SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKTNI IZVJEŠTAJ

PRONALAZAK MUTACIJA POMOĆU TREĆE GENERACIJE SEKVENCIRANJA

Ivan Moštak, Luka Jukić

Zagreb, siječanj 2019.

Sadržaj

Uvod

Sekvenciranje treće generacije dozvoljava DNA očitanja velikih duljina (na razini molekule, za razliku od postojećih metoda koje se koriste kidanjem DNA na fragmente te potom amplifikacije) na brz način, ali uz veliku pogrešku što čini ovu metodu upotrebljivom isključivo za određene primjene[3]. Kako bi se omogućio pronalazak mutacija potrebno je više puta očitati mutirani genom kako bi dobili dovoljno veliki uzorak i pravilnim mapiranjem na referentni genom smanjili učinak pogreške.

Postoje mnoge tehnike mapiranja i uspoređivanja nizova, a u našem radu koristili smo metodu s indeksiranjem k-mera (podniza duljine k) u referentnom genomu i očitanjima.

Opis algoritma

Glavni algoritam

Prvi korak algoritma je izvući sve minimizirajuće k-mere iz referentnog genoma (te za svaki k-mer njegovu poziciju unutar reference i poziciju u vektoru kako bi ubrzali izvođenje). Potom za svako očitanje izvući k-mere te ih mapirati na referentni genom. Ako mapiranje nije uspješno, pokuša se napraviti istu stvar s reverznim komplementom sekvence. No kako je u sekvenciranju treće generacije mogućnost pogreške velika, postoji mogućnost da se niti originalna sekvenca niti njen reverzni komplement ne mogu pravilno poravnati s referentnim genomom, jer se ne podudaraju u dovoljnoj mjeri da bi izvukli dovoljan broj istih k-mera.

Nakon što se mapiraju očitanja, za svaku mapiranu sekvencu traže se mutacije tako da tražimo njihovo poravnanje. Pozicija koja je dobivena mapiranjem koristi se za početak pretrage mutacija između referentnog očitanja i očitanja od interesa. Pretraga kreće uspoređujući k-mer reference (koji kreće nekoliko mjesta prije mapirane pozicije, zbog nepreciznosti mapiranja) s određenim brojem k-merova poravnate sekvence. Razlog zbog kojeg se uspoređuje jedan k-mer reference s nekoliko k-merova sekvence je taj da ako je došlo do neke mutacije ili greške prilikom očitavanja, moglo je doći i do pomaka unutar očitanja što može uzrokovati i ekstrakciju različitih k-merova iz reference i očitane sekvence ili pak pomak k-merova te se ne pojavljuju slijedno jedan iza drugog, već su se ili ubacili novi ili je došlo do gubitka. Tijekom traženja mutacija poravnanjem stringova nalazimo mutacije te pamtimo njihov broj pojavljivanja.

Na kraju ispišemo mutacije koje su se najviše pojavile na nekoj poziciji, odnosno ako do mutacije nije došlo, ne ispisuje se ništa.

Ekstrakcija k-mera

Cilj ekstrakcije k-merova je da sve sekvence budu predstavljene stringovima konačne unaprijed određene duljine k [2]. Cilj je da se u slučaju pojave istog segmenta unutar nekoliko sekvenci odaberu uvijek isti slijedovi, kako bi se sekvence mogle usporediti i poravnati te kako bi se u konačnici mogle odrediti mutacije.

Ako je dano N stringova koje je potrebno međusobno usporediti, prvi je korak izvući male stringove koji će predstavljati originalni string. Nije moguće izvlačenje svih substringova duljine k (preveliki memorijski zahtjevi) pa se stvara minimalni broj stringova duljine k na način da ako dvije sekvence sadrže dovoljno duge subsekvence, moraju se uvijek ekstrahirati isti reprezentativni substringovi duljine k - još nazvani i "*minimizers*" [2]. Osim samog stringa, mora se zabilježiti pozicija k-mera unutar originalnog stringa kao i identifikator očitanja iz kojega je k-mer izvučen [2].Postoji nekoliko pravila za ekstrakciju minimizera duljine k iz w slijednih k-merova. K-mer je (w,k)-minimizer za string S ako je minimizer za slijed od w slijednih k-merova sadržanih u stringu S (slijedni string znači da se svaki sljedeći k-mer pomaknut za jednu bazu u odnosu na prethodni k-mer). Tj. u prozoru (prozor je duljine w+k-1) w slijednih k-merova odabire se onaj najmanji, a način uspoređivanja k-mera se određuje na proizvoljan način. Da bi se izbjegli problemi koje uzrokuje ponavljanje iste baze slijedno unutar sekvence i da bi se uvijek izvukle one najrjeđe kombinacije baza (i da bi u konačnici uvijek bili odabrani isti k-merovi ako stringovi sadrže iste substringove) odabire se dodjela brojeva 0, 1, 2 i 3 bazama C, A, T i G na način da se gleda pozicija baze unutar k-mera [2]. Vrijedi sljedeća dodjela brojeva bazi unutar k-mera radi kasnije usporedbe i sortiranja radi ekstrakcije minimizera koja garantira značajnost izvučenog substringa (tablica 1):

baza	Parna pozicija	Neparna pozicija
C	0	3
A	1	2
T	2	1
G	3	0

Tablica 1. Dodjela vrijednosti nukleotidima ovisno o poziciji unutar k-mera

Da se spriječi nenamjerno preskakanje pojedinih baza prilikom ekstrakcije minimizera, potreban je odabir w<=k kako bi bile pokrivene sve baze, tj. sve baze osim onih na početku i kraju stringa (maksimalno w-1 baza može ostati nepokriveno na svakom kraju). No i taj se problem rješava upotrebom (u,k) end-minimizera [2]. (u, k) end - minimizer je izabran u prozoru veličine u(u = 1,..,w-1), a pretraga se obavlja na oba kraja stringa nezavisno i rezultat je da je kombinacijom (w,k)-minimizera i (u,k) end-minimizera pokrivena svaka baza iz originalne sekvence barem jednom! [2]
Kada se radi s DNA sekvencama potrebno je provjeriti i reverzni komplement sekvence. Primjer ekstrakcije (w,k)=(3,4) k-mera za jedan jednostavan slijed nukletioida

“AGGCGCTTATGACT” dan je u tablici 2. Odabran je K-mer 1 zbog toga jer je njegova vrijednost minimalna (“*ordering number”* mu je najmanji) te je on minimizer za dani slijed nukleotida. Da je slijed nukleotida dulji, u sljedećem koraku se pretraga pomiče za jedan nukleotid u desno te se nukleotid T uključuje u pretragu, a nukleotid A s početka izlazi iz aktivnog “prozora” te se definitivno mora odabrati novi minimizer pošto K-mer 1 više nije dio aktivnog mjesta ekstrakcije (tablica 3). Odabire se K-mer 4 jer ima najmanji “*ordering number”.* Na primjeru je također vidljivo kako je svaka baza očitanja pokrivena barem jednom (nalazi se u minimalno jednom minimizeru), a uključivanjem end-mera (i pretrage krajeva sekvence) može se pokriti svaka baza unutar očitanja.

	A	G	G	C	G	C	T
K-mer 1	**1**	**0**	**3**	**3**			
K-mer 2		3	0	0	0		
K-mer 3			3	3	3	3	
Izvan Aktivnog prozora pretrage				0	0	0	1

Tablica. 2 Ekstrakcija (w=3,k=4) k-mera iz jednostavnog slijeda nukleotida, korak 1 - aktivni prozor pretrage: “AGGCGC”

	A	G	G	C	G	C	T
Izvan Aktivnog prozora pretrage	**1**	**0**	**3**	**3**			
K-mer 2		3	0	0	0		
K-mer 3			3	3	3	3	
K-mer 4				**0**	**0**	**0**	**1**

Tablica. 3 Ekstrakcija k-mera iz jednostavnog slijeda nukleotida, korak 2 - aktivni prozor pretrage:”GGCGCT” (pomak aktivnog prozora pretrage za jedan nukleotid u desno)

Mapiranje

Mapiranje se izvodilo na način da tražimo pojavljuje li se neki k-mer iz očitanja u referentnom genomu uspoređujući njihov “*ordering number*” (vidi tablica 1. radi pojašnjenja dodjele broja), ako se nađe u prvih k k-mera smatramo da je mapiranje uspješno i mapira se na poziciju referentnog k-mera - pozicija očitanog k-mera, ako smatramo da je mapiranje neuspješno, vraćamo nevaljanu poziciju.

Poravnanje

Za poravnanje smo koristili ograničeno lokalno poravnanje[1] s radijusom 2 kako bi ubrzali pronalazak poravnanja jer smo poravnavali uglavnom veoma slične k-mere čime smo uspjeli smanjiti prostor pretraživanja u matrici s dobitkom na brzini izvođenja bez gubitka preciznosti. Kao rješenje poravnanja uzimamo samo jedno poravnanje zbog čega je moguće smanjena preciznost pri traženju mutacija.

Pseudokod:

```
for (int i = 1; i < row; i++) {
                    for (int j =max(1,i-radius); j < column; j++) {

                                if ((abs(i - j) > radius)) {
                                        continue;
                                        }

                                auto index = i * (column)+j;

                                int match_award = -SUB+(kmer_seq[j - 1] == kmer_ref[i - 1])*(4+SUB);

                                operation[MATCH] = matrix[index - column - 1] + match_award;
                                operation[INSERT] = matrix[index - 1] - INSERTCOST;
                                operation[DELETE] = matrix[index - column] - DELETECOST;

                                for (int k = MATCH; k <= DELETE; k++) {

                                        if (operation[k] > matrix[index]) {
                                                matrix[index] = operation[k];
                                                operations[index] = k;
                                        }
                                }
          }
}
```

Na tablici ispod žutom bojom je prikazan prostor pretraživanja, dok su crvenom bojom označeni brojevi korišteni za poravnanje. Kao stringovi za uspoređivanje su korišteni “AATCCGTG” i “AATCCCTG”, a težina za podudaranje je 4, za zamjenu -2, a za umetanje ili brisanje -3.

Ovdje se vidi kako je umjesto pretraživanja 64 polja u matrici (jer pretraživanje počinje od 1, a ne od 0) , korišteno samo 34, tj. 53.125 %, za veći k će i postotak biti manji jer je radijus fiksan, čime se smanjuje ukupno vrijeme potrebno za provođenje algoritma. Što je k veći, iskorišteni prostor će biti relativno manji što dovodi do još veće uštede u odnosu na potpuno pretraživanje.

		A	A	T	C	C	G	T	G
	0	0	0	0	0	0	0	0	0
A	0	**4**	4	1	0	0	0	0	0
A	0	4	**8**	5	2	0	0	0	0
T	0	1	5	**12**	9	6	3	4	1
C	0	0	2	9	**16**	13	10	7	4
C	0	0	0	6	13	**20**	17	14	11
C	0	0	0	3	10	17	**18**	15	12
T	0	0	0	4	7	14	15	**22**	19
G	0	0	0	1	4	11	18	19	**26**

Tablica lokalnog poravnanja za “AATCCGTG” i ”AATCCCTG” uz match=4, indel=-3 sub=-2

Pronalazak mutacija

Prilikom traženja mutacija koristimo 2 poravnata k-mera i početnu poziciju te uspoređujemo znak po znak koja mutacija se dogodila. Prilikom usporedbi pamtimo broj umetanja znakova kako bi na pravilnu poziciju uspjeli staviti mutaciju, koja će na kraju biti početna pozicija + pozicija u znaku - broj umetanja.

Npr. za ulaz AAT_TGCT AATCTGCT i početnu poziciju 50 ćemo spremiti <M,A,50>, <M,A,51>, <M,T,52> , <I,C,52>, <M,T,53> itd.

Nakon što smo spremili sve zapise o mutacijama za svaku poziciju želimo izvući onu koja se najviše puta pojavila te ju ispišemo osim ako se najčešće baza podudarala s bazom referentnog očitanja(najčešća pojava -> MATCH).

Npr. ako je za poziciju 50 situacija sljedeća u datoteku će se zapisati umetanje T u poziciju 50.

Mutacija	Znak	Broj pojavljivanja
M	A	8
I	**T**	**11**
X	-	3

Mjerenja

Lambda

w	k	RAM [MB]	Trajanje [ms]	Jaccard
6	14	58.7	8900	0.50592885
7	14	51.4	**8124**	**0.52016129**
9	14	58.3	8927	0.51012145

Escherichia coli

w	k	RAM [GB]	Trajanje [s]	Jaccard
7	14	6.3	**1576**	**0.6786696**
8	14	6.3	1618	0.6696867
9	14	6.5	1632	0.6597586
7	16	7.7	1758	0.6565123

Usporedba rezultata

Lambda

Jaccard score referentne implementacije iznosi 0.453846153846 naši izmjereni rezultati imaju odstupanja od 10 do 16 %. Vremena izvođenja su relativno bliska zbog toga što lambda nema dugački genom i ima mali broj očitanja te zbog toga što nije bilo prevelike razlike u konačnom broju k-mera koji su izvučeni iz genoma i očitanja pa je i broj usporedbi bio sličan. Do razlika u rješenjima dolazi zbog toga što su izvučeni drugačiji k-meri pa je došlo i do različitih mapiranja što je dovelo do različitih broja mutacija, npr. za w 6 i k 14 je pronađeno brisanje T na poziciji 187, dok za w 7 i k 14 ta mutacija nije zabilježena u izlaznoj datoteci.

Escherichia coli

Jaccard score referentne implementacije iznosi 0.824952583615 što znači da su odstupanja oko 20 % za naše izmjerene rezultate. Vrijeme izvođenja u ovom slučaju dosta ovisi o broju i duljini k-mera koji se izvlače iz referentnog genoma i njegovih očitanja, za veći w je njihov broj porastao dok je za veći k k-mer dulji te će biti potrebno više memorije kako bi se oni spremili i dulje vremena kako bi se iz njih mogao napraviti ordering number i poravnanje. Zbog toga ne treba čuditi povećanje u vremenu od 11 % za isti w, no 14 % veći k.

Zanimljivo je primijetiti kako su najbolji rezultati ostvareni za w 7 i k 14 u oba primjera.

Zaključak

U okviru projekta je napravljena implementacija pronalaženja mutacija pomoću treće generacije sekvenciranje koristeći k-mer indekse. Implementacija je testirana na lambda i Escherichia coli genomima i njihovim simuliranim očitanjima.

Rezultati implementacije su dobiveni pokretanjem na bio-linuxu preko komandne linije, prevedena je pomoću g++ compilera koristeći optimizaciju O3 i library c++11. Sami rezultati su zadovoljavajući jer su glavni ciljevi, vrijeme izvođenja ispod 30 minuta na 1 dretvi i odstupanje manje od 50 % od referentne implementacije, ostvareni.

Implementacija je napravljena u c++-u bez korištenja ikakvih dodatnih biblioteka, tj. korištena je samo standardna biblioteka. Kao moguća poboljšanja je moguće umjesto standardne biblioteke koristiti biblioteke koje nude bolje performanse od standardne (npr. boost za stringove i odgovarajuće strukture podataka), promijeniti bodovanje kod poravnanja tako da se koriste supstitucijske matrice umjesto jednakog bodovanja za svaku supstituciju, pronaći bolji algoritam za mapiranje, tj. bolji uvjet zaustavljanja pretrage i provjeriti gubimo li točna umetanja jer se ona zapisuju na poziciji prije zbog čega može dogoditi da imamo više matcheva od umetanja na toj poziciji pa se to umetanje izgubi.

Literatura

[1] Algoritmi preklapanja - skripta iz bioinformatike, materijali za predavanja, Fakultet elektrotehnike i računarstva, Zagreb

[2] Minimizers - Reducing storage requirements for biological sequence comparison, Roberts, Hayes, Hunt, Mount, Yorke, - https://academic.oup.com/bioinformatics/article/20/18/3363/20214

[3] Third-generation sequencing, https://en.wikipedia.org/wiki/Third-generation_sequencing