Bioinformatika

Faza razmještaja u OLC paradigmi sastavljanja genoma implementirana u Canu asembleru

FER, 2016/2017

Ivan Soldo, 0036471227  
Tomislav Vulić, 0036460396

# Problem

Razvojem strojeva za očitavanje početkom 21. stoljeća u fokus su se vratili asembleri genoma koji kao ulazne podatke primaju relativno kratka očitanja (red veličine 20 kb). Problem kod rekonstruiranja genoma iz takvog skupa ulaznih podataka je taj što zbog relativno malih očitanja proces postaje vrlo osjetljiv na dugačka ponavljanja te šum očitanja. Međutim, ulazni skup očitanja je redundantan te je moguće usrednjavanjem i filtriranjem dobiti rezultate relativno visoke točnosti.

OLC paradigmu čine tri faze: preklapanje (overlap), razmještaj (layout) te konsenzus (consensus). U prvoj se fazi iz ulaznih podataka pronalaze očitanja sa najboljim preklapanjem (najdulje preklapanje sa najvišom razinom sličnosti po nekoj mjeri, npr. Levenshteinova udaljenost). U drugoj fazi, kojom se ovaj rad i bavi, uzimaju se podaci o preklapanju te se izgrađuje graf koji prikazuje moguće putanje između očitanja. Algoritam razmještaja filtrira očitanja koja su nepovoljna za izgradnju grafa, odabire najbolja preklapanja te razrjeđuje inicijalni graf susjedstva očitanja. Nakon razmještanja, OLC treba proglasiti konsenzusni slijed na osnovu višestrukog poravnanja sljedova dobivenih u svakom bloku dobivenom u fazi razmještanja.[[1]](#footnote-1)

# Ulazni podaci

Program za fazu razmještaja (*layout*) prima dvije datoteke. Prva datoteka je u zapis u FASTA formatu, a sadrži niz očitanja (*contig*). Svako očitanje u toj datoteci izgleda ovako:

>2 iid=2

TCAGTCAGGCTGGCGGCATCATTTCCCAAAATACGGTAATTTATTTTCG ...

Prva linija počinje znakom '>' i smatra se komentarom, te se ignorira. Druga linijia sadrži niz slova koja označavaju baze. Za testni primjer, FASTA zapis sastojao se od 162 očitanja (324 linije).

Druga datoteka je zapisana u MHAP formatu, te sadrži informacije o preklapanjima. Za ovaj rad, MHAP datoteka je generirana koristeći graphmap program[[2]](#footnote-2). Taj program uzima FASTA datoteku sa ulaza, te uspoređuje očitanja i nalazi preklapnja između njih. Taj postupak spada pod prvu fazu (preklapanje, *overlap*). Rezultat tog programa je datoteka u MHAP formatu koja za svako preklapanje sadrži slijedeće podatke:

Indeks prvog očitanja u preklapanju

Indeks drugog očitanja u preklapanju

Ocjena kvalitete preklapanja[[3]](#footnote-3)

Broj neprekinutih preklapanja (dijeljeni min-merovi)

Zastavica reverznog komplementa za prvo očitanje

Pozicija početka preklapanja u prvom očitanja

Pozicija završetka preklapanja u prvom očitanju

Duljina prvog očitanja

Zastavica reverznog komplementa za drugo očitanje

Pozicija početka preklapanja u prvom očitanja

Pozicija završetka preklapanja u prvom očitanju

Duljina prvog očitanja

Iz ovih se ulaznih podataka gradi graf preklapanja. Iako je u ovom radu obrađen algoritam koji prima ispravljena očitanja, neka su preklapanja svejedno nepovoljna i nepotrebna. Ovaj algoritam filtrira samo najjednostavnije viškove očitanja, tj. ne uključuje u izgradnju grafa ona preklapanja koja su obuhvaćena.

# Algoritam

Algoritam je implementiran po uzoru na bogart funkciju Canu asemblera, koji je temeljen na *greedy* **BOG** (*best overlap graph)* algoritmu (Miller et al. 2008.). Taj algoritam učitava samo najbolja, tj. najdulja preklapanja između očitanja. BOG ignorira obuhvaćena (*contained)* preklapanja, tj. ona preklapanja u kojem je jedno očitanje nadniz drugoga[[4]](#footnote-4).

Inicijalno filtriranje provodi se na slijedeći način:

*// n – broj zapisa o preklapanjima*

*// C(r) = informacije o preklapanjima sa nekim očitanjem r*

*// C\_f = informacije o filtriranim preklapanjima*

i **<=** 0

**dok** i<n **radi**

**ako je** C(i) obuhvaćen **onda**

**preskoči**

**inače**

C\_f **<=** C(i)

i **<=** i+1

**vrati** {C\_f}

Nakon što su izbačena nepovoljna očitanja, nastavlja se sa procjenom kvalitete preostalih preklapanja. Preklapanja za svako očitanje se soritraju silazno prema sličnosti preklapajućih podnizova. Sada imamo n\_f očitanja i njihova preklapanja. Preklapanja se dalje mogu razvrstati na lijevo, odnosno desno preklapanje s obzirom na promatrano očitanje. Ovime se stvara redundancija u smislu da i-ti čvor koji se preklapa sa j-tim čvorom ima u svojim preklapanjima preklapanje sa j-tim čvorom, te j-ti čvor ima u svojim preklapanjima preklapanje sa i-tim čvorom, samo na suprotne orijentacije.

Čvor je struktura koja se identificira rednim brojem očitanja iz ulaze .fasta datoteke, te sadrži i dvije strukture koje opisuju lijeva, odnosno desna preklapanja:

Čvor

Indeks očitanja

Popis lijevih preklapanja

Popis desnih preklapanja

Struktura preklapanja sadrži sve što se nalazi u zapisu u .mhap formatu

Preklapanje

Sličnost // [0.000000,1.000000]

Broj zajednlčkih min-mera

Indeksi dvaju preklapajućih očitanja

Koordinate preklapanja prvog očitanja

Koordinate preklapanja drugog očitanja

Duljina prvog očitanja

Duljina drugog očitanja

U ovom je radu implementiran vrlo naivan način konstrukcije BOG-a. Kao početni čvor uzima se onaj koji nema lijeva preklapanja, tj. onaj koji ima globalno najmanju sličnost sa svojim lijevim preklapanjem. Ti se čvorovi stavljaju u listu starters te se iz svakog od tih čvorova pokušava napraviti BOG. Potpuno rješenje bilo bi iz svakog od početnih čvorova konstruirati sve moguće putanje, te naći onu putanju koja ostvaruje najbolji rezultat po nekoj metrici. U ovome radu je implementiran jednostavan DFS (*depth first search*) algoritam koji kreće iz svakog od početnih čvorova, te traži put do kraja grafa, tj. do očitanja bey desnog preklapanja. Za svaki se čvor izračuna maksimalan rezultat koji se iz njega može postići (uz proizvoljnu metriku, o čemu će riječi biti u poglavlju Metrika). Na ovaj se način za svaki čvor putanja izračunava točno jedanput. Krajnji desni čvorovi daju najbolju (ili neutralnu) vrijednost metrike. Vremenska složenost iznosi O(n), gdje je n broj čvorova. Pseudo kod algoritma za pronalazak najbolje putanje:

1 **dfs** (čvor)

2 **lista** lokalna\_rješenja = {}

3 **za svaki** čvor.dijete **radi**

4 **rješenje** trenutno\_rješenje

5 **ako je** čvor.dijete.rješenje = nađeno

6 trenutno\_rješenje <= čvor.dijete.rješenje

7 **inače ako je** čvor.dijete.rješenje = {}

8 čvor.rješenje <= u\_ciklusu

9 trenutno\_rješenje <= dfs(čvor.dijete)

10 **inače**

11 // čvor je posjećen, ali rješenje nije nađeno

12 // ciklus

13 **nastavi**

14 **ako je** trenutno\_rješenje != {}

15 lokalna\_rješenja <= trenutno\_rješenje

16 čvor.rješenje <= nađeno

17 **vrati** lokalna\_rješenja

# Metrika

Metrika određuje što se proglašava najboljim rezultatom. Vrlo logična metrika bila bi ukupni rezultat sličnosti među preklapanja na putanji (npr. najveći umnožak koeficijenta sličnosti preklapnja), međutim, takva metrika može najboljom putanjom proglasiti onu putanju koja ima vrlo malen broj preklapanja visoke sličnosti. U ovom je radu implementirana metrika koja preferira dulje rezultate. Gleda se kumulativni zbroj ocejna sličnosti, te se boljom putanjom smatra ona koja ima veći zbroj svih sličnosti. To je, dakako, daleko od idealnog, jer se vrlo lako najboljim putem proglasi onaj koji prolazi kroz najviše čvorova, iako sličnost kod preklapanja tih čvorova može bit vrlo loša. Puno bi bolje bilo kada bi metrika cijenila i duljinu i sličnosti (npr. kvadrat broja očitanja u putanji pomnožen sa kumulativnim umnoškom sličnosti preklapanja).

# Rezultati

Izlaz ovog programa je datoteka u FASTA formatu koja sadrži procijenjeni zapis kompletnog genoma na temelju ulaznih podataka o preklapanju. Nažalost, ovakav algoritam, sa ovakvom metrikom, nije dao zadovoljavajuće rezultate. Za oba primjera, izlaz iz programa sadrži jedan red veličine više baza nego referentni genom. Međutim, po pitanju memorije i vremenskog ograničenja, program radi vrlo dobro (možda zato što ne radi ono što bi trebao raditi). Koristeći program cgmemtime[[5]](#footnote-5) izmjereno je vrijeme izvođenja 6.747 sekundi te zauzeće memorije od 410924 kiB.

# Zaključak

Izlazne datoteke ovoga programa nisu dale željene reyultate u usporedbi sa referetnim genomima. Za to je odgovorna korištena metrika, određivanje samo jednog, i to ukupnog najboljeg puta, te premala količina vremena posvećena ovome projektu od strane studenata.

Ovaj rad ima postavljenu zadovoljavajuću kodnu infrastrukturu za daljnji rad i razvoj bogart algoritma.

Mjesta za napredak ima u promjeni metrike, te u pokušaju konstruiranja manjih komada putanje koji bi imali visoke rezultate po kvalitetnoj metrici. Tako bi se dobila kontizi sa visokom vjerojatnošću točnosti koji bi se mogli spajati skafoldima.

# Literatura

1. Bioinformatika; M. Domazet-Lošo, M. Šikić, Zagreb, 2013
2. Canu: scalable and accurate long-read assemblz via adaptive k-mer weighting and repeat separation; S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, A. M. Phillipy
3. Aggresive assembly of pyrosequencing reads with mates; J. R. Miller, A. L. Delcher, S. Koren, E. Venter, B. P. Waleny, A. Bronwley, J. Johnson, K. Li, C. Mobarry, G. Sutton
4. De novo assembly using long error-prone reads; M. Kostelac, Zagreb, 2016
5. Simplification of the Overlap Graph; B. Rahle, Zagreb, 2014

1. M. Šikić, M. Domazet-Lošo: Bioinformatika, 2013. [↑](#footnote-ref-1)
2. github.com/isovic/graphmap [↑](#footnote-ref-2)
3. Obično se koristi *Jaccard score* [↑](#footnote-ref-3)
4. „Aggressive assembly of pyrosequencing reads with mates“, Miller et al. 2008. [↑](#footnote-ref-4)
5. github.com/isovic/cgmemtime [↑](#footnote-ref-5)