

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DOKUMENTACIJA

**Pronalazak mutacija pomoću treće generacije
sekvenciranja**

Kseniia Ikonnikova i Heidi Sokolovski

Voditelj: *Krešimir Križanović*

Zagreb, svibanj, 2023.

Sadržaj

1. Uvod	3
2. Opis algoritama	4
2.1 Generiranje minimizera	4
2.2 Najdulji rastući podniz (<i>Longest Increasing Subsequence, LIS</i>)	5
2.3 Poravnanje	7
2.4 Pronalazak mutacija	10
3. Rezultati	12
4. Zaključak	13
5. Literatura	14

1. Uvod

Treća generacija sekvenciranja je revolucionirala područje bioinformatike. Prethodne generacije su generirale kratka i precizna očitavanja. Za razliku od njih, očitavanja treće generacije su duga i obuhvaćaju velike dijelove molekule DNA. Ta tehnologija značajno je poboljšala sposobnost točne identifikacije i karakterizacije različitih mutacija u genomu.

Kako je sekvenciranje genoma od sve veće važnosti, tako je važnije i uspješno poravnati referentni genom i razna očitavanja. Mnoge metode koriste analizu baziranu na k -merima što su podsekvence duljine k . One su često brze i efikasne pa su stoga veoma popularne. Jedna od više korištenih vrsta k -mera su minimizeri. Mnogi se bave njihovom optimizacijom [1].

U ovom radu se opisuju algoritmi za pronalazak mutacija pomoću treće generacije sekvenciranja i vizualiziraju na jednostavnom primjeru. Osim toga se proučava vrijeme izvođenja i utrošak memorije te uspoređuje točnost rezultata ove implementacije i izvorne.

2. Opis algoritama

2.1 Generiranje minimizera

Za pronalazak mutacija u genomu koriste se minimizeri. Minimizeri su manji dijelovi sekvence duljine k poznati kao k -meri. Biraju se svakih w k -mera. Od njih se odabere jedan abecedno minimalan k -mer te je to minimizer. Dije se na unutarnje i krajnje minimizere na početku i kraju niza. Osjetljivi su na umetanja i brisanja [2].

Prvi algoritam je zaslužan za stvaranje potrebnih minimizera. On se primjenjuje na referentnom genomu i na svakom pojedinom očitavanju. Zbog toga što DNA molekula ima dva lanca, postoji mogućnost da nisu sva očitavanja s istog lanca. Kako bi se pokrio i taj slučaj, osim generiranja minimizera za dobiveno očitavanje, generiraju se minimizeri i za njegov reverzni komplement. Naravno nije potrebno napraviti isto i za referentni genom.

Algoritam je rješiv u $O(n)$ vremenu.

Za stvaranje minimizera potrebne su veličina minimizera k , broj uzastopnih k -mera koje gledamo w te sekvenca iz koje se vade minimizeri.

Algoritam radi tako da se najprije inicijalizira lista minimizera, pomoćna lista i spremi početni minimizer.

Prva *for*-petlja služi za pronalazak krajnjih minimizera s početka sekvence s indeksom većim od 1 i manjim od w . U pomoćnu listu se dodaju minimizeri jedan po jedan pri svakom prolasku petlje. Trenutni minimizer je minimum svih koji se nalaze u pomoćnoj petlji. Ako je trenutni minimizer različit od prethodnog, dodaje se u listu minimizera te prethodni postane trenutni.

Druga *for*-petlja služi za pronalazak unutarnjih minimizera. Najprije obriše nulti element s pomoćne liste. Ostatak petlje je jednak prvoj.

Treća i zadnja *for*-petlja je za krajnje minimizere na kraju sekvence. U njoj se ne dodaju elementi u pomoćnu listu, ali sve ostalo je kao u drugoj petlji.

Vraćeni rezultat je lista minimizera.

U nastavku je primjer na kojemu je pokazano kako algoritam radi. Traže se (3, 3)-minimizeri za sekvencu = „GTCATGCACG”.

Minimizeri su prikazani masnim, zelenim slovima.

0	1	2	3	4	5	6	7	8	9
G	T	C	A	T	G	C	A	C	G
G	T	C							
G	T	C	A						
G	T	C	A	T					
	T	C	A	T	G				
		C	A	T	G	C			
			A	T	G	C	A		
				T	G	C	A	C	
					G	C	A	C	G
						C	A	C	G
							A	C	G

Vraćeni rezultat je: („GTC”, 0), („CAT”, 2), („ATG”, 3), („CAC”, 5) i („ACG”, 7).

2.2 Najdulji rastući podniz (*Longest Increasing Subsequence, LIS*)

Potrebno je pronaći podniz danog niza tako da su elementi podniza poredani, od najnižeg prema najvišem i u kojemu je podniz najdulji mogući. U podnizu ne trebaju svi elementi biti susjedni niti jedinstveni. Algoritam je rješiv u vremenu $O(n \log n)$ [3].

Korišten je postojeći pseudokod [4].

Pronađu se poklapanja minimizera te se od tih poklapanja treba naći LIS nad pozicijama u drugom nizu.

Najprije se trebaju inicijalizirati varijable. Varijabla p je lista iste veličina kao početna lista minimizera u kojoj se pohranjuju prethodni indeksi elemenata LIS-a. Varijabla m je lista veća od p za jedan i služi za pohranu indeksa koji predstavljaju kraj rastućih

podniza drugačijih duljina. Varijabla l služi za praćenje duljine LIS-a. Sve je postavljeno na nula.

Zatim se u *for*-petlji prolazi po svim elementima liste minimizera. Varijabla koja predstavlja lijevi dio je postavljena na jedan, dok je varijabla za desni dio postavljena na vrijednost l . Koristi se binarno pretraživanje kako bi se našla nova l pozicija gdje se trenutni element liste minimizera može umetnuti u LIS. Ažurira se vrijednost prethodnog indeksa u p na poziciju novi l manje jedan od liste m . Ako je novi l veći od starog, stari postaje novi.

Nakon petlje se kreira nova lista za pohranu LIS-a. Varijabla k se inicijalizira na poziciju l liste m . U *for*-petlji se iterira unazad od pozicije l manje jedan do nule. Postavlja se sljedeći LIS element i ažurira k u prethodni pohranjen u listi p .

Vraća listu u kojoj je pohranjen najdulji rastući podniz.

U nastavku slijedi primjer nad listom minimizera = („CAT”, 2, 1), („CAT”, 2, 5), („ATG”, 3, 3), („CAT”, 7, 1), („CAT”, 7, 5).

Inicijalizacija:

[(„CAT”, 2, 1), („CAT”, 2, 5), („ATG”, 3, 3), („CAT”, 7, 1), („CAT”, 7, 5)]

$p = [0, 0, 0, 0, 0]$

$m = [0, 0, 0, 0, 0]$

Prva *for*-petlja:

Iteracija 0:

$p = [0, 0, 0, 0, 0]$

$m = [0, 0, 0, 0, 0]$

Iteracija 2:

$p = [0, 0, 0, 0, 0]$

$m = [0, 0, 2, 0, 0, 0]$

Iteracija 4:

$p = [0, 0, 0, 0, 2]$

$m = [0, 3, 2, 4, 0, 0]$

Iteracija 1:

$p = [0, 0, 0, 0, 0]$

$m = [0, 0, 1, 0, 0, 0]$

Iteracija 3:

$p = [0, 0, 0, 0, 0]$

$m = [0, 3, 2, 0, 0, 0]$

Iteracija 5:

$p = [0, 0, 0, 0, 2]$

$m = [0, 3, 2, 4, 0, 0]$

Druga for-petlja:

Iteracija 0:

LIS = [0, 0, („CAT”, 7, 5)]

Iteracija 1:

LIS = [0, („ATG”, 3, 3),
 („CAT”, 7, 5)]

Iteracija 2:

LIS = [(„CAT”, 2, 1), („ATG”,
 3, 3), („CAT”, 7, 5)]

Dobiveni najdulji rastući podniz nad pozicijama u drugom nizu je LIS = [(„CAT”, 2, 1), („ATG”, 3, 3), („CAT”, 7, 5)].

2.3 Poravnanje

Poravnanje je najkompleksniji dio kod problema pronalaska mutacija korištenjem minimizera. Kompleksnost proizlazi iz postojanja umetanja i brisanja. Oni mijenjaju broj gena u sekvenci te ih je iz tog razloga teško odrediti. Problem se pokušava riješiti na dva načina. Prvi način je pokušati samostalno odrediti gdje bi mogla biti umetanja, brisanja ili supstitucije. Drugi način je korištenjem dinamičkog programiranja koje bi trebalo dati točniji rezultat.

Kod prvog načina bez dinamičkog programiranja, ideja je bila usporediti duljine dijelova sekvence između pronađenih zajedničkih minimizera. Ako je duljina jednaka, pretpostavlja se mogućnost postojanja supstitucije te se bilježe svi geni s niza očitavanja. U slučaju da je dio referentnog genoma veći od dijela očitavanja, smatra se postojanje brisanja gena. U suprotnom se smatra umetanje.

Kod supstitucije je jednostavno usporediti dva jednaka niza, ali kod brisanja i umetanja situacija nije toliko trivijalna. Najjednostavniji način koji je bio korišten je prolaženje onim dijelom koji je kraći te za svaki gen pronaći indeks prvog gena koji se pojavljuje u duljem dijelu. Ako takav gen ne postoji ili ako postoji, ali se nalazi predaleko tako da je broj neposječenih gena kraćeg dijela veći od u tom slučaju broja neposječenih gena duljeg dijela, onda se gen zapisuje na trenutnu poziciju te se smatra supstitucijom.

Slijedi primjer.

AATG i CACC				AATG i CT				CT i AATG			
A	A	T	G	A	A	T	G	C	-	T	-
C	A	C	C	C	-	C	-	A	A	T	G

Drugi način koristi dinamičko programiranje. Algoritam koji se koristi se zove Smith-Waterman. On služi za lokalno poravnanje sekvenci što znači da ne gleda sekvencu u cijelosti. Algoritam su prvi predložili Temple F. Smith i Michael S. Waterman 1981. godine [5]. Smith-Waterman algoritam je varijacija Needleman-Wunsch algoritma. Međutim, za razliku od Needleman-Wunsch-a, postavlja negativne vrijednosti bodovanja ćelija matrice na nulu što rezultira vidljivim lokalnim poravnanjima [6]. Algoritam je kvadratne kompleksnosti i zato nije jako pogodan za velike probleme.

Prvo je potrebno odrediti matricu supstitucija i vrijednost kazne za umetanja i brisanja. Svakoj supstituciji baza je dodijeljena vrijednost. Podudaranja dobiju pozitivne vrijednosti, a razlike dobiju negativne. Ako je podudaranju dana vrijednost 1, a razlici -1, matrica će biti sljedeća:

	A	C	G	T
A	1	-1	-1	-1
C	-1	1	-1	-1
G	-1	-1	1	-1
T	-1	-1	-1	1

Nakon toga se treba inicijalizirati matrica bodovanja. Njene dimenzije su duljina prve sekvence uvećano za jedan naspram duljine druge sekvence uvećano za jedan. Elementi nultog retka i stupca se inicijaliziraju na nulu. Taj dodatan redak i stupac omogućavaju poravnanje na bilo kojoj poziciji. Inicijalizirana matrica je sljedeća:

		b_1	...	b_m
	0	0	...	0
b_1	0			
...	...			
b_n	0			

Nakon toga se obavlja bodovanje. Boduje se svaki element od lijeva na desno, gore prema dolje. Ako su sve vrijednosti za pojedinu ćeliju negativne, onda se piše nula. Inače se uzima najveća zaračunata vrijednost i bilježi njegovo podrijetlo.

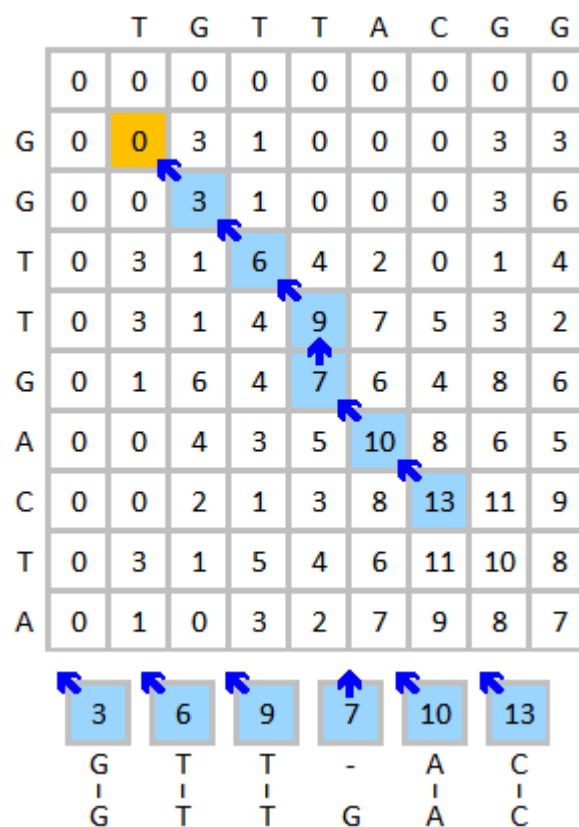
Za sekvence TGTTACGG i GGTGACTA, matrica bodovanja je sljedeća:

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

Sl. 1 – prikaz bodovanja pomoću matrice

Na kraju ostaje pronaći element s najvećim rezultatom i od njega rekurzivno putujući unazad, graditi niz. Dobiveni niz je traženo lokalno poravnanje. Uspješno su pronađene supstitucije, umetanja i brisanja.

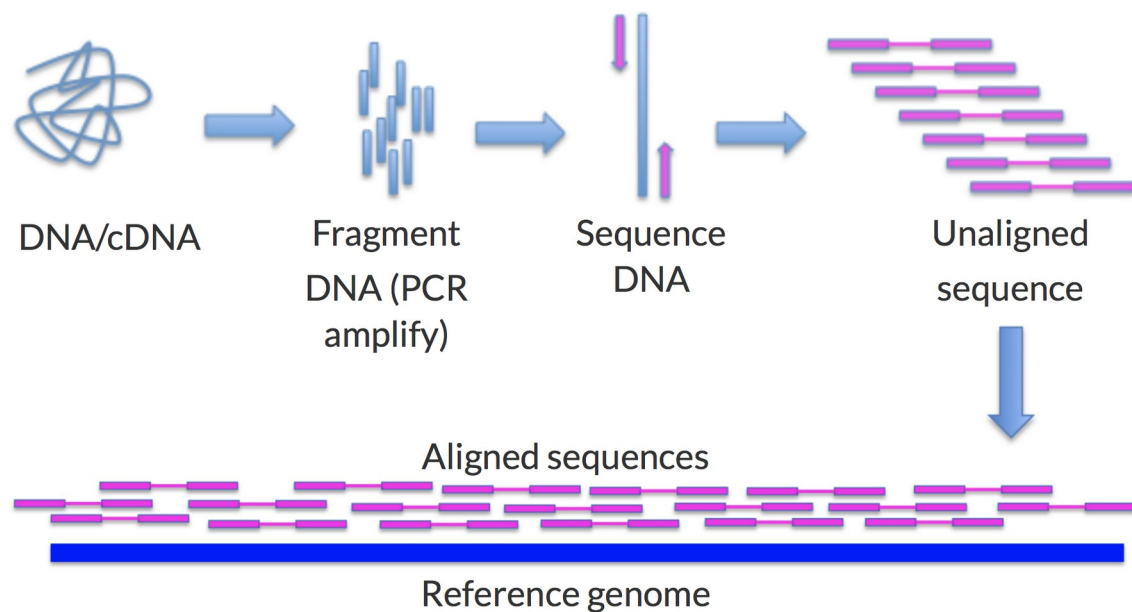
Na slici se može vidjeti izgrađeni niz i kako on služi u poravnanju.



Sl. 2 – prikaz poravnanja korištenjem Smith-Waterman algoritma

2.4 Pronalazak mutacija

Nakon poravnanja svih očitavanja s referentnim genomom i pronalaska umetanja, brisanja i supstitucije, ostaje pronaći koji geni su mutacije.



Sl. 3 – prikaz poravnanja sekvenci očitavanja s referentnim genomom

Mutirani geni se pronalaze na način da se za svaki gen u referentnom genomu pogleda stupac koji sadrži presjek očitavanja na odabranoj poziciji. Zatim se odabere gen najveće frekvencije. On ima najveću vjerojatnost da je točan gen. Ako se odabrani gen razlikuje od gena na referentnom genomu na istoj poziciji, onda je to mutacija. Mutacije mogu biti umetanja, brisanja i supstitucije.

3. Rezultati

Korišten je programski jezik Python te se program izvodio na računalu s procesorom Intel® Core™ i7-6700HQ CPU.

Prvi način poravnavanja sekvenci koji ne koristi dinamičko programiranje je imao loš rezultat. Većina takozvanih mutacija je bilo brisanje. Može se zaključiti da je velika razlika dijelova sekvenci između minimizera glavni razlog za takvu posljedicu. Čak i nakon izbacivanja očitavanja koja sadrže taj problem, rezultat se i dalje sastojao pretežito od mutacija brisanja.

Cjelokupno izvođenje programa trajalo je četrdesetak minuta, što baš i nije dobro.

Drugi način poravnanja je imao mnogo bolje rezultate u odnosu na prvi način. Razlikuje se od izvornog rješenja, ali postoje podudaranja.

Cjelokupno izvođenje programa koristeći drugi način, trajalo je tridesetak minuta, što je malo bolje u odnosu na prvi način, ali nije najbolje.

4. Zaključak

Minimizeri su dobra kombinacija uz treću generaciju sekvenciranja koja pruža duga očitavanja. Koristeći algoritam za najdulji zajednički podniz sekvence referentnog genoma i očitavanja, moguće je poravnati sekvence. Poravnanje se pokušava obaviti na dva načina.

Prvi način nije upotrebljiv, no uvođenje dinamičkog programiranja poboljšava situaciju i pronalazi puno točnije rezultate. Zasigurno postoji neki drugi način za samostalno određivanje pojavljivanja supstitucije, umetanja i brisanja. Ovaj nije bio dobro osmišljen te je zato rezultirao loše određenim mutacijama.

Drugi način je znatno bolji. Dinamičko programiranje u Smith-Waterman algoritmu pruža uspješno lokalno poravnanje. To rezultira boljim konačnim mutacijama.

Može se zaključiti da provjereni algoritmi osmišljeni za poravnanje i pronalazak umetanja, brisanja i supstitucija će skoro uvijek biti bolji od samostalno osmišljenih algoritama. Veoma inteligentni ljudi koji su posvetili puno vremena istraživanju u području bioinformatike su osmislili algoritme poput Smith-Waterman-a. Potrebno je doći na tu razinu za osmišljavanje drugih boljih načina.

Sporo izvođenje je zahvaljujući prirodi programskog jezika Python i ne optimalnosti koda. Pretpostavka je da bi se uvođenjem paralelnog programiranja mogla unaprijediti vremenska performansa.

5. Literatura

Pisana djela:

[1] Shaw, J., Yu, W. Y., Theory of local k-mer selection with applications to long-read alignment, Department of Mathematics, University of Toronto, 9. studeni 2021.

[2] Roberts, M., Hayes, W., Hunt, B. R., Mount, S. M., Yorke, J. A., Reducing storage requirements for biological sequence comparison, Institute for Physical Science and Technology, University of Maryland, College Park, Vol. 20 no. 18 2004, pages 3363 3369 doi:10.1093/bioinformatics/bth408

[3] Schensted, C. (1961), "Longest increasing and decreasing subsequences", Canadian Journal of Mathematics, 13: 179–191, doi:10.4153/CJM-1961-015-3

[4] Longest increasing subsequence, Wikipedia, 24. veljača 2023., link: https://en.wikipedia.org/wiki/Longest_increasing_subsequence
datum pristupa: 31. svibanj 2023.

[5] Smith, Temple F. & Waterman, Michael S. (1981). "Identification of Common Molecular Subsequences". Journal of Molecular Biology. 147 (1): 195–197. CiteSeerX 10.1.1.63.2897. doi:10.1016/0022-2836(81)90087-5. PMID 7265238.

[6] Gagniuc, Paul A. (2021). Algorithms in bioinformatics : theory and implementation (1st ed.). Hoboken, NJ: John Wiley & Sons. pp. 1–528. ISBN 978-1-119-69800-5. OCLC 1240827446.

Slike:

[Sl. 1 i Sl. 2] Smith–Waterman algorithm, Wikipedia, 19. veljača 2023., link: https://en.wikipedia.org/wiki/Smith–Waterman_algorithm

datum prisutpa: 31. svibanj 2023.

[Sl. 3] Reference Genomes and Genomics File Formats, Rockefeller University, Bioinformatics Resource Centre, 7. ožujak 2023. link: https://rockefelleruniversity.github.io/Genomic_Data/presentations/slides/GenomicsData.html#1

datum prisutpa: 31. svibanj 2023.