SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3746

Alat za brzo pretraživanje baza bioloških sljedova

Ivan Vujević

Zagreb, lipanj 2014.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

Hvala mentoru Mili na pomoći i trudu koji je uložio prilikom izrade ovog rada!

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Značajka suvremenih bioloških istraživanja je velika količina podataka. Porastom količine podataka raste i potreba za računalnim metodama koje bi te podatke i 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.

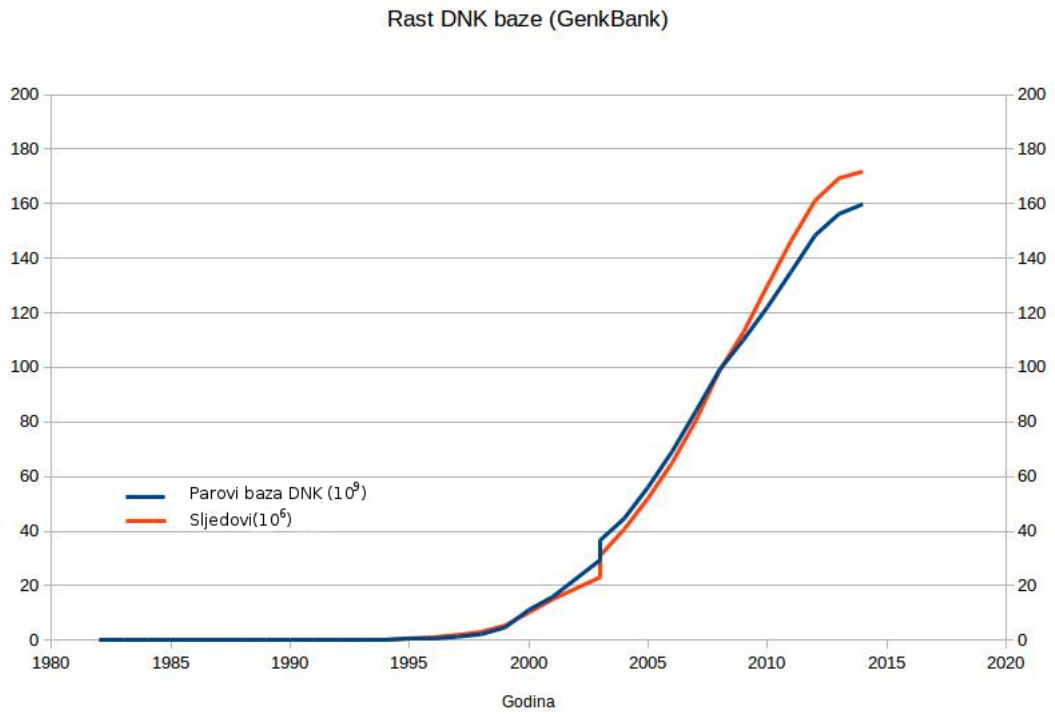
Poglavlje 4 govori o našem rješenju problema. Detaljnije je opisana problematika s kojom se susrećemo prilikom izrade alata te način na koji je dani problem riješen. Algoritam koji koristimo u radu je opisan po koracima.

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 je to iznio 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.

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

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.

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][8].
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.[8]

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 točnost. Ispitivanja su pokazala da BLAST algoritam u prosjeku daje 30% manje izlaznih sljedova u odnosu na stvaran broj podudaranja u bazi[5]. 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 usporedbu bioloških sljedova. Objavili su ga 1970. godine Saul Needleman i Christian

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

Wunsch.[3]

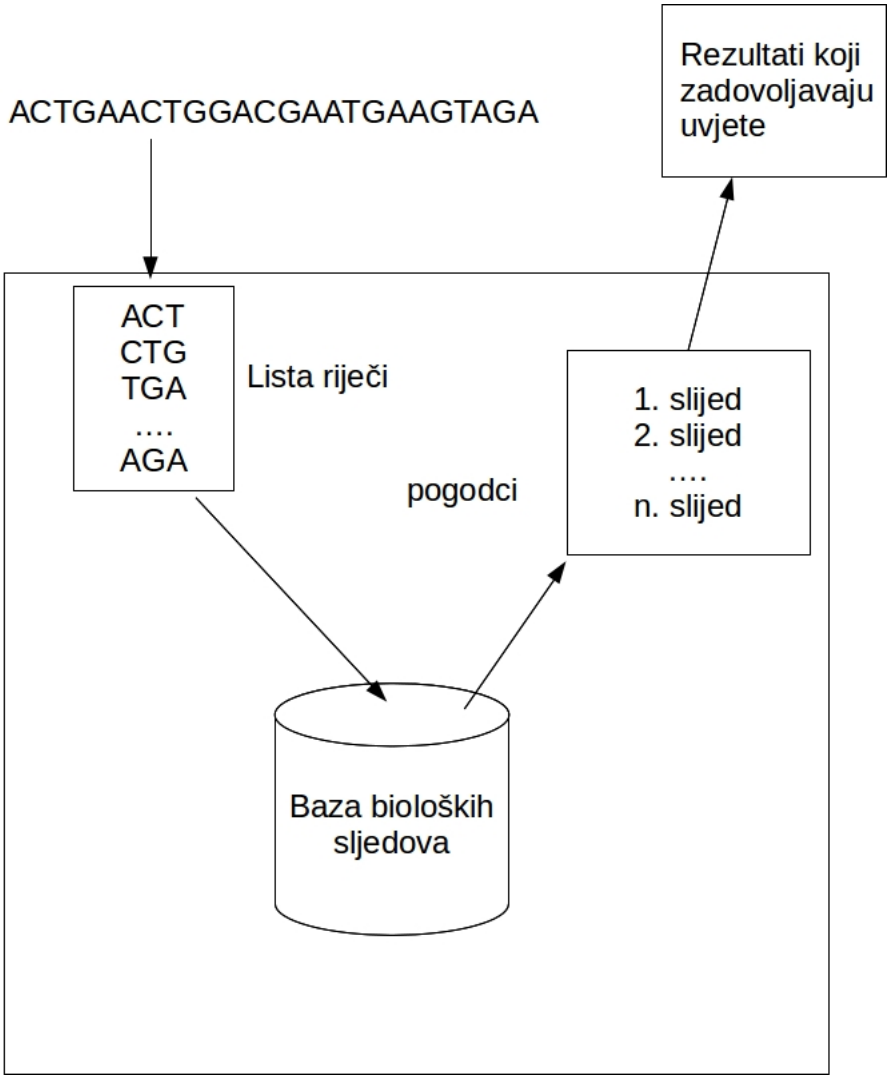
Poravnanjem dvaju sljedova cilj je postići da se što više elemenata (aminokiselina kod proteinskih i nukleotida kod nukleotidnih sljedova) iz jednog slijeda poklopi s elementima iz drugog slijeda. Kod lokalnog poravnanja važno je pronaći regije čije će poravnanje imati najveći rezultat. Kada promatramo 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.word)* 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 retka 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. Kao izlaz, alat nudi dvije mogućnosti za biranje. Jedna mogućnost za pojedini rezultat je ispis opisa i odgovarajućeg slijeda, a druga ispis samo opisa. Kako bi izlaz bio što pregledniji, za pojedini upit se stvara nova datoteka u koju se zapisuju rezultati. Izlazna datoteka je oblika *identifikatorSlijeda.out*.

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 tom 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. 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. Točnost pretrage možemo odrediti podešavajući duljinu riječi te broj riječi koje određuju pojedini slijed. 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.[9] Također, kod brojanja frekvencija pojavljivanja potrebno je paziti da ne brojimo iste preklapajuće riječi.

Metoda koja se koristi u ovom radu sastoji se od dva koraka koji služe za pronalaženje sljedova. 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 dva ponuđena algoritma. Algoritmi koji se koriste u drugom koraku detaljnije će biti opisani u nastavku ovog poglavlja. Za početak spomenimo samo da prvi algoritam uspoređuje dva slijeda na temelju sličnosti riječi koje se u njima pojavljuju dok drugi algoritam za usporedbu sličnosti koristi E-vrijednost kao i alat BLAST.

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. Parametri koji će se koristiti za duljinu riječi i broj riječi koje predstavljaju pojedini slijed ovise o tome radi li se o slijedu proteina ili o slijedu nukleotida. Ispitivanja za proteinske sljedove pokazuju da je kao predstavnike pojedinih sljedova najbolje odabrati **pet** riječi duljine **pet**. [6] Ukoliko se pretražuje baza nukleotidnih sljedova potrebno je odabrati **sedam** riječi duljine **petnaest**.

Razlog zbog kojeg je kod nukleotidnih sljedova potrebno odabrati drugačije parametre u odnosu na proteinske sljedove proizlazi iz činjenice da je vjerojatnost pojavljivanja određenog nukleotida puno veća nego vjerojatnost pojavljivanja određenog proteina, čime se povećava vjerojatnost pojavljivanja određene riječi pa stoga riječ koja ima veliku vjerojatnost pojavljivanja u svakom slijedu ne može biti predstavnik određenog slijeda. Uzmemo li pojednostavljeni matematički model u kojem je jednaka vjerojatnost pojavljivanja svakog nukleotida u bazi nukleotidnih sljedova, odnosno proteina u bazi proteinskih sljedova zaključujemo da je vjerojatnost pojavljivanja jednog nukleotida $\frac{\frac{1}{5}}{\frac{1}{21}} = 4.2$ puta veća od vjerojatnost pojavljivanja jednog proteina. [1]

Algoritam 2: Prvi korak algoritma

Data: Kod proteina $m = 3$, a kod nukleotida $m = 4$

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

Brojač kod svakog pogođenog slijeda povećaj za broj pojavljivanja riječi w među predstavnicima danog slijeda.

Ostavi one sljedove kojima je brojač veći ili jednak m

[1] Broju različitih nukleotida i aminokiselina dodajemo po jedan element koji u bazi označava neprirodne regije.

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 dva načina. Jedan algoritam uzima u obzir samo riječi koje se pojavljuju u oba slijeda, dok drugi u obzir uzima E-vrijednosti izračunatu na temelju rezultata Smith - Waterman poravnanja sljedova.

Metoda koja sličnost računa na temelju riječi iz oba slijeda radi na sljedeći način:

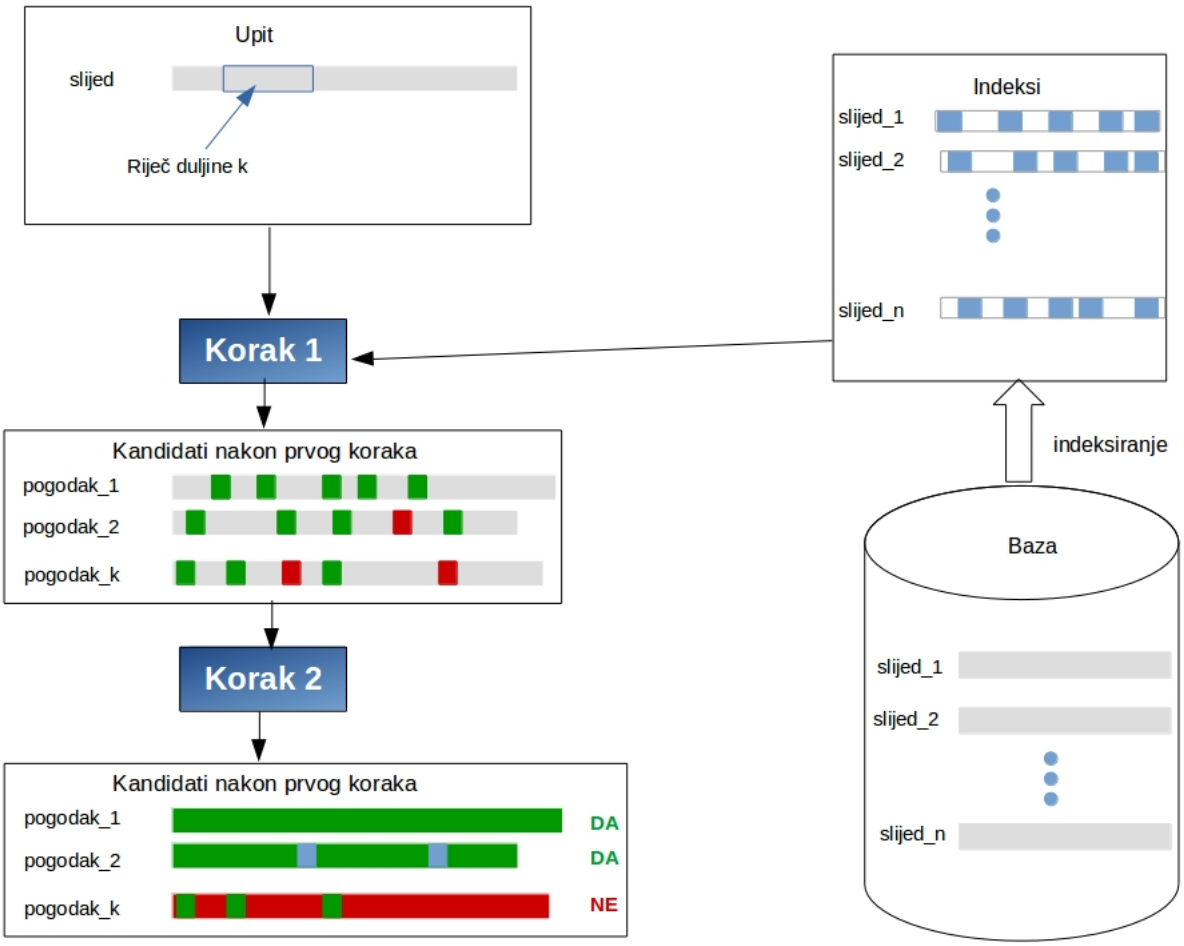
Algoritam 3: Računanje sličnosti

$ukupno$ = broj različitih riječi duljine k koje se pojavljuju u dva slijeda

$brojZajednicko$ = broj različitih riječi duljine k koje se pojavljuju i u jednom i u drugom slijedu

$rezultat = \frac{brojZajednicko}{ukupno}$

Nakon što se algoritmom 3 dobije sličnost između dva slijeda, rezultatom pretrage smatraju se oni sljedovi koji imaju rezultat sličnost veći ili jednak od definiranog praga. Na slici 4.1 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 4.1: Prikaz rada algoritma

4.2.1. E - vrijednost

Računanje E-vrijednost je metoda koju koristi alat BLAST nakon što dobije listu slijedova koji bi mogli odgovarati slijedu iz upita. 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. Pošto 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. 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. U ovom radu, granica E-vrijednosti je 0.05

5. Implementacija

Alat je napisan u programskom jeziku C++ kao višedretveni program. Broj dretvi koje su pokrenute prilagođen je sklopovlju na kojem se program izvodi. U sustavu postoji glavna dretva koja učitava upite iz datoteke koja mora biti u fasta formatu kao i baza koja se indeksirala te ostale dretve koje obrađuju podatke. Iako se prirodno nameće rješenje paralelizacije prvog i drugog koraka algoritma, ispitivanja su pokazala da to ipak nije optimalno rješenje. Veću iskoristivost procesora te samim time i veću brzinu alata dobivamo ukoliko jedna dretva izvodi cijeli algoritam, a različitim dretvama se daju različiti sljedovi iz upita.

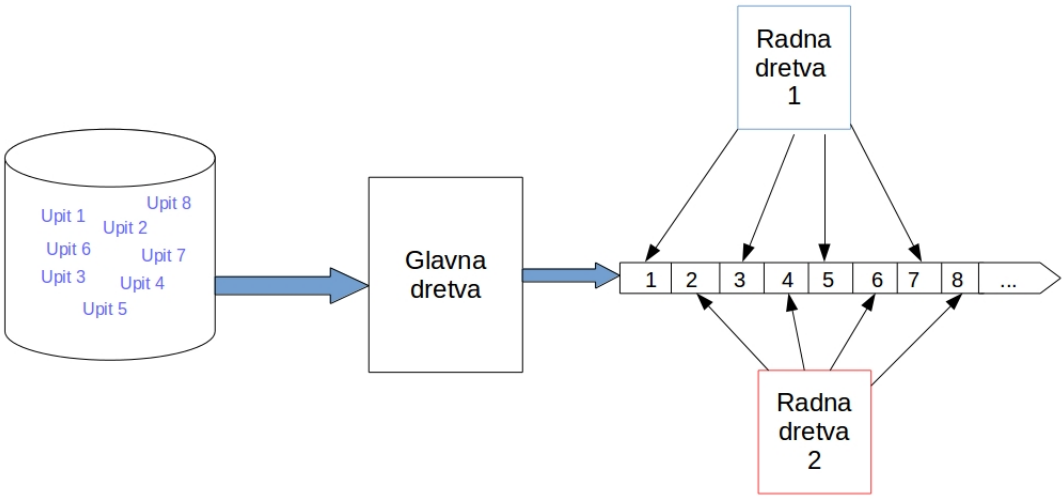
5.1. Raspodjela posla

Razmjena sljedova između glavne dretve i ostalih dretvi koje obrađuju te sljedove ostvarena je koristeći problem proizvođača i potrošača.[1]

Na slici 5.1 prikazana je razmjena sljedova između glavne dretve koja je „proizvođač" i ostalih dretvi koje su „potrošači". Za razliku od osnovnog problema jednog proizvođača i jednog potrošača ovdje imamo situaciju da postoji više potrošača. Uklonit ćemo ograničenje u kojem je spremnik ograničene duljine te pretpostaviti da dretva proizvođača može nesmetano dodavati sljedove u spremnik. Problem koji moramo riješiti je problem pristupa spremniku. Ne smije se dogoditi slučaj u kojem dvije dretve uzimaju isti slijed iz spremnika i kreću ga obrađivati. Ovaj se problem može riješiti na dva načina: jedan je uvođenjem semafora koji će regulirati pristup spremniku, a drugi je način taj da svakoj dretvi dodijelimo određeni dio spremnika u kojem samo ona ima pravo pristupa. Ako bi se odlučili na rješenje sa semaforima dolazimo do

[1] Problem proizvođača i potrošača klasičan je primjer višedretvenog sinkronizacijskog problema. Problem opisuje dva procesora, proizvođač i potrošač, koji koriste ograničen zajednički spremnik. Proizvođač proizvodi niz proizvoda koje stavlja u zajednički spremnik, a te proizvode potrošač troši u jednom trenutku. Problem nastaje u trenutku kada je spremnik pun,a proizvođač želi dodati proizvod, odnosno, u trenutku kada je spremnik prazan, a potrošač želi uzeti proizvod.

problema kojeg uzrokuje sinkronizacija, odnosno do problema u kojem ostale dretve moraju čekati dok jedna dretva uzima slijed za obradu. Zamislimo li situaciju u kojoj u sustavu imamo k dretvi koje čekaju na preuzimanje novog zadatka, u spremniku n sljedova pri čemu je $k < n$, dolazimo do situacije gdje je $k - 1$ dretva blokirana i ne može preuzeti svoj zadatak dok samo jedna dretva preuzima. Navedena situacija ne bi bila kritična u slučaju da se dogodi samo jednom, no ukoliko imamo velik broj upita tada se navedeni događaj ponavlja iz koraka u korak i dolazimo do velikog broja blokiranja čime se bespotrebno troši vrijeme. Drugi način na koji ovaj problem možemo riješiti je taj da svakoj dretvi dodijelimo prostor koji je samo njen te kojem samo ona ima pravo pristupa. Naravno, pravo pristupa ima i proizvođač koji dodaje zadatke u spremnik. Ukoliko bismo unaprijed znali da je u upitu n sljedova te ako u sustavu imamo k dretvi mogli bismo svakoj dretvi dodijeliti dio spremnika veličine $\frac{n}{k}$. Pošto na početku imamo nepoznat broj upita, prostor je dretvama potrebno dinamički dodjeljivati. Kako u sustavu imamo k dretvi, svaka će dretva imati pravo pristupa svako k-tom elementu spremnika od onog koji joj je početno dodijeljen. Na početku će se prvoj dretvi dodijeliti prvi element, drugoj drugi itd. Iz toga slijedi da će prva dretva nakon što obradi prvi slijed iz upita, pristupati $k + 1$ elementu, druga $k + 2$ elementu itd. Naravno u tom slučaju ostaje problem u kojem dretva potrošača želi uzeti proizvod kojeg dretva proizvođača još uvijek nije proizvela. To ćemo riješiti uvođenjem radnog čekanja pojedinoj dretvi dok ne bude spreman njen zadatak. Pošto se čitanje upita i dodavanje upita u spremnik obavlja puno brže od obrade jednog upita radno čekanje će se događati vrlo rijetko, gotovo nikada.



Slika 5.1: Prikazana je komunikacija dretvi u sustavu koji se sastoji od dvije radne dretve. Glavna dretva učitava upite koje zatim sprema u spremnik iz kojeg čitaju dvije dretve.

5.2. Organizacija sustava

U nastavku je prikazan dijagram razreda te dani kratki opisi pojedinih razreda. Metode u dijagramu nisu navedene radi preglednosti te je njihov opis dostupan u kodu.

Base - Apstraktni razred iz kojeg se kasnije izvodi baza proteinskih i nukleotidnih sljedova. Sustav za vrijeme izvođenja komunicira s bazom koja je ovog tipa, ali ima referencu na bazu proteinskih ili nukleotidnih sljedova ovisno o parametrima kojima je sustav pokrenut. Prilikom inicijalizacije elementa koji je ovog tipa potrebno je predati parametre baze, odnosno potrebno je predati putanju do baze u kojoj su pohranjeni sljedovi, do datoteke u kojoj je zapisana reducirana baza (njeni indeksi), te do datoteke u kojoj je zapisan broj pojavljivanja pojedine riječi. Ovaj tip nudi metode za indeksiranje baze te čuva zapis baze i potrebnih indeksa u memoriji.

BaseP - Razred koji je izveden iz razreda *Base* te predstavlja bazu proteinskih slje-

dova. Dotatno je implementirana metoda za indeksiranje baze proteinskih sljedova.

BaseN - Razred koji je izveden iz razreda *Base* te predstavlja bazu nukleotidnih sljedova. Dotatno je implementirana metoda za indeksiranje baze nukleotidnih sljedova.

FindIndex - Apstraktni razred koji ima ulogu pronalaska riječi koje određuje pojedini slijed. Ovisno o tipu sljedova koji se nalaze u bazi iz njega se izvodi razred FindIndexP i razred FindIndexN. Kao što se vidi na dijagramu razreda referenca na objekt ovog tipa nalazi se u tipu Base. Objekt tipa Base će nakon što ima poznate duljinu svih riječi i ako ne postoji datoteka sa zapisanim indeksima, za svaki slijed iz baze pozvati odgovarajuće metode iz ovog razreda koje će obraditi taj slijed i vratiti riječi koje određuju taj slijed.

FindIndexP - Razred koji je izveden iz razreda *FindIndex* te služi za pronalazk riječi koje određuju proteinski slijed.

FindIndexN - Razred koji je izveden iz razreda *FindIndex* te služi za pronalazk riječi koje određuju nukleotidni slijed.

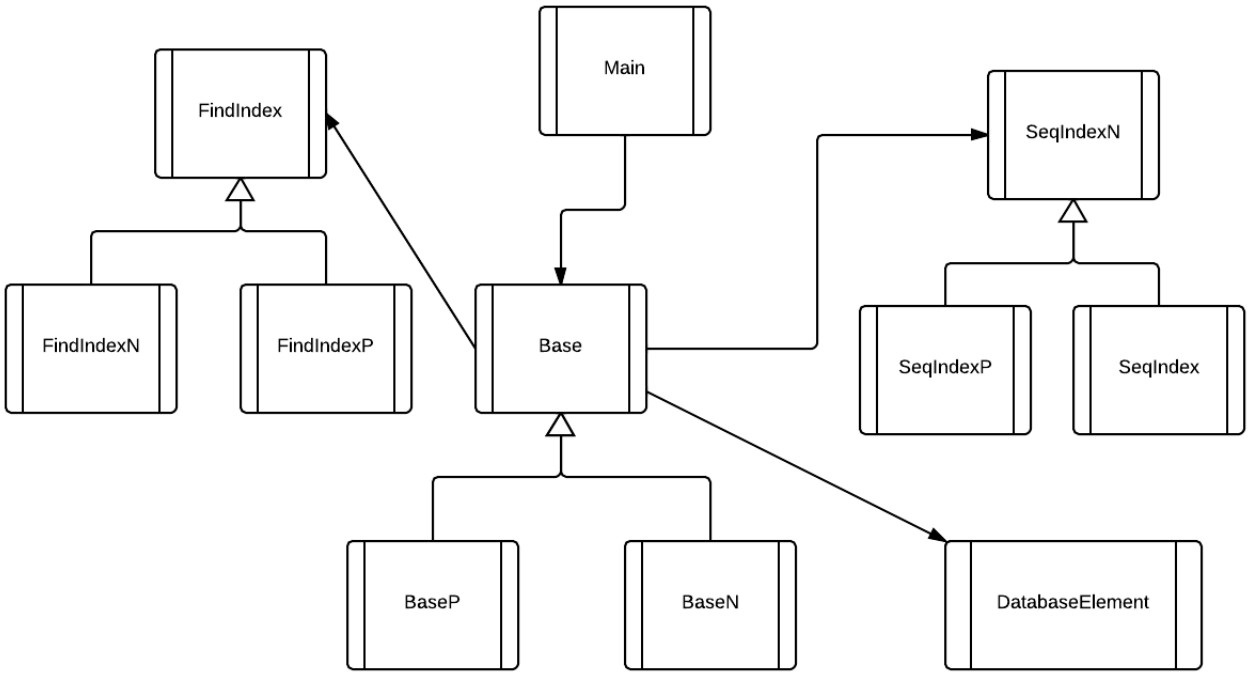
SeqIndex - Apstraktni razred koji predstavlja riječi kojima je određen pojedini slijed. Ovisno u vrsti slijed iz ovog razreda se izvodi razred *SeqIndexP* i razred *SeqIndexN*.

SeqIndexP - Razred koji je izveden iz razreda *SeqIndex* te označuje riječi koje predstavljaju određeni proteinski slijed.

SeqIndexN - Razred koji je izveden iz razreda *SeqIndex* te označuje riječi koje predstavljaju određeni slijed nukleotida.

DatabaseElement - Razred koji predstavlja jedan zapis iz datoteke *fasta* formata. Jedan zapis u fasta formatu se sastoji od opisa koji počinje znakom „>”, te slijeda nukleotida ili proteina.

Util - definira konstante koje se koriste u sustavu.



Slika 5.2: Dijagram razreda od kojih je alat sastavljen

6. Rezultati

6.1. Sljedovi proteina

Za ispitivanje alata korištena je NR baza proteinskih sljedova koja se može preuzeti s adrese *ftp://ftp.ncbi.nlm.nih.gov/blast/db/FASTA/nr.gz*. U trenutku ispitivanja u bazi je bilo 34927437 sljedova. Napravljena su ispitivanja sljedova različitih duljina i sljedova koji potječu od različitih vrsta.

Na slici 6.1 prikazana je frekvencija pojavljivanja sljedova određenih duljina u bazi. Kao što se sa slike može vidjeti, u bazi su najzastupljeniji sljedovi koji imaju duljinu približno 200.

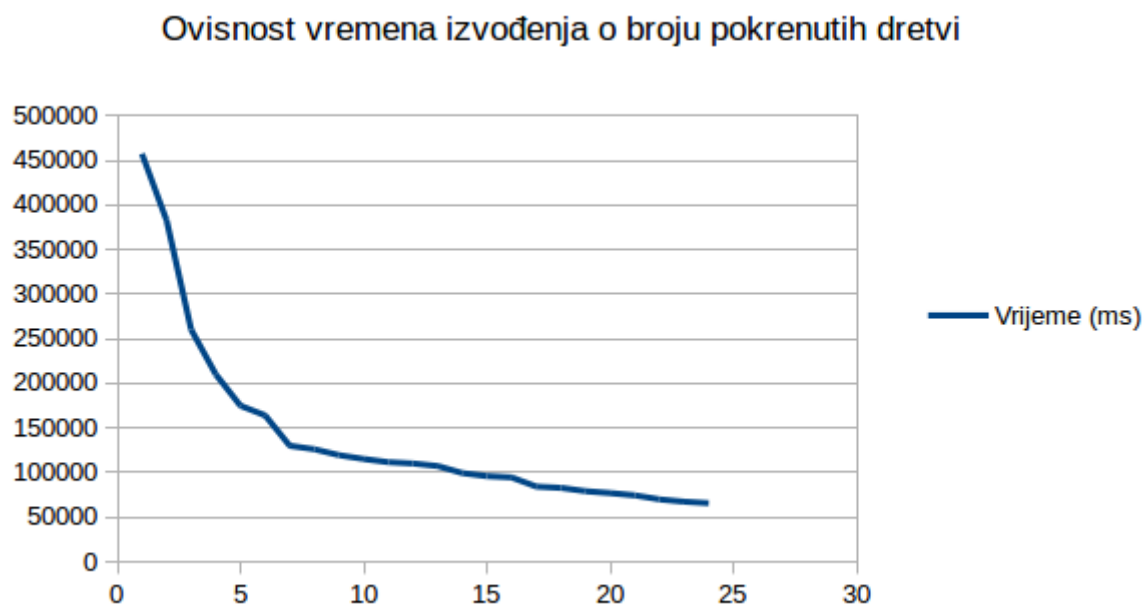
Slika 6.1: Frekvencija pojavljivanja sljedova određene duljine.

Prije pretrage bilo je potrebno provesti indeksiranje baze. Indeksiranje NR baze trajalo je približno 12 sati. Vrijeme indeksiranja, ali i pretrage baze ovisi najviše o sklopovlju na kojem se izvodi. Pošto se radi o paraleliziranom algoritmu, na vrijeme izvođenja najveći utjecaj će imati broj dostupnih procesora. Sklopovlje korišteno prilikom ispitivanja je sljedeće:

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

200 GB radne memorije

Na slici 6.2 prikazana je vremenska ovisnost izvođenja algoritma o maksimalnom broju kreiranih dretvi koje se mogu istodobno izvoditi. Ispitivanje ovisnost provedeno je na istom skupu kao i ispitivanja koja će biti prikazana u nastavku rada.



Slika 6.2: Prikazana je eksponencijalna ovisnost vremena izvođenja o broju pokrenutih dretvi. Na **x** osi prikazan je broj dretvi, a na **y** osi ukupno vrijeme izvođenja. Izgled grafa može varirati ovisno o upitima, no ovisnost vremena o broju dretvi će uvijek biti eksponencijalna.

6.1.1. Ispitni skup

U ispitni skup odabrano je 426 slučajno odabranih sljedova iz baze podataka pazeći pri tome da razdioba duljina u ispitnom skupu bude kao i u bazi. Ispitivanje je provedeno na praleliziranoj i neparalleliziranoj verziji algoritma te su dobiveni rezultati uspoređeni s rezultatima koje daje alat BLAST. Kod uspoređivanja u obzir je uzeta brzina

izvođenja algoritma te broj pogodaka u bazi. Također, kako bi se ostvario što efikasniji algoritam, bilo je potrebno naći što bolji omjer između pogodaka u prvom i drugom koraku. U sljedećim tablicama prikazani su rezultati dobiveni ispitivanjem, nakon čega je napisana i analiza dobivenih rezultata.

Broj pogodaka			
Prvi korak	Drugi korak	Duljina	Vrijeme (s)
801.85	500.13	812.17	443

Tablica 6.1: Prikazani su rezultati neparalelizirane verzije algoritma za skup od 426 sljedova.

Broj pogodaka			
Prvi korak	Drugi korak	Duljina	Vrijeme (s)
801.85	500.13	812.17	65

Tablica 6.2: Prikazani su rezultati paralelizirane verzije algoritma za skup od 426 sljedova.

U tablici 6.1 prikazani su rezultati dobiveni pokretanjem programa u neparaleliziranoj verziji. Postoji samo jedna dretva koja obrađuje podatke. U prvom i drugom stupcu prikazan je broj pogodaka nakon prvog, odnosno drugog koraka. Broj pogodaka nakon drugog koraka je broj sljedova koji su rezultat upita. U trećem stupcu prikazana je prosječna duljina sljedova iz upita, a u zadnjem stupcu prikazano je ukupno vrijeme pretrage. Rezultati prvog i drugog koraka su očekivano isti kao i u paraleliziranoj verziji prikazanoj tablicom 6.2 no vidljiva je razlika u vremenu. Iz prikazanih tablica vidljiva je osjetna razlika u vremenskom izvođenju paralelizirane i neparalelizirane verzije. U ovom slučaju paralelizirana verzija brža je za više od 6 puta od neparalelizirane verzije. Na slici 6.2 dobro se uočava eksponencijalna ovisnost vremena izvođenja i broja pokrenutih dretvi.
U sljedećoj tablici prikazani su rezultati metode koja prilikom određivanja liste riječi u obzir uzima duljinu slijeda.

Broj pogodaka			
Prvi korak	Drugi korak	Duljina	Vrijeme (s)
500	430	812.17	67

Tablica 6.3: Priazani su rezultati metode kojoj broj riječi ovisi o duljini slijeda.

Sljedovi su podijeljeni u tri kategorije na temelju svoje duljine. Cilj je bio da broj sljedova u svakoj kategoriji bude približno jednak. Sljedovima koji su se nalazili u prvoj kategoriji pridijeljena je lista od 5 riječi, onima u drugoj kategoriji lista od 7 riječi, a onima u trećoj kategoriji lista od 8 riječi. Za listu od 5 riječi traženo je minimalno 3 pogotka, za listu od 7 riječi 4 pogotka, a za listu od 8 riječi 5 pogodaka. Iako je metoda imala bolji omjer prvog i drugo koraka, odbačena je iz razloga što je na izlazu imala manji broj sljedova u odnosu na metodu opisanu u ovom radu.

6.1.2. Usporedba rezultata s alatom BLAST

Kao alat s kojim će se usporediti dobiveni rezultati odabran je jedan od najpoznatijih alata koji koriste heuristički pristup rješavanja problema, a to je alat BLAST. Za ispitivanje su odabrana dva slijeda iz NR baze prikazana u tablici 6.4

Oznaka	Porijeklo	Duljina
5031611	Čovjek	118
20162566	Čovjek	179

Tablica 6.4: Sljedovi proteina za usporedbu.

Alat BLAST pokrenut je u svojoj paraleliziranoj verziji koristeći 24 dretve. Vrijeme dobiveno alatom BLAST je 436 sekundi, a vrijeme dobiveno alatom opisanim u ovom radu je 0.58 sekundi. Ukupan broj pogodaka prikazan je u sljedećoj tablici.

Slijed	BLAST	Alat iz ovog rada
5031611	500	86
20162566	80	51

Tablica 6.5: Prikazana je usporedba rezultata dobivenih alatom BLAST i alatom ostvarenim u ovom radu.

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 BLAST. Iz njih se vidi ostvarenje značajnog ubrzanja u odnosu na alat BLAST. Osim metode opisane u poglavlju 4 pokušane su još neke metode koje se ipak nisu pokazale tako dobre. Jedna od metoda nije imala fiksno određeni prag od 3 pogotka kao niti fiksno određeni broj riječi već je veličine prilagođavala ovisno o duljini slijeda. Ta metoda je odbačena iz razloga što je davala manje izlaznih sljedova u odnosu na ovdje opisanu metodu. Druga metoda koja se pokušala ostavariti ima princip rada isti kao i metoda opisana u ovom radu s razlikom što za strukturu podataka ne koristi tablicu raspršenog adresiranja već polje kojim se pokušao ostvariti pristup elementu u složenosti $O(1)$. Iako bi teoretski ova metoda trebala davati brže rezultate, pokazala se nepovoljnom zbog razlike u pohrani elemenata polja i tablice raspršenog adresiranja u memoriju procesora. Ako u polju podataka želimo pristupiti elementu na
proizvoljnom mjestu koji se ne nalazi u memoriji procesora tada će se u memoriju učitati svi elementi od početnog do traženog, čime se značajno usporava izvođenje algoritma.

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

[6] 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.

[7] Matija Korpar. Implementacija smith waterman algoritma koristeći grafičke kartice s cuda arhitekturom. 2011.

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

[9] 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 bioloških 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 koji ne pronalaze optimalno rješenje, ali ovi algoritmi su puno brži od determinističkih algoritama. 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 dva koraka. U prvom koraku, pretragom liste riječi pronalaze se potencijalni kandidati koji se zatim šalju u drugi korak. U drugom koraku, koristeći jedan od dva algoritma, odabiremo sljedove za izlaz.

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 fastier 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