SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3741

Alat za poravnanje genoma

Petar Žuljević

Zagreb, lipanj 2014.

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

Zahvaljujem mentoru Mili na uloženom trudu i vremenu.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Bioinformatika je multidisciplinarna grana znanosti koja iziskuje potrebu za korištenjem suvremenih računalnih alata radi efikasne obrade brojnih podataka koji su izdvojeni prilikom bioloških istraživanja. Jedan od problema koji se javljaju u bioinformatici naziva se poravnanje genetskih sljedova.

Poravnanje genetskih sljedova još uvijek je problem koji nije u potpunosti riješen. Naime, radi se o usporedbi dvaju genetskih nizova čija se duljina mjeri u nekoliko stotina milijuna pa čak i milijardi dušičnih baza. Upravo zbog tako velike količine podataka potrebno je razviti algoritme i strukture podataka kojima će se moći u razumnom vremenu i uz ograničena računalna sredstva obaviti poravnanje. Sljedovi su prikazani kao nizovi znakova abecede koji označavaju dušične baze (A,C,T,G).

Cilj poravnanja je pronalazak sličnih ili identičnih regija u genomima koje bi mogle upućivati na genetsku sličnost, svojevrsne mutacije koje su se dogodile ili pak ukazivati na neke bitne funkcionalnosti zastupljene u kodirajućim regijama genoma.

Postojeća rješenja zasnivaju se na heuristikama koje obavljaju poravnanja koristeći različite metode. Trenutno najpopularniji alati su MUMmer, BLASTZ i LASTZ.

U ovom radu obrađena je problematika poravnanja dvaju genetskih sljedova te je opisan alat koji obavlja tu funkciju. Također su prikazani i rezultati poravnanja nad kromosomima čovjeka i miša.

2. Poravnanje genetskih sljedova

Poravnanje je postupak pronalaska međusobne sličnosti dvaju ili više nizova. Za zadana dva genetska niza sastavljena od znakova abecede (A, C, T, G) možemo konstruirati poravnanje na način da umetnemo praznine (oznaka '-') na određena mjesta tako da zadani nizovi postanu iste duljine. Znakovi koji nisu praznine predstavljaju poklapanja ili supstitucije. U genetskim nizovima koriste se oznake A, C, T i G jer su tim slovima određene dušične baze

Za rad navedenih algoritama potrebno je definirati parametre koji će se koristiti pri poravnanju. Parametri određuju cijenu umetanja praznine te cijene supstitucija, a pri tome se koriste unaprijed određene supstitucijske matrice.. Supstitucijske matrice dobivene su empirijskim metodama u biološkim istraživanjima te na svojevrstan način opisuju vjerojatnosti zamjene jedne dušične baze s drugom.

Primjerice ako su zadana dva genetska slijeda „ACTTCCAGA“ i „AGTTCCGGAGG“, poravnanje bi moglo izgledati kao u tablici 2.1 uz cijene umetanja praznine -1, cijenu supstitucije 0 te cijenu poklapanja 1. Ukupna ocjena sličnosti može se reprezentirati sumom cijena izvršenih supstitucija, poklapanja i umetnutih praznina što bi u prikazanom jednostavnom prikazanom slučaju iznosilo 5.

Niz1	A	C	T	T	C	C	-	-	A	G	A
Niz2	A	G	T	T	C	C	G	G	A	G	G
Cijena	1	0	1	1	1	1	-1	-1	1	1	0

Tablica 2.1: Primjer jednostavnog poravnanja

Poravnanje navedeno u tablici 2.1 nije jedino moguće rješenje, naime postoji velik broj poravananja, ali su interesiratna samo ona koja imaju najveće ocjene. Pri tome nije moguće koristiti egzaktne algoritme koji će dati optimalno rješenje jer je vrijeme njihovog izvršavanja neprihvatljivo te se stoga koriste heuristike koje će relativno brzo doći do dovoljno dobrih rješenja.

2.1. Uvod u terminologiju

U ovom odjeljku navedeni su izrazi i njihove definicije koje će se koristiti u nastavku rada.

MUM *(engl. Maximal unique match)* - najdulji podudarajući podniz koji se pojavljuje točno jednom u oba ulazna niza.

MEM *(engl. Maximal exact match)* - najdulji podudarajući podnizovi koji se pojavljuju više puta u oba ulazna niza.

Kritična duljina - Broj znakova u kojima se MUMovi ili MEMovi moraju podudarati da bi bili uzeti u razmatranje.

Netaknute regije - Područja na kojima nisu pronađeni MUMovi niti MEMovi.

MUMovi šuma (*engl. Noisy MUMs*) - MUMovi koji su rezultat čiste slučajnosti, javljaju se ukoliko se generiraju MUMovi s nedovoljno velikim parametrom kritične duljine.

Reverzni komplement - Niz koji dobijemo kada ga obrnemo i svaku dušičnu bazu zamijenimo s njenom komplementarnom bazom.

Lokalno poravnanje - Poravnanje u kojem pronalazimo dijelove nizova koji su "najsličniji", odnosno najviše se podudaraju.

Globalno poravnanje - Poravnanje prilikom kojeg koristimo sva slova oba niza

2.2. Popularni alati

Poznati su algoritmi za optimalno lokalno i globalno poravnanje koji se zasnivaju na dinamičkom programiranju (Smith-Waterman, Needleman-Wunsch), međutim oni nisu efikasni za poravnanje nizova čije su duljine većeg reda veličine zbog kvadratne vremenske složenosti u kojoj se izvršavaju. Zbog toga je potrebno pronaći druge načine za obavljanje poravnanja. Danas su najpopularniji alati koji se zasnivaju na heurističkom pristupu poput alata MUMmer, BLASTZ i LASTZ.

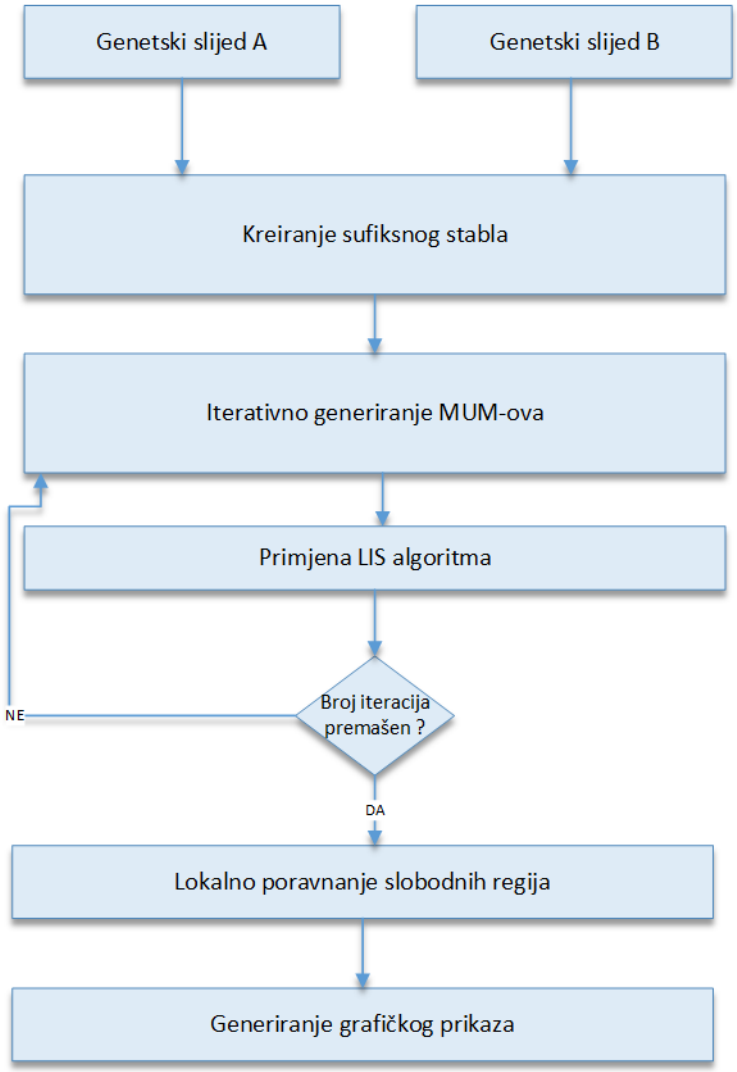
2.2.1. MUMmer

MUMmer je jedan od najbržih alata za poravnanje čitavih genoma. Zadnja inačica je MUMmer 3.0, a zasniva se na izgradnji strukture podataka koja se zove sufiksno stablo.

Rad MUMmera započinje pronalaskom jedinstvenih poklapajućih regija koje se točno jednom pojavljuju u oba zadana genetska slijeda te se poklapaju u barem K

dušičnih baza. Navedena poklapanja nazivaju se MUM-ovi (engl. Maximal Unique Matching). Parametar K može se utvrditi i eksperimentalno te je bitno da je dobro određen jer nisu poklapanja svih duljina jednako značajna i vjerojatna. Sufiksno stablo omogućuje u linearnoj vremenskoj složenosti određivanje navedenih poklapanja, ali loša strana je potrošnja memorije za njegovo kreiranje.

Nakon pronalaska MUM-ova postupak se ponavlja na područjima između identificiranih MUM-ova kroz zadani broj iteracija. Zatim se vrši proces filtriranja MUM-ova pomoću LIS algoritma i u konačnici se izvršava proces lokalnog poravnanja regija koje su između MUM-ova.



Slika 2.1: Tijek procesa poravnavanja kod MUMmera

2.2.2. BLASTZ

BLASTZ [1] je alat koji omogućuje poravnanje dvaju genetskih nizova. Sastoji se od idućih faza: pronalazak identičnih regija u genomima, proširenje identičnih regija, povezivanje proširenih regija, rekurzivno ponavljanje prethodnog na regijama koje su ostale netaknute.

Algoritam 1: BLASTZ

1. Uklanjanje ponavljajućih razasutih regija iz oba niza.
2. Za sve dvanaestorke koje su identične ili se razlikuju u jednoj bazi u oba niza treba raditi iduće:
   (a) Proširiti poravnanje oko pogotka u lijevo i desno, sve dok rezultat poravnanja ne počne padati ispod zadanog praga.
   (b) Ukoliko je rezultatna ocjena poravananja veća od recimo 3000 tada:
      i. Ponavaljanje 2. koraka, tj vršiti proširenje uz dozvoljavanje praznina.
      ii. Ukoliko je ocjena poravnanja veća od recimo 5000 zabilježiti poravnanje.
3. Između svaka dva para susjednih poravnanja iz koraka 2, ponoviti korak 2, ali uz korištenje senzitivnijih parametara (primjerice potraga za sedmorkama) i uz korištenje nižih pragova za poravnanje sa i bez dozvoljenih praznina.
4. Prilagoditi slijed pozicija rezultatnih regija poravnanja tako da se mogu smjestiti u originalne nizove.
5. Filtrirati rezultatne poravnate regije tako da se pronađe najbolji odgovarajući slijed.

2.2.3. LASTZ

LASTZ podržava sve mogućnosti koje nudi i BLASTZ te ih još i proširuje. Pri tome pruža širi skup strategija obrade identičnih regija s kojima započinje proces poravnanja te ima manju memorijsku potrošnju. Bazira se na pretprocesiranju jednog genetskog slijeda, a zatim obavlja poravnanje nad drugim genetskim slijedom.

Memorijska poboljšanja ostvarena su na način što prilikom obrade invertirajućih regija u memoriji ne drži zasebno reverzno komplementiran genetski slijed već uvodi

[1] Schwartz, et al., 2003

4 dodatne funkcije (jedna za svaki smjer poravnanja u oba niza) koje obrađuju regije u nizovima tako da je konačni rezultat isti. Međutim, takvo poboljšanje urokuje teže održavanja koda.

Drugo memorijsko poboljšanje ostvareno je tako što su DNA nizovi pakirani po 4 baze u bajtu podataka umjesto prijašnjeg bajta po bazi. Također koristi i metode raspršenog adresiranja te dodatne strukture podataka koje iskorištavaju znatno manje memorije nego primjerice sufiksno stablo.

3. Podaci

Prilikom testiranja alata iz prethodnih poglavlja korišteni su genetski sljedovi koji se mogu preuzeti u FASTA formatu s poslužitelja Ensembl [1]. Moguće je preuzeti tekstualne zapise svih kromosoma čovjeka i brojnih ostalih životinjskih vrsta. FASTA format jednostavan je način zapisa, u prvom retku nalazi se opis a u svim ostalim redcima zapisan je genetski slijed. Genetski slijed sastoji se od pet znakova od kojih četiri označavaju dušične baze, a peti znak 'N' označava da nije poznato o kojoj se dušičnoj bazi radi (tablica 3.1).

Prilikom poravnanja znak 'N' tretira se tako da se praktički ignorira u fazi pronalaska MUMova, dok je kod korištenja Smith-Watermana moguće tretirati znakove 'N' uz negativne cijene slično kao i praznine.

Znak	Značenje
A	Adenin
C	Citozin
T	Timin
G	Guanin
N	Nepoznato

Tablica 3.1: Značenje znakova u genetskim sljedovima

Genetski sljedovi kromosoma čovjeka variraju po veličini ali okvirno su dugački po 150 milijuna dušičnih baza te takve datoteke zauzimaju i više od 100 MB memorijskog prostora.

Primjer FASTA datoteke [2] :

>9 dna:chromosome chromosome:GRCh37:9:1:141213431:1 REF

...NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGAATTCCAGAACAC...

[1] `http://www.ensembl.org/info/data/ftp/index.html`

[2] izječak iz 9. kromosoma, Homo sapiens

4. Metode

Rad programa opisanog u ovom radu bazira se na ideji alata MUMmer, a najznačajnija razlika je u tome što se koristi struktura podataka sufiksno polje koje bi uz efikasnu implementaciju trebalo koristiti mnogo manje memorije. Za ovu svrhu sufiksno polje nudi iste funkcionalnosti kao i sufiksno stablo te ga je moguće realizirati u linearnoj vremenskoj složenosti.

Alat radi sa dva zadana niza, ali također generira reverzni komplement drugog zadanog niza te se paralelno obrađuju invertirajuće i neinvertirajuće regije. Za efikasno traženje MUMova koristi se sufiksno i LCP polje. Alatu je potrebno predati parametar koji predstavlja minimalnu duljinu koju MUMovi trebaju imati da bi se uzeli u obzir odnosno da bi bili dio rezultatnog poravnanja. Nakon toga iterativno se pronalaze MEMovi i MUMovi manje duljine od zadane na područjima između parova prethodno generiranih MUMova. Ovaj dio ključna je točka u procesu poravnanja te parametar kritične duljine bitno utječe kako na kvalitetu poravnanja tako i na trajanje izvođenja programa te memorijsku potrošnju.

Nakon generiranja MUMova i MEMova pojedina područja u ulaznim nizovima ostala su neporavnata te je nad njima potrebno provesti neki od dinamičkih algoritama poravnanja, konkretno Smith-Waterman algoritam. Uz dobro određene parametre kritičnih duljina te ako se zadani nizovi podudaraju u određenom postotku Smith-Waterman algoritam bi se trebao izvršiti vrlo brzo jer će biti proveden nad regijama koje su znanto manjih duljina u odnosu na duljine početno zadanih nizova.

U konačnici se generira grafički prikaz rezultata poravnanja na kojem je moguće vizualno odrediti podudaranja, invertirajuće regije i translokacije pojedinih područja u genomu.

4.1. Konstrukcija SA i LCP polja

SA polje *(engl. Suffix Array)* je niz koji redom sadržava indekse sufiksa zadanog niza u rastućem leksikografskom poretku. Postoje razne implementacije, ali u radu programa iskorištena je implementacija čija je vremenska složenost konstrukcije linearna [1].

***D*efinicija**:

Neka je zadan niz T duljine N gdje je T[i] sufiks koji započinje na indeksu i pri čemu je $i \in [0, N-1]$.

Tada vrijedi: $T[SA[i]] \leq T[SA[j]]\ \forall i, j \in [0, N-1], i \leq j$

LCP polje *(engl. Longest Common Prefix)* dobije se vrlo lako nakon kreiranja SA polja te predstavlja temelj na kojem se zasniva funkcionalnost programa predstavljenog u ovom radu. Pojedini element LCP polja na nekom indeksu i predstavlja duljinu zajedničkog prefiksa od sufiksa na indeksu i te sufiksa na indeksu i - 1. Ukoliko je LCP[i] = k, tada se T[SA[i]] i T[SA[i - 1]] podudaraju u prvih k znakova.

Prilikom konstrukcije sufiksnog polja za dva zadana niza potrebno je prosljediti konkatenaciju ulaznih nizova pri čemu se na kraj prvog niza priljepi znak $ koji je leksikografski najmanje težine. Isti postupak potrebno je napraviti i sa reverzno komplementiranim drugim nizom. Nakon ove procedure pripremljene su strukture podataka koje su potrebne u idućoj fazi pronalaska MUMova.

Primjer: Neka je zadan niz s_0= "banana$".

i	0	1	2	3	4	5	6
$s_0[i]$	b	a	n	a	n	a	$
$SA[i]$	6	5	3	1	0	4	2
$s_0[SA[i]]$	$	a	a	a	b	n	n
		$	n	n	a	a	a
			a	a	n	$	n
			$	n	a		a
				a	n		$
				$	a		
					$		
$LCP[i]$	0	0	1	3	0	0	2

Tablica 4.1: Sufiksno i LCP polje za niz "banana$"

[1] `https://github.com/elventear/sais-lite-lcp`

4.2. Generiranje MUMova

Ovaj postupak se vrši u nekoliko iteracija kroz 3 dijela, a može se provesti tek kad su priređeni SA i LCP polje:

1. Pronalazak MUMova duljih od kritične duljine K, koristeći SA i LCP polje
2. Filtriranje MUMova pomoću LIS algoritma
3. Kreiranje novih SA i LCP polja za netaknute regije

Prvo obrađujemo cjelokupan niz, a zatim područja između parova MUMova. Iteriranje se radi dokle god to ima smisla, a u programu koji se opisuje ostavljena je mogućnost određivanja i tog parametra.

4.2.1. Pronalazak MUMova

Nakon priređenog SA i LCP polja može se započeti s fazom pronalaska MUMova koji su dulji od kritične duljine K. Sam postupak generiranja MUMova svodi se na prolaz kroz LCP polje i očitavanje elemenata LCP i SA polja. U nastavku slijedi algoritam:

Algoritam 2: Algoritam pronalaska MUMova pomoću SA i LCP polja

```
Data: Znakovni niz s_0, niz LCP, niz SA, K
Result: Lista MUMova
i = 1;
N = broj elemenata u LCP polju;
while i < N - 1 do
    očitaj SA[i -1], LCP[i], SA[i];
    if T[SA[i]] i T[SA[i-1]] sufiksi istog niza or LCP[i] < K then
        continue;
    else
        if LCP[i] > LCP[i-1] and LCP[i] > LCP[i + 1] then
            if T[SA[i] - 1] != T[SA[i-1] - 1] then
                dodajUListuMUMova(SA[i], SA[i - 1]);
```

Navedeni algoritam prolaskom kroz LCP polje traži sufikse koji se podudaraju u barem K početnih znakova te su nužno iz različitih početnih ulaznih nizova (tj. jedan sufiks treba biti na poziciji prije znaka $ a drugi element tada treba biti na poziciji većoj od pozicije znaka $). Isto tako treba biti zadovoljen uvjet maksimalnosti,

stoga se provjerava okolina elementa LCP[i], a ako je taj uvjet (uvjet lokalnog maksimuma) zadovoljen tada još treba provjeriti da pronađeni MEM nije dio nekog većeg MUMa. Ukoliko su znakovi na pozicijama neposredno prije pronađenih potencijalnih MUMova različiti tada MUM dodajemo u listu.

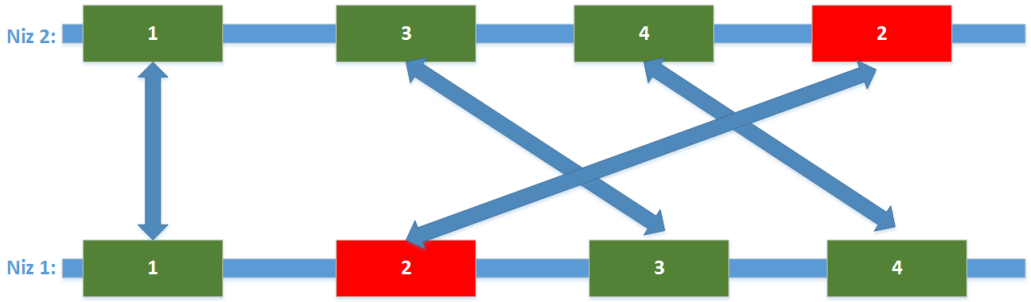
Važno je za uočiti da će broj pronađenih MUMova bitno ovisiti o parametru kritične duljine K. Ukoliko je parametar K premalen tada će se stvoriti prevelik broj MUMova čije je postojanje rezultat puke slučajnosti (*engl. Noisy MUMs*). U slučaju prevelikog parametra K postoji mogućnost da će se pronaći premalo MUMova te se u idućoj iteraciji uz zadano smanjenje kritične duljine neće pronaći niti jedan novi MUM što će uzrokovati odmah prelazak na fazu poravnanja netaknutih regija koje su u tom slučaju prevelike za Smith-Waterman algoritam.

Kritične duljine mogu varirati te ne ovise samo o veličini ulaznih nizova nad kojima se radi poravnanje, već i o međusobnoj sličnosti nizova.

4.2.2. Filtriranje MUMova

MUMove koji su generirani u svakoj iteraciji prethodne faze potrebno je filtrirati i svrstati ih u sortiran poredak s obzirom na pozicije u prvom i drugom ulaznom nizu. To je nužno ukoliko želimo jednoznačno odrediti područja rasprostiranja netaknutih regija i nad njima raditi daljnje iteracije.

Za početak sortiraju se MUMovi s obzirom na pozicije u referentnom nizu, a nakon toga eliminiraju se oni MUMovi čiji poredak nije u odgovarajućem redoslijedu. Ukratko rečeno primjenjuje se algoritam LIS (*engl. Largest Incereasing Subsequence*) na MUMove s obzirom na pozicije u drugom nizu.



Slika 4.1: Primjena algoritma LIS

Algoritam 3: Filtriranje MUMova

Data: listaMUMova

Result: filtriranaListaMUMova

i = 1;

N = broj MUMova;

sortirajPoPozicijiX (listaMUMova);

while *i* < *N* **do**

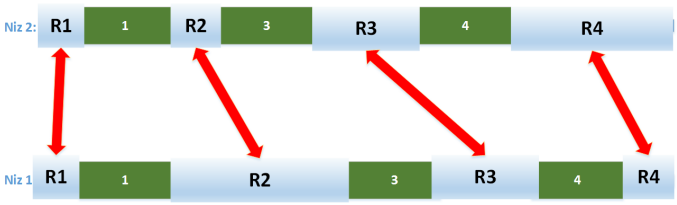
    očitaj listaMUMova[i];

    if *listaMUMova[i].getPozicijaY()* > *listaMUMova[i - 1].getPozicijaY()* **then**

        listaMUMova[i].izbaci();

4.2.3. Generiranje listi LCP i SA polja za netaknute regije

Svaki uzastopni par generiranih MUMova razapinje po jednu netaknutu regiju u svakom od ulaznih nizova. Za svaku takvu regiju provodi se ekstrakcija elemenata polja SA koji joj pripadaju. Primjerice ukoliko se raspolaže s N MUMova u općenitom slučaju postojat će N + 1 takva regija (slika 4.2, označeno s Rx).



Slika 4.2: Netaktnute regije za koje je potrebno kreirati zasebna LCP i SA polja

Kreiranje zasebnih LCP i SA polja za netaknute regije i nije toliko trivijalan postupak. Potrebno je kreirati nekoliko pomoćnih listi (jedna koja će pamtiti za svaku poziciju u konkateniranom nizu kojoj regiji pripada znak na toj poziciji te još 2(N + 1) dodatna lista za svaku netaknutu regiju u koju će se pohraniti LCP i SA polja).

Problem generiranja novih SA polja može se riješiti prolaskom kroz početno SA polje tako da se za svaki element SA[i] ispita kojoj netaknutoj regiji pripada prvi znak sufiksa T[SA[i]] te se u pripadajuću listu umetne element SA[i].

Generiranje novih LCP polja za navedene regije funkcionira na način da se napravi prolaz po odgovarajućem SA polju pripadne regije i zatim se napravi upit koji odgovora na pitanje kolika je duljina zajedničkog prefiksa za SA[i-1] i SA[i]. Odgovor na to pitanje dobiva se u logaritamskoj složenosti koristeći segmentno stablo koje se samo jednom kreira na početku. Segmentno stablo vraća minimum na intervalu između LCP-ova koji se odnose na SA[i-1] i SA[i].

Algoritam 4: Generiranje listi LCP i SA polja za netaknute regije

Data: SA, LCP, konkatenacija nizova T, netaktnute regije REG

Result: listaLCPpolja, listaSApolja

```
for ∀ c ∈ T do
    označiPripadnostOdgovarajućojRegiji();
for ∀ sa ∈ SA do
    listaSApolja.dodaj(sa);
for ∀ regija ∈ REG do
    for ∀ sa ∈ regija.getSA() do
        minimum = segmentnoStablo.getMinimum(sa, sa.getOneBefore());
        listaLCPpolja.dodaj( minimum );
```

4.3. Smith-Waterman

Algoritam Smith-Waterman zasniva se na dinamičkom programiranju te pronalazi najbolje lokalno poravnanje za zadana dva niza, odnosno regije koje će imati najveći rezultat. U ovom programu koristi se kao završna faza u kojoj se poravnavaju netaknute regije.

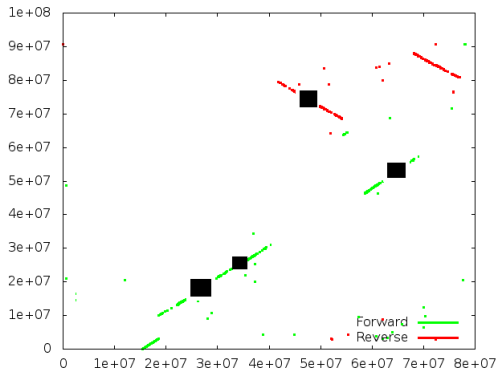
Smith-Waterman algoritam funkcionira po sljedećoj formuli:

$$V(i,j) = max \begin{cases} 0 & i = 0 \parallel j = 0 \\ max \begin{cases} 0 \\ V(i-1, j-1) + w(s_i, t_j) \\ V(i-1, j) + d \\ V(i, j-1) + d \end{cases} & i > 0 \,\&\, j > 0 \end{cases}$$

Pri čemu je:

1. $w(s_i, t_j)$ cijena zamjene
2. d je cijena umetanja ili brisanja
3. V je maksimalna vrijednost između podnizova s i t nad kojima se radi poravnanje na područjima $s[1...i]$ i $t[1...j]$

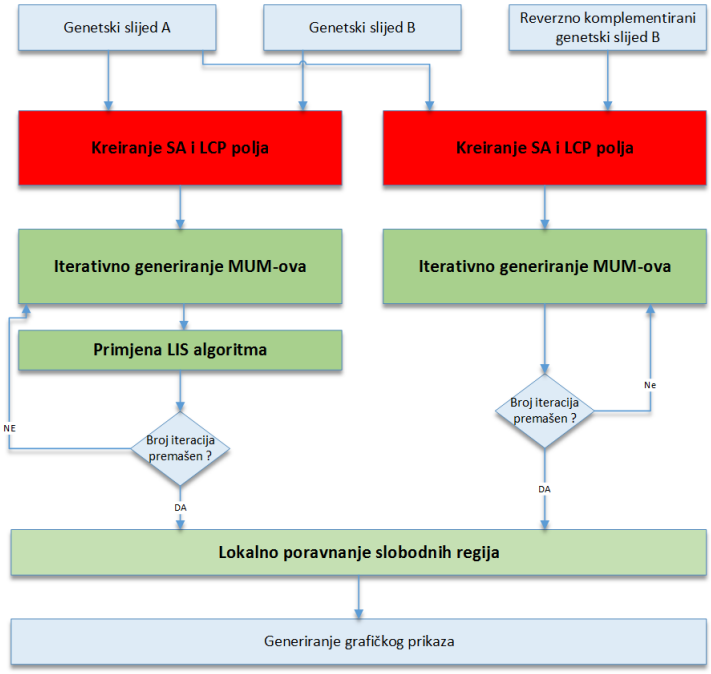
Na slici 4.3 prikazana su područja koja se trebaju poravnati koristeći Smith-Waterman algoritam. Zbog poprilične sličnosti genetskih nizova ta područja nisu pretjerano velika te će se poravnanje izvršiti relativno brzo.



Slika 4.3: Područja za poravnanje Smith-Watermanom

5. Implementacija

Alat koji je tema ovog rada napisan je u programskom jeziku C++ te je realiziran kao višedretveni program. Prilagođen je za izvođenje na Linux platformi. Prilikom pokretanja programa aktiviraju se dvije dretve koje paralelno generiraju sufiksno i LCP polje za normalne i reverzno komplementarne regije genoma. Nakon toga se paralelno izvode i generiraju MUMove te pri tome kreiraju još proizvoljan broj dretvi koje traže MUMove na netaknutim regijama prve iteracije, a zatim vrše Smith-Waterman algoritam.Izložena problematika s navedenim pristupom prirodno je takva da dretve mogu paralelno raditi na odvojenim dijelovima ulaznih nizova tako da nije bilo potrebe za sinkronizacijskim mehanizmima.



Slika 5.1: Dijagram toka opisanog programa

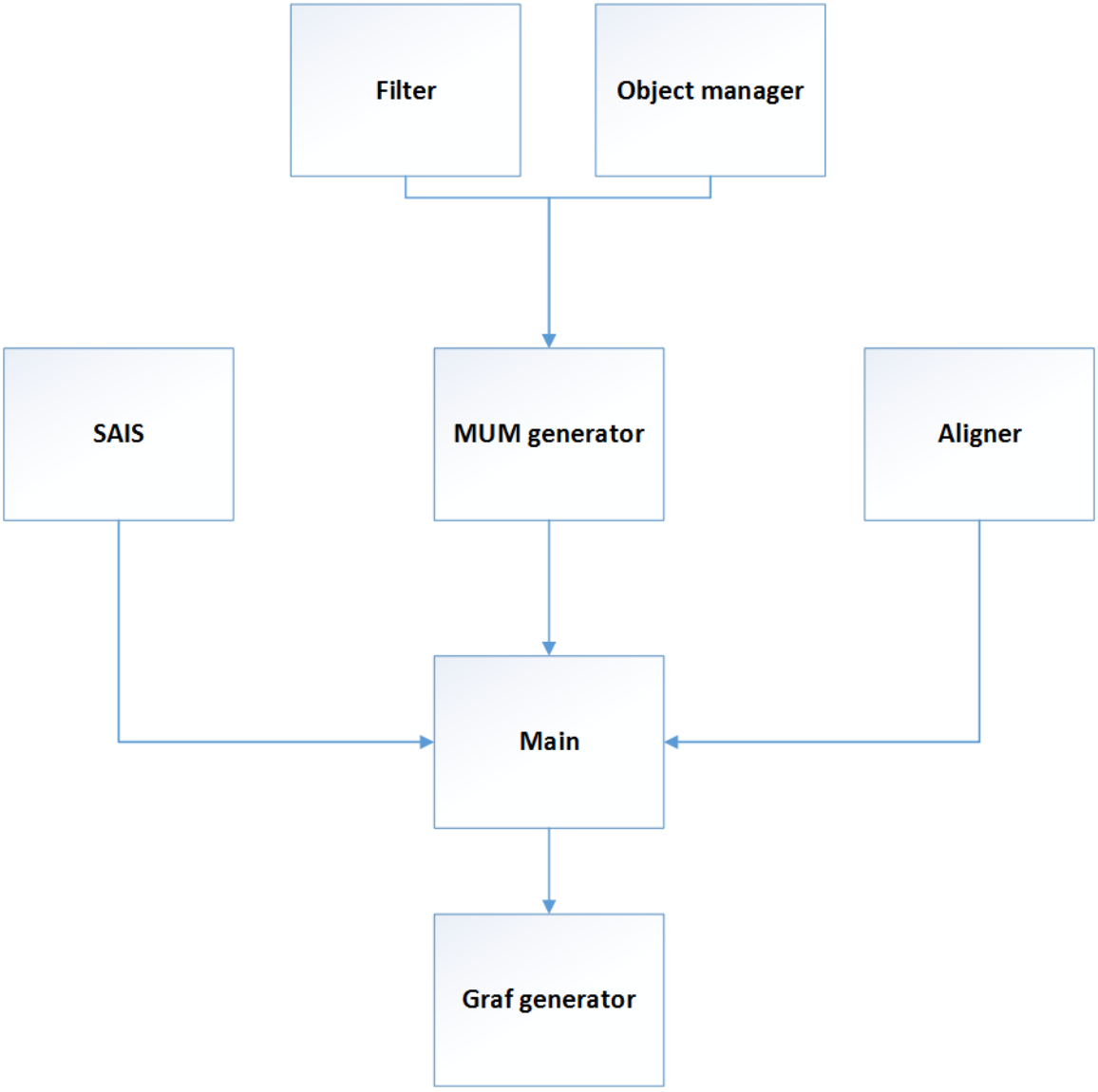
Crveno obojane dijelove na slici 5.1 obavljaju dvije dretve, a zeleno obojane dijelove proizvoljan broj dretvi. Međutim do lokalnog poravnanja slobodnih regija dolazi se tek nakon što su izgenerirani svi MUMovi.

5.1. Arhitektura sustava

Sustav se može promatrati kao skup nekoliko modula. S obzirom da se problematika razlaže na nekoliko manjih komponenti bilo je prirodno rješavati pojedine probleme zasebno. U nastavku je dat i grafički prikaz arhitekture sustava pored tekstualnog opisa.

Opis bitnih modula u sustavu:

1. **Main** - Glavni modul koji prima ulazne parametre te ih prosljeđuje ostalim modulima. Upravlja brojem dretvi te predstavlja pristupnu točku u sustav.
2. **SAIS** - Implementacija konstrukcije sufiksnog polja u linearnoj složenosti. Glavni modul mu predaje konkatenirane stringove, a kao rezultat dobiva SA i LCP polje.
3. **Aligner** - Kao parametre prima stringove i listu pozicija nad kojima treba izvršiti poravnanje. Kao rezultat vraća listu pozicija nad kojima je poravnanje izvršeno.
4. **MUM generator** - Za zadanu konkatenaciju ulaznih nizova te SA i LCP polje vraća listu generiranih MUMova.
5. **Filter** - Kao parametre prima listu MUMova, a za rezultat vraća profiltrirane MUMove.
6. **Object manager** - Kreira segmentno stablo, liste LCP i SA polja vezane uz netaknute regije, sadrži sve bitne strukture podataka i njihove definicije.
7. **Graf generator** - Graf generator je modul koji od Main-a prima set točaka dužina koje treba iscrtati na grafu kao i ostale postavke grafa (izgled, boje, nazivi itd.). Koristi alat "gnuplot" za ispis grafa.



Slika 5.2: Arhitektura sustava

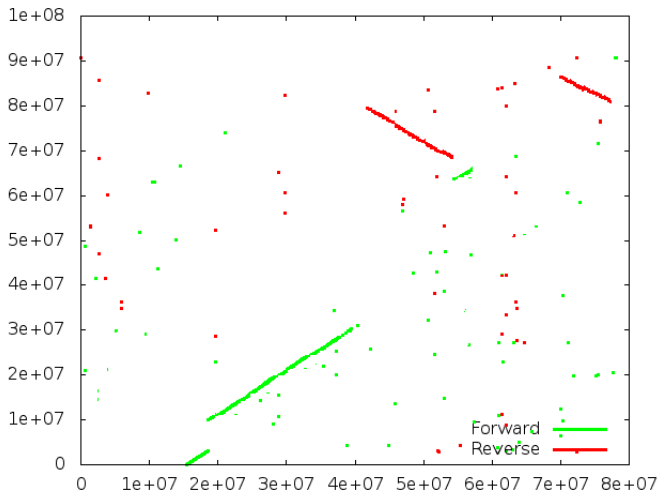
6. Rezultati

Prilikom testiranja alata poput MUMmera i alata opisanog u ovom radu korišteni su genetski sljedovi koji se mogu preuzeti s već navedenog Ensembl poslužitelja. Provedena su testiranja na kromosomima čovjeka i miša čije se duljine mjere u stotinu do dvije stotine milijuna dušičnih baza. Također su rezultati provjereni s onima na web stranici na kojoj su objavljena poravnanja za sve kromosme čovjeka s odgovarajućim kromosomima miša `http://www.broadinstitute.org/annotation/mouse/index.html`. Sva testiranja obavljena su na serverskom računalu s 24 procesorske jezgre po 2.40GHz uz 200 GB radne memorije.

6.1. Ispitni skup

U nastavku su prikazani rezultati poravnanja kromosoma čovjeka s kromosomom miša.

"Miš/Čovjek - 18. kromosom"



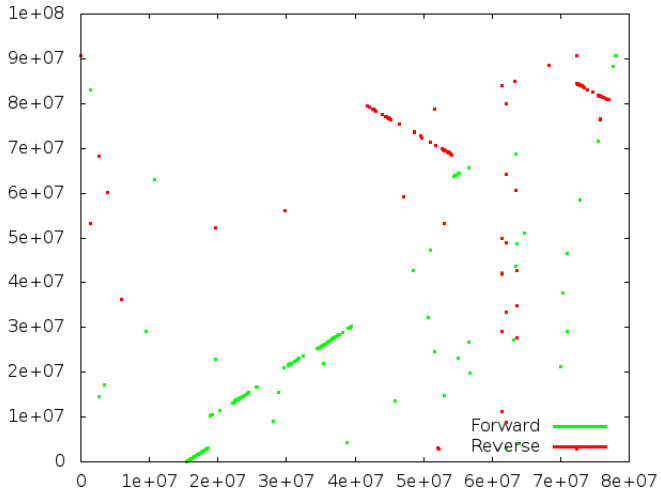
Slika 6.1: Miš/Čovjek - 18. kromosom - kritična duljina 80

Zelenom linijom su prikazani dijelovi koji se poklapaju u genomima koji nisu reverzno komplementirani, dok crvene linije predstavljaju poklapanja referentnog niza

sa reverzno komplementiranim drugim nizom. Na slici se mogu primjetiti MUMovi šuma.

Mišji kromosom 18 je nešto kraće duljine nego čovjekov.

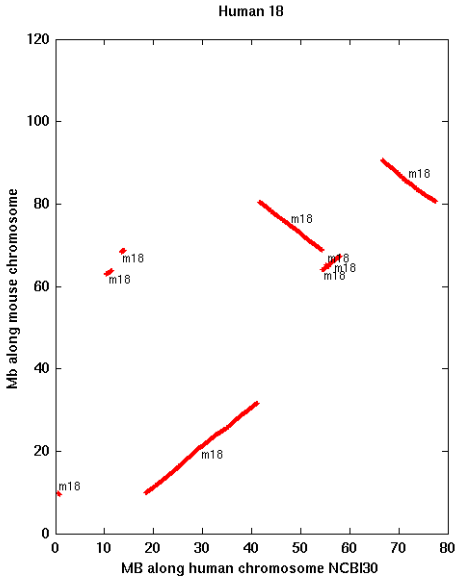
"Miš/Čovjek - 18. kromosom"



Slika 6.2: Miš/Čovjek - 18. kromosom - kritična duljina 120

Može se primjetiti smanjenje MUMova šuma prilikom korištenja znatno veće kritične duljine. Na slici 6.2 prikazan je rezultat pronalaska MUMova nakon prve iteracije.

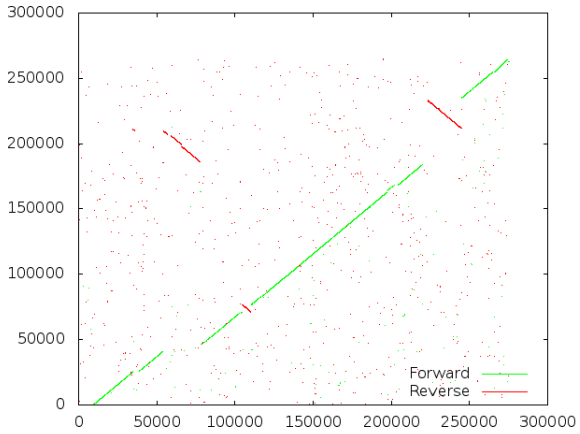
"Miš/Čovjek - 18. kromosom - rezultati s Ensembl poslužitelja"



Slika 6.3: Miš/Čovjek - 18. kromosom - rezultati s Ensembl poslužitelja

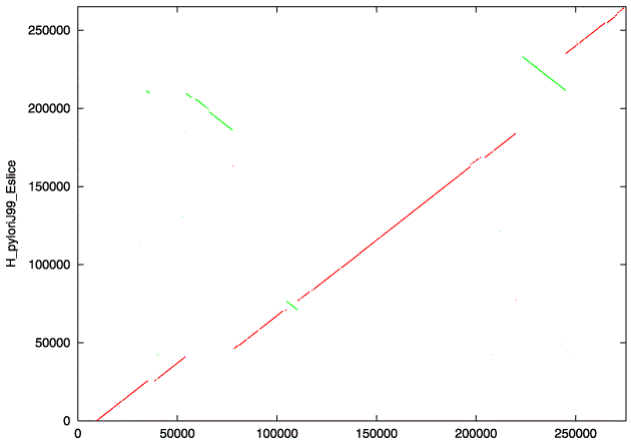
6.2. Usporedba s alatom MUMmer

Alati su testirani na različitim test primjerima. Na primjerima koji su manji od 5M dušičnih baza gotovo da i nema neke značajnije razlike u performansama.



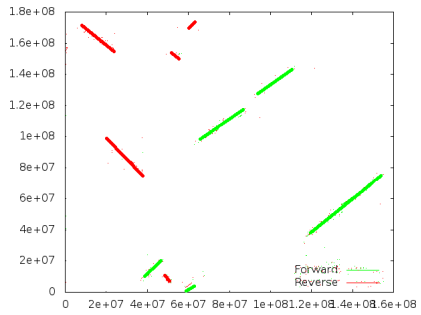
Slika 6.4: Usporedba s MUMmerom - Hpylori - program opisan u radu

Na slici 6.4 vide se crvene točke koje predstavljaju reverzne MUMove, rezultat je takav jer se nad reverznim regijama nije primjenio LIS algoritam, a početna kritična duljina je bila relativno mala.

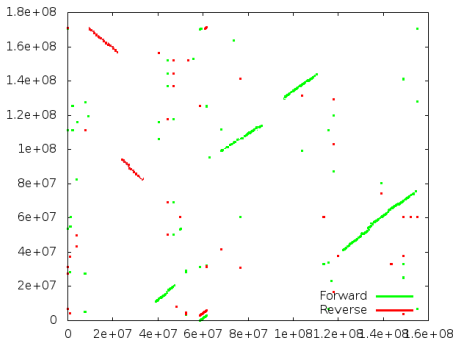


Slika 6.5: Usporedba s MUMmerom - Hpylori - MUMmer

6.2.1. Poravnanje kromosoma X Miš (173.9MB) / Čovjek / (157.9MB):



Slika 6.6: Usporedba s MUMmerom - chrX - rješenje iz rada



Slika 6.7: Usporedba s MUMmerom - chrX - MUMmer

1. **Vrijeme** - Vrijeme za cjelokupno poravnanje kod MUMmera za kromosom X iznosi 99 minuta, a rješenju iz rada treba 20 - 40 minuta ovisno o postavljenim parametrima.

2. **Memorija** - Za dani primjer konkatenacija nizova kromosoma čovjeka i miša zauzima 300MB. MUMmer koristi do 20GB memorije prilikom poravnanja, dok predloženo rješenje iskoristi 25GB. Prilikom konstrukcije sufiksnog stabla MUMmer zauzme 4,8GB dok alat iz rada pri konstrukciji sufiksnog polja za reverzno-komplementirani niz i običan zauzme 5,9GB.

3. **Točnost** - Rješenje opisano u radu poklapa se u prosjeku 90% s MUMmerom ovisno o namještenim parametrima. Procjena je obavljena na temelju generiranja slučajnih primjera veličine 30 milijuna dušičnih baza.

7. Zaključak

Problematika ovog rada bila je poravnanje dvaju genetskih sljedova. Naglasak je bio na razvijanju efikasne metode pomoću koje će se dobiti što bolje i što preciznije rješenje u što kraćem vremenu. Pokazano je da algoritmi koji pronalaze optimalno rješenje nisu efikasni i mogu se koristiti samo na manjim skupovima podataka. Trenutačno najpopularniji alati baziraju se na heuristikama i dosta ih radi po principu da odrede sjemena (*engl. seed*) koja predstavljaju čvrste fiksne točke za daljnji proces poravananja. Ideja koja stoji iza alata u ovom radu bila je riješiti problem slično kao i alat MUMmer, ali uz efikasnije memorijsko zauzeće korištenjem strukture sufiksno polje umjesto sufiksnog stabla te uz korištenje mehanizma paralelizacije. Pokazalo se da to vodi na dobar put, a paralaleizirana verzija alata je za konstantu (broj aktivnih dretvi) brža od neparalelizirane verzije. U budućnosti je potrebno pokušati osmisliti načine gdje i kako još poboljšati ovdje opisan alat te osmisliti matematički model po kojem će se procjenjivati ključni parametri.

LITERATURA

[1] Dan Gusfield. *Algorithms on Strings, Trees, and Sequences Computer Science and Computational Biology*. Cambridge University Press, 1997.

[2] Ge Nong, Sen Zhang, and Wai Hong Chan. *Two Eficient Algorithms for Linear Time Suffix Array Construction*. 1999.

[3] Scott Schwartz, W. James Kent, Arian Smit, Zheng Zhang, Robert Baertsch, Ross C. Hardison, David Haussler, Webb Miller. *Human–Mouse Alignments with BLASTZ*. 2003.

[4] Robert S. Harris. *IMPROVED PAIRWISE ALIGNMENT OF GENOMIC DNA*. 2007.

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

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

[7] Mile Šikić i Mirjana Domazet-Lošo. *Bioinformatika*. 2013.

[8] L Deicher, Kasif, D. Fleischmann, Peterson, White, L. Salzberg. *Alignment of whole genomes*. 1997.

Alat za poravnanje genoma

Sažetak

Poravnanje genoma je jedan od većih problema u bioinformatici koji još uvijek nije efikasno riješen. Iako su poznati egzaktni algoritmi kojima bi se moglo doći do optimalnog rješenja ne koriste se jer bi njihovo izvođenje zahtijevalo previše vremena.

U radu je opisan alat za poravnanje genoma ostvaren kao višedretveni program u programskom jeziku C++. Alat se bazira na pronalasku identičnih podnizova zadanih nizova pomoću sufiksnog polja, a nakon toga na primjeni algoritma Smith-Waterman na područja u kojima nisu pronađeni MUMovi.

Ključne riječi: Poravnanje, genom, sufiksno polje, paralelizacija, heuristika, bioinformatika

Title

Abstract

Genome alignment is one of the major problems in bioinformatics that still have not been effectively resolved. Although exact algorithms which would give optimal solution are known they are not used becouse of their long running time.

In this thesis is described tool for genome alignment realized like multithreaded application written in C++. Tool is based on locating identical interval regions in given sequences by using suffix array, and after that by using Smith-Waterman algorithm on regions in which MUMs were not found.

Keywords: Alignment, genome, suffix array, parallelization, heuristics, bioinformatics