SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 4187

Alat za poravnanje dugačkih očitanja

Marko Ratković

Zagreb, lipanj 2015.

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

Zahvaljujem mentoru Mili Šikiću i Ivanu Soviću na strpljenju i korisnim savjetima.

Zahvaljujem roditeljima i prijateljima na podršci.

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

1. Uvod

Ovaj rad se bavi problemom poravnanja dugačkih očitanja na referentni genom te opisuje razvijeni alat koji rješava taj problem. U kratkom uvodu biti će izložen kratki pregled pojmova iz bioinformatike kako bi se razumjela pozadina problema.

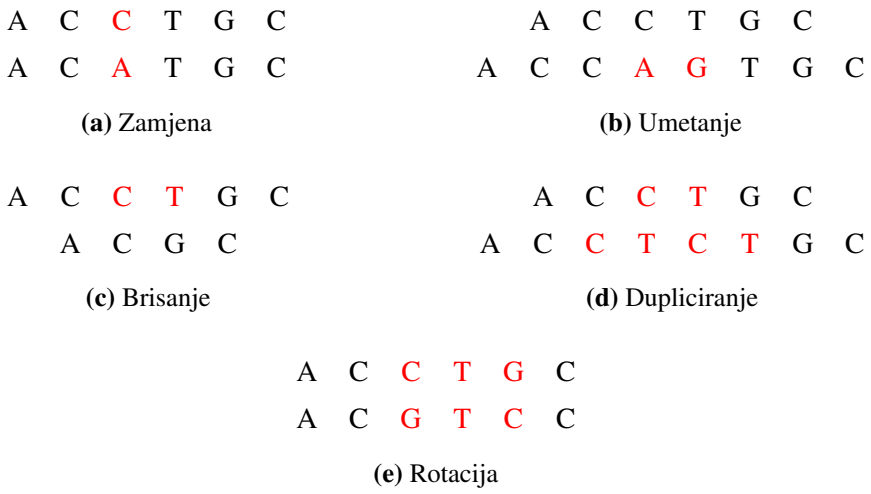
1.1. DNA

Deoksiribonukleinske kiseline (engl. *Deoxyribonucleic acid*) ili skraćeno DNA odnosno DNK su dugački linearni polimeri sastavljeni od nukleinskih kiselina. Nukleinske kiseline (RNA i DNA) sadrže sve genetske informacije stoga je razumijevanje njihove strukture i uloge od temeljne važnosti za razumijevanje nasljednih bolesti, evolucije između vrsta te bilo kakvog tipa bioloških interakcija organizama.

DNA molekule su dvostruke uzvojnice sastavljene od velikog broja povezanih nukleotida od kojih se svaki nukleotid sastoji se od šećera, fosfata i baze. Nukleinske baze ili nukleotidi su adenin, citozin i timin koje označavamo slovima A, C, G i T. Dva lanca su međusobno povezana na način da se adenin spaja s timinom, a gvanin sa citozinom.

S vremenom na molekuli DNA mogu se pojaviti razlicite mutacije. Dominantni tipovi mutacija su:

- **zamjena** - najčešća mutacija, promjena jedne ili više baza s nekom drugom
- **umetanje** - umeće se baza ili slijed baza.
- **brisanje** - iz niza se briše baza ili slijed baza, naziva se još i ispuštanje
- **duplikacija** - ponavljanje sekvence dva ili više puta zaredom, iznimno rijedak tip mutacije
- **rotacija** - okretanje dijela sekvence, iznimno rijedak tip mutacije



(a) Zamjena **(b)** Umetanje **(c)** Brisanje **(d)** Dupliciranje **(e)** Rotacija

Slika 1.1: Mutacije

1.2. Sekvenciranje

Sekvenciranje je metoda kojom se određuje poredak pojedinih gradivnih elemenata u molekuli. Kod DNA i RNA određuje se poredak nukleotida, dok se kod sekvenciranja proteina određuje poredak aminokiselina.

Prvi pokušaj sekvenciranja ljudskog DNA, poznat pod nazivom *Human Genome Project (HGP)*, započet je 1990. godine te je uspješno završio 13 godina poslije. Objavljeni rezultati su pokazali da se ljudski genom sastoji od 3 milijarde parova baza A, C, G i T. Cijena čitavog projekta se procjenjuje 3 milijarde američkih dolara[5].

Rezultati su potakli razvoj metoda sekvenciranja i nagli pad cijena te je danas cijena za sekvenciranje ljudskog genoma pala na svega 1000 dolara[1].

Danas dominantna metoda sekvenciranja je tzv. *shotgun* sekvenciranje. Kod te metode, DNA se lomi na slučajan broj malih fragmenata (duljina do nekoliko tisuća baza). Tako dobiveni fragmenti se očitavaju. Višestruka očitanja se dobiju tako da se početna sekvenca više puta klonira. Računalni programi koristeći preklapajuće krajeve očitanja pokušavaju ih sastaviti u jednu kontinuiranu sekvencu.[11]

Kljucni parametri uređaja za sekvenciranje su duljina očitanja, točnost i naravno cijena. Uređaji su podijeljeni na tri generacije. Pod 3. generaciju spadaju *PacBio* i *Oxford Nanopore* uređaji. Prethodne generacije uređaja pružale su visoku točnost (98-99% *Ian Torrent* i *Illumina*) ali male duljine očitanja (svega nekoliko stotina baza) uz visoku cijenu. 3. generacija uređaja omogućava znatno veće duljine očitanja uz manju

[1] Illumina HiSeq X Ten System `http://www.illumina.com/systems/hiseq-x-sequencing-system/system.html`

cijenu opreme i cijenu samog sekvenciranja, ali uz smanjenje točnosti.
PacBio uređaji omogućavaju prosječne duljine očitanja od 5-10Kbp[2] te maksimalne duljine do 40Kbp. Postotak greške kod takvih očitanja seže do $\sim 15\%$.
Novo predstavljeni *Oxford Nanopore* uređaji omogućavaju čak i duža očitanja. Predstavljeni *Oxford Nanopore MiniON* je dimenzijama malen uređaj koji se spaja s računalom putem USB konektora. Konačna cijena i specifikacije očitanja nastalih takvim uređajem nisu još u potpunosti poznati. Očekuje se da će cijena jednog takvog uređaja biti u rangu 1000 dolara. Dosada objavljeni radovi spominju očitanja reda veličine nekoliko desetaka Kbp pa do stotine Kbp. Podatak o točnosti nije moguće još sa sigurnosti iznijeti jer se mogu naći radovi u kojima se spominje stopa greške i do 30%, ali i radovi koji spominju grešku manju od 15% ili pak 10%[19][16]. Konačni podaci biti ce dostupni u narednih nekoliko mjeseci što je i za očekivati jer se ipak radi o novom uređaju.

1.3. Poravnanje

Poravnanja sljedova odnosno algoritmi ocjenjivanja sličnosti imaju jednu od ključnih uloga u bioinformatici. Njihova zadaća je poravnanje dvaju ili više sljedova DNA, RNA i proteina te davanje ocjene sličnosti na temelju poravnanja. Važnost im leži u tome što njihovi rezultati mogu ukazivati na funkcionalne, strukturalne i evolucijske veze između danih sljedova odnosno samih organizama kojima pripadaju.

Poravnanje omogućava da na temelju visoke ocjene sličnosti ocijenimo sljedove evolucijski srodnima odnosno organizme klasificiramo kao evolucijski srodne što bi daljnjom analizom s drugim sljedovima moglo rezultirati u pronalasku zajedničkih predaka. Poravnanjem mutiranih ili novo otkrivenih DNA, RNA i proteina s sljedovima poznatih svojstava te analizirajući sličnost možemo pretpostaviti svojstva novih sljedova.

1.4. Pregled postojećih rješenja

Sljedove možemo shvatiti kao niz znakova zadanih nad određenom abecedom. Naizgled jednostavan problem ocjenjivanja sličnosti predstavlja velik problem zbog

[2]bp (engl. *base pare*) predstavlja broj nukleinskih parova baza

velike duljine sljedova koji se promatraju. Genomi jednostavnijih organizama dugi su nekoliko milijuna baza dok je genom čovjeka dug 3 milijarde baza. Iz tog razloga i jednostavni algoritmi zahtijevaju ogromne količine resursa (vrijeme i memorija).

Klasični postupci poravnanja (algoritmi Needleman–Wunsch[17] i Smith–Waterman[18]) nisu prihvatljivi kada se radi o jako dugačkim nizovima Izravno poravnavanje korištenjem primjerice *Smith-Waterman* algoritma izravno nad čitavim nizovima nije moguće zbog velike vremenske složenosti. Jedan od načina rješavanja tog problema je na nekakav način izolirati kandidatne regije koje imaju veću vjerojatnost da bas u njima nalazi potencijalno poravnanje na genom. Nakon što su određene regije provedu se spomenuti algoritmi i odabere se najbolje poravnanje.
Postoji čitav niza alata čija je zadaća poravnati očitanja dobivena kao izlaz uređaja za sekvenciranje na referentni genom. Svi ili barem velika većina postojećih alati ovog tipa korištenjem različitih heuristika odabiru kandidatne regije. Najpopularniji pristup je tzv. *seed-and-extend*. Prvo se izgradi nekakva vrsta indeksa koji omogućava pronalazak sljedova duljine k u genomu. Indeks se nakon izgradnje pohrani na disk kako bi se mogao koristiti koristiti više puta bez potrebe na ponovnom izgradnjom.
Očitanja se obraduju na način da se promatra svaki niz od k uzastopnih znakova. Takvi nizovi se često nazivaju k-torke (engl. *kmer*). Koristeći prethodno izgrađeni indeks, pronađu se pozicije pojavljivanja k-torka u genomu. Takve pozicije su zapravo začeci (engl. *seed*) poravnanja. Promatra se područje oko začetaka te pokušava proširiti (engl. *extend*) u slučaju da se okolina poklapa s bazama očitanja. Na taj način se formiraju kandidatne regije. Koriste se različite ocijene kvalitete kako bi se odabrale selektirale potencijalnije kandidatne regije.

Najpoznatiji alati ove primjenu su *BWA-MEM*, *BWA-SW*, *SNAP*, *Bowtie* i *LAST* [8][9][20]. Postojeći alati su uglavnom prilagođeni za poravnanje kratkih očitanja s malim postotkom pogreške. Problem s javlja kada je potrebno poravnati očitanja nastalima sekvenciranjem treće generacije. Neki od alata(*BWA-MEM* i *LAST*) su nadograđeni kako bi podržavali i način rada prilagođen dugačkim očitanjima. Još uvijek postoji problem nedovoljne preciznosti postojećih alata te dugog vremena izvođenja za takva očitanja.

Cilj ovog rada implementirati alat koji ce biti prilagođen poravnavanje dugačkih očitanja nastalih uređajima za sekvenciranje kao što su *PacBio* i *Oxford Nanopore* koji imaju velik postotak greške (10% do 30%) vodeći računa o točnosti i vremenu izvođenja. Moja izvedba takvog alata predložena je u poglavlju 3.1, dok je u poglavlju 4 moguće vidjeti rezultate usporedbe s nekima od gore spomenutih alata.

2. Algoritmi i strukture

U ovom poglavlju dan je pregled algoritama i struktura podataka bitnih za razumijevanje ideje i implementacije alata za poravnavanje koji ovaj rad obraduje.

2.1. Sufiksno polje

Sufiksno polje je jednostavna struktura koja omogućava pronalazak svih pozicija traženog podniza u ulaznom tekstu.
Predstavljeno je 1990. godine od dvojca Manber, Myers [10] kao memorijski manje složena alternativa sufiksnom stablu (strukturi podataka koja rješava isti problem pronalazaka pojavljivanja podniza).
Sufiksno polje je u suštini jednostavno polje ne-negativnih cijelih brojeva koji predstavljaju početne pozicije abecedno poredanih sufiksa ulaznog teksta.

Definicija: Neka $S = S[1]S[2]....S[n]$ označava ulazni niz gdje $S[i]$ predstavlja znak na poziciji i u nizu. S_i označava sufiks koji počinje na poziciji i.

$$S_i = S[i]S[i+1]S[i+2]...S[n]$$

Sufiksno polje SA izgrađeno nad nizom S je definirano na način da element $SA[i]$ sadrži i-ti po leksikografskom poretku sufiks od niza S. Odnosno za svaki i, $1 < i \leq n$ vrijedi relacija:

$$S_{SA[i-1]} < S_{SA[i]}$$

Primjer
Uzmimo da je ulazni niz $'banana'$. Na kraj niz dodajemo jedinstveni znak \$ koji se ne nalazi u abecedi Σ nad kojom je zadan ulazni niz. Pretpostavimo da je \$ leksikografski

manji od svih znakova iz Σ.

$$S = banana\$$$

Tablica 2.1: Sufiksi niza

(a) Po poziciji u nizu

Sufiks	Indeks
$banana\$$	1
$anana\$$	2
$nana\$$	3
$ana\$$	4
$na\$$	5
$a\$$	6
$\$$	7

(b) Leksikografski sortirani

Sufiks	Indeks
$\$$	7
$a\$$	6
$ana\$$	4
$anana\$$	2
$banana\$$	1
$na\$$	5
$nana\$$	3

Nakon leksikografskog sortiranja sufiksa dobijemo poredak prikazan tablicom 2.1b. Sufiksno polje čine upravo početne pozicije tako sortiranih sufiksa što je i prikazano tablicom 2.2.

Tablica 2.2: Sufiksno polje

i	1	2	3	4	4	6	7
$SA[i]$	7	6	4	2	1	5	3

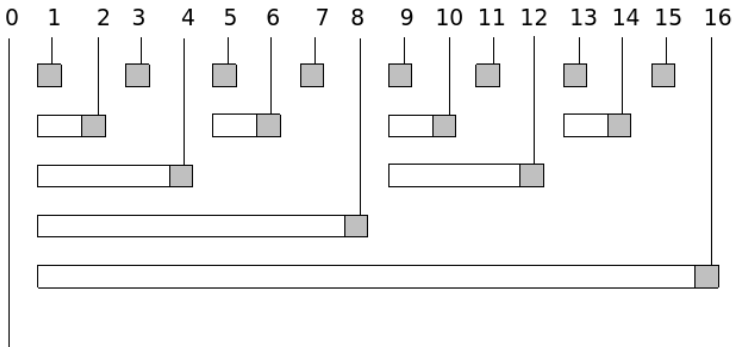
Naivno rješenje izgradnje sufiksnog polja rezultiralo bi u složenost $O(|S|^2 log|S|)$. Korištenje binarne pretrage za pronalazak najdužeg zajedničkog prefiksa dvaju sufiksa u kombinaciji s hash funkcijom koja bi u $O(1)$ vremenskoj složenosti davala odgovor na pitanje dali su dva niza jednaka rezultiralo bi poboljšanjem složenosti na $O(|S|log^2|S|)$[3].
Postoji čitav niz sofisticiranih algoritama koji omogućavaju izgradnju u linearnom vremenu [7][13]. Postoje još i algoritmi s vremenskom složenosti $O(|S|log|S|)$ u najgorem slučaju koji su se pokazali u praksi brži od linearnih[13].
Memorijska složenost većine algoritama je linearna. Za pohranu sufiksanog polja i

ulaznog niza velika većina algoritama zahtjeva približno $5n$ okteta memorijskog prostora.

Algoritam za traženje svih pojavljivanja niza T u ulaznom tekstu S zasniva se na dvije binarne pretrage koje određuju raspon pozicija sufiksnog polja. Na tim pozicijama zapisane su vrijednosti koje odgovaraju početnim pozicijama u ulaznom nizu. Vremenska složenost takvog pristupa je $O(|T|log|S|)$.
Moguće je postići vremensku složenost $O(|T| + log|S|)$ koristeći informaciju o najdužem zajedničkom prefiksu dvaju podnizova. Algoritam su predložili Manber i Myers u svom radu [10].

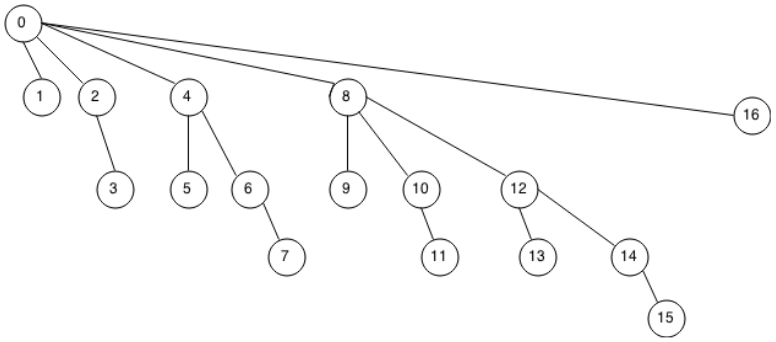
2.2. Fenwickovo stablo

Fenwickovo stablo[2](poznato po nazivima logaritamska struktura i *binary indexed tree*) je struktura koja omogućuje u složenosti $O(logn)$ ubacivanje elemenata te odgovaranje na pitanje: *'Koliko je suma elemenata od indeksa a do indeksa b unutar strukture?'*.
Struktura je organizirana na način prikazan na slici 2.1. Svaki element zadužen je za pamćenje sume iz nekog intervala stabla.
Primjerice element na indeksu 6 pamti kolika je suma elemenata ubačenih na indeks 5 i 6, element na indeksu 8 pamti sume elemenata ubačenih na indekse 1 do 8.
Fenwickovo stablo na indeksu i pamti kolika je suma u intervalu $[i - 2^t + 1, i]$, gdje je 2^t najmanja potencija broja dva sadrzana u broju i.



Slika 2.1: Dijagram odgovornosti

Kako bi smo dohvatili sumu elemenata iz intervala $[1, x]$ potrebno je odrediti sumu



Slika 2.2: Stablo ispitivanja

onih elemenata strukture koji zajedno pokrivaju čitav zadani interval. Primjerice ukoliko želimo sumu iz intervala [1, 7] potrebno je vratiti sumu vrijednosti zapisanih u elementima 7, 6 i 4 (indeks 7 zadužen samo za sebe, 6 zadužen za 5 i 6, 4 zadužen za interval 1 do 4).

Redoslijed ispitivanja prikazan je na slici 2.2. Kako bi se pokrilo cijelo područje potrebno je od zadanog čvora doći do korijena (od čvora 7 do korijena na putu su čvorovi 7, 6 i 4 kao što je pokazano u prethodnom primjeru). Kako bi se dobila suma za interval $[1, x]$ potrebno je od čvora x doći do korijena i sumirati vrijednosti u čvorovima na putu. Faktor grananja svakog čvora jednak je broju pratećih nula ((engl. *trailing zeroes*))u binarnom zapisu.

Ukoliko želimo dohvatiti sumu iz intervala $[a, b]$ to ćemo učiniti na način da dohvatimo sumu iz intervala $[1, b]$ i od nje oduzmemo sumu iz intervala $[1, a - 1]$:

$$\sum_{i=a}^{b} A[i] = \sum_{i=1}^{b} A[i] - \sum_{i=1}^{a-1} A[i]$$

za $1 < a \leq b \leq n$, n broj elemenata koji se pohranjuju u strukturu.

Kod ubacivanja elementa u strukturu, kako je vidljivo iz slike 2.1, nije dovoljno ubaciti na taj indeks, već je potrebno uvećati i sve one indekse koji su odgovorni za taj. Krenemo od indeksa koji je jednak indeksu na koji ubacujemo vrijednost. Uvećamo taj element strukture za zadanu vrijednost, te indeks uvećamo za njegovu najmanju potenciju broja dva. Time smo dobili slijedeći čvor koji je odgovoran za čuvanje vrijednosti. To ponavljamo dok ne izađemo iz granica polja.

Čitav algoritam ubacivanja i dohvaćanja sume na intervalu dan je u nastavku. $lobit(i)$ predstavlja operaciju dohvaćanja najmanje potenciju broja dva zadanog broja i.

Algoritam 1 Operacije Fenwickovog stabla za odredivanje sume

function LOBIT(i)
    return $i\& \sim i$
end function

function DODAJ(i, x)
    if $i >$ $S.velicinaPolja$ **then** return
    end if
    $S_i \leftarrow S_i + x$
    Dodaj($i + lobit(i), x$)
end function

function UPIT(b)
    if $b ==$ 0 **then** return 0
    end if
    return S_b + Upit(b - lobit(b))
end function

Fenwickovo stablo često se koristi u brojnim algoritmima za optimiziranje unutrašnjih petlji. Osim računanja sume, struktura se može prilagoditi da omogućuje upite oblika *'Koliko je elemenata u strukturi je manje od x?'* ili *'Koji je maksimalna vrijednost nad intervalom $[1, b]$?'*.
U nastavku je prikazana prilagodba operacija ubacivanja u strukturu i upita kako bi se omogućilo dohvaćanje maksimalne vrijednosti u intervalu $[1, b]$.

Algoritam 2 Operacije Fenwickovog stabla za odredivanje maksimalne vrijednosti

function DODAJ(i, x)
  if $i >$ $S.velicinaPolja$ **then** return
  end if
  $S_i \leftarrow Max(S_i, x)$
  Dodaj($i + lobit(i), x$)
end function

function UPIT(b)
  if $b ==$ 0 **then** return 0
  end if
  return Max(S_b, Upit(b - lobit(b)))
end function

2.3. Kvantificiranje sličnosti dvaju nizova

U nastavku su analizirani neki od algoritama koji mogu služiti kao praktična mjera sličnosti dvaju nizova.

2.3.1. LCS

Definicija: Označimo s X i Y dva znakovna niza, *LCS*(engl. *longest common subsequence*) ili najduža zajednička sekvenca predstavlja dužinu najduljeg ne nužno povezanoga podniza koji se javlja i u X i u Y.

Primjer:
Uzmimo kao primjer nizove $AGCAT$ i $TGAC$. Moguće je odmah primijetiti da je najduža sekvenca koja se javlja u oba niza duljine 2, a to mogu biti sekvence GA, AC, GC.
Način na koji je moguće riješiti ovaj problem je korištenjem dinamičkog programiranja. S $dp(i, j)$ označimo dužinu najdulje zajedničke sekvence prefiksa $X_{1..i}$ i $Y_{1..j}$. Dodatno definirmo početne uvijete $dp(0, j) = 0, \forall j$ i $dp(i, 0) = 0, \forall i$. Tada je moguće

postaviti slijedeću rekurzivnu relaciju:

$$dp(i,j) = max \begin{cases} 0 & \\ dp(i-1,j) & ,i \geq 1 \\ dp(i,j-1) & ,j \geq 1 \\ dp(i-1,j-1)+1 & ,X[i] = Y[j] \end{cases}$$

Duljina najdulje zajedničke sekvence je pohranjena u $dp(|X|,|Y|)$. I vremenska i memorijska složenost su jednake $O(|X| \cdot |Y|)$.
Moguće je smanjiti memorijsku složenost na linearnu u ovisnosti o zbroju dužina podnizova korištenjem *Hirschbergova* algoritma na rekonstrukciju rješenja [4].
Ukoliko je potrebno pamtiti samo duljinu bez informacije o tome koji su nizovi zajednički, tada je moguće memorijsku složenost smanjiti na $O(min(|X|,|Y|))$ jer je potrebno pamtiti samo prethodni redak za izračun trenutnog.

Tablica 2.3: LCS tablica na nizove $AGCAT$ i $TGAC$

	∅	**A**	**G**	**C**	**A**	**T**
∅	0	0	0	0	0	0
T	0	0	0	0	0	1
G	0	0	1	1	1	1
A	0	1	1	1	2	2
C	0	1	1	2	2	2

2.3.2. LCSk++

LCSk++[14] predstavlja generalizaciju *LCS* algoritma. Predstavlja mjeru sličnosti dvaju nizova a definira se kao duljina najduže zajedničke sekvence čiji su najkraći kontinuirani dijelovi duži ili jednaki vrijednosti k.

S $dp(i,j)$ označimo $LCSk++(X_{1...i}, Y_{1...j})$, gdje su $X_{1...i}$ i $Y_{1...j}$ prefiksi nizova X i Y. Definiramo početne uvjete $dp(0,j) = 0, \forall j$ i $dp(i,0) = 0, \forall i$. Moguće je postaviti

slijedeću rekurzivnu relaciju:

$$dp(i,j) = max \begin{cases} 0 & \\ dp(i-1,j) & ,i \geq \\ dp(i,j-1) & ,j \geq 1 \\ dp(i-q,j-q)+q & ,\forall q \geq k \; s.t. \; X_{i-q\ldots i-1} = Y_{j-q\ldots j-1} \end{cases}$$

$dp(i-1,j)$ i $dp(i,j-1)$ predstavljaju preuzimanje prethodno određenog maksimuma dok $dp(i-q,j-q)+q$ za slučaj kad je prethodnih q znakova jednako, odgovara pokušavaju proširenja $LCSk++(X_{1\ldots i-q-1}, Y_{1\ldots j-q-1})$ s slijedećih q jednakih znakova. Rezultat za nizove X i Y je pohranjena u $dp(|X|,|Y|)$.

Naivna implementacija po definiciji bi rezultirala vremenskom slozenošću $O(|X| \cdot |Y \cdot |min(X|,|Y|))$.
Efikasnija implementacija je predložena u [14] te se zasniva na pronalasku svih neprekinutih podnizova duljine k koji se javljaju oba niza. Svaki pronađeni podniz predstavi se parom (i,j) gdje je i pozicija na kojoj popinje taj podniz u nizu $|X|$, a j početak u nizu $|Y|$. Takvi parovi se sortiraju te se algoritam provodi linearno u ovisnosti o broju parova.
Definiraju se relacije $prethodi(P,G)$ i $nastavlja(G,P)$ nad parovima $P = (i_p, j_p)$ i $G = (i_g, j_g)$

Definicija: Neka su P i G prethodno opisani parovi. Kažemo da vrijedi $prethodi(P,G)$ ako vrijede slijedeći izrazi:

$$i_p + k \leq i_g$$

$$j_p + k \leq j_g$$

Definicija: Kažemo da vrijedi $nastavlja(G,P)$ ako $i_p - j_p = i_g - j_g$ i ako vrijedi:

$$i_g - i_p = 1$$

Korištenjem tako definiranih pojmova možemo iskazati rekurzivnu relaciju za $dp(P)$,

odnosno $LCSk++(X_{1...i}, Y_{1...j})$

$$dp(P) = max \begin{cases} k & \\ max_G(dp(G)) + k & \forall G,\ prethodi(P, G) \\ dp(G) + 1 & \exists G,\ nastavlja(G, P) \end{cases}$$

Korištenjem prethodno opisane strukture Fenwickovog stabla moguća optimizacija pri traženju maksimalne izračunate vrijednost do određenog indeksa (određivanje drugog člana u gornjem izrazu). Vremenska složenost algoritma predloženog u u [14], koji koristi spomenutu optimizaciju, je $O(m+n+rlog(r)+rlog(n))$. Prostorna složenost $O(m+n+r)$, gdje su m i n duljine ulaznih nizova, a r broj neprekinutih podnizova duljine k koji se javljaju oba niza.

3. Implementacija

3.1. Ideja rješenja

Razvijeni alat jednostavnog naziva *MAPPER* koristi prethodno opisanu strukturu sufiksno polje kao indeks pretrage. Sufiksno polje je odabrano iz razloga što omogućava promjenjivu vrijednost k (duljina *seed*-a), što nije slučaj kod raznih *hash* struktura. Također pruža kompromis između vremenske složenosti pretraživanja i memorijskog zauzeća. Memorijsko zauzeće je približno 5 puta veće od genoma nad kojim je izgrađeno. Valjda napomenuti da je implementacija jednostavnija nego primjerice sufiksnog stabla ili pak FM-indeksa.
Sam proces poravnanja očitanja odvija se u tri faze:

- pronalazak k-torki
- određivanje kandidatnih regija
- poravnanje očitanja na odabrane regije te selektiranje one s najboljom ocjenom

Problem klasičnoga načina pronalaženja svih pozicija pojavljivanja svake k-torke je u tome što ukupan broj pozicija može biti izrazito velik. Ukoliko odaberemo k koji je prevelik, zbog greški u očitanjima, gubimo značajan broj pozicija te u samoj konačnici gubimo i na preciznosti. Ukoliko je s druge strane k premali, tada imamo velik broj pozicija te potencijalno veću preciznost, ali smo time povećali vrijeme izvođenja. Odabir male vrijednosti k u kombinaciji s dugačkim genomom i očitanjima s visokim postotkom greške dovodi do javlja se velikog broja *slučajno točnih* pojavljivanja k-torke te se time dodatno povećava vremensko izvođenje.
Kako bi riješio problem odabira povoljne vrijednosti parametra k odlučio sam se na pristup kod kojeg mijenjam k ovisno o broju pojavljivanja. Zadaje se početna vrijednost parametra k preko parametra k_poc te se k-torka traži u genomu korištenjem sufiksnog polja. Ukoliko je broj pojavljivanja veći od zadanog parametra h, k se inkrementira sve dok broj pojavljivanja ne padne ispod vrijednosti parametra h. Ukoliko je broj pojavljivanja pao ispod minimalnoga broja koji je zadan parametrom l, tada se

uzimaju pozicije iz prethodnog koraka (za k - 1).
Nakon što su dobivene pozicije za trenutnu k-torku, kreću se određivati pozicije slijedeće s vrijednosti parametra $k = min(k\_poc, k-2)$. Svaku takvu k-torku jednoznačno predstavljam trojkom (i, j, k), gdje je i pozicija početka u genomu, j početna pozicija u očitanju i k duljina.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
C	A	G	C	G	T	T	T	C	G	C	C	G	C
		C	G	G	T	T	T	C	A	C			
		0	1	2	3	4	5	6	7	8			

Slika 3.1: k-torka koja odgovara trojki $(4, 2, 5)$

Kako bi se odredile kandidatne regije potrebno je pronaći one dijelove genoma koji imaju gušće raspoređene mapirane k-torke.
Nazovimo regijama svaki podniz genoma ćija je duljina dvaput veća od duljine očitanja. Ideja je klizati pomični prozor po genomu te grupirati sve one trojke koje upadaju u određenu regiju. Nad takvom regijom izračunati vrijednost $LCSk++$.
Prva faza računanja *LCSk++* (traženje zajedničkih kontinuiranih podnizova duljine k) može se preskočiti jer već imamo određene pozicije te ih kao takve predajemo algoritmu.
Nema potrebe pomicati prozor po jednu bazu i za svaku regiju (svaki podniz) računati $LCSk++$ već je dovoljno *skupljati* trojke koje koje pripadaju istoj regiji. Sortiraju se trojke po prvoj vrijednosti (pozicija početka u genomu) te se koristi se varijanta *sweep line* algoritma. Na taj način pomični prozor ne pomičemo zapravo po genomu već po trojkama čiji je broj znatno manji od broja baza genoma.
Kako se koristi varijabilan k, parovi koji se koriste u algoritmu za računanje $LCSk++$ spomenutno u poglavlju 2.3.2 i objavljenom u radu [14] prelaze u trojke jer se pamti i duljina zajedničkog podniza. Potrebno modificirati spomenuti alogoritam na način da umjesno fiksnog k, koristi vrijednost pohranjenu u svakoj od predanih trojki.
Za sve prođene regije računa se ocjena sličnosti (vrijednost $LCSK++$) te se kao kandidatne uzimaju one čija je ocjena do $X\%$ lošija od najbolje ocjenjene regije za to očitanje. Maksimalan broj regija koje se uzimaju kao kandidatne i spomenuti X su parametri koji se predaju programu.
Neka m označava duljina genoma, n duljina očitanja, a r broj trojku koje predstavljaju broj neprekinutih podnizova, duljina većih ili jednakih početnom k, koji se pojavljuju

oba niza. Vremenska složenost računanja $LCSk++$ je $O(m+n+rlog(r)+rlog(n))$, a kako pomičemo prozor po trojkama kojih ima r tada dolazimo do najgoreg slučaja vremenske složenosti $O(rm + rn + r^2log(r) + r^2log(n))$.
Kako je računanje $LCSk++$ vremenski skupa operacija pokušava se smanjiti broj računanja iste te preskočiti one regije za koje je moguće odmah zaključiti da neće moći rezultirati ocjenom koja je do $X\%$ lošija od najbolje. Ukoliko znamo da je trenutno najbolja ocjena max, a regija koju promatramo ima zbroj duljina trojki koje njoj pripadaju manji od $(1 - X/100) \cdot max$. Najveća moguća vrijednost $LCSk++$ koju takva regija može dobiti jednaka je upravo zbroju duljina trojki(i to samo u slučaju da se niti jedna trojka ne poklapa s drugom). Ako je taj zbroj manji od spomenute vrijednosti $(1 - X/100) \cdot max$ možemo zaključiti da ta regija nikako neće imati vrijednost $LCSk++$ unutar $X\%$ od trenutne najbolje vrijednosti. Za takvu regiju možemo preskočiti računanje $LCSk++$.

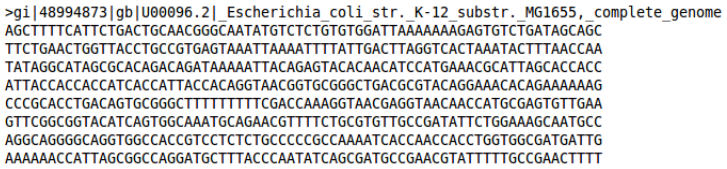
Završna faza se svodi na računanje poluglobalnog poravnanja očitanja za najbolje kandidatne regije i odabira one koja ima najmanju cijenu poravnanja. Takvu poziciju proglasimo poravnanjem.
Preciznost ovakve izvedbe uvelike ovisi o odabiru odabiru početne vrijednosti parametra k te gore navedenih parametara l i h koji izravno utječu na broj pozicija koje su uzimaju u obzir. O tome ce biti više riječ u poglavlju 4 pri analiziranju rezultata testiranja.

3.2. Formati datoteka

Program na svom ulazu prima referentni genom na koji vrši poravnanje. Ulaz se daje u obliku FASTA datoteke (ekstenzija *.fasta* ili *.fa*). Format je opće prihvaćen u bioinformatičarskim krugovima. Za prikaz sekvenci koriste se slova koja predstavljaju nukleinske baze, ali se također koriste i dodatna slova u slučajevima kad se ne može jednoznačno odrediti o kojoj bazi je riječ. Zapis se sastoji od zaglavlja - linije koja započinje znakom $>$ i jedne ili više linija slijeda. U datoteci može biti više od jednog zapisa. Čitava specifikacija dostupna je na stranicama *NCBI*[1]. Slika 3.2 prikazuje primjer jedne datoteke.

Očitanja koja se poravnaju daju se u FASTQ datoteci. Radi se o tekstualnoj

[1] `http://www.ncbi.nlm.nih.gov/BLAST/blastcgihelp.shtml`

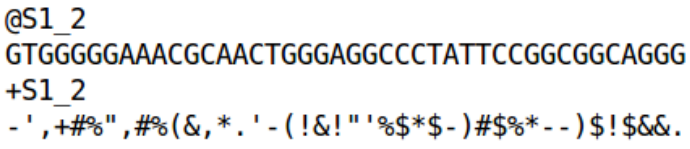


Slika 3.2: Primjer FASTA datoteke

datoteci koja uz identifikator i slijed sadrži još i podatak o kvaliteti svake pojedine baze slijeda. Datoteka najčešće sadrži više zapisa sljedećeg oblika:

1. započinje znakom @ nakon kojeg slijedi identifikator slijeda te opis koji nije obavezan
2. slijed
3. zapocinje znakom + nakon koje slijedi identifikator
4. zapis o kvaliteti za svaku bazu

Kvaliteta predstavlja vjerojatnost točnog očitanja te je generirana prilikom sekvenciranja. Prikazana je ASCII znakovima u rasponu od 33('!') do 126(' ') gdje veća ASCII vrijednost predstavlja višu kvalitetu očitanja. Na slici 3.3 dan je primjer jedne takve datoteke.



Slika 3.3: Primjer FASTQ datoteke

Izlaz *MAPPER*-a su informacije o pozicijama unutar referentne sekvence na kojoj se javlja očitanje. Alat informacije o svim očitanjima, pozicijama, podatku o orijentaciji(ocitanje se moze mapirati kao referzni komplement) i poravnanju na izlazu daje u SAM formatu. SAM (S*equence Alignment/Map*) format je generički format za spremanje poravnanja. Sastoji se od zaglavlja koje sadrži informacije o referentnom slijedu (naziv, dužina, identifikator...), alatu za poravnanje (naziv, argumenti s kojima

je pozvan, opis, verzija) te linije za svako poravnanje. Zapis poravnanje se sastoji od imena, cijelobrojne vrijednosti koja predstavlja bit mapu s informacijama o poravnanju (mapirano - ne mapirano, radi li se o reverznom komplementu, radi li se o prvom ili zadnjem zapisu...), ime referentnog slijeda na koji se obavilo poravnanje, indeks početka odnosno pozicije u slijedu, ocijene kvalitete poravnanja u rasponu od 0-255, CIGAR niz, duljine, slijeda i zapisa o kvaliteti očitanja preuzetih iz zapisa u FASTQ datoteci. Osim same pozicije bitan podatak je CIGAR koji predstavlja zapis o poravnanju, točnije zapis koji govori dali su se na određenoj poziciji baze referentnog slijeda i očitanja jednak, različite ili se radi o umetanju ili brisanju. Detaljne informacije o SAM formatu i CIGAR-u mogu se pronaći na stranicama *SAMTools*[2]

3.3. Programska izvedba

Alat je razvijen u programskom jeziku C++ po C++11 standardu. Čitav kod je dokumentiran i stavljen na Github pod MIT licencom i dan na slobodno korištenje.
Od dodatnih biblioteka potrebno je imati samo *libdivsufsort*. *libdivsufsort* pruza brzu izgradnju sufikskog polja (vremenska složenost $O(nlog(n))$, memorijska O(5n)). Upute za instalaciju te dodatne informacije kao i rezultati usporedbi s sličnim bibliotekama dostupni su na Github stranici[3].
Korištena je i biblioteka *edlib*[4] koja omogućava brzo poravnavanje dvaju sekvenci korištenjem algoritma baziranog na ideji izloženoj u radu [12]. Koristi se u završnom koraku se za traženje poluglobalnog poravnanja očitanja na preostale kandidatne regije.
Iskorištena je *kseq*[5] mala biblioteka za jednostavan i brz rad s FASTA i FASTQ datotekama. *edlib* i *kseq* su objavljene pod MIT licencom te dolaze u sklopu alate pa nije potrebna dodatna instalacija.
Program podržava paralelno izvođenje na razini mapiranja očitanja odnosno paralelno poravnavanje više očitanja. Za paralelizaciju se koristi *OpenMP*. To je aplikacijsko programsko sučelje (engl. *application programming interface*) koje omogućava putem jednostavne sintakse (*pragma* direktive) kreiranje paralelnih regija i čitavog niza sinkronizacijskih mehanizama. U radu je pomoću OpenMP-a implementiran mehaniza

[2] `https://samtools.github.io/hts-specs/SAMv1.pdf`
[3] `https://github.com/y-256/libdivsufsort`
[4] `https://github.com/Martinsos/edlib/`
[5] `http://lh3lh3.users.sourceforge.net/kseq.shtml`

proizvođač-potrošač s redom poslova. Jedna dretva predstavlja predstavlja proizvođača koji učitava očitanja iz FASTQ ulazne datoteke te ih stavlja u red poslova, dok preostale dretve su potrošači te uzimaju prvi slobodan posao(očitanje) iz reda poslova te ga poravnaju na referentni genom. Za samu sinkronizaciju i štićenje kritičnog odsječka zadužen je *OpenMP*.

3.3.1. Korištenje

Stvaranje indeksa

Indeks odnosno sufiksno polje koje se koristi za brzo pretraživanje referentnog slijeda izgrađuje se sljedećom naredbom.

```
mapper –m index <referenti_slijed_fasta> <indeks>
```

Vrijeme izgradnje indeksa ovisi o veličini slijeda. Detaljnije o tome u poglavlju 4. Memorijsko zauzeće indeksa je 4 puta veće od veličine slijeda nad kojim je izgrađen (32 bitne vrijednosti korištenje za spremanje indeksa, ne sprema znakove slijeda). Indeks je dovoljno jednom izgraditi za određeni slijed izgraditi jednom te se prilikom poravnanja koristi prethodno izgrađeni indeks.

Poravnanje očitanja

Očitanja zadana kao zapisi u FASTQ datoteci se poravnavaju na referentni slijed korištenjem slijedeće naredbe.

```
mapper –m map [opcije] <referenti_slijed_fasta> <index>
                  <ocitanja_fastq> <sam_izlaz>
```

Opcije predstavljaju dodatne parametre koje je moguće specificirati kako bi se podesio željeni omjer preciznosti odnosno točnosti poravnanje i vremena izvođenja. Sve zastavice i njihova uloga su već prethodno spomenute u poglavlju 3.1.

Tablica 3.1: Dodatne zastavice

Argument	Značenje	Pretpostavljena vrijednost
-t N	broj dretvi	broj jezgri
-k N	početnu duljinu k-torke (seed)	15
-h N	gornja granica broja ponavljanja k-torke	45
-l N	donja granica broja ponavljanja k-torke	20
-kf f	faktor koji definira koliko slabiji rezultati ce se čuvati u međukoracima	0.8
-pos N	maksimalan broj pozicija koje se čuvaju u međukoracima	80

4. Rezultati

4.1. Testiranje

U ovom poglavlju iznosim rezultate testiranja *MAPPER*-a nad genomima različitih duljina preuzetih s *NCBI* (*The National Center for Biotechnology Information*). Korišteni su genomi i njihove veličine iskazane u milijunima baznih parova:

- E. Coli [1] $\sim$ 4.64Mbp
- C. Elegans [2] $\sim$ 100.2Mb
- Human genome hg19, version 38, chromosome 3 [3] $\sim$ 200Mbp

Testna očitanja generirana su alatom *PBSIM*[4] koji se često koristi u bioinformatičkim krugovima prilikom testiranja. Razlog odabira upravo *PBSIM*-a je njegova mogućnost zadavanja različitih parametara prilikom generiranja očitanja (minimalna, maksimalna i prosječna duljina očitanja, standardna devijacija duljine, postotak greške koji se unosi u generirana očitanja, udio tipova greške kao što su zamjena, umetanje i brisanje). Uz generirana očitanja očitanja, *PBSIM* generira i *MAF* (*Mutation Annotation Format*) datoteku koja sadrži podatke o mutacijama(greškama) koje su ubačene u generirana očitanja.

Kako većina alata za poravnavanje na svom izlazu daje prethodno opisanu SAM datoteku, koristim skriptu za konverziju MAF datoteke u SAM koja je dio alata *LAST*. Tako generirana SAM datoteka mi predstavlja očekivani izlaz koji uspoređujem s izlazima svakog pojedinog alata za poravnanje koji testiram i provjeravam dali je određeno očitanje poravnano na točnu poziciju. Uz preciznost(omjer točno poravnatih očitanja i

[1] `ftp://www.ncbi.nlm.nih.gov/nuccore/48994873?report=fasta`

[2] `ftp://ftp.ncbi.nih.gov/genbank/genomes/Eukaryotes/invertebrates/Caenorhabditis_elegans/WBcel235/Primary_Assembly/assembled_chromosomes/FASTA/`

[3] `ftp://ftp.ncbi.nlm.nih.gov/genbank/genomes/Eukaryotes/vertebrates_mammals/Homo_sapiens/GRCh38/Primary_Assembly/assembled_chromosomes/FASTA/`

[4] `https://code.google.com/p/pbsim/`

poravnatih očitanja) i točnost(omjer točno poravnatih očitanja i ukupnog broja očitanja) također se mjeri i vrijeme izvođena te memorijsko zauzeće.
Svi testovi provedeni na računalu opremljenom sa 64GB RAM-a i 12-jezgrenim Intel(R) Xeon(R) E5645 procesorom radnog takta 2.40GHz koje pokreće Ubuntu 14.04.2 LTS operacijski sustav.

4.2. Usporedba

U nastavku su dati rezultati *MAPPER*-a kao i rezultati drugih alata navedenih u uvodu. Provodi se usporedba razvijenog alata *MAPPER* s trenutno vodećim alatima *BWA-MEM* i *LAST* te alata *GraphMap* koji se pokazao dobrim za poravnanja očitanja ovog tipa.
Tablice 4.1 i 4.2 sadrže naredbe kojima je pokrenut svaki od alata. Svim alatima koji podržavaju paralelizaciju zadan je jednak maksimalni broj dretvi koji odgovara broju fizičkih jezgri računala na kojem je testiranje provedeno (12 dretvi).

Tablica 4.1: Naredbe za izgradnju indeksa

Naziv alata	Naredba za izgradnju indeksa
BWA-MEM	bwa index reference.fa
GraphMap	graphmap -I -r reference.fa
LAST	lastdb index.db reference.fa
MAPPER	mapper -m index reference.fa index.db

Tablica 4.2: Naredbe za poravnanje očitanja

Naziv alata	Naredba za poravnanje očitanja
BWA-MEM	Pacbio: bwa mem -t 12 -x pacbio reference.fa reads.fq > out.sam ON: bwa mem -t 12 -x ont2d reference.fa reads.fa > out.sam
GraphMap	graphmap -v 5 -t 12 -B 0 -b 3 -r reference.fa -d reads.fq -o out.sam
LAST	lastal -v -q 1 -r 1 -a 1 -b 1 index.db reads.fa > out.sam
MAPPER	mapper -m map -t 12 -k 15 ref.fa index.db reads.fq out.sam mapper -m map -t 12 -k 10 ref.fa index.db reads.fq out.sam

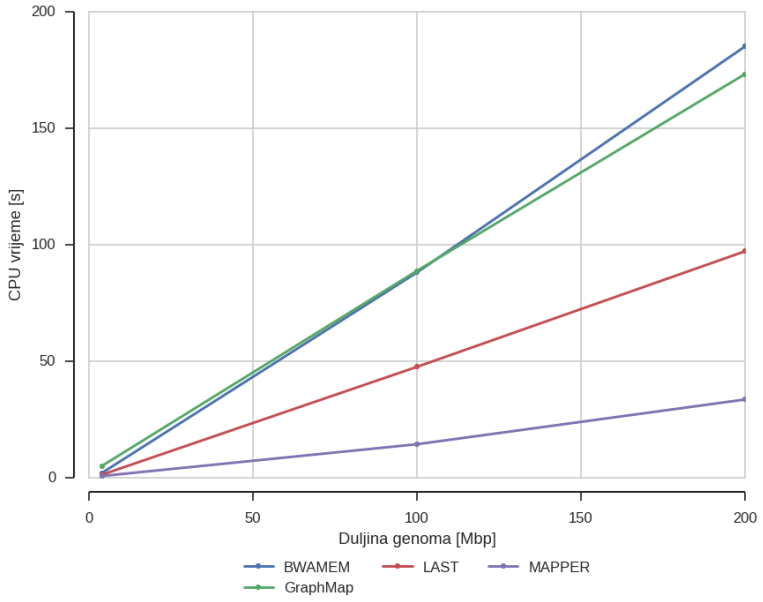
4.3. Rezultati testiranja

4.3.1. Izgradnja indeksa

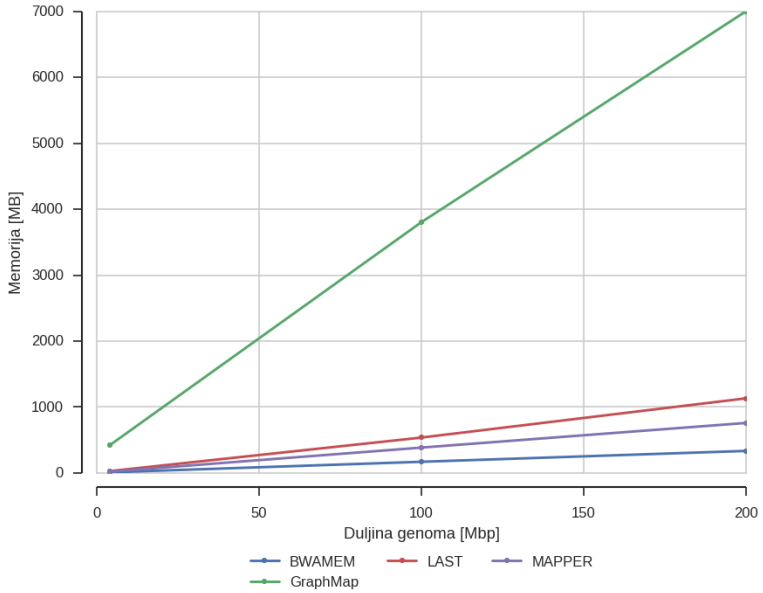
Kao što je prethodno spomenuto, svaki od alata za poravnanje koristi vlastitu izgrađenu strukturu indeksa. Slika 4.1 prikazuje vrijeme potrebno za izgradnju indexa u ovisnosti o duljini slijedna nad kojim je izgrađeno dok slika 4.2 memorijsko zauzeće izgrađenog indeksa.

Razvijeni *MAPPER* se pokazao brzim prilikom izgradnje te memorijski štedljiv pri izgradnji indeksa. Glavni razlozi toga su korištenje jednostavne strukture sufiksnog polja i biblioteke *libdivsufsort* koja se pokazala izrazito brza. Samo vrijeme izgradnje ne predstavlja bitno svojstvo iz razloga što se indeks jednom izgradi te se koristi za sva poravnanja.

GraphMap koristi dva indeksa pri poravnanju kako bi postigao veću preciznost stoga je moguće prepoloviti memorijsko zauzeće prikazano na slici 4.2 ukoliko manje precizan način rada koji koristi samo jedan indeks.



Slika 4.1: Vrijeme izgradnje indeksa



Slika 4.2: Memorijsko zauzeće indeksa

4.3.2. Poravnanje

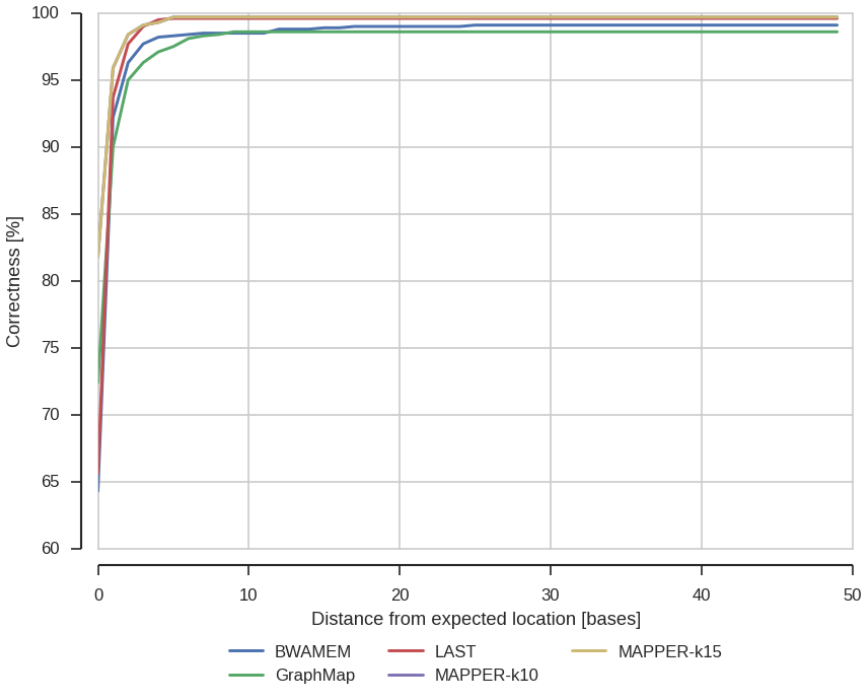
U nastavku su navedeni grafovi koji prikazuju preciznost poravnanja odnosno postotak točno poravnatih nizova u ovisnosti od stvarne pozicije u slijedu. Razvijeni alat *MAPPER* testiran je korištenjem različitih parametara. Prikazani su rezultati za početnu duljinu k-torka 10 i 15. U tablici 4.3 su navedena svojstva ispitnih skupova očitanja nad kojima su alati testirani. Svaki ispitni skup sastoji se od 1000 generiranih očitanja. Zbog velikog broja grafova u nastavku slijedi samo jedan dio koji prikazuje ponašanje alata za karakteristične slijedove i očitanja. Preostali grafovi i tablice s rezultatima dostupne su, uz alat, dokumentaciju i izvorni kod, u GitHub repozitoriju.

Tablica 4.3: Spefcifikacije korištenih očitanja

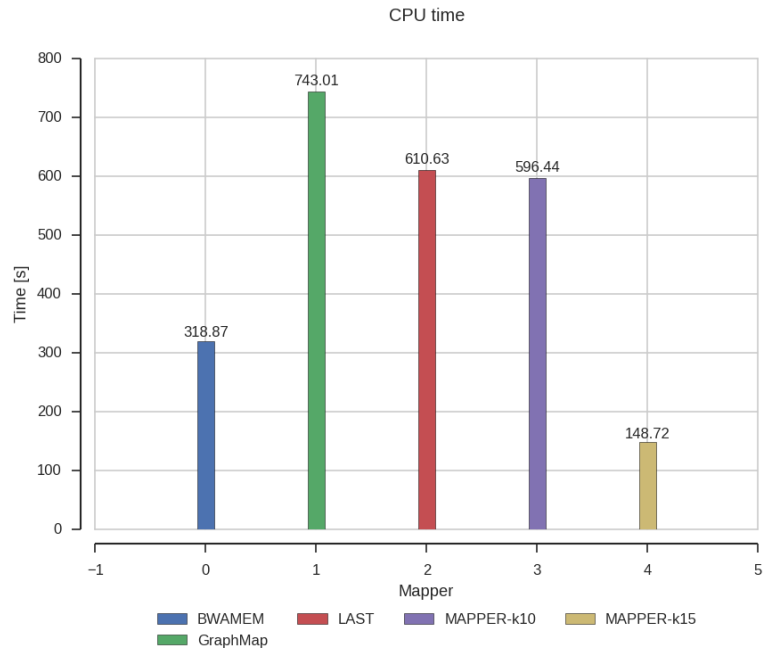
Ispitni skup	Postotak greške	std[5]	M:I:D[6]	Srednja duljina	Min	Max
Skup 1[7]	17.6	5	30:27:43	5600	100	100000
Skup 2	22.2	2	10:60:30	3000	100	25000
Skup 3[8]	31	9	55:17:28	2006.14	42	28601
Skup 4	41	5	51:11:38	3629.76	57	31299

[5]standardna devijacija greške
[6]Omjer tipova unesene greške *mismatch:insertion:deletion*
[7]Modelirano po rezultatima objavljenim u [6]
[8]Modelirano po rezultatima objavljenim u [15]

Ispitni skup 1. sadrži očitanja s najmanjim postotkom greške i svi alati imaju visok postotak točnosti kod manjih sljedova (oko 99.7%). Kod malo većeg slijeda (ljudski kromosom) javlja se mali pad točnosti alata *LAST* i *BWA-MEM*(98%) dok *GraphMap* i *MAPPER* (s oba tipa postavki) imaju točnosti i dalje preko 99%. (slika 4.3). Manje osjetljiva opcija *MAPPER*-a pokazala je jednaku točnosti uz znatno manje vrijeme izvođenja u odnosu na konkurenciju (slika 4.4).
Ispitni skup 2. sadrži neznatno veći postotak pogreške te su rezultati gotovo identični onima ispitnog skupa 1.

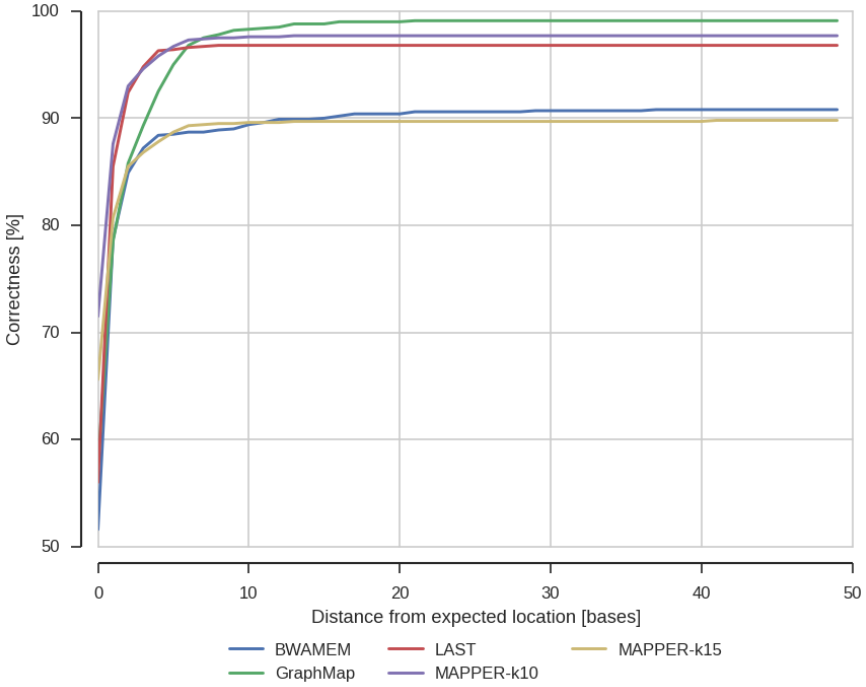


Slika 4.3: Točnost, HG19, Skup 1

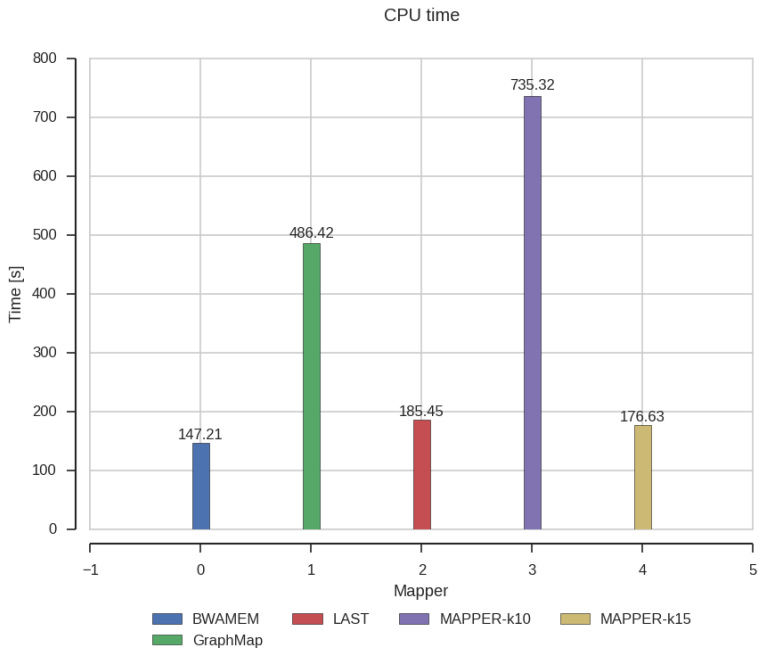


Slika 4.4: CPU vrijeme izvodenja, HG19, Skup 1

Ispitni skup 3. sadrži očitanja s približno 30% greške što dovodi do smanjenja točnosti svih alata. Najbolje rezultate daje *GraphMap*(99.1%). Kod ovako visokog stupnja greške javlja se razlika u točnosti kod *MAPPER*-a ovisno o zadanom parametru k. Za parametar k jednak 10 postiže se visoka točnost (97.7%) usporediva s alatom *LAST*.

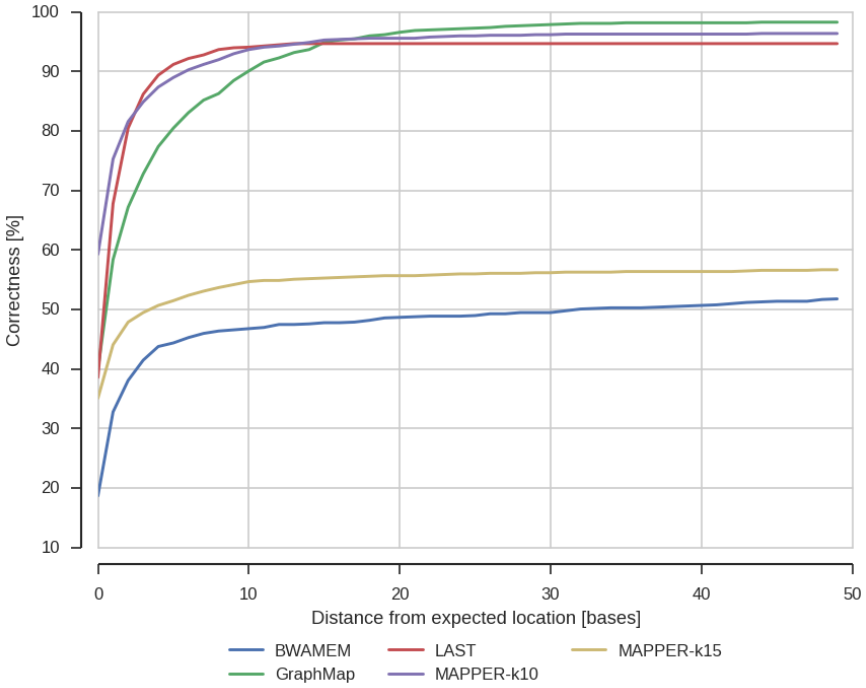


Slika 4.5: Točnost, HG19, Skup 3

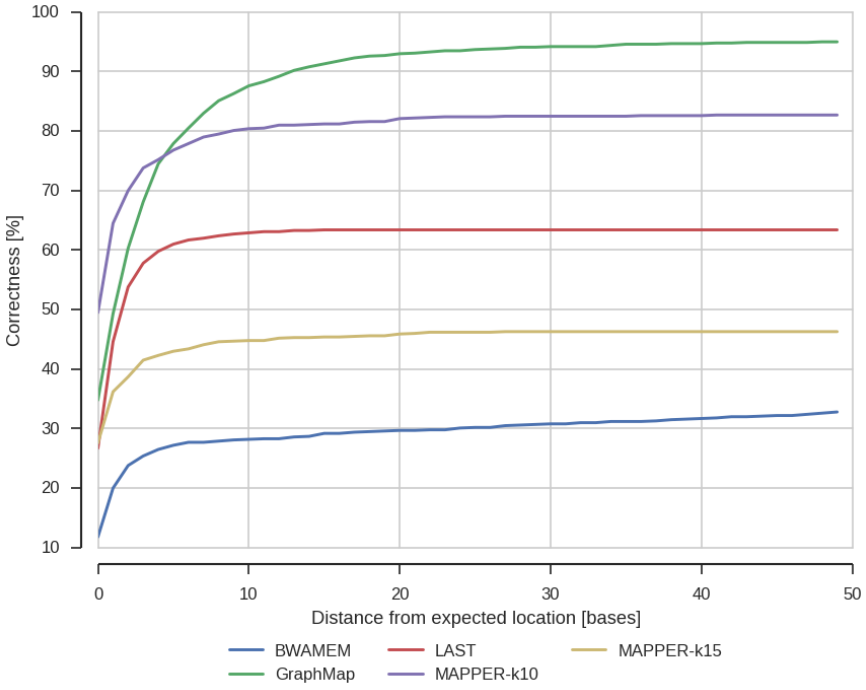


Slika 4.6: CPU vrijeme izvodenja, HG19, Skup 3

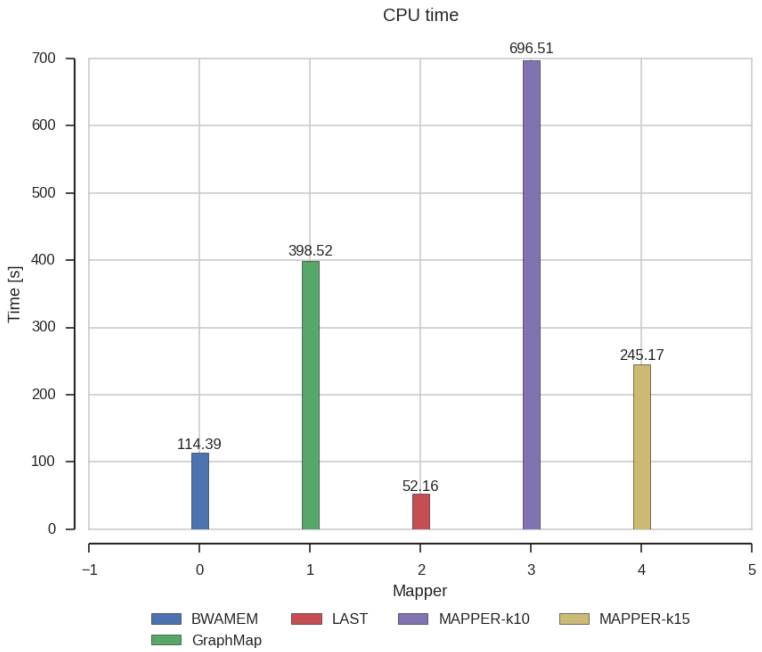
Ispitni skup 4. sadrži očitanja najvećeg stupnja greške *LAST*. Kod ovakvih očitanja točnost opada kod dužih slijedova što je vidljivo iz pada rezultata za genom E.coli i kromosom čovjeka (grafovi 4.7 i 4.8). *GraphMap* ima najvišu točnosti (95%). *MAPPER* je, uz parametar k jednak 10, pokazao zadovoljavajuću točnost od preko 80% ali pod cijenu rasta vremena izvođenja (slika 4.9 Kod očitanja visokog stupnja greške (30% i više), manje osjetljiv način rada *MAPPER*-a(k jednak 10) se podovoljno dobar te je postotak točno poravnatih očitanja oko 50%.



Slika 4.7: Točnost, E.Coli, Skup 4



Slika 4.8: Točnost, HG19, Skup 4



Slika 4.9: CPU vrijeme izvodenja, HG19, Skup 4

5. Zaključak

Cilj ovog rada je bio razviti alat koji ce moći poravnavati dugačka očitanja nastala uređajima za sekvenciranje treće generacije kao što su *PacBio* i *Oxford Nanopore*. Glavni izazov bio je razviti takav alat koji ce s velikom točnosti moći poravnati i očitanja koja imaju visoke stope greške.
Razvijeno rješenje je pokazalo dobre rezultate (i potrošnja resursa i točnost) kod očitanja čiji stupanj greške do 20-ak%. Kod očitanja s većim stupnjem greške moguće je povećati točnost adekvatnom kombinacijom parametara (prvenstveno parametra k) pod cijenu povećanja vremena izvođenja. Na taj način je moguće postići da alat i za greške reda veličine 30%-35% daje zadovoljavajuće rezultate u usporedbi s postojećim rješenjima.
Kako još nisu u potpunosti poznate konačne specifikacije *Oxford Nanopore* očitanja, može se samo očekivati povećanje broja novih alata i nadogradnja postojećih alata s opcijom za poravnanje dugih očitanja.

U svrhu optimizacije predstavljenog rješenja moguće je zamijeniti sufiksnog polje nekom strukturom koja omogućava brzu pretragu uz cijenu povećanja memorijskog zauzeća (primjerice sufiksno stablo ili kombinacija *hash*-a i koncepta *double-indexing* predstavljena u [1]. Trebalo bi detaljnije istražiti metode koje bi omogućavale da određene k-torke ne razmatramo prilikom određivanja regija (k-torke s velikim brojem pojavljivanja nam ne daju pretjerano korisne informacije).
Uz postojeću ocjenu sličnosti za kandidatne regije, bilo bi dobro razmotriti i nekakvu sekundarnu ocjenu koja bi pomogla pri otklanjanju problema većeg broja regija s jednakom ocjenom.
Valja spomenuti i mogućnost implementacije koja će iskorištavati masivnu paralelnost koju omogućava grafičke kartice (*CUDA* tehnologija) i time postići bolje performanse.

Poznavanje specifikacija očitanja (srednja duljina, postotak greške i tipova greške) te srodno tome specificiranje parametara kao rezultat bi imalo povećanje preciznosti

alata, ali uz nužno smanjenje općenitosti, točnosti i/ili performansi pri poravnanju očitanja drugačijih specifikacija.

LITERATURA

[1] Benjamin Buchfink, Chao Xie, i D. H Huson. Fast and sensitive protein alignment using diamond. *Nature Methods*, 12:59–60, 2015. URL http://www.nature.com/nmeth/journal/vaop/ncurrent/full/nmeth.3176.html. Published online 17 November 2014.

[2] Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software: Practice and Experience*, 24:327–336, 1994.

[3] Malcolm C. Harrison. Implementation of the substring test by hashing. *Commun. ACM*, 14(12):777–779, Prosinac 1971. ISSN 0001-0782. doi: 10.1145/362919.362934. URL http://doi.acm.org/10.1145/362919.362934.

[4] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, Lipanj 1975. ISSN 0001-0782. doi: 10.1145/360825.360861. URL http://doi.acm.org/10.1145/360825.360861.

[5] IHGSC. Finishing the euchromatic sequence of the human genome. *Nature*, 431 (7011):931–945, 2004. IHGSC: Intl. Human Genome Sequencing Consortium.

[6] Fiddes Ian T Miga Karen H Olsen Hugh E Paten Benedict Akeson Mark Jain, Mite. Improved data analysis for the minion nanopore sequencer, 2015.

[7] Juha Kärkkäinen i Peter Sanders. Simple linear work suffix array construction. U *Proceedings of the 30th International Conference on Automata, Languages and Programming*, ICALP'03, stranice 943–955, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40493-7. URL http://dl.acm.org/citation.cfm?id=1759210.1759301.

[8] Heng Li i Richard Durbin. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

doi: 10.1093/bioinformatics/btp324. URL http://bioinformatics.oxfordjournals.org/content/25/14/1754.abstract.

[9] Heng Li i Nils Homer. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics*, 11(5):473–483, 2010. doi: 10.1093/bib/bbq015. URL http://bib.oxfordjournals.org/content/11/5/473.abstract.

[10] Udi Manber i Gene Myers. Suffix arrays: A new method for on-line string searches. U *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, stranice 319–327, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics. ISBN 0-89871-251-3. URL http://dl.acm.org/citation.cfm?id=320176.320218.

[11] Mirjana Domazet-Lošo Mile Šikić. *Bioinformatika*. Prateći materijal za predmet Bioinformatika, Fakultetu elektrotehnike i računarstva, 2013.

[12] Gene Myers. A fast bit-vector algorithm for approximate string matching based on dynamic programming. *J. ACM*, 46(3):395–415, Svibanj 1999. ISSN 0004-5411. URL http://doi.acm.org/10.1145/316542.316550.

[13] Ge Nong, Sen Zhang, i Wai Hong Chan. Two efficient algorithms for linear time suffix array construction. *IEEE Trans. Comput.*, 60(10):1471–1484, Listopad 2011. ISSN 0018-9340. doi: 10.1109/TC.2010.188. URL http://dx.doi.org/10.1109/TC.2010.188.

[14] Filip Pavetic, Goran Zuzic, i Mile Sikic. $lcsk$++: Practical similarity metric for long strings. *CoRR*, abs/1407.2407, 2014. URL http://arxiv.org/abs/1407.2407.

[15] J; Loman N.J; Quick. Bacterial whole-genome read data from the oxford nanopore technologies minion™ nanopore sequencer., 2014. URL http://dx.doi.org/10.5524/100102.

[16] Aaron Quinlan, Josh Quick, i Nicholas Loman. A reference bacterial genome dataset generated on the minion™ portable single-molecule nanopore sequencer. *bioRxiv*, 2014. doi: 10.1101/009613.

[17] Saul B. Needleman, Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(7011):443–453, 1970.

[18] Temple F. Smith i Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981. URL `http://gel.ym.edu.tw/~chc/AB_papers/03.pdf`.

[19] John M Urban, Jacob Bliss, Charles E Lawrence, i Susan A Gerbi. Sequencing ultra-long dna molecules with the oxford nanopore minion. *bioRxiv*, 2015. doi: 10.1101/019281.

[20] Matei Zaharia, William J. Bolosky, Kristal Curtis, Armando Fox, David A. Patterson, Scott Shenker, Ion Stoica, Richard M. Karp, i Taylor Sittler. Faster and more accurate sequence alignment with SNAP. *CoRR*, abs/1111.5572, 2011. URL `http://arxiv.org/abs/1111.5572`.

Alat za poravnanje dugačkih očitanja

Sažetak

Poravnavanje bioloških sljedova predstavlja jedan od najstarijih i najviše istraživanih problema u bioinformatici. Cilj je identificirati slične regije koje mogu predstavljati funkcionalnu, strukturalnu ili evolucijsku vezu između dvaju organizama. Neprekidnim razvojem tehnologije napreduju i uređaji za sekvenciranje te su moguća sve duža očitanja. Javlja se potreba za razvojem alata koji ce moći poravnati takva dugačka očitanja koja potencijalno mogu imati i visok postotak greške.
U ovom radu je predloženo jednostavno rješenje koje problem svodi na problem izdvajanja iz ulaznog slijeda kandidatnih regija korištenjem vrijednosti $LCSk++$ dvaju podnizova. U radu je dat algoritam, implementacijski detalji razvijenog alate te usporedba s postojećim alatima.

Ključne riječi: bioinformatika, poravnanje, dugacka očitanja, LCSk++, sufiksno polje

Long read aligner

Abstract

Sequence allimgent is one of the first and most researched problems in bioinformatics. It is used to identify regions of similarity that may indicate functional, structural or evolutionary relationship between two organisms
With the advancement of technology, sequencing machines are constantly improving and they are producing longer and longer reads. Need for new aligners that are able to work with those long error-prone exists.
This Thesis presents simple tool that solves the problem of aligning reads by selecting potential regions in a sequence that have higher similarity score. $LCSk++$ between two subsequences is used as similarity score. Described algorithm and implementation details are presented in this Thesis as well as comparison with existing aligners.

Keywords: bioinformatics, sequence aligment, long error-prone reads, LCSk++, suffix array