SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Pronalazak mutacija pomoću treće generacije sekvenciranja

Josip Kasap, Mislav Jakšić
Voditelj: *Mag. ing. Robert Vaser*

Zagreb, siječanj 2019.

SADRŽAJ

1. Uvod

Sekvenciranje genoma je čest postupak u računalnoj biologiji. Kako bi se odredio izgled genoma, to jest sekvencirao, iz stanice jedinke treba izolirati deoksiribonukleinsku kiselinu. Kada je DNK izolirana ona se umnaža te se kemijskim postupcima umnožene DNK "režu" na manje komadiće. Sada je potrebno očitati svaki komadić DNK.

Sangerov postupak je jedna od prvih metoda očitavanja DNK. Sangerova metoda je vrlo precizna, ali spora metoda koja proizvodi kratka očitanja od po nekoliko desetaka nukleotidnih baza. Navedene nedostatke pokušali su otkloniti postupci sekvenciranja druge generacije. Metode druge generacije očitavaju komadiće DNK s greškom do 3% ali su očitanja dulja, do dvije stotine nukleotidnih baza. Trgovačko društvo koje prednjači u opisanom sekvenciranju je Illumina. U nadi da će se dodatno smanjiti troškovi sekvenciranja razvijeni su postupci sekvenciranja treće generacije. Ove metode očitavaju komadiće DNK s do 30% pogreške, ali su očitanja jako velika, do tisuću nukleotidnih baza. Nažalost, bez posebnih postupaka ispravljanja očitanja, ona su beskorisna zbog velike pogreške [2].

U daljnjem tekstu objašnjeni su algoritmi koji koriste očitanja dobivena trećom generacijom sekvenciranja. Algoritmi obrađuju očitanja, poravnaju ista s referentnim genomom i zatim razluče koje nukleotidne baze su mutirane, a koje su pogrešno očitane.

2. Minimizacija K-mera

2.1. Pozadina

Potrebno je odrediti gdje se podniz znakova nalazi u nizu znakova. Naivni način potrage bila bi uzastopna usporedba znaka podniza s znakom niza sve dok svi uzastopni znakovi nisu isti ili dok ne dođemo do kraja niza u kojem slučaju bi zaključili da se podniz ne nalazi u nizu. Vremenska složenost naivnog algoritma je O(N) gdje je N duljine niza u kojem tražimo podniz.

U naivnom algoritmu krije se nevidljiva pretpostavka: oba niza su nemutirana. Ako postoji i jedan mutirani znak, naveden postupak postaje nemoguće izvesti usprkos tome što se niz i podniz poklapaju u svim ostalim znakovima. U računalnoj biologiji, savršeni, nemutirani podnizovi su rijetkost, pa je potreban drugačiji postupak usporedbe.

Postoji bolji način traženja podniza u nizu, a to je uspoređivanjem K-mera. K-mer je podniz uzastopnih znakova duljine K. Tako svi K-meri duljine 3 niza 2310343 su 3-meri: 231, 310, 103, 034 i 343.

Umjesto da uspoređujemo znak sa znakom, usporedit ćemo na kojim se mjestima u nizu K-meri podudaraju. Sada, čak i ako je dio niza mutiran, i dalje je moguće naći barem neki dio koji je nemutiran i jednak u nizu i podnizu. Nažalost, K-meri ne garantiraju da se na nekom mjestu podniz i niz preklapanje zbog čega se K-meri nekad nazivaju i "sjemenke".

Kako bi našli mjesto najboljeg preklapanja niza i podniza potrebno je provjeriti okolinu za svaki K-mer i potom izabrati ono mjesto gdje je preklapanje najbolje. Memorijska složenost K-mer algoritma je O(K*N) gdje je K duljina K-mera i N duljina niza.

Kako bi smanjili memorijsku složenost, umjesto da spremamo svaki K-mer u memoriju, možemo spremiti samo neke pametno odabrane K-mere. Najmanji K-mer u

skupu od W K-mera, gdje je W broj uzastopnih K-mera koji se uspoređuju, zove se minimizacijski K-mer ili samo minimizator. U prošlom primjeru, sortiranjem svih K-mera dobivamo sljedeći slijed: 034, 103, 231, 310 i 343. Ako je W jednak 5 onda je minimizator prošlih K-mera 034 [1].

2.2. Implementacija

Algoritam kao ulaz uzima niz znakova, te parametre K koji određuje duljinu K-mera i L koji određuje duljinu prozora unutar kojeg se traži K-mer minimizator.

Za svaki lanac DNK algoritam izluči K-mere unutar prozora veličine L. Među svim K-merima algoritam odabire najmanji K-mer kao minimizator. Nakon što je K-mer minimizator pronađen na poziciji P, prozor skače na poziciju P + 1, te se postupak minimizacije nastavlja dok ne dođe do kraja niza.

Razlog zašto prozor može skočiti na poziciju P + 1 bez straha da će neki K-mer minimizator biti izostavljen je sljedeći: ako postoji K-mer minimizator između trenutne pozicije prozora i pozicije P + 1, onda bi pozicija K-mer minimizator bila manja od P + 1. To znači da je pozicija P nužno pozicija na kojem se nalazi minimizator, što je u skladu s definicijom.

Memorijska složenost algoritma je O(1), dok je vremenska složenost O(N), gdje je N duljina ulaznog niza znakova.

Niz znakova:	2	3	1	0	3	4	3
	2	3	1	-	-	-	-
	-	3	1	0	-	-	-
	-	-	1	0	3	-	-
Minimizator:	-	-	-	0	3	4	-
	-	-	-	-	3	4	3

Tablica 2.1: K = 3, L = 7, minimizator niza je 034

3. Poravnanje svih fragmenata s referentnim genomom

Kao ulazni podatak dobivamo listu svih fragmenata zajedno s njihovim minimizatorima te indeks minimizatora za referentni genom, a očekivani izlaz je lista svih mutacija u CSV formatu (mutacija, pozicija, zamjenska_baza). Da bismo dobili takav izlaz prvo trebamo poravnati svaki fragment s referentnim genomom. Za svaki fragment i za njegove minimizatore pomoću indeksa minimizoatora iz referentnog genoma dobijemo poziciju gdje se taj minimizator nalazi u referentnom genomu. Ako ta pozicija postoji onda se provjerava mala okolina oko minimizatora (5 nukleotidnih baza u oba smjera) koja mora biti identična u fragmentu i referentnom genomu tako da preklapanje minimizatora nije slučajno.

Na ovaj način dobijemo listu svih pozicija nukleotidnih baza na referentnom genomu i na fragmentu koje označavaju pozicije u kojima su im minimizatori i okolina uz minimizatore isti. Označimo sa l listu, a s n broj elemenata te liste. Element liste l[i] (i<n) ima 2 vrijednosti. Prva vrijednost označava poziciju preklapajućeg minimizatora u referentnom genomu, a druga vrijednost označava poziciju preklapajučeg minimizatora u fragmentu. Označimo te dvije vrijednosti s l[i].x i l[i].y. Fragment se na ovaj način dijeli na n+1 djelova, (kao da ga režemo nožem n puta) u kojem svaki dio predstavlja određenu sekvencu koju trebamo poravnati algoritmom globalnog poravnanja. Npr. jedan mogući dio za poravnanje bi bio referentni genom od pozicije l[1].x + k -1 do pozicije l[2].x, s fragmentom od pozicije l[1].y + k -1 do pozicije l[2].y. Sada ako želimo poravnati cijeli fragment s genomom, umjesto da ih poravnavamo kao 2 ogromna stringa kojima neznamo ni početnu ni krajnju poziciju (za referentni genom) i takvo poravnanje bi trajalo predugo i bilo vremenski prezahtjevno, možemo poravnati n + 1 djelova fragmenta kojima točno znamo početne i krajnje pozicije. Problem nam predstavljaju prvi i zadnji djelovi, u prvom djelu ne znamo koja je točna pozicija početka u referentnom genomu, a u zadnjem djelu neznamo koja je točna pozicija kraja

u referentnom genomu. Prvi dio možemo gledati u obrnutom poretku, na način da kraj prvog djela (p[0].x za genom i p[0].y za fragment) bude tretiran kao početak, a njegov kraj je onda pozicija 0 za fragment, samo što cijeli taj string gledamo u obrnutom poretku. Gledajući problem na taj način, problem poravnanja početka je jednak kao problem poravnanja kraja i njih moramo riješiti zasebno.

Za poravnanje 2 stringa koji nisu početni i krajnji koristimo algoritam globalnog poravnanja, a za poravnanje 2 string koji su početni i krajnji algoritam poluglobalnog poravnanja [2]. Jedina modifikacija jest da se koristi ograničeno dinamičko programiranje radi uštede na prostoru i vremenu. Kada se fragment poravna s referentnim genomom za njega se može dobiti minimalna lista svih prijelaza kojih je potrebno napraviti na referentnom genomu da se dobije fragment, i ta lista je lista mutacija za fragment. Nakon što dobijemo listu svih mutacija za fragment svaka mutacija iz liste se ubacuje u mapu svih mutacija. U mapi svih mutacija se mapira pozicija mutacije s listom svih mutacija iz svih fragmenta na toj poziciji. Ova mapa je jako bitna, budući da je pogreška fragmenata otprilike 15%, da bi saznali koja je prava mutacija na određenoj poziciji u referentnom genomu (ako je uopće i ima) moramo pogledati u mapu svih mutacija na toj poziciji te izabrati mutaciju koja se najčešće ponavlja. Da bismo odlučili koja je mutacija bitna, a koja nije potrebna nam je još jedna mapa koja mapira poziciju referentnog genoma s brojem fragmenata koji ga prekrivaju. Ako se najčešća mutacija ponavlja u barem pola svih fragmenata koji prekrivaju tu poziciju onda za nju možemo reći da je ispravna mutacija.

Nakon što dobijemo listu svih ispravnih mutacija nju možemo vratiti kao izlaz programa.

4. Rezultati

Na ulaznim podacima bakteriofaga, dobiveni su sljedeći rezultati na procesoru Intel Core i3-4330 (Q3'13):

K	L	Vrijeme[sec]	Memorija[MB]	Jaccardov indeks
20	50	162	∼260	0,73
40	50	209	∼380	0,74
20	100	242	∼200	0,71
40	100	267	∼300	0,72

Tablica 4.1: Reultati mjerenja

K je veličina K-mera, te kako se ona povećava, točnost se također povećava jer veći K-mer znači veće područje podudaranja. L je veličina prozora unutar kojeg se traže K-mer minimizatori. Kako se L smanjuje, tako se zauzeće memorije povećava jer je potrebno pamtiti više K-mera.

5. Zaključak

Kao što rezultati pokazuju, program traje dugo i zauzima puno memorije.

Indeks minimizatora trenutno je implementiran kao tablica ključeva. Ako bi se implementirao samostojeći FM-indeks, zauzeće memorije bi se znatno smanjilo.

6. Literatura

[1] Brian R. Hunt Stephen M. Mount i James A. Yorke Michael Roberts, Wayne Hayes. Reducing storage requirements for biological sequence comparison, 16. Travnja 2004.

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