SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

Projekt

Pronalazak mutacija pomoću treće generacije sekvenciranja

Hari Barić, Lucija Belić

14. Sječnja 2019.

Sadržaj

1.Uvod

Usporedba sekvenci jedan je od temeljnih alata moderne bioinformatike, te se aplicira u područjima bioinformatike kao što su pronalazak mutacija. Jedan od većih izazova predstavljaju dugi nizovi sekvenci koje je potrebno pročitati, te razne operacije koje se tada provode nad njima.

U ovom radu implementiran je algoritam koji pronalazi mutacije između referentnog niza koji predstavlja potpuni genom bakterije *Escherichia coli,* te niza sekvenci koje predstavljaju manje dijelove referentnog niza dobivene trećom generacijom sekvenciranja. U daljnjem tekstu objašnjeno je kako se uz pomoć generiranja k-mer nizova, te traženja tzv. minimizera može efektivno indeksirati veliki podniz, te zatim uporabom algoritma poravnanja razlučiti mutacije između referentnog i sekvenciranog niza.

2. Opis korištenih algoritama

2.1. K-mer Minimizers

Iz skupa očitanja dobivenih sekvenciranjem mutiranog genoma te referentnog genoma potrebno je pronaći razliku između referentnog genoma te sekvenciranog mutiranog genoma, a kako bismo to postigli koristi se '*seed-and-extend*' metoda. '*Seeds*' su kraći podnizovi koji se pojavljuju u nizu nukleotida i koriste se za indeksiranje, pretraživanje ili sastavljanje genoma.

Sve podnizove nizova iz skupa nizova nukleotida duljine *k* nazivamo *k*-merima. Količina *k*-merova za niz duljine *L* je *L-k+1*, što iziskuje veliku količinu memorije.

Kako bismo taj problem izbjegli tražimo predstavnika iz grupe susjednih k-merova tako da dva različina niza odabiru istog predstavnika ako dijele dovoljno dugačak podniz. Izabrani *k*-mer nazivamo *minimizer*. Dakle od svih mogućih *k*-merova koristimo samo dio njih, a ti minimizeri imaju svojstvo:

"If two strings have a significant exact match, then at least one of the minimizers chosen from one will also be chosen from the other." [1]

Skup *w* uzastopnih k-merova koji su pomaknuti za jedan znak (A, C, G, T) od prethodnog, pokrivaju niz uzastopnih znakova veličina prozora *w+k-1*. Minimizere tražimo na način da promatramo *w* uzastopnih *k*-merova i odabiremo najmanjeg. Iz opisanog svojstva minimizera zaključujemo da dva niza koja imaju zajednički podniz duljine *w+k-1* dijele isti minimizer.

Po nizu se pomičemo za jedno mjesto udesno te unutar prozora duljine *w+k-1* leksikografski tražimo najmanji *k*-mer koji je ujedno i minimizer. Postupak traženja minimizera prikazan je na slici.

Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	2	3	1	0	3	2	1	0	1	2	3	3	1	0	1
	2	3	**1**	**0**	**3**										
		3	1	**0**	**3**	**2**									
			1	**0**	**3**	**2**	1								
				0	**3**	**2**	1	0							
					3	2	**1**	**0**	**1**						
						2	1	**0**	**1**	**2**					
							1	**0**	**1**	**2**	3				
								0	**1**	**2**	3	3			
									1	**2**	**3**	3	1		
										2	**3**	**3**	1	0	
											3	3	**1**	**0**	**1**

Slika 1: Postupak određivanja minimizera [1]

Minimizeri su korišteni za poravnanje dugih očitanja, u probabilističkom de Bruijnov grafu, za pretraživanja kroz mikrobne genome i drugo.
U sklopu ovog projekta konstruirani su indeksi podnizova duljine nađenih minimizera svakog očitanja i reference. U našem algoritmu generiramo parove *(i,j)*, gdje *i* predstavlja poziciju *k*-mera, odnosno minimizera u referentnom nizu, dok *j* predstavlja poziciju istog minimizera u sekvenciranom nizu.

2.2. Longest Increasing Subsequence (LIS)

Primarna ideja implementacije našeg algoritma je pronalazak najdulje regije koja se podudara u referentnom nizu i sekvenciranom nizu koji očitavamo kako bi ih tada usporedili, te na kraju našli njihove razlike, tj. mutacije. Za tu svrhu koristimo LIS algoritam (*engl. Longest Increasing Subsequence*) kako bi pronašli najdulji niz uzastupnih zajedničkih *k*-mer minimizera između referentnog i sekvenciranog niza, te gradimo listu parova (*seq_index, ref_index*) gdje *seq_index* označava poziciju minimizera u sekvenciranom nizu, dok ref_index označava poziciju u referentnom.

Nakon konstruiranja sortirane liste parova (*seq_index, ref_index*) po prvom članu *seq_index*, koristi se LIS algoritam koji uspoređuje drugi član para *ref_index* kako bi pronašli najdulji uzastopni niz indeksa minimizera koji se nalaze u referentnom genomu. Nakon pronađenog najduljeg niza, izbacujemo one parove koji imaju jednak susjedni

seq_index, te provjeravamo ako je razmak (engl. *gap*) između pronađenih minimizera u referentnom nizu veći od parametra *gap* kako bi izbjegli pretjerano velike podnizove koji se neće dobro poravnati, tj. podnizove koji stvaraju veću grešku te usporavaju izvršavanje programa. U slučaju da nije pronađen niti jedan razmak veći od parametra *gap* , najboljom regijom poravnanja proglasujemo prvi i zadnji par (*seq_index, ref_index*) koji smo dobili, te zatim slijedi algoritam poravnanja koji je zadužen za poravnanje podniza iz referentog i sekvenciranog nizaSadržaj.

2.3. Needleman-Wunsch algoritam

U molekularnoj biologiji najčešće se postavlja pitanje postoji li međusobna sličnost između dvaju bioloških nizova, kako bi se identificirala njihova struktura ili funkcija. Odgovor na ovo pitanje je u usporedni nizova, odnosno u računanju sličnosti nizova. Algoritam koji rješava zadani problem poravnanja niza predložili su Saul B. Needleman i Christian D. Wunsch 1970.[3] Algoritam se temelji na dinamičkom programiranju te se danas naziva Needleman-Wunsch algoritam. Algoritam Needleman-Wunsch je algoritam za globalno poravnanje s obzirom da traži poravnanje od *(0,0)* do *(n,m)*, gdje *n* i *m* predstavljaju dužinu dvaju nizova. Vremena i memorijska složenost algoritma je *O(n*m)*.
Algoritam koristi sustav bodovanja, a temelji se na relaciji prikazanoj na slici. [2]

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

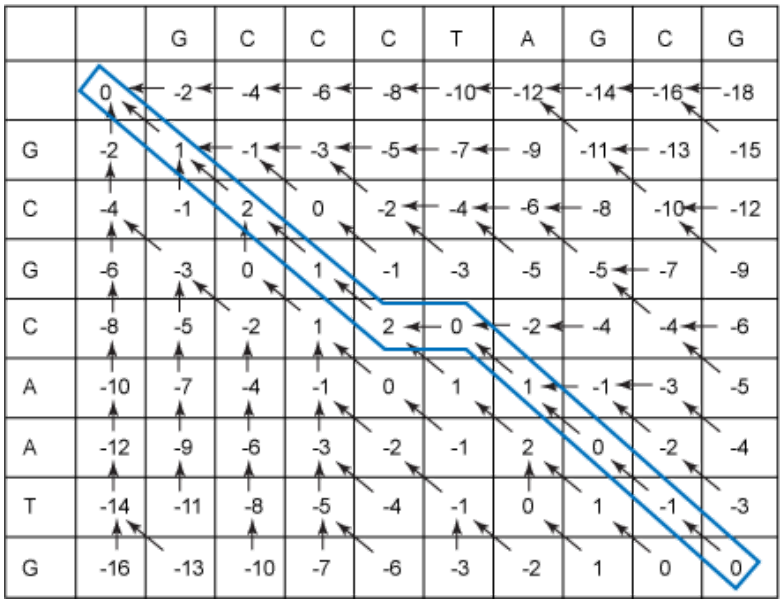
Slika 2: Algoritam za izračun V matrice

Matrica se popunjava počevši od gornjeg lijevog kuta *(0,0)* te pronalazi maksimalan broj bodova svake ćelije, odnosno traži se maksimalna cijena puta do vrha *(n,m)*.
Vrijednosti za prvi redak i prvi stupac predstavljaju početne uvjete. Cijena svakog pomaka je *d*, pa su vrijednosti za rubne uvjete linearni s *d*. Vrijednosti pojedinog člana matrice računamo poznavanjem vrijednosti za njegove susjede lijevo, gore i lijevo gore. Uzima se maksimalna među njima i popunjava s dobivenim rezultatom. Računanje poravnanja svodi se na izračun vrijednosti u matrici dimenzija *(n + 1)×(m + 1)*.
Označimo tu matricu s V, a *d* označava cijenu kretanja desno i dolje (umetanje ili brisanje), a *w(Ai,Bj)* predstavlja cijenu zamjene dva znaka odnosno kretanja dijagonalno dolje. Informaciju o poravnanju dobijamo tako da pamtimo za svaki element smjer od kuda se do njega došlo te pretragom unatrag od *(n,m)* elemeta do *(0,0)* korištenjem informacije o smjerovima, rekonstruiramo poravnanje.
Pseudokod za rekostrukciju [4]:

```
AlignmentA ← ""
AlignmentB ← ""
i ← length(A)
j ← length(B)
while (i > 0 or j > 0)
{
  if (i > 0 and j > 0 and F(i,j) == V(i-1,j-1) + w(A_i, B_j))
  {
    AlignmentA ← A_i + AlignmentA
    AlignmentB ← B_j + AlignmentB
    i ← i - 1
    j ← j - 1
  }
  else if (i > 0 and F(i,j) == F(i-1,j) + d)
  {
    AlignmentA ← A_i + AlignmentA
    AlignmentB ← "-" + AlignmentB
    i ← i - 1
  }
  else
  {
    AlignmentA ← "-" + AlignmentA
    AlignmentB ← B_j + AlignmentB
    j ← j - 1
  }
}
```

Kod Needleman-Wunschovog algoritma, zbog bioloških razloga, vrijednosti za kažnjavanje zamjene, brisanja i umetanja se razlikuju. Vrijednost za slaganje je pozitivna, dok su vrijednosti koje kažnjavaju brisanje, umetanje i zamjenu negativne vrijednosti. Vrijednosti slaganja i neslaganja mogu varirati ovisno o parovima nukleotida, a razlikuju se i od vrijednosti za umetanje i brisanje. U našem projektu vrijednost za brisanje i umetanje postavljene su na -2 , tj. $d = -2$, dok je vrijednost za slaganje 4, a neslaganje, odnosno zamjenu -1.

Slika predstavnja popunjavanje tablice Needleman-Wunschovim algoritmom, te su označene informacije o smjerovima pomoću kojeg rekonstruiramo poravnanje.



Slika 3: *Needleman-Wunsch algoritam sa vrijednostima d=-2, w=1 za podudaranje, -1 za zamjenu*

Dobiveno poravnanje je :

G C G C - A A T G

G C C C T A G C G

Poravnavanje je provedeno za sva očitanja na referentnom genom, te je za svaki indeks nađen odnos podnizova očitanja za podnizove reference. Pronađena mutacija je najčešći odnos baze referentnog genoma prema bazi sekvenciranih genoma.

3. Analiza točnosti, vremena izvođenja i zauzeća memorije

U ovom poglavlju objašnjeno je na koji način je algoritam testiran, te su dani rezultati testiranja.

Testiranje se provelo na danom skupu ulaza koji uključuje dvije datoteke. Jedna datoteka predstavlja potpuni referentni genom na kojemu se traže mutacije, dok druga datoteka predstavlja niz različitih očitanja tog genoma.

Za usporedbu naše i izvorne implementacije koristila se skripta koja računa *Jaccard score* , tj. postotak jednakih očitanja mutacija. Kako bi našli najbolje parametre za pokretanje našeg programa, prvo se radilo testiranje za slučaj *lambda* kako bi se uštedilo na vremenu pošto je taj skup znatno manji od realnog primjera *ecoli.* Algoritam se može testirati zadavajući različite parametre, a to su redom:

k - duljina *k*-mer podstringa pomoću kojeg se traže minimizeri
w - broj uzastopnih *k*-mer podstringa od kojih se uzima minimizer
c_tres - minimalan broj pronađenih mutacija koje se uzimaju kao valjane
gap - maksimalni razmak između pozicije 2 minimizera pronađena u podnizu
file - 0 za slučaj lambda, 1 za slučaj *ecoli*

Za duljinu *k* izabrali smo vrijednost 15, dok smo za vrijednost *w* izabrali vrijednost 5. Pomoću njih dobili smo najbolje rezultate, bez obzira na preporučene vrijednosti *k*=*w*=20 [1]. Kako bi ubrzali algoritam, te isto tako povećali njegovu točnost, uveden je parametar *gap* , koji u slučaju da se pronađe preveliki razmak između 2 minimizera, preskače se cijela dobivena regija koja je predviđena za poravnanje, te se time štedi na vremenu pošto se ne mora raditi algoritam za poravnanje koji je kvadratne složenosti nad velikom duljinom znakovnog niza u toj regiji.

c_tres parametar	gap parametar	Trajanje programa [s]	Avg. RAM [MB]	Jaccard score [%]
13	10	46	~ 400	35.68
8	12	104	~ 500	55.52
10	12	112	~ 1300	66.96
12	15	132	~ 1300	68.44

Tablica 1. Usporedba izvršavanja za slučaj lambda uz parametre k=15, w=5

c_tres parametar	gap parametar	Trajanje programa [min]	Avg. RAM [GB]	Jaccard score [%]
8	11	92	~ 4	49.48
11	14	158	~ 6	78.28

Tablica 2. Usporedba izvršavanja za slučaj ecoli uz parametre k=15, w=5

Iz priloženih tablica možemo zaključiti da što manjim odabirom vrijednosti parametra *gap*, dobivamo znatno brže izvršavanje algoritma jer izbjegavamo poravnanje prevelikih regija, dok povećanjem *c_tres* parametra dobivamo bolji Jaccard score u slučaju da nađemo više poravnanja za istu regiju u referentnom nizu. Ugađanjem ta 2 parametra, možemo birati između brzine izvođenja i točnosti.

4. Zaključak

Korištenjem k-mer minimizera kako bi indeksirali dijelove dugačkog niza znatno možemo uštedjeti na memoriji, te na efikasan način možemo naći dijelove niza koji nas zanimaju. U našem radu uspjeli smo pomoću minimizera pronaći najveće regije podudaranja između sekvenciranog niza i referentnog, kako bi zatim pomoću algoritma za poravnanje pronašli mutacije, tj. njihove razlike. Najbolji *Jaccard score* koji smo uspjeli dobiti između naše i izvorne implementacije nad pravim primjerom referentog genoma bakterije *Escherichia coli* je 78.28%. Za to je trebalo 92 minute, stoga nas je predugačko izvođenje algoritma spriječilo u daljnim ugađanjem parametara c_tres i gap kako bi pronašli najbolji mogući rezultat.

5.Literatura

[1] https://academic.oup.com/bioinformatics/article/20/18/3363/202143

[2] M. Šikić, M. Domazet-Lošo, Bioinfomatika: http://www.fer.unizg.hr/predmet/bio

[3] Needleman, Saul B. & Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". *Journal of Molecular Biology*. 48 (3): 443–53.

[4] Wikipedia: https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm

[5] vlab.amrita.edu,. (2012). Global alignment of two sequences - Needleman-Wunsch Algorithm. Retrieved 15 January 2019, from vlab.amrita.edu/?sub=3&brch=274&sim=1431&cnt=1