SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5752

Brzo pretraživanje sličnih proteinskih sljedova

Dario Babojelić

Zagreb, lipanj 2018.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
ODBOR ZA ZAVRŠNI RAD MODULA

Zagreb, 15. ožujka 2018.

ZAVRŠNI ZADATAK br. 5752

Pristupnik: **Dario Babojelić (0036491127)**
Studij: Računarstvo
Modul: Računarska znanost

Zadatak: **Brzo pretraživanje sličnih proteinskih sljedova**

Opis zadatka:

Pretraživanje proteinskih baza podataka je vremenski zahtjevan zadatak. U zadnjih 10 godina broj novo pronađenih sljedova brzo raste i time nastaju novi izazovi za algoritme za pretraživanje. Algoritme u području moguće je podijeliti u one kojima je naglasak na brzini i one kojima je naglasak na osjetljivosti. U ovom radu naglasak je brzom pretraživanju. Potrebno je razviti rješenje koje će brzo pretraživati bazu na način da se u prvom koraku heurističkom metodom pronađe manji podskup kandidatnih sljedova, dok se u drugom koraku računaju poravnanja koristeći postojeći alat za optimalno poravnanje sljedova OPAL. Za određivanje kandidatnih sljedova koristiti podnizove duljine k koji imaju minimalnu leksikografsku vrijednost (engl. minimizer). Za povećanje osjetljivosti pretrage koristiti reduciranu abecedu aminokiselina.
Rješenje treba biti napisano u programskom jeziku C/C++. Programski kod je potrebno komentirati i pri pisanju pratiti neki od standardnih stilova. Kompletnu aplikaciju postaviti na repozitorij Github.

Zadatak uručen pristupniku: 16. ožujka 2018.
Rok za predaju rada: 15. lipnja 2018.

Mentor:
Izv. prof. dr. sc. Mile Šikić

Djelovođa:
Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za završni rad modula:
Prof. dr. sc. Siniša Srbljić

Hvala mentoru Mili na pomoći tijekom izrade ovog rada!

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Tehnološkim napretkom naglo je porasla količina podataka u istraživačkoj biologiji. To je dovelo do potrebe za sve složenijim računalnim rješenjima koja mogu te podatke obrađivati. Pretraga baze proteina jedan je od glavnih problema bioinformatike. Današnje baze proteina mogu sadržavati milijune proteinskih sljedova što odgovara nekoliko desetaka GB[1]. To čini pretraživanje takvih baza vremenski vrlo zahtjevnim zadatkom.

Cilj ovog rada je razviti programsko rješenje koje će vremenski efikasno pretraživati bazu. Iako je i točnost pretrage bitna, prednost dajemo brzom pretraživanju baze.

1.1. Opis problema

Gradivni dijelovi proteina su aminokiseline. Svaku aminokiselinu označavamo jednim slovom, dakle protein promatramo kao niz slova. Duljina proteina varira između svega

Tablica 1.1: Aminokiseline

alanin - **A**	arginin - **R**	asparagin - **N**	asparaginska kiselina - **D**
cistein - **C**	glutamin - **Q**	glutaminska kiselina - **E**	glicin - **G**
histidin - **H**	izoleucin - **I**	leucin - **L**	lizin - **K**
metionin - **M**	fenilalanin - **F**	prolin - **P**	serin - **S**
treonin - **T**	triptofan - **W**	tirozin - **Y**	valin - **V**

desetak pa sve do nekoliko desetaka tisuća aminokiselina. Najčešće proteini sadrže nekoliko stotina aminokiselina.

Standardna reprezentacija proteina je u FASTA formatu. To je jednostavan tekstualni format u koji možemo zapisati jedan ili više proteina. Svaki protein zapisuje se u dva retka. Prvi redak započinje znakom '>' iza kojeg slijedi opis proteina. U drugom

[1] Primjer je NCBI NR baza koja sadrži preko 54 000 000 proteina i veličine je 32 GB

```
>sp|Q6GZV1|024R_FRG3G Uncharacterized protein 024R OS=Frog virus 3
MWQYLPILLMTMISQLEWTVAAVKRYPAGGFITGDKLSRVFEALPWRVAVVSDEPEKYEG
FPILTEEDPAVFEDADCILFAVSDPKCVTGAMKSVFMASSKTAWVVYDGTETRATVRSWM
RRLWRAETYVPLLTHRGFVTDVCVYSQPDSERYVSVMTATAHFYSNRLEVLEEMAFVPHL
AYAKLAMGRYTVLDGCMSVKGSADVAPLNRSMWFLTAAAIPHGEIDTDSLFSDPGAVYSC
GSALREALGSLPEGSTSVVAVRNSSYRKYVRGILGPNFRVETFTNVVKTWGVYDYVLLPM
GISDSYKQGRDLMEKLEMPGGHRVVTFAPENYTVNEVHLNRPLKYAIKRMDLITPMVLRH
VSLNK
>sp|Q197D5|025R_IIV3 Uncharacterized protein 025R OS=Invertebrate iridescent virus 3
MNYSVIWAITILILGLVLTLAWARQNPTHPINPLVLNYHTKPSPKRHRMVLVVESFASVD
ALVELVENILSQTIRVASITVVSQRPDHLRQVPLLHQTCTFSRASGLSALFKETSGTLVV
FISKEGFHHFQSPTLLETIDQRGVTAEQTLPGIVLRNTDMPGIDLTTVYRQQRLGLGN
```

Slika 1.1: Primjer FASTA datoteke

retku nalazi se niz aminokiselina koje grade taj protein. Preporučeno je da drugi redak zapisa bude razlomljen na više redaka od po 80 znakova. Na slici 1.1 prikazan je primjer FASTA datoteke koja sadrži dva proteina.

Cilj je napraviti alat koji će obrađivati dvije FASTA datoteke. Prva datoteka sadrži bazu proteina. U drugoj datoteci su proteini za koje vršimo upite u bazu. Zadatak je za svaki protein iz druge datoteke pronaći najsličnije njemu u bazi. Postavlja se pitanje što znači sličnost dvaju sljedova. Sličnost se može definirati na razne načine, a možda najjednostavniji primjer bi bio Levenshteinova udaljenost nizova. Tada bi u bazi željeli pronaći proteinske sljedove koji imaju najmanju Levenshteinovu udaljenost s upitnim proteinom. Za pronađene slične nizove želimo odrediti i optimalno poravnanje. Na slici 1.2 prikazano je optimalno poravnanje dva proteina. Sada

```
M V Q Y L M A P A G - - - G F I T A G
M V - - L M A P A G D E C G F I W A G
+ + - - + + + + + + - - - + + + - + +
```

Slika 1.2: Optimalno poravnanje

umjesto Levenshteinove udaljenosti možemo koristiti sličnost koju računamo iz poravnanja. Možemo to pokazati na jednostavnom primjeru. Svaku identičnu aminokiselinu na odgovarajućoj poziciji nagradit ćemo s 5 bodova. Svako otvaranje pukotine unutar proteina penalizirat ćemo s 10 bodova. Svaki nastavak pukotine penalizirat ćemo s 1 bodom. Različite aminokiselina na odgovarajućim pozicijama penalizirat ćemo s 3 boda. U takvom sustavu bodovanja poravnanje sa slike 1.2 ostvarilo bi $5+5-10-1+5+5+5+5+5+5-10-1-1+5+5+5-3+5+5 = 39$ bodova. Više bodova znači i bolje poravnanje. Takav način računanja sličnosti prilagođeniji je sljedovima proteina. Sličan sustav bodovanja koristi algoritam Smith-Waterman [7] koji ćemo koristiti u rješenju.

1.2. Postojeća rješenja

Promotrimo li pretraživanje nekoliko tisuća upita u bazama današnje veličine uviđamo vremensku zahtjevnost ovog problema. Zbog toga su kroz zadnje desetljeće heuristička rješenja prevladala nad onima koja se baziraju na deterministčkim algoritmima. Deterministički algoritmi se danas jako rijetko koriste kao samostalna rješenja zbog vremenske neefikasnosti.

Primjer determinističkog algoritma je Smith-Waterman[7] koji za dva slijeda duljine n i m u vremenskoj složenosti $\mathcal{O}(nm)$ pronalazi optimalno lokalno poravnanje. Lokalno znači da je algoritmu cilj pronaći regije čije će poravnanje dati najbolji rezultat. Iz njegove složenosti vidljivo je da algoritam ne može biti samostalno rješenje za današnje veličine baza proteina.

Primjer heurističkog rješenja i najpopularniji alat za pretragu baze proteina je BLAST (engl. *Basic Local Alignment Search Tool*) [2]. BLAST znatno povećava brzinu izvođenja tako da najprije smanji prostor pretraživanja. Umjesto usporedbe slijeda upita sa svakim slijedom iz baze, BLAST koristi kratke podnizove (engl. *word*) koji služe kao sjeme (engl. *seed*) u daljnjem poravnanju (engl. *seed-and-extend approach*). Pronađena poravnanja rangira prema mjeri koja se naziva *E-vrijednost* (engl. *Expect(E) value*)[5]. To je statistička mjera koja za neko poravnanje predstavlja očekivani broj istih takvih ili boljih poravnanja u bazi. Što je ta mjera manja, poravnanje je značajnije. Izraz za izračun *E-vrijednosti* nekog poravnanja je:

$$E = Kmne^{-\lambda S}$$

gdje su m i n duljine sljedova koji daju to poravnanje, S je broj bodova koje to poravnanje ostvaruje u sustavu sličnom onome iz kraja prethodnog poglavlja, a K i λ su konstante koje BLAST određuje iz svojstava baze s kojom trenutno radi. Iz formule vidimo da je poravnanje to značajnije što je brojčana vrijednost poravnanja S veća i što su duljine sljedova n i m manje. Ti parametri ne utječu jednako. Porastom S *E-vrijednost* opada eksponencijalno, dok smanjivanjem duljina n i m opada linearno.

1.3. Opis rješenja

Rješenje koje ćemo razviti će raditi u dva koraka.

U prvom koraku ćemo heurističkom metodom smanjiti prostor pretraživanja. Od svih proteina u bazi izdvojit ćemo kandidatne sljedove koji bi mogli biti slični upitnom proteinu. Za određivanje kandidatnih sljedova koristit ćemo podnizove duljine k koji

imaju minimalnu leksikografsku vrijednost (engl. *minimizer*). Dodatno ćemo povećati osjetljivost pretrage korištenjem reducirane abecede aminokiselina. O svemu tome više u sljedećem poglavlju.

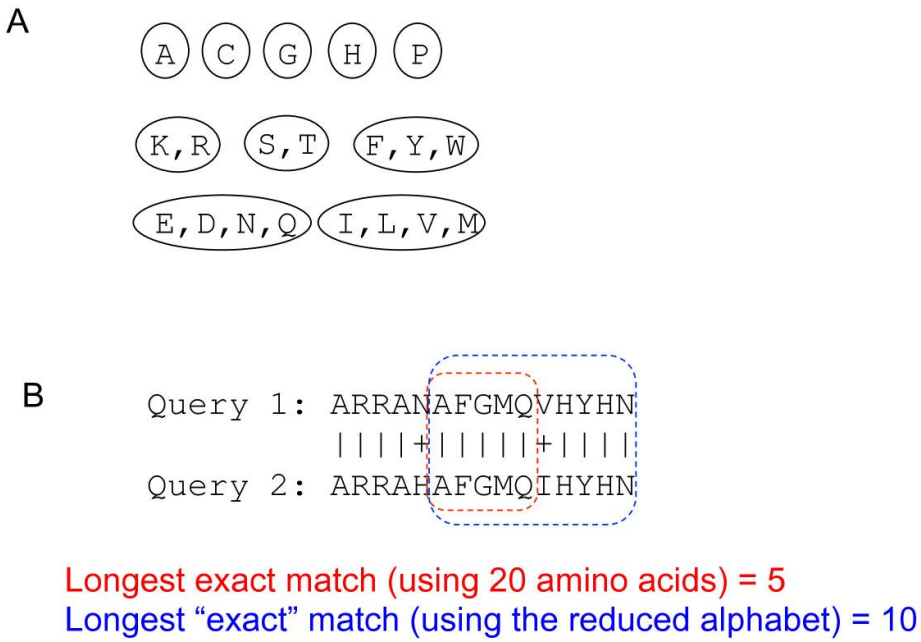
U drugom koraku računat ćemo poravnanja kandidatnih sljedova koristeći postojeći alat za optimalno poravnanje OPAL [10]. Alat nudi četiri različita algoritma kojima se pronalazi poravnanje. Koristit ćemo spomenuti Smith-Waterman algoritam koji traži optimalna lokalna poravnanja jer želimo rješenje uspoređivati s BLAST-om.

2. Metode

Metode opisane u poglavljima 2.1 i 2.2 su heurističke metode koje služe smanjenju prostora pretraživanja, tj. filtriraju kandidatne sljedove. Metoda opisana u poglavlju 2.3 je deterministička metoda koja daje optimalna poravnanja upitnog proteina s kandidatnim sljedovima.

2.1. Reducirana abeceda aminokiselina

S ciljem povećanja osjetljivosti aminokiseline ćemo podijeliti u grupe. Sve aminokiseline unutar iste grupe proglasit ćemo jednakima. Pokazano je da takva ideja doprinosi osjetljivosti pretrage sličnih proteina [6]. Na slici 2.1 prikazana je jedna moguća abe-



Slika 2.1: **(A)** Prikaz grupiranih aminokiselina. **(B)** Primjer iz kojeg je vidljivo kako reducirana abeceda djeluje na osjetljivost pretrage. [9]

ceda i njen utjecaj na ocjenu sličnosti proteina. Kada koristimo standardu abecedu od 20 aminokiselina najduži identični podniz dva slijeda je duljine 5. Korištenjem reducirane abecede ta duljina je 10.

Razvijane su razne abecede koje su imale različite namjene. Moramo odlučiti koja abeceda je najprilagođenija ovom slučaju. Primjeri nekih abeceda prikazani su u tablici 2.1. Prva abeceda *all.*20 nije reducirana, predstavlja standardnih 20 aminokiselina. Tu

Tablica 2.1: Abecede aminokiselina

Ime abecede	Veličina abecede	Grupacija aminokiselina
all.20	20	P G E K R Q D S N T H C I V W Y F A L M
gbmr.4	4	G [ADEKNQEST] [CFHILMVWY] P
gbmr.10	10	G D N [AEFIKLMQRVW] Y H C T S P
murphy.5	5	[LVIMC] [ASGTP] [FYW] [EDNQ] [KRH]
murphy.10	10	A [KR] [EDNQ] C G H [ILVM] [FYW] P [ST]

abecedu nećemo koristiti jer želimo povećati osjetljivost. Pokazano je da povećanje osjetljivosti najbolje postižu sažete abecede poput *gbmr.4* [6]. Međutim, takve abecede smanjuju efikasnost pretraživanja. Rezultiraju pronalaskom velikog broja potpuno različitih proteina, tj. nepotrebno povećavaju prostor pretrage [9]. Iz tog razloga sažete abecede nisu dobar izbor za rješenje ovog problema gdje naglasak stavljamo na brzinu pretrage. Abecedu koja je odabrana u RAPSearch-u [9] koristit ćemo i ovdje. To je abeceda *murphy.10* koja je prikazana na slici 2.1, a nalazi se i u tablici 2.1. Ona donosi znatno povećanje osjetljivosti uz minimalan gubitak efikasnosti pretrage.

2.2. Podnizovi minimalne leksikografske vrijednosti (engl. *minimizer*)

Ovo metoda je glavni dio ovog rada, znatno smanjuje prostor pretrage i tako ubrzava rješenje. Ideja je preuzeta iz [4] gdje je služila za smanjenje prostora pretrage sljedova nukleotida. Budući da ne možemo svaki upitni protein uspoređivati sa svakim proteinom iz baze, moramo pronaći način da brzo identificiramo one sljedove iz baze koji prema nekoj mjeri imaju najveće šanse da budu slični upitnom proteinu.

Ideja je da svaki protein zamijenimo nekim značajnim predstavnicima. Predstavnici su podnizovi proteina koji imaju najmanju leksikografsku vrijednost prema definiranoj funkciji koja svakoj aminokiselini pridružuje neku brojčanu vrijednost (engl. *hash*

value). Umjesto proteina uspoređivat ćemo njihove predstavnike. Ako proteini imaju dovoljno (više od neke definirane granice) istih predstavnika, proglasit ćemo ih kandidatima za sličnost.

2.2.1. Oznake

Neka je $\Sigma = \{P, G, E, K, R, Q, D, S, N, T, H, C, I, V, W, Y, F, A, L, M\}$ abeceda svih aminokiselina. Reduciranu abecedu aminokiselina lako ćemo ukomponirati u ovo rješenje u sljedećim poglavljima (aminokiselinama iste grupe dodijelit ćemo istu brojčanu vrijednost). Proteini koje promatramo su tada elementi skupa Σ^* i oblika su $s = a_1a_2a_3...a_n$ gdje je $a_i \in \Sigma$ i n je duljina tog proteinskog slijeda. Uzastopni podniz proteina duljine k nazivat ćemo k-podniz. Dakle, k-podniz je element skupa Σ^k. Oznakom s_i^k označavat ćemo k-podniz proteinskog slijeda s koji počinje na i-toj poziciji. Primjer proteinskog slijeda i nekih njegovih k-podnizova:

$$s = MEKRCIVWALFYWKKRSD$$

$$s_2^3 = EKR, \quad s_8^5 = WALFY$$

2.2.2. Računanje k-podnizova najmanje vrijednosti

Definiramo funkciju $\nu : \Sigma \to \mathbb{Z}$ koja aminokiselinama pridružuje brojčanu vrijednost na sljedeći način:

$$\nu(P) = 0, \quad \nu(G) = 1, \quad \nu(E) = 2, \quad \nu(K) = 3, \quad ..., \nu(L) = 18, \quad \nu(M) = 19$$

Definiramo funkciju $\phi : \Sigma^k \to \mathbb{Z}$ koja nizovima aminokiselina pridružuje brojčanu vrijednost kao:

$$\phi(s) = B^{k-1} * \nu(a_1) + B^{k-2} * \nu(a_2) + ... + B * \nu(a_{k-1}) + \nu(a_k)$$

$$B = |\Sigma|$$

Gore navedene definicije vrijede kada radimo sa standardnom abecedom od 20 aminokiselina. Za reduciranu abecedu morat ćemo samo promijeniti vrijednosti funkcije ν tako da one aminokiseline koje su smještene u istu grupu imaju istu vrijednost. Dodatno potrebno je promijeniti konstantu B na veličinu reducirane abecede. Definicija kada radimo s odabranom abecedom *murphy.10* iz tablice 2.1:

$$\nu_M(A) = 0, \quad \nu_M(K) = \nu_M(R) = 1, \quad \nu_M(E) = \nu_M(D) = \nu_M(N) = \nu_M(Q) = 2,$$

$$\nu_M(C) = 3, \quad \nu_M(G) = 4, \quad \nu_M(H) = 5, \quad \nu_M(I) = \nu_M(L) = \nu_M(V) = \nu_M(M) = 6,$$
$$\nu_M(F) = \nu_M(Y) = \nu_M(W) = 7, \quad \nu_M(P) = 8, \quad \nu_M(S) = \nu_M(T) = 9$$
$$B_M = 10$$

Odabirom takve funkcije svaki slijed duljine k preslikava se u jedinstveni cijeli broj. Budući da nema kolizija, možemo zaključiti da vrijedi:

$$\phi(s_1) = \phi(s_2) \Longleftrightarrow s_1 = s_2$$

Sada ćemo svaki proteinski slijed s zamijeniti skupom svih (w, k)-predstavnika tog slijeda $M(s)$ i te skupove ćemo koristiti kako bismo brže ustvrdili postoji li potencijalna sličnost između sljedova. (w, k)-predstavnik slijeda $s, |s| \geq w + k - 1$, je par (h, i) takav da postoji j, $max(1, i - w + 1) \leq j \leq min(i, |s| - w - k + 1)$, za koji vrijedi:

$$h = \phi(s_i^k) = min\{\phi(s_{j+p}^k) : 0 \leq p < w\}$$

Drugačije rečeno, (h, i) ulazi u skup $M(s)$ ako $\phi(s_i^k) = h$ i ako postoji okruženje od w susjednih k-podnizova oko i takvo da među njima ne postoji k-podniz koji daje manju vrijednost funkcije ϕ. Algoritam 1 prikazuje računanje skupa predstavnika za parametre w, k i ulazni proteinski slijed u vremenskoj složenosti $\mathcal{O}(w\,|s|)$. U poglavlju vezanom za implementaciju pokazat ćemo kako to napraviti u složenosti $\mathcal{O}(|s|)$. Za svaki prozor od w uzastopnih k-podnizova tražimo sve takve koji daju minimalnu vrijednost funkcije ϕ i bilježimo ih u skupu M dodajući tu vrijednost i poziciju tog k-podniza. Treba primijetiti da u prozoru od w uzastopnih k-podnizova može biti više onih s minimalnom vrijednosti funkcije ϕ. Zato algoritam 1 brine o pronalasku svih minimalnih.

Pokazat ćemo računanje skupa $M(s = MEKRCVWALYSD)$ za reduciranu abecedu *murphy.10* i $w = 6, k = 3$. Prvo trebamo izračunati vrijednosti funkcije ϕ za sve 3-podnize.

$$\phi(s_1^3) = \phi(MEK) = B^2 * \nu_M(M) + B * \nu_M(E) + \nu_M(K) = 100 * 6 + 10 * 2 + 1 = 621$$

$$\phi(s_2^3) = \phi(EKR) = 211, \quad \phi(s_3^3) = 113, \quad \phi(s_4^3) = 136, \quad \phi(s_5^3) = 367,$$
$$\phi(s_6^3) = 670, \quad \phi(s_7^3) = 706, \quad \phi(s_8^3) = 67, \quad \phi(s_9^3) = 679, \quad \phi(s_{10}^3) = 792$$

Sada za svaki prozor od 6 uzastopnih 3-podniza odaberemo minimalne brojčane vrijednosti i ažuriramo skup $M(s)$ tim vrijednostima i odgovarajućim pozicijama.

$$min(\phi(s_1^3), \phi(s_2^3), \phi(s_3^3), \phi(s_4^3)\phi(s_5^3), \phi(s_6^3)) = 113 = \phi(s_3^3), \quad M \leftarrow M \cup \{(113, 3)\}$$

Algoritam 1 Računanje skupa predstavnika $M(s)$

Ulaz: parametri w, k i slijed proteina $s, |s| \geq w + k - 1$
Izlaz: skup (w, k)-predstavnika $M(s)$
$M \leftarrow \emptyset$
for $(i \leftarrow 1;\ i \leq |s| - w - k + 1;\ i++)$ **do**
  $m \leftarrow \infty$
  for $(j \leftarrow 0;\ j < w;\ j++)$ **do**
    $m \leftarrow min(m, \phi(s^k_{i+j}))$
  end for
  for $(j \leftarrow 0;\ j < w;\ j++)$ **do**
    if $\phi(s^k_{i+j}) = m$ **then**
      $M \leftarrow M \cup \{(m, i + j)\}$
    end if
  end for
end for
return M

$$min\{\phi(s^3_i) : 2 \leq i \leq 7\} = 113 = \phi(s^3_3), \quad M \leftarrow M \cup \{(113, 3)\}$$
$$min\{\phi(s^3_i) : 3 \leq i \leq 8\} = 67 = \phi(s^3_8), \quad M \leftarrow M \cup \{(67, 8)\}$$
$$min\{\phi(s^3_i) : 4 \leq i \leq 9\} = 67 = \phi(s^3_8), \quad M \leftarrow M \cup \{(67, 8)\}$$
$$min\{\phi(s^3_i) : 5 \leq i \leq 10\} = 67 = \phi(s^3_8), \quad M \leftarrow M \cup \{(67, 8)\}$$
$$min\{\phi(s^3_i) : 6 \leq i \leq 11\} = 67 = \phi(s^3_8), \quad M \leftarrow M \cup \{(67, 8)\}$$
$$min\{\phi(s^3_i) : 7 \leq i \leq 12\} = 67 = \phi(s^3_8), \quad M \leftarrow M \cup \{(67, 8)\}$$
$$M(ME\mathbf{KRC}VW\mathbf{ALY}SD) = \{(113, 3),\ (67, 8)\}$$

$(6, 3)$-predstavnici slijeda $MEKRCVWALYSD$ su podnizovi KRC i ALY. U rješenju ćemo koristiti $w = 13, k = 4$.

Za upitne proteine odredimo predstavnike na isti način (Algoritmom 1). Vremenski preskupo bi bilo za svaki upitni protein uspoređivati njegov skup predstavnika sa svakim skupom predstavnika proteina iz baze. Zato ćemo predstavnike proteina iz baze indeksirati u tablicu raspršenog adresiranja gdje će ključevi biti brojčane vrijednosti predstavnika, a vrijednosti lista parova $(id, pozicija)$. id je identifikator proteina u bazi, a $pozicija$ je mjesto unutar tog proteina gdje se nalazi taj predstavnik. Tada ćemo za svaki upitni protein proći samo po bitnim predstavnicima iz baze. Algoritam 2 prikazuje način indeksiranja proteina iz baze.

Algoritam 2 Indeksiranje proteina iz baze

Ulaz: lista svih proteina iz baze $P = \{s_1, s_2, ..., s_p\}$, konstante w i k

Izlaz: tablica raspršenog adresiranja koja indeksira sve predstavnike iz baze

$H \leftarrow prazna\,tablica$

for $(j \leftarrow 1;\ j \leq p;\ j++)$ **do**

  $M \leftarrow Algoritam1(w, k, s_j)$

  for all $(h, i) \in M$ **do**

    $H[h].append((j, i))$

  end for

end for

return H

2.2.3. Ocjena sličnosti upitnog proteina i proteina iz baze

Sada je potrebno ocijeniti sličnost upitnog proteina sa svakim proteinom iz baze na temelju njihovih predstavnika.

Moguća ocjena sličnosti između dva proteina bila bi broj predstavnika iste brojčane vrijednosti. Na primjer, proteini s_1 i q imali bi sličnost 3.

$$M(s_1) = \big\{(\mathbf{168}, 2),\ (\mathbf{123}, 6),\ (253, 11),\ (\mathbf{183}, 14)\big\}$$

$$M(s_2) = \big\{..., (\mathbf{123}, 57), ..., (\mathbf{382}, 126), ..., (\mathbf{183}, 246), ..., (\mathbf{168}, 323), ...\big\}$$

$$M(q) = \big\{(168, 3),\ (123, 8),\ (382, 14),\ (183, 18)\big\}$$

Problem ovakvog načina ocjenjivanja je što ne uzimamo u obzir pozicije u proteinima za koje smo pronašli iste predstavnike. Tako bi jako dug protein s_2 koji nema zajedničkih predstavnika sa q, osim prikazanih, imao veću sličnost s q (jednaku 4). Naravno, želimo da s_1 ima veću sličnost s q.

Za svaki protein iz baze stvorit ćemo jedan niz brojeva u koji ćemo upisivati razlike u pozicijama predstavnika identičnih brojčanih vrijednosti. Nazivat ćemo taj niz NPR (niz pozicijskih razlika). Ocjena sličnosti bit će duljina najveće ne nužno uzastopne sekvence NPR-a takve da razlika između maksimalne i minimalne vrijednosti te sekvence ne prelazi konstantu d. Primjerice, uz $NPR = [-\mathbf{7}, -\mathbf{8}, 435, 123, -\mathbf{4}, 435, -\mathbf{6}]$ i $d = 5$ ocjena sličnosti bila bi 4. Ocjena sličnosti za proteine s_1, q i $d = 5$ tada je jednaka 3.

$$NPR\big(M(s_1),\ M(q)\big) = \big[2 - 3, 6 - 8, 14 - 18\big] = \big[-1, -2, -4\big]$$

Cijeli niz zadovoljava uvjet da je razlika između maksimalne i minimalne vrijednosti tog niza manja ili jednaka $d = 5$. Stoga je sličnost između s_1 i q jednaka duljini niza. Ocjena sličnosti za proteine s_2 i q uz isti d jednaka je 1.

$$NPR\big(M(s_2),\, M(q)\big) = \big[57 - 8, 126 - 8, 246 - 8, 323 - 8\big] = \big[49, 118, 315\big]$$

U ovom nizu ne postoji sekvenca veličine veće od 1 koja zadovoljava gornje uvjete.

Ovim načinom ocjenjivanja zapravo tražimo najbolje lokalno poravnanje između dva slijeda. Velik broj pozicijskih razlika koje su u intervalu d ukazuju na veliku šansu da sljedovi na tim područjima daju dobro lokalno poravnanje.

$$M(s_3) = \big\{(\mathbf{168}, 1),\, (\mathbf{123}, 6), \ldots (\mathbf{382}, 384),\, (\mathbf{183}, 390)\big\}$$

Protein s_3 iako ima 4 identična predstavnika kao i q imati će sličnost jednaku 2 uz $d = 5$.

$$NPR\big(M(s_3),\, M(q)\big) = \big[1 - 3, 6 - 6, 384 - 14, 390 - 18\big] = \big[-2, 0, 370, 372\big]$$

Dvije sekvence ovdje ostvaruju maksimalnu veličinu: $\big[-2, 0\big]$ i $\big[370, 372\big]$. Obje su veličine 2, dakle ocjena sličnosti je 2. Postavlja se pitanje zašto želimo ocijeniti s_1 većom sličnošću nego s_3. Razlog je u tome što tražimo lokalna poravnanja. Poravnanje s_1 i q vjerojatno će biti značajnije jer postoje tri predstavnika koja su bliska u oba proteina. Poravnanje s_3 i q vjerojatno će biti manje značajno jer na dva mjesta postoji grupa od dva bliska predstavnika u oba proteina. Lokalno poravnanje s_1 na q trebalo bi biti značajnije od oba poravnanja s_3 na q. U rješenju koristimo $d = 30$.

2.2.4. Određivanje kandidatnih sljedova

Sada od svih proteina iz baze moramo odrediti one s najvećom ocjenom sličnosti s upitnim proteinom. Svaki protein kojem će ocjena biti veća od neke unaprijed definirane konstante κ proći će u sljedeći korak algoritma.

Algoritam 3 pokazuje kako jednim prolaskom po predstavnicima upitnog proteina stvaramo nizove pozicijskih razlika za sve proteine iz baze. U listu L dodajemo identifikatore onih sljedova iz baze čiji nizovi pozicijskih razlika zadovoljavaju gore opisani uvjet, tj. imaju ocjenu sličnosti veću od κ.

Da bismo izračunali ocjenu sličnosti, trebamo algoritam koji za niz cijelih brojeva i konstantu d vraća duljinu najveće sekvence takve da razlika maksimalnog i minimalnog elementa ne prelazi d. To ostvarujemo algoritmom 4 koji nakon sortiranja niza koristi tehniku dva pokazivača (engl. *two pointers technique*) i u složenosti $\mathcal{O}(n)$

Algoritam 3 Reduciranje baze - pronalazak svih kandidata

Ulaz: indeksirani svi proteini iz baze (izlaz algoritma 2) H, upit q, konstante κ, w, k, d

Izlaz: lista identifikatora sljedova iz baze koji su zadovoljili ovaj dio algoritma L

$L \leftarrow [\,]$

$NPRs \leftarrow prazna\,hash\,tablica$

$M_q \leftarrow Algoritam1(w, k, q)$

for all $(h, i) \in M_q$ **do**

  for all $(id, pos) \in H[h]$ **do**

    $NPRs[id].append(i - pos)$

  end for

end for

for all $id \in keys(NPRs)$ **do**

  if $Algoritam4(NPRs[id], d) \geq \kappa$ **then**

    $L.append(id)$

  end if

end for

return L

Algoritam 4 Određivanje duljine najdulje sekvence kojoj je razlika maksimuma i minimuma manja ili jednaka nekoj konstanti

Ulaz: lista brojeva A i konstanta d

Izlaz: duljina tražene sekvence

$ret \leftarrow 0$

$sort(A)$

$p1 \leftarrow p2 \leftarrow 0$

while $p2 < size(A)$ **do**

  while $A[p2] - A[p1] > d$ **do**

    $p_1 + +$

  end while

  $ret \leftarrow max(ret, p2 - p1 + 1)$

  $p_2 + +$

end while

return ret

(gdje je n duljina ulaznog niza) pronalazi najdulju sekvencu traženog uvjeta. Složenost traženja je linearna jer u svakoj iteraciji inkrementiramo barem jedan od pokazivača, znači napravimo najviše $2 * n$ iteracija. Ukupna složenost algoritma 4 je $\mathcal{O}(n \log_2 n)$ jer trebamo sortirati ulazni niz.

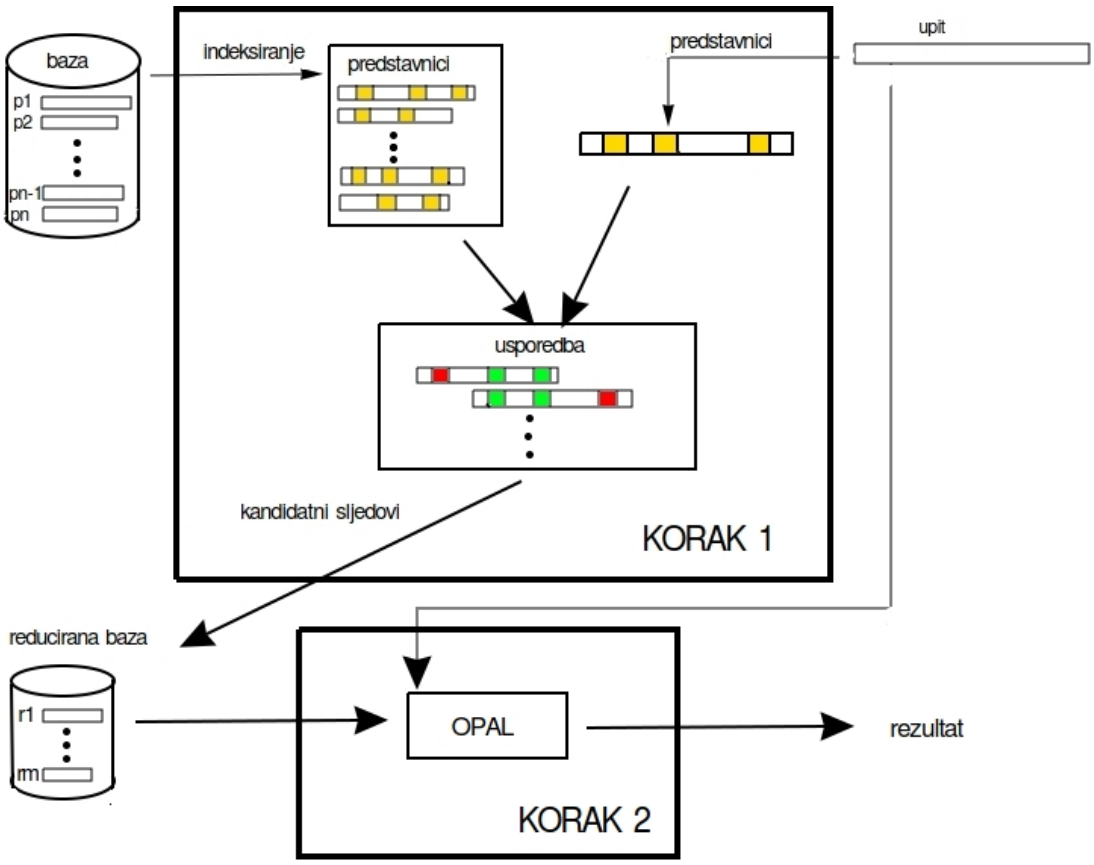
2.3. OPAL

Svi proteinski sljedovi iz baze, koji zadovolje prvi heuristički korak algoritma, dolaze do ovog koraka. Broj proteina koji zadovolje prvi korak obično bude oko 1% ukupnog broja proteina u bazi. Zbog velikog smanjenja prostora pretrage u mogućnosti smo koristiti deterministički alat za optimalno poravnanje. U ovom poglavlju pod pojmom *baza proteina* podrazumijevat ćemo reduciranu originalnu bazu, tj. sve one proteine koji su zadovoljili prvi korak algoritma.

OPAL je SIMD (engl. *Single instruction, multiple data*) C/C++ knjižnica za pronalazak optimalnih poravnanja jednog upita na više proteina. Podržava više algoritama optimalnog poravnanja. Mi ćemo koristiti već spomenuti Smith-Waterman.

OPAL koristimo tako da zadamo bazu proteina i upitni protein, a on nam za svaki protein iz baze vrati rezultat poravnanja s upitom. Iz rezultata poravnanja možemo dohvatiti brojčanu vrijednost poravnanja, to je mjera koliko je poravnanje dobro. Što je mjera veća poravnanje je značajnije. Ta vrijednost dobiva se na sličan način kao što je pokazano na kraju poglavlja 1.1. Upravo tu mjeru koristimo kao krajnju ocjenu kvalitete poravnanja, rezultate koje ispisujemo rangiramo prema toj vrijednosti. U rezultatima poravnanja nalaze se i informacije o poziciji tog poravnanja unutar proteina, postotku identičnih aminokiselina i rekonstrukciji poravnanja (slično kao na slici 1.2).

Na slici 2.2 prikazana je skica cijelog opisanog sustava.



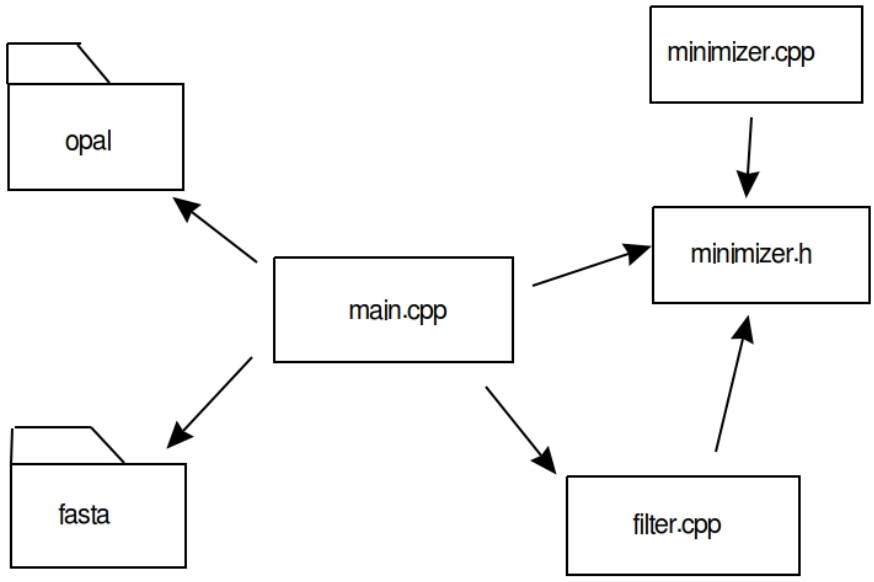
Slika 2.2: Grafički prikaz cijelog algoritma

3. Implementacija

Alat je napisan u programskom jeziku C++. Implementacija je dostupna na Githubu[1].

3.1. Organizacija sustava

Programski kod podijeljen je na tri datoteke: *main.cpp*, *minimizer.cpp* i *filter.cpp*. Dodatno su korištene dvije knjižnica. Knjižnica *fasta*[2] predstavlja jednostavnu implementaciju čitača datoteka u FASTA formatu. Druga korištena knjižnica je *opal*[10].



Slika 3.1: Dijagram ovisnosti komponenata rješenja

[1]https://github.com/dbabojelic/FPSSS
[2]https://github.com/casalebrunet/fasta

U datoteci *minimizer.cpp* nalazi se metoda koja implementira pronalazak svih predstavnika proteinskog slijeda kao što to prikazuje algoritam 1. Implementacija je nešto drugačija od prikazane u pseudo kodu i manje je vremenski složenosti. U pseudo kodu algoritma 1 nakon svakog pomaka trenutno promatranog prozora nanovo računamo minimalnu vrijednost. U stvarnoj implementaciji koristimo algoritam pomičnog prozora s održavanjem minimalne vrijednosti (engl. *Sliding Window Minimum/Maximum*). Algoritam koristi strukturu podatka red (engl. *queue*) kako bi u amortiziranoj konstantnoj složenosti odgovorio na pitanje koji je najmanji element trenutno prisutan u pomičnom prozoru. Pomicanje prozora, tj. ubacivanje/dodavanje novog elementa također je amortizirane konstantne složenosti. To ukupan algoritam čini linearne složenosti u duljini slijeda proteina. U istu metodu ukomponiran je i algoritam 2 na način da kada nađemo predstavnika odmah ga stavljamo i u tablicu raspršenog adresiranja.

U datoteci *filter.cpp* zapravo je implementirana metoda ekvivalentna algoritmu 3. I prikazana je sljedećim programskim isječkom:

```
vector<int> getSimilar(vector<minimizer::Minimizer> Mq,
                       minimizer::IndexTable &indexTable,
                       const int D, const int K) {

    unordered_map<int, vector<int>> NPRs;

    // stvorimo sve nizove pozicijskih razlika
    for (minimizer::Minimizer mini: Mq) {
        for (auto& element: indexTable[mini.h]) {
            NPRs[element.sequenceIndex].
            push_back(element.position - mini.position);
        }
    }

    vector<int> ret;

    //za svaki niz pozicijskih razlika pronademo
    //najveci podskup koji zadovoljava uvjet
    //da je max - min <= D
    //
    //ako je taj najveci podskup veci od K
```

```
    //ID proteina stavljamo u odgovor
    int sz = Mq.size();
    for (auto& npr: NPRs) {
        sort(npr.second.begin(), npr.second.end());

        //algoritam dva pokazivaca
        int p1 = 0;
        int p2 = 0;
        int sz = npr.second.size();
        int maxSegmentSize = 0;
        while (p2 < sz) {
            while (npr.second[p2] - npr.second[p1] > D)
                p1++;
            maxSegmentSize = max(maxSegmentSize, p2 - p1 + 1);
            p2++;
        }

        if (maxSegmentSize >= K)
            ret.push_back(npr.first);
    }

    return ret;
}
```

Datoteka *main.cpp* ulazna je točka programskog rješenja. U njoj se povezuju funkcionalnosti iz gore opisanih knjižnica i datoteka.

4. Rezultati

Za testiranje korištena je Uniprot Swiss-Prot baza proteinskih sljedova dostupna na adresi *http://www.uniprot.org/downloads*. Baza sadrži 557012 proteinskih sljedova prosječne duljine 359 aminokiselina i veličine je 275 MB. Kao upitne proteine koristit ćemo tri različita skupa proteina *HumDiv_neutral*, *HumVar_neutral*[1] i *Escherichia_Coli_07798*[3]. Osnovne informacije o ta tri seta prikazane su u tablici 4.1. Testiranje je provedeno na jednom procesoru *Intel(R) Xeon(R), 2.40 GHz* i 252 GB

Tablica 4.1: Skupovi upita za testiranje

Ime seta	Broj proteina	Ukupni broj aminokiselina
HumDiv	315	235 219
Escherichia	4969	1 393 750
HumVar	3400	2 263 906

radne memorije.

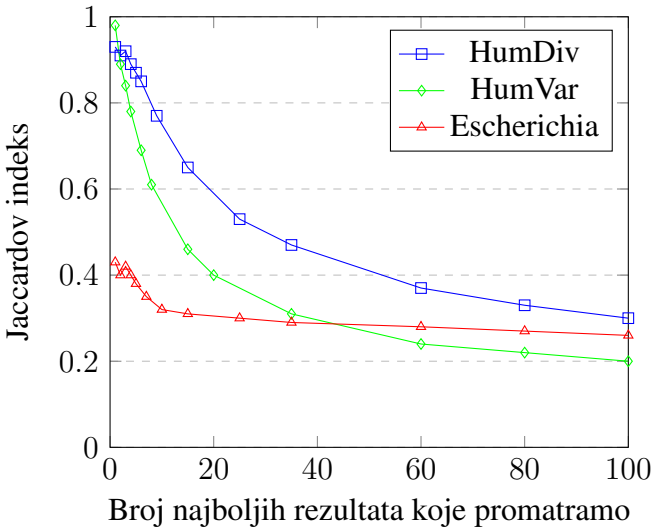
4.1. Usporedba s BLAST-om

Rezultate ćemo ispisivati u standardnom BLAST-ovom tabularnom formatu[1]. Ocjenu sličnosti rezultata definirat ćemo Jaccardovim indeksom[2]. U našem slučaju to će biti omjer $|R \cap B| \,/\, |R \cup B|$, gdje je R skup rezultata dobiven našom metodom, a B skup rezultata dobiven BLAST-om.

Za ispitne skupove *HumDiv* i *HumVar* dobili smo slične rezultate. Kada tražimo manje od 10 najboljih poravnanja dobiveni rezultati odgovaraju onima koje zabilježi i BLAST. Povećanjem broja rezultata koje promatramo indeks pada, što je i razumljivo budući da su pronađene sličnosti sve manje značajne i lakše dolazi do razlika u pronađenim poravnanjima. Za ispitni skup *Escherichia* dobili smo nezadovoljavajuće

[1] http://www.metagenomics.wiki/tools/blast/blastn-output-format-6

[2] https://en.wikipedia.org/wiki/Jaccard_index



Slika 4.1: Jaccardov indeks u ovisnosti o broju najboljih rezultata

rezultate. To možemo objasniti povećanim brojem mutacija kod bakterija. Zbog toga je teže pronaći značajna poravnanja. Budući da je BLAST također alat zasnivan na heurističkom algoritmu i ne daje nužno optimalne rezultate, ne možemo zaključiti da je naš alat lošiji ili bolji od BLAST-a, već je ovo samo usporedba rezultata.

Tablica 4.2: Usporedba vremena izvođenja

Ime seta	Vrijeme (min)	BLAST-ovo vrijeme (min)
HumDiv	56	45
Escherichia	483	225
HumVar	542	436

Na tablici 4.2 prikazana je usporedba vremena izvođenja. Napomenimo još da je prije testiranja bilo potrebno indeksirati bazu proteina. Alatu iz ovog rada za to je bilo potrebno pedesetak sekundi, dok je BLAST indeksiranje izvršio u tridesetak sekundi.

5. Zaključak

Iz prethodnih poglavlja možemo zaključiti da je problem pretrage baze proteinskih sljedova složen i nije jednostavno napraviti alat koji radi zadovoljavajuće. Deterministički algoritmi prespori su za upotrebu na današnjim baza, stoga je zadnjih godina naglasak na heurističkim metodama. Za usporedbu koristili smo najpopularniji heuristički alat BLAST koji je nastao još 1990. i od tada je usavršavan. Iz rezultata je vidljivo da smo se uspjeli približiti BLAST-u samo na lakšim upitnim skupovima (ako promatramo samo manji broj najboljih poravnanja) i da ima još podosta mjesta za napredak.

LITERATURA

[1] Ivan A Adzhubei, Steffen Schmidt, Leonid Peshkin, Vasily E Ramensky, Anna Gerasimova, Peer Bork, Alexey S Kondrashov, i Shamil R Sunyaev. A method and server for predicting damaging missense mutations. *Nature methods*, 7(4): 248, 2010.

[2] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, i David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17): 3389–3402, 1997.

[3] 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*, 41(D1):D48–D55, 2012.

[4] Heng Li. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences. *Bioinformatics*, 32(14):2103–2110, 2016.

[5] NCBI. The statistics of sequence similarity scores. URL `https://www.ncbi.nlm.nih.gov/BLAST/tutorial/Altschul-1.html`.

[6] Eric L Peterson, Jané Kondev, Julie A Theriot, i Rob Phillips. Reduced amino acid alphabets exhibit an improved sensitivity and selectivity in fold assignment. *Bioinformatics*, 25(11):1356–1362, 2009.

[7] Temple F Smith i Michael S Waterman. Comparison of biosequences. *Advances in applied mathematics*, 2(4):482–489, 1981.

[8] Robert Vaser, Dario Pavlović, i Mile Šikić. Sword—a highly efficient protein database search. *Bioinformatics*, 32(17):i680–i684, 2016.

[9] Yuzhen Ye, Jeong-Hyeon Choi, i Haixu Tang. Rapsearch: a fast protein similarity search tool for short reads. *BMC bioinformatics*, 12(1):159, 2011.

[10] Martin Šošić. *OPAL - SIMD C/C++ library for massive optimal sequence alignment*, 2015. URL `https://github.com/Martinsos/opal`.

Brzo pretraživanje sličnih proteinskih sljedova

Sažetak

Pretraga baza proteinskih sljedova jedan je od glavnih problema bioinformatike. Do sada su razmatrane razne deterministične i heuristične metode za rješavanje ovog problema. Deterministične metode daju optimalne rezultate, ali su vrlo spore. Heuristične metode mnogo su brže, ali ne pronalaze optimalne rezultate. Eksponencijalni rast broja proteinskih sljedova prisilio nas je da više istražujemo heuristične metode.

U ovom radu predstavljamo heuristički algoritam za efikasno pretraživanje baze proteinskih sljedova. Algoritam je implementiran u dva koraka. U prvom koraku koristimo heurističku metodu predstavljenu u radu [4] kako bi znatno smanjili prostor pretrage. U drugom koraku koristimo OPAL[10], knjižnicu koja nam pruža implementacije algoritama optimalnog poravnanja.

Ključne riječi: bioinformatika, heuristički algoritmi, baza proteinskih sljedova

Fast protein sequence similarity search

Abstract

Protein database search is one of the fundamental problems in bioinformatics. For decades, it has been explored and solved using different exact and heuristic approaches. Deterministic approaches yield optimal result but are really slow. Heuristic approaches are a lot faster but don't find optimal result. Exponential growth of data in recent years has forced us to explore mostly heuristic approaches.

In this paper we present heuristic algorithm for efficient protein database search. Algorithm was implemented as a two-step approach. In the first step we use heuristic method introduced in [4] to significantly reduce database size. In the second step we use OPAL[10], a library which provides implementation of optimal alignment algorithms.

Keywords: bioinformatics, heuristic algorithm, protein database