

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

Akadska godina: 2019./2020.

Pronalazak mutacija pomoću treće generacije sekvenciranja

Ilija Domislović, Andrej Mijić, Zorica Žitko

Zagreb, siječanj 2020.

Sadržaj

1. Uvod.....	3
2. <i>Minimzeri</i> [1].....	3
3. Implementacija i rezultati.....	7
4. Zaključak	9
5. Literatura	10

1. Uvod

Prepoznavanje mutacija sve više dobiva na važnosti u modernoj medicini – bilo da se radi o ranom prepoznavanju raka, bakterijama koje postaju otporne na antibiotike ili pak nasljednim bolestima. Moderne metode sekvenciranja i daljnji napretci u bioinformatičari sa sobom donose obećanje precizne ili personalizirane medicine – pristupa koji bi omogućio izradu terapije za svakoga pacijenta u ovisnosti o posebnostima njegovog genetskog profila uzimajući u obzir i vanjske čimbenike.

Za uspješnu detekciju mutacija nužno je poravnavanje niza očitavanja s referentnim genomom. Očito, riječ je o postupku koji je vremenski i memorijski zahtjevan, posebno u slučaju duljih očitavanja kakva dobivamo sekvenciranjem treće generacije. Roberts i drugi u svome su članku „Reducing storage requirements for biological sequence comparison” [1] predstavili jedno moguće poboljšanje tehnike začetaka (eng. *seeds*) koje su nazvali *minimizerima*. Osnovnu ideju toga algoritma i nekoliko jednostavnih primjera donosimo u poglavlju koje slijedi.

2. *Minimzeri* [1]

Recimo da želimo pronaći slične podnizove dvaju dugih nizova N_1 i N_2 . Pretpostavka je da možemo koristiti kraće podnizove, začetke, za traženje potencijalno duljih preklapanja nizova N_1 i N_2 . Stoga je prvi korak ovakvoga postupka odabir skupa začetaka iz N_1 i N_2 koji dobro predstavljaju te nizove. Koristimo začetke koji su neprekinuti podnizovi duljine k znakova, tzv. k -meri. Najjednostavniji učestali pristup traženju k -mera zajedničkih dvama nizovima (ili skupu nizova) pamćenje je svakog k -mera koji se pojavljuje u pojedinom nizu na poziciji i . Promotrimo li niz *ARNGHS*, vidimo da on sadrži 3 4-mera: *ARNG*, *RNGH*, *NGHS*. Popišemo li sve k -mere dobivene iz nekog skupa nizova i sortiramo li ih, primjetit ćemo da su se isti k -meri našli jedni uz druge, što je idealno ishodište za traženje duljih preklapanja. Jasno je da pamćenje svih k -mera iziskuje značajnu količinu memorije – broj k -mera u nizu N_i iznosi $|N_i| - k + 1$, pri čemu je $|N_i|$ duljina niza, a dodatno moramo pamtit i niz kojemu pojedini k -mer pripada, kao i poziciju u tome nizu na kojoj počinje, tj. pamtimo k -mer uređenu trojku (s, i, p) ¹. Očito je da bi pamćenje

¹ s – niz znakova duljine k , i – redni broj promatranog niza, p – pozicija u nizu i na kojoj počinje s

manje k -mera rezultiralo manjim zahtjevima na memoriju. k -mere bi trebalo birati tako da i dalje preklapanja uočavamo čim je baza k -mera sortirana, odnosno, htjeli bismo da nizovi N_1 i N_2 dijele neki reprezentativni k -mer ako dijele i dovoljno dugačak podniz. Preciznije, takvi posebno odabrani k -meri, zovemo ih *minimizeri*, zadovoljavaju sljedeće svojstvo: „Ako dva niza imaju značajno preklapanje, tada će barem jedan od *minimizera* izdvojenih iz prvog niza biti izdvojen i iz drugog.“ [1]

Za odabir *minimizera* nužno je odabrati pravilo za sortiranje k -mera, najčešće jednostavno leksikografski poredak, pa bi tako AAAA bio najmanji 4-mer. Razlikujemo unutarnje (eng. *interior*) i krajnje (eng. *end*) *minimizere*.

Skup w uzastopnih k -mera čini niz duljine točno $w + k - 1$ znakova, pri čemu su dva uzastopna k -mera jedan u odnosu na drugoga posmaknuti za jedan znak. *Minimizer* je najmanji od w k -mera (uz poštivanje odabranog poretka). U slučaju izjednačenja svaki od najmanjih k -mera biva proglašen *minimizerom*. Kažemo da je k -mer uređena trojka (s, i, p) (w, k) -*minimizer* niza N_i ako je *minimizer* za prozor veličine w uzastopnih k -mera koji ju sadrži. Iz toga izravno slijedi: „Ako dva niza imaju zajednički podniz duljine $w + k - 1$, tada dijele i (w, k) -*minimizer*.“ [1]

Tablica 1. Odabir $(6, 7)$ -*minimizera* između 6 uzastopnih 7-mera u nizu duljine 12 ($w = 6, k = 7, l = 12$), prema [1]

POZICIJA	1	2	3	4	5	6	7	8	9	10	11	12
SEKVENCA	1	4	1	2	4	8	5	6	1	1	2	3
	1	4	1	2	4	8	5					
		4	1	2	4	8	5	6				
			1	2	4	8	5	6	1			
				2	4	8	5	6	1	1		
					4	8	5	6	1	1	2	
						8	5	6	1	1	2	3

U Tablici 1. prikazan je jednostavan primjer biranja $(6, 7)$ -*minimizera*, pri čemu svaki redak predstavlja jedan 7-mer u prozoru veličine 6, a odabrani *minimizer* je podebljan. Slično, u Tablici 2. prikazan je odabir $(4, 3)$ -*minimizera* iz niza duljine 15. Možemo uočiti sljedeće: uzastopni prozori često dijele *minimizere*, ali moguć je i nastanak procijepa između *minimizera*. Procijep

nastaje kada su *minimizatori* dvaju susjednih prozora udaljeni više od k pozicija, što rezultira neporkivenošću pojedinih pozicija *minimizatora*. Želimo li spriječiti nastanak procijepa, dovoljno je odabrati $w \leq k$. Ipak, to ostavlja mogućnost da sa svake strane (sa svakoga kraja) najviše $w - 1$ znak ostane izvan bilo kojeg *minimizera*, što je razlog za uvođenje krajnjih *minimizera*. (u, k) -krajnji *minimizer* je (u, k) -*minimizer* odabran iz prozora veličine u koji započinje na jednome kraju niza, a skup k -krajnjih *minimizera* sastoji se od svih takvih (u, k) -*minimizera* čija se veličina prozora kreće od 1 do neke najveće vrijednosti v . Krajnji *minimizatori* zadovoljavaju sljedeće svojstvo: „Ako krajevi dvaju nizova imaju preklapanje duljine između k i $k + v - 1$ znakova, tada oni dijele barem jedan k -krajnji *minimizer*.“ [1]

Kombinirajući (w, k) -*minimizere* s (u, k) -krajnjim *minimizerima* uz $u = 1, \dots, w - 1$ i $w \leq k$ postićemo prekrivanje svakoga znaka u nizu barem jednim *minimizerom*.

Tablica 2. Odabir $(4, 3)$ -*minimizera*, pri čemu jedan redak predstavlja sadržaj cijeloga prozora čiji je *minimizer* podebljan. Pojaljuju se procijepi. Prema [1]

POZICIJA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SEKVENCA	5	4	2	1	7	6	1	0	1	4	6	6	2	1	2
	5	4	2	1	7	6									
		4	2	1	7	6	1								
			2	1	7	6	1	0							
				1	7	6	1	0	1						
					7	6	1	0	1	4					
						6	1	0	1	4	6				
							1	0	1	4	6	6			
								0	1	4	6	6	2		
									1	4	6	6	2	1	
										4	6	6	2	1	2

Tablica 3. Odabir (3, 3)-minimizera za istu sekvencu kao u Tablici 2. kako bi se izbjeglo stvaranje procijepa između minimizera, pri čemu jedan redak predstavlja sadržaj cijeloga prozora čiji je minimizer podebljan. U najgorem slučaju krajnjih $w - 1$ znak nije pokriven niti jednim minimizerom. Prema [1]

POZICIJA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SEKVENCA	5	4	2	1	7	6	1	0	1	4	6	6	2	1	2
	5	4	2	1	7										
		4	2	1	7	6									
			2	1	7	6	1								
				1	7	6	1	0							
					7	6	1	0	1						
						6	1	0	1	4					
							1	0	1	4	6				
								0	1	4	6	6			
									1	4	6	6	2		
										4	6	6	2	1	
											4	6	2	1	2

Tablica 4. Odabir 3-krajnjih minimizera za lijevi kraj niza. Biramo po jedan (u, 3)-minimizer za veličine prozora od 1 do najviše $l - k + 1$, gdje je l duljina niza. Prema [1]

POZICIJA	1	2	3	4	5	6	7	8	9	10
SEKVENCA	4	3	2	1	4	3	2	1	2	3
	4	3	2							
	4	3	2	1						
	4	3	2	1	4					
	4	3	2	1	4	3				
	4	3	2	1	4	3	2			
	4	3	2	1	4	3	2	1		
	4	3	2	1	4	3	2	1	2	
	4	3	2	1	4	3	2	1	2	3

Tablica 5. Krajnji minimizeri za niz iz Tablice 3. Kombinirajući ova dva minimizera s onima određenima u Tablici 3. osiguravamo pokrivenost svake baze barem jednim minimizerom. Prema [1]

POZICIJA	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SEKVENCA	5	4	2	1	7	6	1	0	1	4	6	6	2	1	2
	5	4	2												
													2	1	2

3. Implementacija i rezultati

Ulaz u program dvije su .FASTA datoteke, jedna sadrži referentni genom, a druga skup očitavanja dobiven sekvenciranjem mutiranog genoma, dok je izlaz .CSV datoteka koja sadrži listu mutacija u odnosu na referencu kako je navedeno u Tablici 6. Dodatno, uz .FASTA datoteke za pokretanje nužno je navesti i argumente koji se odnose na veličinu prozora i *minimizera*, broj dretvi te minimalan broj očitavanja koja se „slažu“ oko promjene u odnosu na referentni genom.

Tablica 6. Opis izlaza iz programa

MUTACIJA	LINIJA U .CSV DATOTECI		
SUPSTITUCIJA	X	Pozicija u referenci na kojoj se dogodila supstitucija	Zamjenska nukleotidna baza
UMETANJE	I	Pozicija u referenci prije koje se dogodila supstitucija	Umetnuta nukleotidna baza
BRISANJE	D	Pozicija u referenci na kojoj se dogodilo brisanje	-

Nakon učitavanja referentnog genoma i skupa očitavanja, traže se *minimizeri* za referentni genom i pridružuju im se indeksi, koji kasnije zamjenjuju konkretne nizove nukleotidnih baza. Potom se za svako od očitavanja pronađu *minimizeri* i očitavanja se također indeksiraju, pri čemu se *minimizerima* koje sada pronalazimo u očitanjima, a koji su prethodno zabilježeni u referentnom genomu, pridružuju indeksi jednaki onima *minimizera* u referentnome genomu. S obzirom na mogućnost da je očitavanje dobiveno sekvenciranjem reverznog komplementa niza koji se koristi kao referentni genom, generira se reverzni komplement svakog od očitavanja i provodi se isti postupak traženja *minimizera* i indeksiranja kao i za očitavanja. Kako ne bismo

morali pretraživati cijeli referentni genom za preklapanja, algoritmom najduljeg zajedničkog podniza (eng. *Longest Common Subsequence, LCS*)² pokušavamo utvrditi na kojemu bi se dijelu referenca najbolje mogla preklapati s pojedinim očitanjem, odnosno reverznim komplementom tog očitanja³. LCS vraća indekse u referenci na kojima očekivano preklapanje počinje i završava, kao i duljinu dobivenog podniza. Ako je rezultat izvođenja takav da bi između početnog i završnog indeksa reference trebalo postojati preklapanje barem dvostruko dulje od promatranog očitanja, zaključujemo da preklapanje ne postoji i takvo očitanje ne pokušavamo poravnati s referentnim genomom u kasnijem koraku. Ustanovimo li da preklapanje postoji, dio reference određen dobivenim indeksima i pojedino očitanje poravnavaju se Hirschbergovim algoritmom⁴. Jednom kada su sva očitanja poravnata s referencom, za pojedinu poziciju u referentnom genomu provodimo „glasanje“ – gledamo što se na toj poziciji nalazi u svakom pojedinom očitanju i kao rezultat zapisujemo promjenu koja se pojavljuje u najvećem broju očitanja. Da bismo promjenu iz očitanja smatrali mutacijom uvodimo prag – barem n očitanja mora na toj poziciji sadržavati istu promjenu. Intuitivno, što je veća pokrivenost, to bi se više očitanja trebalo „složiti“ kako bi smo odstupanje od reference na pojedinoj poziciji zapisali kao mutaciju. Jednom kada su sve promjene pobrojane, konstruira se izlazna datoteka na način opisan u Tablici 6. Za procjenu kvalitete rezultata koristi se Jaccardov indeks [2].

² Implementiran prema pseudokodu na [3]

³ U slučaju da je dulji podniz dobiven korištenjem reverznog komplementa očitanja, on se koristi i u ostatku postupka. Zbog jednostavnosti, u tekstu se nadalje riječ „očitanje“ koristi za niz za koji je algoritmom LCS ustanovljeno bolje preklapanje s referencom, bilo da se radi o stvarnom očitanju ili o njegovom reverznom komplementu.

⁴ Implementiran prema pseudokodu iz [4]

Rezultati testiranja programa ispitnim skupovima lambda i E. coli nalaze se u Tablici 7.

Tablica 7. Rezultati dobiveni pokretanjem programa s veličinom prozora 100 (kmer_size), veličinom minimizera 20 (minimizer_size), 8 dretvi za ispitni skup E. coli, 4 dretve za ispitni skup Lambda (number_of_threads) i pragom 5 za prihvaćanje promjene u odnosu na referentni genom (mutation_voting_threshold)

Ispitni skup	Vrijeme [min]	Memorija [MB]	Jaccardov indeks	Jaccardov indeks (referentna implementacija)
Lambda	0.4	300	0.6919	0.4538
E. coli	~180	550	0.8194	0.8249

4. Zaključak

Prepoznavanje mutacija važan je, ali memorijski je i vremenski zahtjevan postupak čijem se smanjenju složenosti može pristupiti na različite načine. U okviru ovoga projekta razmatrani su *minimizeri*, koji predstavljaju unaprjeđenje osnovne metode začetaka. Ideja je duge nizove predstaviti njima karakterističnim podnizovima bez pamćenja velikog broja takvih podnizova. Iz rezultata, kao i iz rada u kojemu je navedena metoda prvi puta opisana [1], vidljivo je da je riječ o metodi koja zahtijeva značajno manje računalnih resursa, a da istovremeno ne narušava kvalitetu rezultata.

5. Literatura

- [1] M. Roberts, W. Hayes, B. R. Hunt, S. M. Mount and J. A. Yorke, "Reducing storage requirements for biological sequence comparison," *Bioinformatics*, vol. 20, no. 18, p. 3363–3369, 2001.
- [2] Wikipedia, "Jaccard index," 22 October 2019. [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index. [Accessed 14 January 2020].
- [3] Wikipedia, "Longest common subsequence problem," 31 December 2019. [Online]. Available: https://en.wikipedia.org/wiki/Longest_common_subsequence_problem. [Accessed 14 January 2020].
- [4] M. Šikić and M. Domazet-Lošo, "Bioinformatika," Sveučilište u Zagrebu, Fakultet elektrotehnike i računarstva, Zagreb, 2013.