SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

Faza razmještaja u OLC paradigmi sastavljanja genoma

Domagoj Boroš, Ivan Jurin, Mateja Škriljak

Voditelj: *prof.dr.sc. Mile Šikić*

Zagreb, siječanj, 2017.

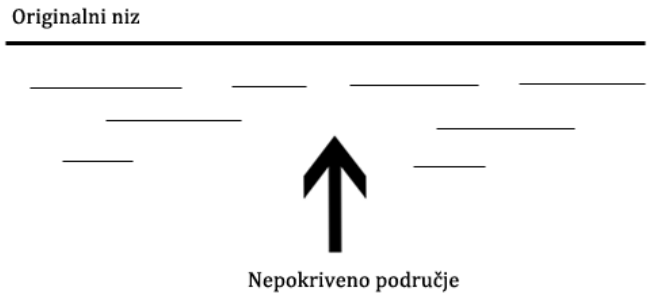
Sadržaj

1. Uvod 1
2. OLC paradigma sastavljanja genoma 2
3. *Miniasm* algoritam 3
   3.1 Čišćenje očitanja 3
   3.2 Sastavljanje grafa 3
   3.3 Čišćenje grafa 4
      3.3.1 Uklanjanje tranzitivnih bridova 4
      3.3.2 Uklanjanje šiljaka 4
      3.3.3 Uklanjanje mjehurića 5
      3.3.4 Sastavljanje unitiga 6
4. Rezultati i zaključak 7
5. Literatura 9
6. Sažetak 10

1. Uvod

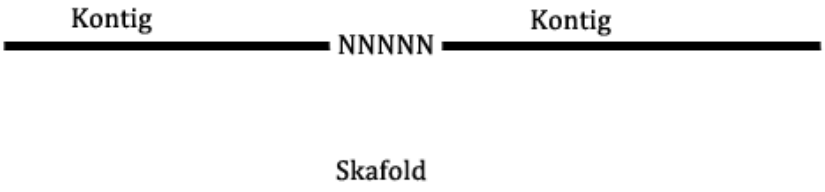
Danas postoje razne metode sekvenciranja genoma koje nude rješenja prihvatljive kvalitete. Problem je iznimno složen te različita rješenja nude različite duljine očitanja i varijabilnu točnost. Susrećemo se i s problemom različite razine pokrivenosti očitanja, pri čemu se javlja problem razlomljenosti slijeda u više dijelova s procijepima između te sastavljanjem ne možemo dobiti jedinstveno potpuno rješenje.

Poseban problem je *de novo* sastavljanje, kada se sastavlja genom koji nam nije unaprijed poznat te nemamo referentni genom za provjeru očitanja, čime je čitav postupak dodatno otežan.



Slika 1. Pojedini dijelovi skafolda nisu pokriveni [2]

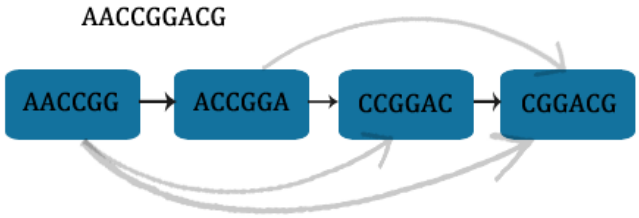
Rezultantni sastavljeni genom nakon većine postupaka nije potpun kontinuirani slijed, stoga se predstavlja hijerarhijskom strukturom koja opisuje način mapiranja očitanih nizova u rekonstruirani ciljni genom, pri čemu je mapiranje zapravo dobiveno većim dijelom heuristički. U takvoj strukturi osnovna jedinica su očitanja, koja se zatim grupiraju u kontige, koji se grupiraju u skafolde. Kontig je pritom niz uzastopnih očitanja koja se preklapaju, a skafold skupina kontiga, pri čemu se definira njihov poredak, orijentacija i veličine procijepa između njih. Veličina i točnost kontiga i skafolda definira kvalitetu sastavljenog genoma. [2]



Slika 2. Skafold koji se sastoji od dva kontiga između kojih je procijep od 5 znakova [2]

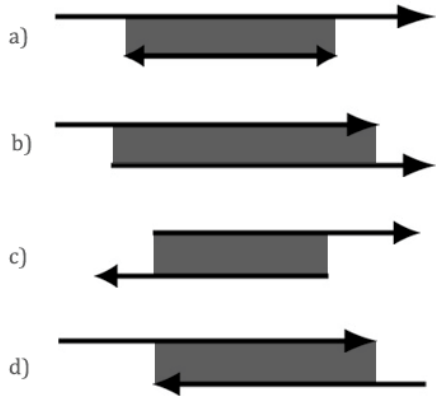
2. OLC paradigma sastavljanja genoma

Dvije glavne metode sastavljanja genoma temeljene na algoritmima nad grafovima su OLC (engl. *Overlap-Layout-Consensus*), odnosno Preklapanje-Razmještaj-Konsenzus metode te metode nad *de Bruijn* grafovima.



Slika 3. Graf preklapanja s vrhovima za svako očitanje. Bridovi predstavljaju preklapanja. Sivi bridovi su tranzitivni, ta preklapanja su sadržana u drugima. [2]

OLC pristup pronalazi preklapanja očitanja te gradi graf preklapanja. Iz grafa se zaključuje o obliku ciljnih struktura. Takav graf se sastoji od vrhova koji odgovaraju očitanjima i bridova koji povezuju očitanja ukoliko između njih postoji preklapanje. Putovi u grafu su tada potencijalni kontizi te svaki ima svoju „zrcalnu sliku", odnosno niz koji mu je reverzni komplement. Slika 4. prikazuje vrste preklapanja nizova. Preklapanje može ujedno biti i obuhvaćanje, ako je čitav *B* sadržan u *A*. Kod drugih preklapanja prefiks ili sufiks jednog niza je poravnat s prefiksom ili sufiksom drugoga.



Slika 4. a) Obuhvaćanje, b) Regularni lastin rep, c) Prefiksni lastin rep, d) Sufiksni lastin rep [2]

Nakon izgradnje grafa slijedi njegovo pojednostavljenje. Cilj druge faze, a ujedno i ovog rada, jest izgraditi graf te mu smanjiti broj bridova i vrhova bez smanjenja prostora potencijalnih rješenja. Uklanjaju se tranzitivni bridovi te „mjehurići" i vršna očitanja.

Za generiranje preklapanja korišten je alat *minimap*. [1] Algoritam *miniasm* radi s neispravljenim očitanjima, te su i rezultantni kontizi neispravljeni.

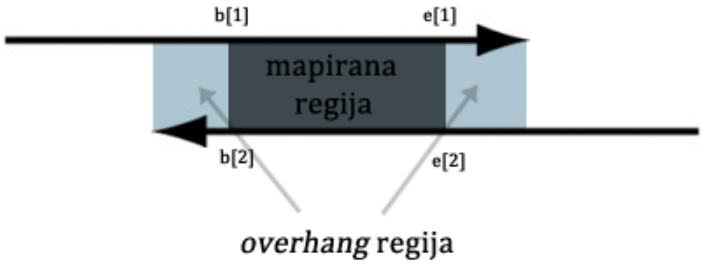
3. *Miniasm* algoritam

Promatraju se preklapanja nizova (očitanja) v i w. Ako se v može preslikati na podniz unutar w, w sadrži v, a ako se sufiks od v i prefiks od w preslikavaju, govorimo o preklapanju $v \rightarrow w$ koje ujedno i definira brid između tih očitanja. Duljina preklapanja je tada duljina prefiksa v koji se ne preklapa s w. Graf preklapanja je usmjereni graf $G = (V, E, l)$ gdje je V skup očitanja, E je skup preklapanja među njima, a l je funkcija koja svakom bridu pridružuje njegovu duljinu. Ulazni stupanj čvora je uznačen s $deg^{+}(v)$, a izlazni s $deg^{-}(v)$.

3.1 Čišćenje očitanja

Prvi korak je reduciranje greški u očitanjima. Izravnavaju (engl. *trimming*) se očitanja pronalaženjem najdužeg slijeda pokrivenog s tri ili više preklapanja zadane duljine te se odbacuju baze u očitanjima koje su tada izvan pronađenog slijeda. Također se filtriraju kimerna preklapanja.

3.2 Sastavljanje grafa



Slika 5. Svijetlo područje označava regiju koja bi bila mapirana da je preklapanje savršeno [1]

Gradi se graf preklapanja, pri čemu se preklapanja klasificiraju prema prikazanom algoritmu. Ulazni parametri su: duljine očitanja *l*, početni indeks preklapanja *b*, završni indeks *e*, maksimalna duljina *overhang* područja *o* te maksimalan omjer duljine *overhang* područja i duljine preklapanja *r*.

```
KlasificirajPreklapanje(l, b, e, o, r):

overhang ← min(b[1], b[2]) + min(l[1] - e[1], l[2] - e[2])

maplen ← max(e[1] - b[1], e[2] - b[2])

ako overhang > min(o, maplen * r):

   vrati UNUTARNJE_PREKLAPANJE

ako b[1] ≤ b[2] i l[1] - e[1] ≤ l[2] - e[2]:
```

```
    vrati PRVI_SADRŽAN
ako b[1] ≥ b[2] i l[1] - e[1] ≥ l[2] - e[2]:
    vrati DRUGI_SADRŽAN
ako b[1] > b[2]
    vrati PREKLAPANJE_PRVI_PREMA_DRUGOM
    vrati PREKLAPANJE_DRUGI_PREMA_PRVOM
```

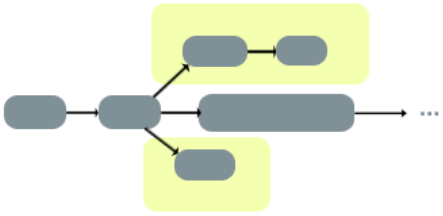
Unutarnja preklapanja (definirana ovisno o duljini vanjskog (engl. *overhang*) područja) i očitanja sadržana u drugima se ignoriraju, a ostala očitanja se dodaju u graf. [1]

3.3 Čišćenje grafa

3.3.1 Uklanjanje tranzitivnih bridova

Brid $v \to w$ je tranzitivan ako postoji $v \to u$ i $u \to w$. Uklanjanje takvih bridova ne utječe na povezanost očitanja u grafu. Slika 1. prikazuje primjer takvih bridova. Tranzitivni bridovi su uklonjeni prema [3]

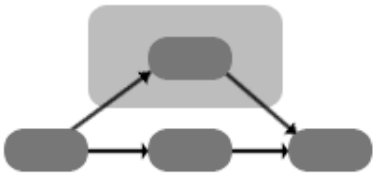
3.3.2 Uklanjanje šiljaka



Slika 6. Žuto su označeni potencijalni šiljci u grafu.

Čvor v je šiljak ako $deg^+(v) = 0$ i $deg^-(v) > 0$. Većina šiljaka je uzrokovana pogrešnim očitanjem ili nedetektiranim preklapanjima [1].

3.3.3 Uklanjanje mjehurića



Slika 7. Mjehurić.

Mjehurić u grafu je aciklički podgraf s jednim izvorom v i jednim ponorom w, pri čemu postoje barem dva puta između v i w. Algoritam pronalaženja mjehurića iz [1] započinje pretragu iz potencijalnog izvora i posjećuje čvor kad posjeti sve njegove ulazne bridove.

Ulazni parametri su: graf $G = (V, E)$, početni čvor v_0 i maksimalna dubina istraživanja *d.* Algoritam vraća ponor mjehurića ako je pronađen, ili *nil* ako nije. Uklanjanje je implementirano prema [4].

```
PronađiMjehurić(V, E, v₀, d):
ako deg⁺(v₀) < 2: vrati nil
za v ∈ V: δ[v] ← ∞                    ■ minimalna udaljenost od v₀ do v
δ[v₀] ← 0
S ← prazan stog                       ■ čvorovi čiji su ulazni bridovi posjećeni
PUSH(S, v₀)
p ← 0                                 ■ broj posjećenih čvorova izvan S
dok S nije prazan:
   v ← POP(S)
   za svaki v → w ∈ E:
      ako w = v₀: vrati nil           ■ ciklus
      ako δ[v] + l(v → w) > d: vrati nil
      ako δ[w] = ∞:                   ■ još neistraženo
         γ[w] ← deg⁻(w)
```

```
            p ← p + 1
        ako δ[v] + l(v → w) < δ[w]:
            δ[w] ← δ[v] + l(v → w)
        γ[w] ← γ[w] − 1
        ako γ[w] =0:
            ako deg⁺(w) ≠ 0:
                PUSH(S, w)
            p ← p − 1
    ako |S| = 1 i p = 0:
        vrati POP(S)
vrati nil
```

3.3.4 Sastavljanje unitiga

Neka je $v_1 \to v_2 \to \cdots \to v_k$ put u grafu, duljine *k.* Spajanjem podnizova u čvorovima dobivamo $v_1[1, l(v_1 \to v_2)] \circ v_2[1, l(v_2 \to v_3)] \circ \ldots \circ v_{k-1}[1, l(v_{k-1} \to v_k)] \circ v_k$, gdje je $v[i, j]$ podniz određen indeksima i i j, uključivo, a $\circ$ je operator konkatenacije nizova. U grafu s uklonjenim tranzitivnim bridovima, unitig je put $v_1 \to v_2 \to \cdots \to v_k$ gdje $deg^+(v_i) = deg^-(v_{i+1}) = 1$ te $v_1 = v_k$ ili $deg^-(v_1) \neq 1$ i $deg^-(v_k) \neq 1$. Unitig je zapravo maksimalni put na kojem se susjedna očitanja mogu združiti tako da se ne utječe na povezanost u originalnom grafu. [1]

4. Rezultati i zaključak

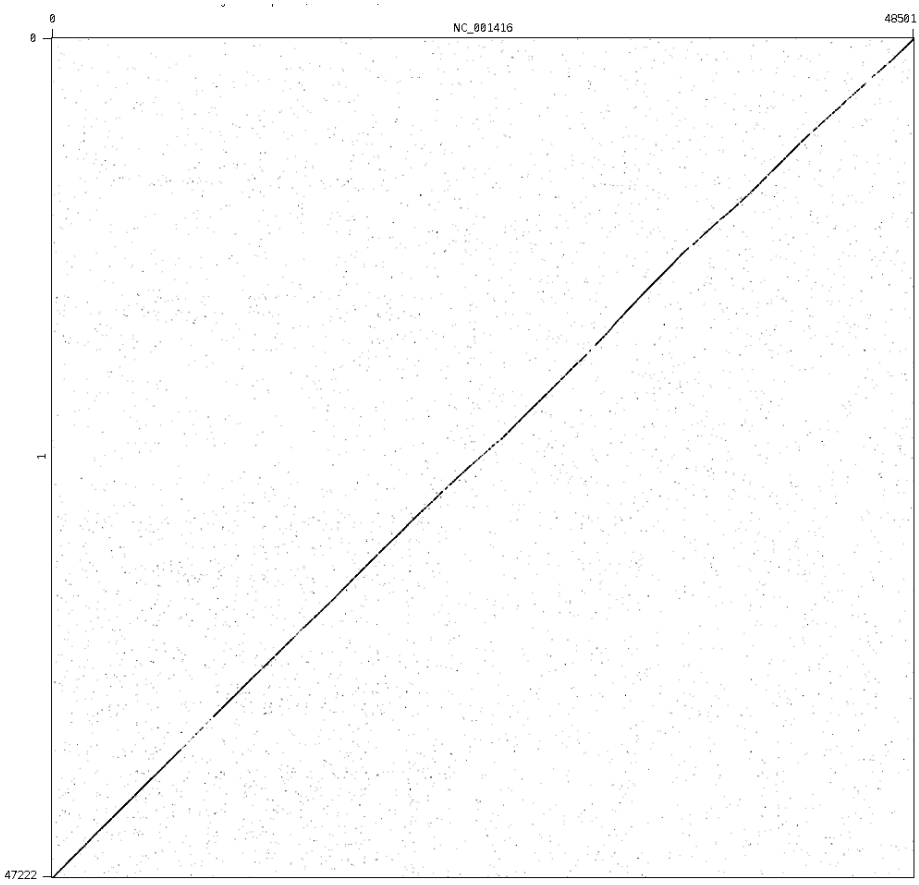
Testni podaci	Vlastita implementacija	Originalna implementacija
Lambda	0.02 s	0.01 s
E. coli	6.69 s	3.37 s

Tablica 1. Usporedba vremena potrebnog za izvršavanje naše implementacije i originalne na dva skupa testnih podataka.

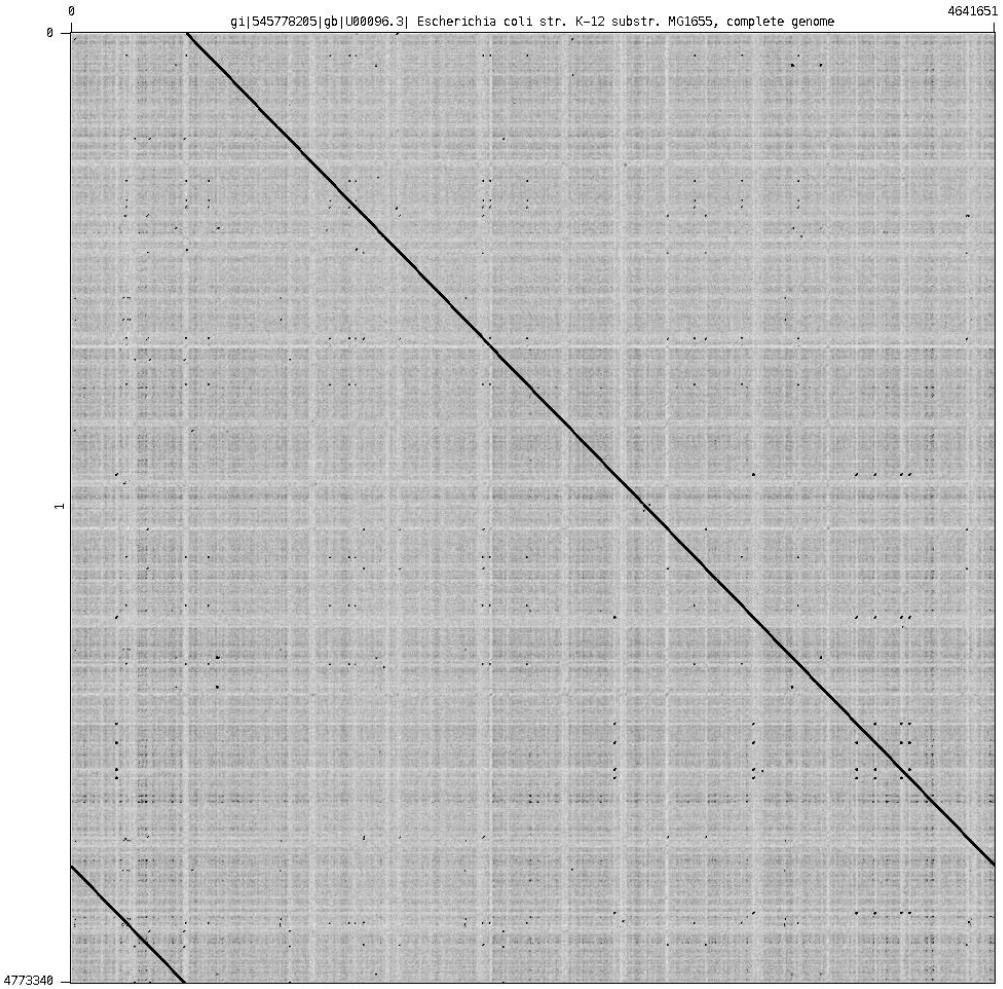
Testni podaci	Vlastita implementacija	Originalna implementacija
Lambda	1988 kB	1976 kB
E. coli	189,816 kB	94,588 kB

Tablica 2. Usporedba memorije potrebne za izvršavanje naše implementacije i originalne na dva skupa testnih podataka.

Originalna implementacija algoritma je efikasnija, što je posebno vidljivo na većem skupu testnih podataka.



Slika 8. Rezultantne kontige na *Lambda* skupu.



Slika 8. Rezultantne kontige na *E. coli* skupu.

Miniasm algoritam ne ispravlja pogrešna očitanja. U najboljem slučaju, pogreška u sastavljanju nizova koji predstavljaju pojedinačne unitige je jednaka pogrešci u samom isčitavanju početnih sekvenci. Teorijski zaključci pokazuju [1] potencijalna poboljšanja u sastavljanju unitiga iskorištavanjem znanja o preklapanjima u očitanjima. No, takav alat nije definiran u sklopu ove metode.

5. Literatura

[1] Li, H. Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences, Bioinformatics (2016) 32 (14): 2103-2110.

[2] Šikić, M., Domazet-Lošo M., Bioinformatika (2013), skripta s predmeta Bioinformatika, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu

[3] Myers, E. W. The fragment assembly string graph. Bioinformatics (2005), 21 Suppl 2:ii79–85.

[4] Zerbino, D. R., Birney, E. Velvet: algorithms for de novo short read assembly using de Bruijn graphs (2008). Genome Res, 18:821–9.

6. Sažetak

Metode sastavljanja genoma koje rade nad dugačkim očitanjima se obično sastoje od četiri stadija: pronalazak preslikavanja uspoređujući sve nizove sa svima ostalima, ispravljanje očitanja, sastavljanje ispravljenih očitanja i konsenzus oko sastavljanja kontiga. U ovom radu razmatraju se algoritmi koji rade bez koraka ispravljanja. Preklapanja su generirana iz neispravljenih očitanja alatom *minimap* nakon čega je implementiran algoritam *miniasm* kao korak razmještaja u Preklapanje-Razmještaj-Konsenzus paradigmi.

Implementirani algoritam pokazuje potencijal sastavljanja kontiga iz neispravljenih očitanja. *Miniasm* je brz i efikasan te daje dobre rezultate.