalnog broja pogođenih riječi. U ovom slučaju svaki je slijed određen s 5 riječi i minimalno mora biti pogođen s 3 riječi da bi mogao ići u drugi korak. Na kraju su prikazni rezultati nakon drugog korak algoritma.



Slika 4.2: Prikaz rada algoritma

4.2.1. E - vrijednost

Kako bi izveli formulu za računanje E-vrijednosti krenimo od jednog matematičkog problema.

Zadana su dva niza znakova, prvi duljine m, drugi duljine n. Razdioba znakova u znakovnom nizu ravna se po jednolikoj razdiobi tako da je p vjerojatnost pojavljivanja jednog znaka. Iz prvog niza odabiremo podniz duljine l te nas zanima vjerojatnost pojavljivanja tog podniza u drugom nizu. Pogledajmo prvo vjerojatnost da se odabrani podniz duljine l pojavi na početku drugog niza, između znaka na poziciji 1 i l. Vjerojatnost tog događaja je p^l. Budući u drugom nizu ima $n-l+1$ podniz duljine l, slijedi da je vjerojatnost koju smo tražili jednaka $\sum_{i=1}^{n-l+1} p^l = (n-l+1) \cdot p^l$

Pogledajmo sada kako se prethodni izračun koristi za računanje E-vrijednosti kod alata BLAST. U 2.2 spomenuto je da alat BLAST koristi listu riječi kojima radi pretragu nad bazom podataka. Lista je dobivena iz početnog niza. Recimo sada da je upitni niz duljine m, niz iz baze duljine n, a duljina riječi neka je l. Želimo izračunati očekivan broj preklapanja riječi iz ulaznog niza s nizom iz baze. Izračunali smo da je vjerojatnost pojavljivanja jedne riječi $p_r = (n - l + 1) \cdot p^l$. U upitnom slijedu ima $m - l + 1$ riječ pa je očekivanje onda $E = (m - l + 1) \cdot p_r$. Nizovi duljine n i m su puno dulji od l i od 1 tako da se $m - l + 1$ može pisati kao m, a $n - l + 1$ kao n.[4] Konačno pišemo

$$E = m \cdot n \cdot p^l \tag{4.1}$$

što se još može zapisati kao

$$E = m \cdot n \cdot e^{-\lambda l} \tag{4.2}$$

gdje je $\lambda = -\log(p)$. Prikazane formule predstavljaju očekivano preklapanje riječi iz jednog niza s riječima iz drugog niza. Alat BLAST uvodi još neke parametre kojima računa očekivanje, tako da je formula koju alat BLAST koristi za računanje E-vrijednosti:

$$E = K \cdot m \cdot n \cdot e^{-\lambda S} \tag{4.3}$$

gdje su λ i K Karlin-Altschulovi statistički parametri određeni svojstvima baze podataka.

Razlog zbog kojeg računamo i koristimo E-vrijednost je taj što želimo vidjeti koliko smo pretragu dobro napravili, odnosno u drugom koraku eliminirati sljedove koji ne odgovaraju slijedu iz upita, koje smo slučajno dobili. Vjerojatnost da smo slučajnim putem dobili rezultat poravnanja S je:

$$p = 1 - e^{-E} \tag{4.4}$$

Nakon što izračunamo E-vrijednost za pojedino poravnanje, kao izlaz dajemo sljedove koji imaju malo očekivanje iz čega slijedi da imaju i malu vjerojatnost slučajnog pogotka. Granica očekivanja iznad koje ćemo pogodak smatrati slučajnim ovisi o tome koliko su srodni sljedovi koji se uspoređuju.

5. Implementacija

Ovisno o želji korisnika aplikacija se može koristiti kao klijent-server aplikacija ili kao aplikacija koja će nakon obavljenih upita završiti svoje izvođenje. Broj dretvi koje su pokrenute prilagođen je sklopovlju na kojem se aplikacija izvodi. Prije pokretanja cijelog algoritma potrebno je u memoriju učitati bazu i indekse. Taj posao je raspodijeljen na tri dretve. Jedna zadužena za čitanje baze, a druge dvije istovremeno čitaju indekse iz skupa riječi visoke i niske frekvencije.

Paralelizacija glavnog algoritma obavlja se na način da imamo jednu dretvu koja čita upite i raspoređuje ih među dretvama koje obavljaju cijeli posao. Svaka dretva je zadužena za svoj upit i unutar radne dretve se obavljaju prvi i drugi korak algoritma. Za paralelizaciju je korištena biblioteka OpenMP[15] koji nam omogućuje jednostavan rad i komunikaciju među dretvama. Kako bismo izbjegli potrebu za sinkronizacijom i bespotrebnim čekanjem glavna dretva svakoj radnoj dretvi dodijeli prostor unutar kojeg će upisati svoje dobivene rezultate. Nakon što sve radne dretve završe, glavna dretva pokupi dobivene rezultate po dretvama i spoji ih u jedan rezultat koji je kasnije prikazan korisniku. U nastavku je prikazana funkcija koja će će stvoriti radnike i svakom od njih dodijeliti pojedini zadatak.

Algoritam 5.1: Stvaranje radnih dretvi

```
void search(ChainSet& queries, Type type, OutSet &results,
        AlignmentType align_type, double max_evalue,
         int max_out, std::shared_ptr<EValue> evalue_params,
         std::shared_ptr<ScoreMatrix> scorer) {

  #pragma omp parallel
  {
    #pragma omp single
    {
      for(int i =0; i < queries.size();i++) {
```

```
            #pragma omp task shared(results)
            {
              findInDatabase(queries[i],type,std::ref(results[i]),align_type,ma
              int resLen = results[i].size();
              results[i].resize(min(resLen,max_out));
              #pragma omp critical
              cout<<i<<"\t"<<resLen<<endl;
            }
          }
        }
        #pragma omp barrier
    }
}
```

5.1. Organizacija sustava

Sustav se sastoji od četiri osnovna modula od kojih svaki implementira specifičnu zadaću. Prvi modul nosi ime "makedb" i služi za pripremu baze. Učitava bazu s diska i stvara novu, reduciranu, bazu. Taj modul je potrebno pokrenuti samo jednom, prilikom prvog inicijaliziranja baze. Drugi i treći modul služe za pretragu baze i nose naziv "blastp" i "blastx". Modul "blastp" omogućuje poravnanje proteinskih lanaca na bazu proteina, a modul "blastx" omogućuje mapiranje DNK-a lanaca na bazu proteina. Zadnja dva modula omogućuju rad sustava na klijent-server arhitekturi. Jedan modul implementira funkcionalnost servera koji čeka na upit s odgovarajućim parametrima i putanjom do datoteke koju želimo poravnati s bazom proteina. Taj upit mu šalje dio sustava implementiran u četvrtom modulu.

Sustav je organiziran po klasam koristeći se praksom objektno-orijentiranog dizajna i poštivajući konvenciju pisanja C++ koda. U nastavku je dan popis klasa i kratki opis svake od njih.

Base - Razred koji predstavlja bazu podataka koja je spremljena na disku i koja ima svoje indekse. Unutar tog razreda implementirane su sve radnje koje je potrebno raditi s bazom koja se nalazi na disku (učitavanje baze, indeksiranje, učitavanje i spremanje indeksa). Za vrijeme izvođenja, baza algoritma komunicira s radnim dretvama.

DatabaseElement - Razred koji predstavlja jedan zapis iz datoteke *fasta* formata.

Jedan zapis u fasta formatu se sastoji od opisa koji počinje znakom „>”, te niza znakova koji predstavljaju proteinski lanac. Kako bi ubrzali izvođenje, razred još pamti duljinu imena, i duljinu pripadnog lanca. Također, svaki razred ima jednoznačno određeni broj koji pripada samo tom razredu.

Evalue - Razred koji pohranjuje parametre i implementira funkciju za računanje e-vrijednosti.

ScoreMatrix - Razred koji pohranjuje i implementira rad sa substitucijskim matricama koje se koriste prilikom poravnanja dva niza.

Alignment - Razred koji pohranjuje i obrađuje sve podatke dobivene poravnanjem, potrebne prilikom prikaza rezultata korisniku.

Algorithm - Razred u kojem je implementirana glavna funkcionalnost algoritma.

Util - Razred koji nudi implementaciju radnji koje je potrebno napraviti iz više razreda.

Seg - Razred koji nudi implementaciju alata za maskiranje regija niske složenosti proteinskih lanaca

Writer - Razred koji omogućuje prikaz dobivenih rezultata korisniku.

6. Rezultati

Dobiveni rezultati uspoređeni su s, danas dva najpopularnija alata za pretražaivanja baza bioloških sljedova, DIAMOND[5] i BLASTP. DIAMOND je relativno novi alat koji je svojim rezultatima opasno potukao konkurenciju i ozbiljno se približio BLASTP-u od kojeg je brži nekoliko redova veličina.

Korištena je NR baza proteinskih sljedova veličine 32GB i koja sadrži 54,183,042 proteinskih lanaca. Ukupna duljina svih proteinskih lanaca iznosi 19531459180 znakova, što daje prosječno 360 znakova po lancu.

Kako bi imali što vjerodostojnije testove brzine i senzitivnosti odabrali smo 4 različita seta. To su HumDiv, HumVar [9], oba neutralne varijante i setove iz Ensemble anotacije gena: Escherichia Coli, Plasmodium Falciparum. Broj proteinskih lanaca i ukupna duljina svih proteina unutar svakog od setova prikazana je u tablici 6.1

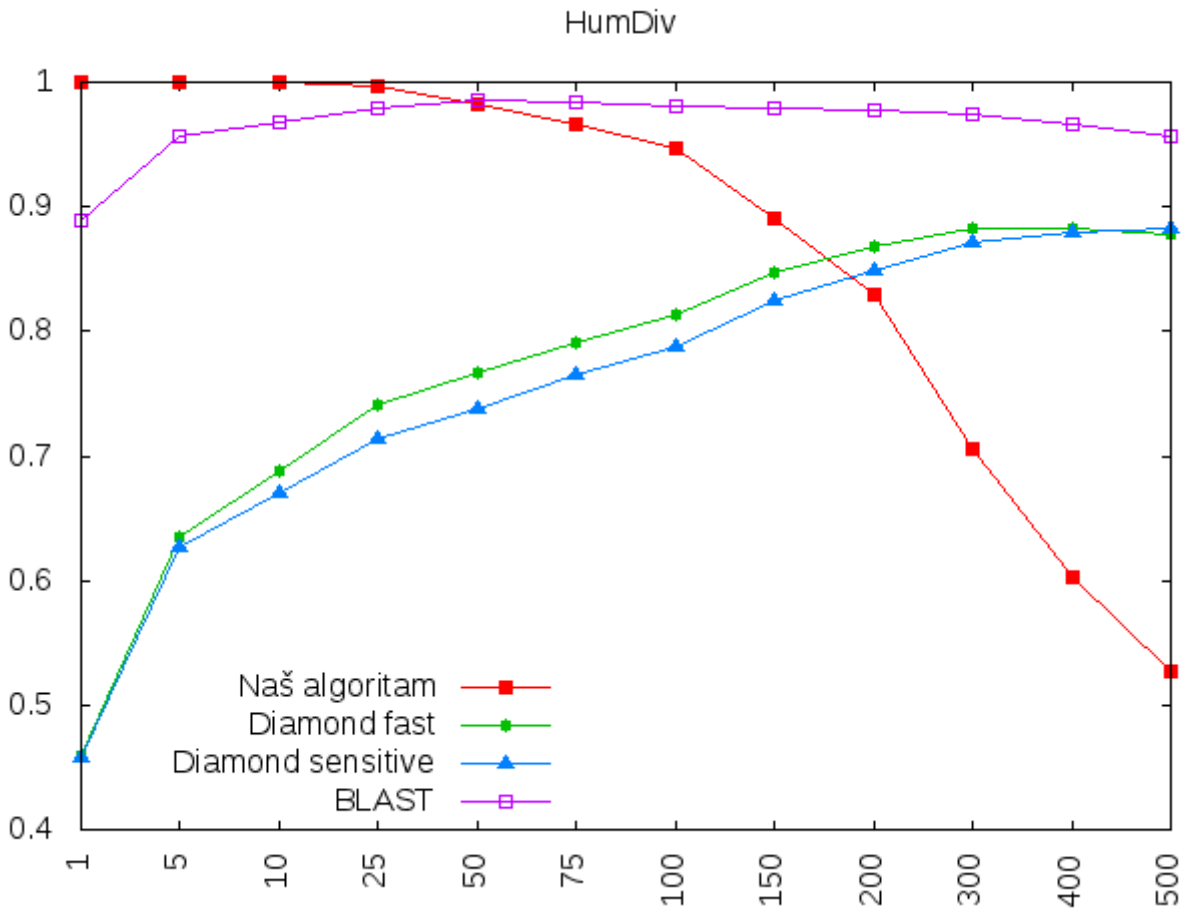
Tablica 6.1: Sljedovi proteina za usporedbu.

Ime skupa	Broj proteinskih lanaca	Duljina
HumDiv	315	235,219
HumVar	4,969	2,263,906
Escherichia Coli	4,969	1,393,750
Plasmodium	5,405	4,112,977

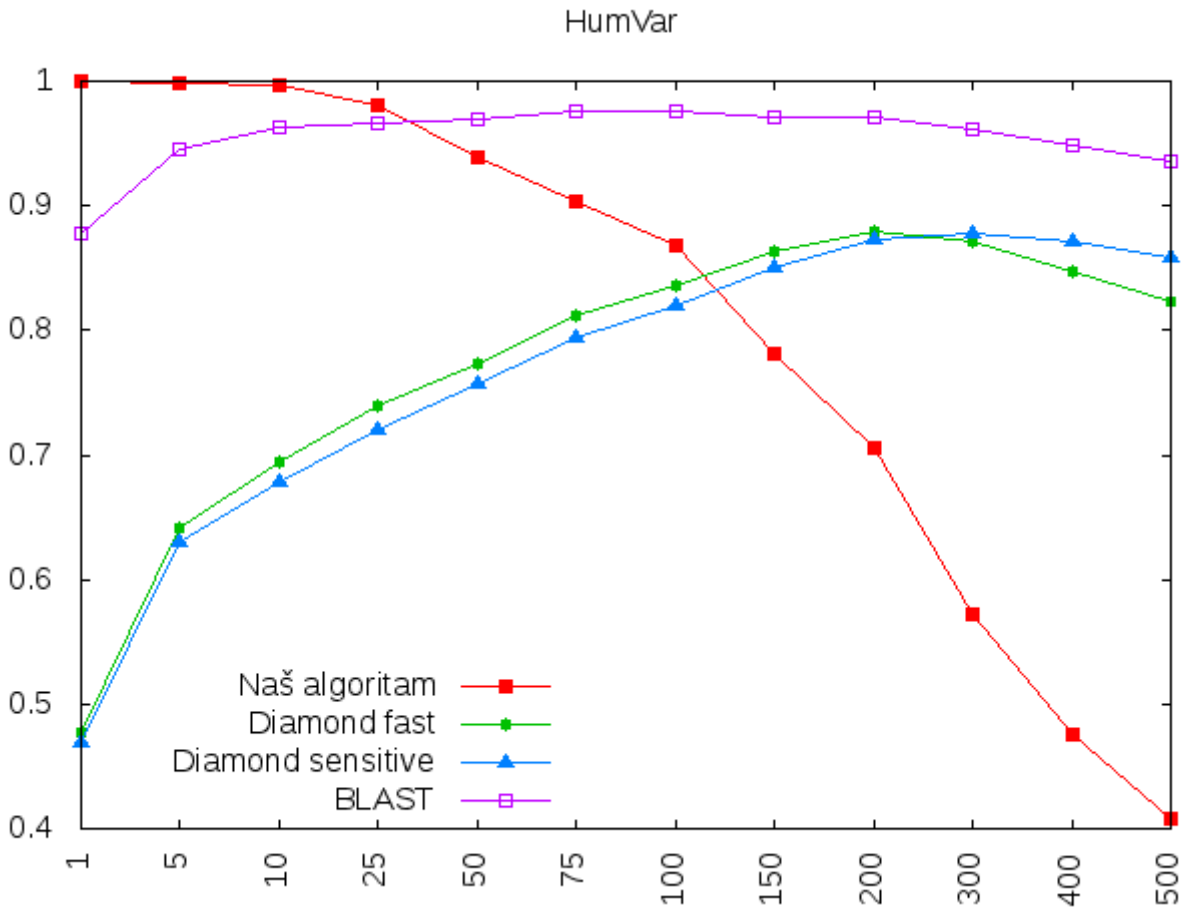
6.1. Testiranje senzitivnosti

Senzitivnost smo mjerili s obzirom na rang dobivenih rezultata. Kao referentni skup smo uzeli rezultate dobivene alatom SW# koji koristi Smith-Waterman poravnanje između upita i baze te na taj način garantira pronalazak optimalnih rezultata. Budući da DIAMOND ima 2 tipa rada, fast i sensitive, oba smo uključili u testiranja. Mjerenje smo obavili na način da smo gledali koliki postotak unutar određenog broja prvih SW# izlaza daje svaki od algoritama. Graf ima 12 točki mjerenja, odabranih iz skupa

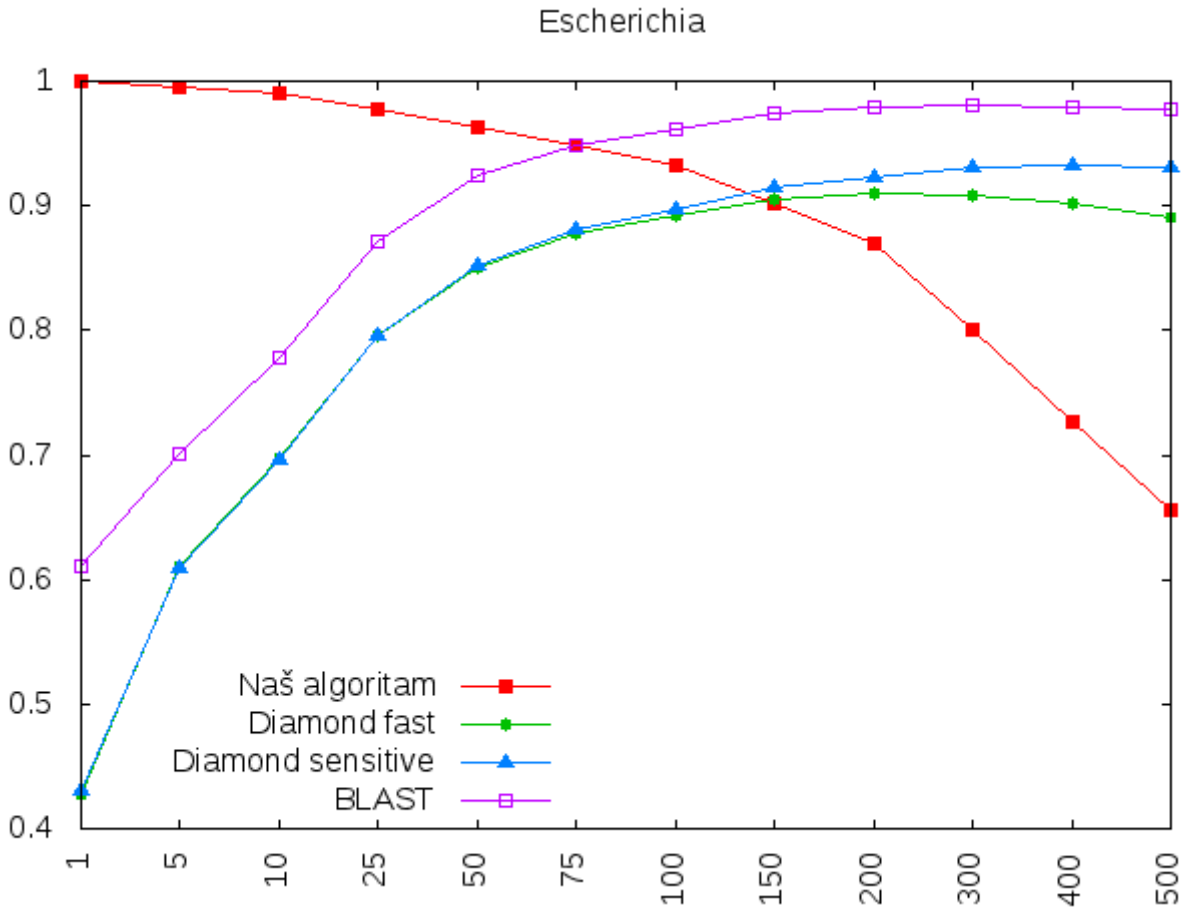
$\{1, 5, 10, 25, 50, 75, 100, 150, 200, 300, 400, 500\}$. Na sljedećim grafovima prikazani su odnosi između pojedinih algoritama za svaki od 4 ispitna skupa.



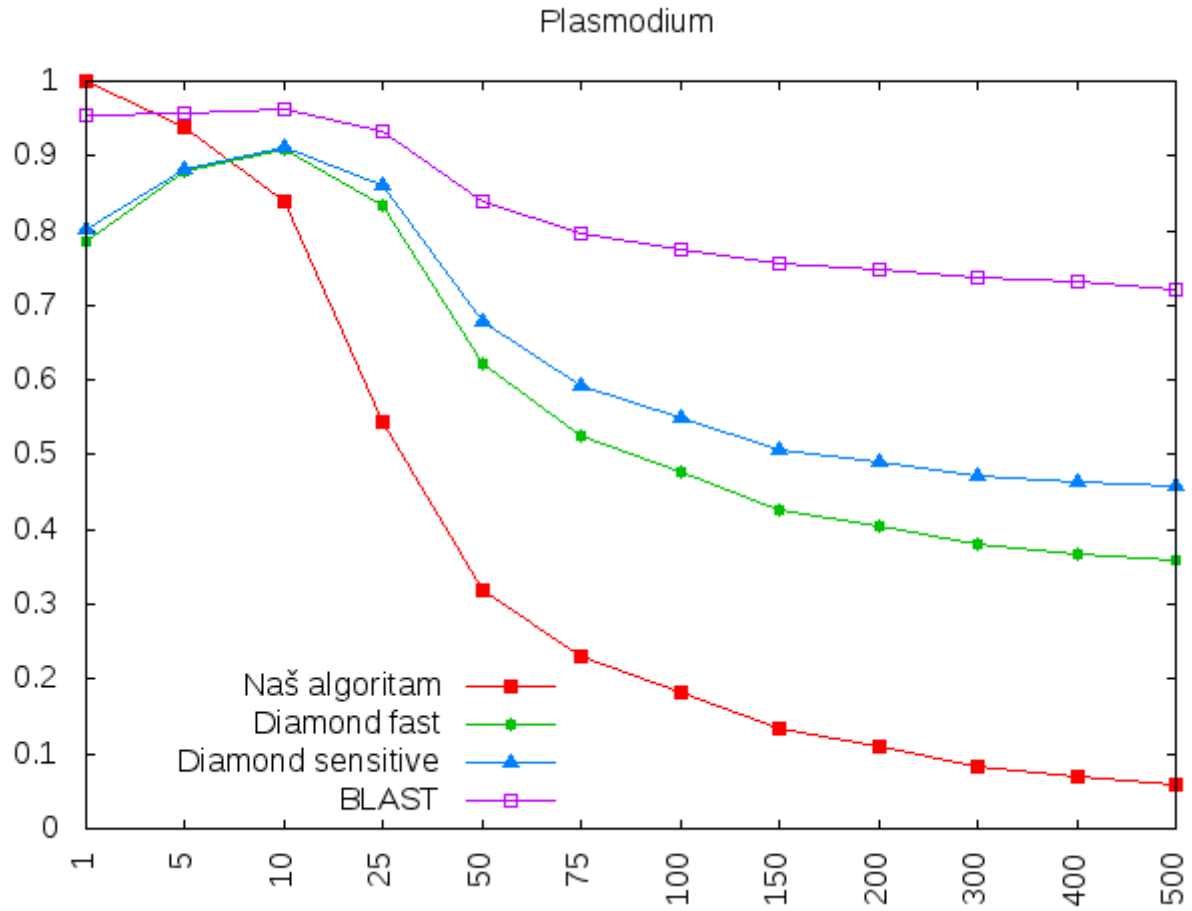
Slika 6.1: Prikaz senzitivnosti na skupu HumDiv



Slika 6.2: Prikaz senzitivnosti na skupu HumVar



Slika 6.3: Prikaz senzitivnosti na skupu Escherichia



Slika 6.4: Prikaz senzitivnosti na skupu Plasmodium

Iz prethodnih grafova je vidljivo da smo po senzitivnosti bliži SW#-u na svakom skupu za prvih 5 poravnanja u odnosu na DIAMOND. Bitno je uočiti da dajemo bolje rezultate čak i u odnosu na DIAMOND sensitive način rada, ali također smo i za 3 skupa bolji od BLASTP-a za prvih 5 poravnanja, dok smo na 4. skupu zanemarivo ispod njega.

6.2. Testiranje brzine

Brzinu smo testirali na računalu sa sljedećim sklopovljem:

Intel(R) Xeon(R) CPU 2.40GHz x 24

200 GB radne memorije

U proračun smo uzeli vrijeme proteklo od pokretanja aplikacije do dobivanja rezultata. Za prikaz izlaznih rezultata odabran je "bm9" format, kojeg imaju implementirana

sva tri algoritma. U sljedećoj tablici prikazana je usporedba brzina našeg algoritma i fast verzija DIAMOND-a.

Tablica 6.2: Vrijeme u sekundama za 4 skupa podataka

Algoritam	*HumDiv*	*HumVar*	*Escherichia*	*Plasmodium*
Diamond-fast	1532	2058	1797	1928
Naš algoritam	416	1457	1037	640

Iz tablice je vidljivo minimalno ubrzanje od 1.41 puta na skupu HumVar, a dok je najveće ubrzanje postignuto na skupu HumDiv i iznosi 3.68. Uz ovakve vremenske rezultate i prethodno prikazanu osjetljivost poboljšanje je i više nego očigledno.

7. Zaključak

U ovom radu bavili smo se problemom pretrage baza podataka bioloških sljedova. Između determinističkog i heurističkog pristupa kojim se može obaviti pretraga baze u ovom radu odabran je heuristički algoritam. Heuristički algoritam donio je ubrzanje, a kao posljedica ubrzanja na izlazu je bio manji broj sljedova. U poglavlju 6 prikazana je usporedba rezultata dobivenih ispitivanjem ovog alata i rezultata dobivenih alatom BLASTP i DIAMOND. Iz njih se vidi ostvarenje značajnog ubrzanja u odnosu na alat BLASTP, ali i ubrzanje u odnosu na DIAMOND. Isto tako je vidljivo i poboljšanje u osjetljivosti u odnosu na oba alogoritma.

Značajno ubrzanje smo dobili implementirajući dva filtra koji obavljaju filtriranje nakon što su dobivene potencijalne sekvence na temelju skupa riječi. Radi se o filtrima koji filtriraju na temelju rasporeda riječi u ulaznom nizu i nizu iz baze i filtru koji propušta dalje samo one sekvence koje riječima pogađaju jednu od dijagonala barem 2 puta.

Osim metode opisane u poglavlju 4 pokušane su još neke metode koje se ipak nisu pokazale tako dobre. Kako bi poboljšali senzitivnost pokušali smo povećavati broj riječ i smanjivati duljinu riječi. Također, pokušali smo indekse tražiti i unutar regija određene duljine. Svaka od tih metoda je bila ili zanemarivo osjetljivija od metode opisane u 4 ili je bila čak i gora. Uz to, bila je i znatno sporija.

Rad je dostupan na internetu, na stranici `https://github.com/ivujevic/fast_alignment`.

8. Zahvale

Ovim putem zahvaljujem mentoru Mili Šikiću na trudu koji je uložio u protekle dvije-tri godine dok smo radili na ovom radu i promišljali kako ga unaprijediti. Zahvaljujem i svima drugima koji su na bilo koji način doprinijeli da ovaj rad ugleda svjetlo dana.

LITERATURA

[1] Warren John Ewens and Gregory R Grant. *Statistical methods in bioinformatics: an introduction*, volume 746867830. Springer, 2005.

[2] Neil C Jones and Pavel Pevzner. *An introduction to bioinformatics algorithms*. MIT press, 2004.

[3] Mile Šikić and Mirjana Domazet-Lošo. *Bioinformatika*. 2013.

[4] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Englewood Cliffs, 1995.

[5] Benjamin Buchfink, Chao Xie, and Daniel H Huson. Fast and sensitive protein alignment using diamond. *Nature methods*, 12(1):59–60, 2015.

[6] David W Mount. Sequence and genome analysis. *New York: Cold Spring*, 2004.

[7] Joshua Tan, Durga Kuchibhatla, Fernanda L Sirota, Westley A Sherman, Tobias Gattermayer, Chia Yee Kwoh, Frank Eisenhaber, Georg Schneider, and Sebastian Maurer-Stroh. Tachyon search speeds up retrieval of similar sequences by several orders of magnitude. *Bioinformatics*, 28(12):1645–1646, 2012.

[8] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[9] Paul Flicek, Ikhlak Ahmed, M Ridwan Amode, Daniel Barrell, Kathryn Beal, Simon Brent, Denise Carvalho-Silva, Peter Clapham, Guy Coates, Susan Fairley, et al. Ensembl 2013. *Nucleic acids research*, page gks1236, 2012.

[10] Naomi K Fox, Steven E Brenner, and John-Marc Chandonia. Scope: Structural classification of proteins—extended, integrating scop and astral data and classification of new structures. *Nucleic acids research*, 42(D1):D304–D309, 2014.

[11] Jin Xiong. *Essential bioinformatics*. Cambridge University Press, 2006.

[12] Robert Vaser, Dario Pavlović, Matija Korpar, and Mile Šikić. Sword-a highly efficient protein database search. *bioRxiv*, page 014654, 2015.

[13] Matija Korpar and Mile Šikić. Sw#–gpu-enabled exact alignments on genome scale. *Bioinformatics*, 29(19):2494–2495, 2013.

[14] Martin Šošić. OPAL-SIMD c/c++ library for massive optimal sequence alignment, 2015.

[15] The openmp® api specification for parallel programming.

[16] NCBI. `http://www.ncbi.nlm.nih.gov/blast/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=FAQ#LCR`, 2014.

Alat za brzo pretraživanje baza proteinskih sljedova

Sažetak

Kod pretrage baza bioloških sljedova postoje dvije vrste algoritama. Jedna vrsta su deterministički, koji koriste rigorozne algoritme za pronalazak optimalnog rješenja. Druga vrsta su heuristički algoritmi. To su algoritmi puno brži od determinističkih, ali ne daju uvijek optimalno rješenje. Potrebno je pronaći kompromis između brzine i točnosti pretrage.

U ovom radu opisan je heuristički algoritam pretrage. Na početku se za svaki slijed odredi lista riječi te se tako dobije smanjena baza. Smanjenje cijele baze na listu riječi omogućuje nam bržu pretragu. Algoritam se sastoji od koraka. U prvom koraku, pretragom liste riječi pronalaze se potencijalni kandidati koji se zatim šalju u drugi korak. U drugom koraku, koristeći jedan od algoritama za poravnanje odabiremo izlazne rezultate.

Ključne riječi: heuristički algoritam, bioinformatika, paralelizacija, biološke baze

Tool for fast searching of protein sequences in databases

Abstract

In database searching are two fundamental types of algorithm. One is deterministic type, which uses a rigorous algorithm to find the best solution. Another is the heuristic type which don't find optimal solution, but these algorithms are more faster than deterministic type algorithms. We have to find a compromise between speed and accuracy.

In this paper we describe heuristic type of algorithms. At the beginning, for each sequence we have to find word list to get reduced base as a result. Reducing the whole base allows us faster searching. Algorithm has two steps. In the first step we find potential candidate and in the second step we do selection between potential candidate using one of two offered algorithms.

Keywords: heuristic algorithm, bioinformatics, parallelization, biological database