SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT - BIOINFORMATIKA

Layout faza OLC sastavljanja genoma

Matea Pejčinović, Fran Stanić

Voditelj: prof.dr.sc. *Mile Šikić*

Zagreb, siječanj, 2016.

Sadržaj

1. Uvod

Proteklo stoljeće obilježeno je jako velikim razvojem područja genetike. Mnoga otkrića i znanstvene uspjehe paralelno je pratio razvoj posve nove računalne discipline, bioinformatike. U današnje vrijeme u mnogim je znanstvenim institutima u tijeku dešifriranje i sekvenciranje milijardi parova genoma. Cilj ovih istraživanja jest stvoriti baze podataka kako bi rezultati rada znanstvenika bili dostupni svim zainteresiranim istraživačima širom svijeta.

Upravo ovakve baze omogućuju implementaciju raznih algoritama i računalnih programa koji se bave sekvenciranjem i sastavljanjem gena. Budući da je tema ovog rada sastavljanje genoma, potrebno je prikazati određeni uvod u problematiku.

Najčešći pristup u sekvenciranju je tzv. *shotgun* sekvenciranje koje, kao što sam naziv kaže, „*siječe*“ genom, i to ne jednom, već mnogo puta na slučajan način. U tu svrhu potrebno je prethodno načiniti kopije genoma kako bi se imalo što više segmenata u trenutku kad započne proces sastavljanja genoma. Taj proces je izuzetno zahtijevan jer je potrebno jako velik skup segmenata DNK spojiti u što kraći i kvalitetniji niz DNK koji će ih sve sadržavati.

Sastavljanje genoma ostvaruje se na dva načina:

1. Mapiranjem na postojeći referentni genom,
2. *De novo* sastavljanjem.

Za prvi se način obično koristi referentni genom dobiven Sangerovom metodom od više donora. Ovaj način je za sad dominantna tehnika u odnosu na Maxim-Gilbertovu tehniku i nešto noviju metodu tzv. pirosekvenciranja. Poznata je i kao dideoksi metoda ili metoda terminacije sinteze lanaca. Temelji se na činjenici da se jednolančane DNK koje se razlikuju za samo jedan nukleotid mogu razdvojiti jedan od drugog postupkom elektroforeze korištenjem akriamidnog gela s poprilično velikom rezolucijom. Također je bitno naglasiti da ovaj postupak uzima u obzir komplementarnost baza te da postoje mnogi računalni programi koji provode dijelove ovog postupka s velikom točnošću.

De novo sastavljanje je postupak rekonstrukcije genoma bez pomoći referentnog (originalnog) genoma. Jedan od načina provođenja ovog načina sastavljanja jest OLC postupak (*Overlap – Layout - Consensus*). Kao što se može i pretpostaviti, ovaj pristup je znatno složeniji od prethodnog i spada u skup NP-teških (eng. *NP-hard*) problema. Problemi na koje možemo naići prilikom provođenja *de novo* sastavljanja genoma su postojanje ponavljajućih regija unutar DNK niza koji se očitava što kod asemblera može dovesti do toga da se predvidi postojanje višestrukog položaja takvih regija, ali i određivanje optimalnog koeficijenta pokrivenosti prilikom čitanja genoma kako bismo smanjili broj slijednih sekvenci.

Ovaj postupak se može izvesti pohlepnim pristupom ili asemblerima temeljenim na grafovima.

U ovom radu će naglasak biti na *de novo* sastavljanju genoma pomoću algoritma zasnovanog na grafu – OLC pristupu (eng. *Overlap – Layout – Consensus*). Zapravo, tema ovog projektnog zadatka je implementacija faze razmještaja (eng. *layout*) očitanja na temelju poravnanja za što se kao ulazni podaci koriste rezultati prethodne faze.

2. OLC pristup

OLC pristup (*Overlap – Layout – Consensus*) je jedna od najpoznatijih metoda za *de novo* sastavljanje genoma. Sastoji se od tri faze:

1. preklapanja (eng. *overlap*),
2. razmještaja (eng. *layout*) i
3. konsenzusa (eng. *consensus*).

Cilj je pronaći preklapanja poravnanjem sekvenci očitanja, razmjestiti očitanja na temelju poravnanja te postići konsenzus spajanjem svih sekvenci očitanja ujedinjavanjem preklapanja. U fazi preklapanja se izračunava i gradi osnovni graf dok razmještaj vrši kompresiju grafa na temelju koje se tijekom faze konsenzusa određuje sekvenca genoma.

Kao što je već i navedeno,OLC pristup pripada grupi algoritama temeljenih na grafovima. Ova skupina predstavlja preklapanja kao usmjerene netežinske grafove.

Graf je kod ovog pristupa definiran na sljedeći način:

- očitanja su čvorovi,
- ako se sufiks od čvora A značajno preklapa s prefiksom čvora B, onda se može reći da postoji brid od čvora A prema čvoru B,
- želimo pronaći Hamiltonov put u grafu – put koji posjećuje svaki čvor samo jednom.

2.1 Faza preklapanja

U ovoj fazi se sekvenca svakog očitanja uspoređuje sa svim drugim očitanjima u oba smjera. No, ukoliko se želi primijeniti ovaj postupak u implementaciji, suočit ćemo se s kvadratnom složenošću što nikako nije opcija uzevši u obzir činjenicu da očitanja mogu biti prilično duga.Umjesto ovog pristupa, nerijetko se koristi BLAST i slični

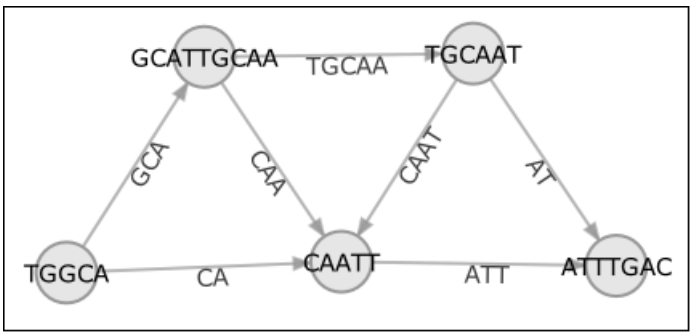
algoritmi koji smanjuju složenost. Čak je i s njima nužno rabiti paralelno programiranje i višestruke procesore zbog dužine očitanja. Neka od uobičajenih pojednostavljenja odnose se na zahtjeve da očitanja dijele kraći niz baza fiksne duljine unutar strukture. Ona očitanja koje nemaju niti jedan isti takav niz, ne podliježu usporedbi.

U radu [1] se za preklapanje između dva očitanja definira zajednički dio baza pomoću dva intervala: [*o.f.beg*, *o.f.end*] i [*o.g.beg*, *o.g.end*]. Pod bazama se misli na slijed slova koja čine DNK, a njihovi položaji se indeksiraju od 0. Također je jako bitno primijetiti da se krajnja točka intervala smatra ekstremnom ako iznosi 0 ili je zapravo jednaka duljini relevantnog očitanja.

Svako preklapanje ima barem dvije ekstremne vrijednosti završnih točaka i vrijedi

$|o.f.end - o.f.beg| \approx |o.g.end - o.g.beg|$.

Različiti OLC algoritmi definiraju različite kriterije za preklapanja. Primjerice, *Celera Assembler* prepoznaje preklapanje i dodaje ga kao brid u graf ako se barem 40 nukleotida s najmanje 94%-tnom sličnošću preklapa.



Slika 1. Jednostavni graf nakon faze preklapanja

Slika prikazuje graf u kojem su čvorovi očitanja, a bridovi povezuju ona očitanja koja se preklapaju. U stvarnom su OLC grafu očitanja i preklapanja zapravo mnogo veća, u ovom prikazu su skraćeni radi jasne predodžbe o pristupu u sastavljanju genoma.

2.2 Faza razmještaja

Ovaj postupak nije nikako jednostavan jer, kao i *de novo* postupak, jedan je od poznatih NP-teških problema. Stoga se za pronalazak Hamiltonovog ciklusa u grafu

koristi se postupni pristup izravnavanja grafa dobivenog u prethodnoj fazi. Želimo izgraditi put kroz graf kroz svaki čvor što nije trivijalan zadatak. Radi smanjenja grafa, u ovoj fazi se provodi niz postupaka koji trebaju dovesti do stvaranja kontiga, djelomično sastavljenog niza od nekoliko očitanja. U grafu bi to bio podgraf ili skup međusobno višestruko povezanih vrhova. Jednom kad uspijemo identificirati ovaj podgraf, vrhovi i bridovi se sažimaju u jedan čvor ili kontig. Na taj način dolazimo do slijedne sekvence jer svaki čvor ima jednog susjeda u jednom smjeru.Rekurzivno se za sve čvorove traži sekvenca koja će završit čim naiđemo na čvor koji ima više od jednog susjeda.

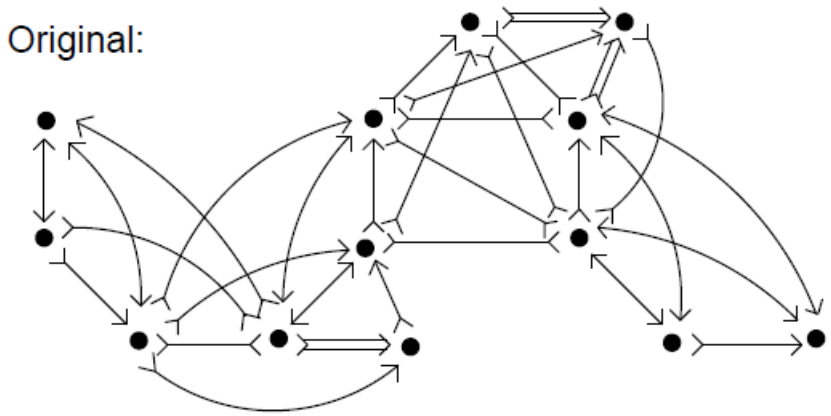
Na ovaj način se dođe do većeg broja slijednih sekvenci koje spajanjem postaju skafoldi. Kako bi se mogao nastaviti postupak sastavljanja genoma, potrebno je doći do više informacija na različite načine, a jedan od njih je definitivno grupiranje očitanja dobivenih od istog isječka genoma iz faze preklapanja u parove čime se može odrediti međusobni relativni položaj slijednih sekvenci.

Budući da je ova faza dosta vremenski složena, koristi se niz pojednostavljenja i heuristika koje omogućavaju dobivanje dovoljno dobrih rješenja sa smanjenjem trajanja izračuna. U nastavku je pomnije objašnjen pristup kojeg smo koristili u izradi projekta.

2.2.1 Implementacija

Najprije je bilo potrebno učitati podatke koji će sadržavati informaciju o preklapanjima između pojedinih sekvenci. Takve informacije su dobivene iz datoteke u MHAP formatu koji je pomnije objašnjen u zasebnom poglavlju. Radi izgradnje grafa bilo je potrebno provesti nekoliko koraka unutar faze razmještaja.

Koristeći dolje prikazani graf, bit će objašnjen niz koraka faze razmještaja.



Slika 2. Originalni graf na kojem će se vizualno moći uočiti pojedini korak faze razmještaja

2.2.1.1 Uklanjanje sadržanih nizova baza (eng. containment)

Cilj je izgraditi graf u kojem će sva očitanja i spojevi parova očitanja koji se preklapaju biti sadržani u njemu. Očito je da će se dogoditi određena ponavljanja, tj. bit će potrebno ukloniti određena očitanja jer će već postojati putanja u grafu kojom ćemo doći do njih. Ovo pojednostavljenje uvelike pridonosi smanjenju zauzeća memorije i vremenskoj složenosti čitavog postupka.

U implementaciji u okviru projekta koristi se ovo pojednostavljenje kako bismo mogli nastaviti s provođenjem faze razmještaja. Podaci koji su dobiveni u datoteci nisu onakvi kakvi bi trebali biti da se možemo referirati na [1]. Naime, budući da svako preklapanje mora imati barem dva ekstrema, najprije smo morali načiniti određena pojednostavljenja u algoritmu.

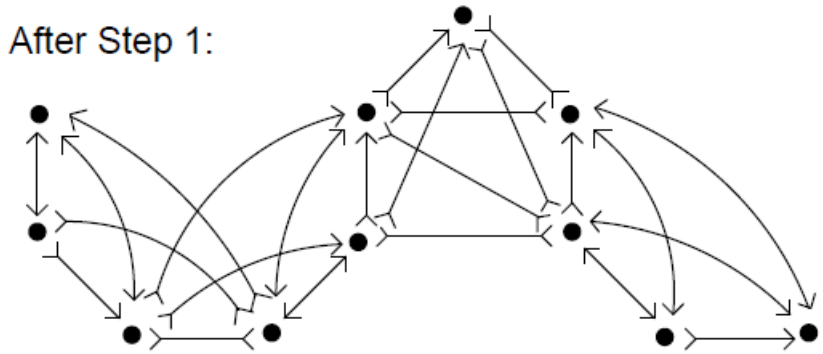
Preklapanje se smatra „*containmentom*" ako su oba kraja očitanja ekstremi, tj. kraj intervala je ili 0 ili jednak duljini relevantnog očitanja. Kako bismo odredili ekstreme, bilo je potrebno naći dijelove u očitanjima koji nisu dio preklapanja. Gledali smo dijelove s desne i lijeve strane i našli za početak manji od njih kako bi se odredio ekstrem koji će biti 0. Onaj koji će predstavljati drugi ekstrem će biti veći od dva. Pri tome je bitno bilo gledati orijentaciju; ako su oba usmjerena u istom smjeru, tad uspoređujemo početke očitanja i krajeve, inače se uspoređuju početak jednog očitanja i kraj drugog i obrnuto.

Prethodni korak omogućava izgradnju dijelova koji predstavljaju preklapanja za koja potom trebamo odrediti jesu li „*containmenti*” ili ne. Kako je prethodno napisano, u slučaju da jesu već sadržani, moraju zadovoljiti određene uvjete. U našoj implementaciji se radilo o sljedećoj provjeri:

```
length()>=read.length()*(1-percentMargin/100)
```

Dakle, *percentMargin* je korišten kao parametar kojeg smo mijenjali kako bismo vidjeli koliko utječe na dobivene rezultate. Gornji izraz zapravo evaluacija duljine preklapanja gdje se provjerava je li ona dulja ili jednaka duljini očitanja pomnoženoj s izrazom na koji utječe *percentMargin*. Vrijednost *absoluteMargin* je postavljen na 20 i za njega su dobiveni poprilično dobri rezultati.

Utrošak memorije smo smanjili time što očitanja ne učitavamo na početku, nego poslije izbacivanja svih već sadržanih baza.



Slika 3. Graf nakon uklanjanja sadržanih bridova

Uspoređujući originalni graf i gore prikazani, vidljivo je da već u ovoj fazi određen broj bridova biva eliminiran. Uglavnom se radi o onim bridovima koji predstavljaju put već sadržan u grafu. Prilikom uklanjanja tih bridova,eliminiraju se i vrhovi iz kojih ne izlaze bridovi niti ima ulaznih bridova u njih.

2.2.1.2 Uklanjanje tranzitivnih bridova

Kad dođemo do ovog koraka, u grafu još uvijek ima više bridova nego što nam je zapravo potrebno. Sljedeći korak se sastoji od uklanjanja tranzitivnih bridova što

smanjuje broj bridova u grafu za neki faktor c. Time ćemo dobiti graf koji će biti puno bolji u smislu relevantnosti informacija koje dobivamo iz saznanja da je pojedini brid unutra, no i sada se može pojaviti određen broj lanaca koji nemaju nikakvu mogućnost grananja.

```
constant FUZZ ← 10

for v ∈ V do
 { mark[v] ← vacant
   for v → w ∈ E do
       reduce[v → w] ← false
 }

for v ∈ V do
 { for v → w ∈ E do
       mark[w] ← inplay
   longest ← max_w len(v → w) + FUZZ
   for v → w ∈ E in order of length do
       if mark[w] = inplay then
           for w → x ∈ E in order of length and
                   len(w → x) + len(v → w) ≤ longest do
               if mark[x] = inplay then
                   mark[x] ← eliminated

   for v → w ∈ E in order of length do
       for w → x ∈ E in order of length and
             (len(w → x) < FUZZ or
               w → x is the smallest edge out of w) do
           if mark[x] = inplay then
               mark[x] ← eliminated

   for v → w ∈ E do
     { if mark[w] = eliminated then
           reduce[v → w] ← true
       mark[w] ← vacant
     }
 }
```

Slika 4. Algoritam za uklanjanje tranzitivnih bridoba

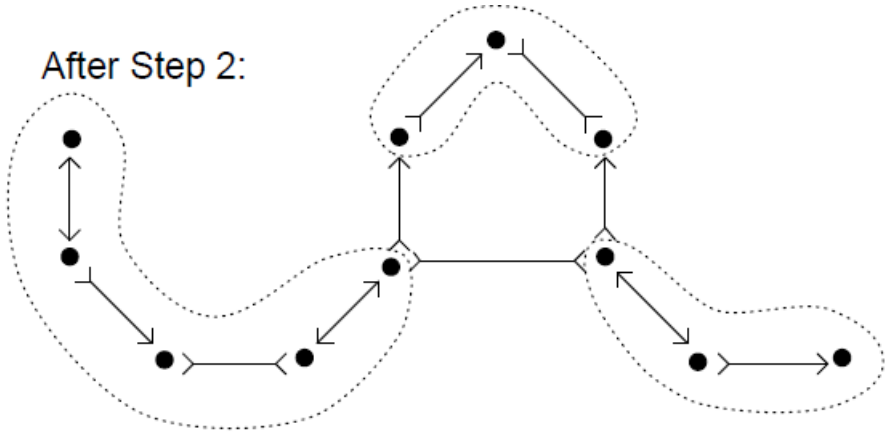
Način na koji ovaj algoritam radi može se objasniti na vrlo jednostavnom primjeru. Uzmimo vrh *v* i bridove koji izlaze iz tog vrha: $v \rightarrow w_1$, $v \rightarrow w_2$, ..., $v \rightarrow w_n$. Duljina ($v \rightarrow w$) je duljina znakovnog niza kojim je brid označen, dakle i duljina brida.

Na početku se svi vrhovi označe kao „*vacant*”, dakle prazni i spremi se informacija da nije potrebno reducirati niti jedan brid.

Sljedeći dio algoritma se odnosi na svaki vrh u skupu vrhova. Najprije se svi vrhovi do kojih se može doći iz pojedinog vrha *v* označe kao „*inplay*”, kao da su „*u igri*”. Nađemo najdužu sekvencu uzevši u obzir i dodatnu vrijednost koju unosi faktor FUZZ. Za svaku sekvencu *v→w* po duljini vrijedi da ako je *w* označen kao *inplay*, onda je potrebno proći svaki vrh koji izlazi iz w_i i označiti za eliminaciju vrhove koji su na početku takvog brida, a koji su inicijalno označeni kao *inplay*. Uvjet koji je korišten u algoritmu odnosi se na provjeru duljine bridova koji izlaze iz w_i. Obrada se završava ako je brid predug za eliminaciju bridova koji izlaze iz *v*.

Krajnji dio algoritma za svaki vrh koji je u listi susjednih *v*-a vrši ispitivanje; ako je bio označen za eliminaciju, vrši se reduciranje pripadnog brida, a *w* se označi kao *„vacant”*.

Kao vrijednost za FUZZ korišten je broj 10.



Slika 5. Graf nakon uklanjanja tranzitivnih bridova

Na ovoj slici je jasno kako je jako velik broj bridova otpao prilikom eliminacije tranzitivnih.

2.2.1.3 Generiranje kontiga

Ova faza je zapravo završni dio faze razmještaja. Ovdje najprije mičemo unutarnje vrhove; one kod kojih je samo jedan unutra i jedan van brid. Ovaj dio nije neophodan,

no provodimo ga radi jednostavnosti gradnje konačnog grafa i smanjenja broja bridova.

Potom se u više navrata miču vrhovi koji nemaju nijedan brid koji bi bio ni unutra ni van.

Jedan od zadnjih zadataka prije gradnje kontiga jest određivanje koji su bridovi obuhvaćeni. Problem se formulira kao najmanji trošak toka mreže gdje želimo naći minimalan broj obuhvaćenih elemenata tako da se poštuju granice bridova i zadrži se tok (jednakost broja izlaznih i ulaznih bridova).

U rekonstrukciji se promatraju granične sekvence vodeći računa o tome da postoje praznine zbog nemogućnosti ove faze da ih popuni. Za svaku od tih sekvenci vrijedi da postupak kreće od 0-tog ulaznog vrha i završava u izlaznom vrhu s indeksom 0. Radi ispravnog formuliranja problema, dodaju su ε-labelirani metabridovi kao ulazni i izlazni u svaki čvor iz posebnog vrha *s*. Ne postoje granice na ovim bridovima pa nema potrebe za nekim dodatnim pojednostavljenjima već možemo pokušati naći ciklične puteve gdje ne postoji prekid kontiga kad god je *s* zaobiđen.

Formalno se ovaj problem može formulirati na sljedeći način:

ULAZ: *Za svaki brid e, gornja franica toka je c(e) ≥ 0, a trošak po jedinici toka je v(e). Za svaki vrh v u toku zadovoljenje, odnosno zahtijev je označen s b(v).*

IZLAZ: *Tok x(e) za svaki brid definiran kao* $\sum_e v(e)x(e)$ *jest minimalan za x(e)* $\in$ *[0, c(e)]*

i $\sum_{u\to v} x(u \to v) + b(v) = \sum_{v\to w} x(v \to w)$.

Problem kojeg rješavamo je zapravo formuliran na sljedeći način:

c(e) = u(e) - l(e)

b(v) = $\sum_{u\to v} l(u \to v) - \sum_{v\to w} l(v \to u)$

v(e) = 1

t(e) = l(e) + x(e)

Jednostavna struktura granica bridova u našem slučaju (=1, ≥1 ili ≥0) vodi jednostavnom problemu toka. Određena pojednostavljenja za algoritam složenosti *O(EV)* su:

1. Za svaki (=1)-brid podrazumijevajući *c(e)* = 0, postavi se *x(e)* = 0 i brid se uklanja.
2. Ako vrh s *b(v)* = 0 nema ni ulaznih ni izlaznih bridova, postavi se *x(e)* = 0 za svaki brid *e* susjedan vrhu. Potom se ukloni vrh i svi susjedni bridovi.
3. Ako vrh *v* ima jedinstven izlazni brid $v \to w$ i *b(v)* > 0, tad se doda *b(v)* *x(*$v \to w$*)*, doda se ili oduzme *b(v) b(w)* ovisno o usmjerenju brida u *w*(za ulazne se zbraja, za izlazne se oduzima) te se postavi *b(v)* na 0.
4. Ako vrh *v* ima jedinstven ulazni brid $u \to w$ i *b(v)* < 0, tad se doda *b(v)* *x(*$u \to w$*)*, doda se ili oduzme *b(v) b(u)* ovisno o usmjerenju brida u *u*(za ulazne se oduzima, za izlazne se zbraja) te se postavi *b(v)* na 0.

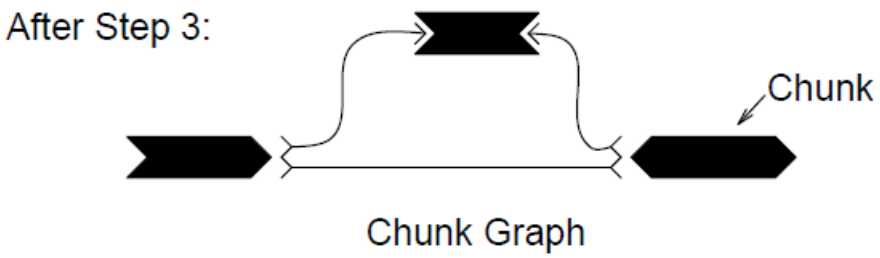
Bridovi koji ostanu nakon gore navedenih pojednostavljenja zovu se netrivijalni bridovi.

Iako se pojednostavljenja čine prilično jednostavna, poprilično su efikasna u smanjenju veličine problema izgradnje grafa.

Za generiranje kontiga koristimo dva algoritma. Prvi uzme nasumični vrh i ide u oba smjera preko sortiranih bridova te tako gradi kontig. Drugi algoritam pak uzme sve vrhove i isprobava koji najduži kontig može napraviti. On nije potpun jer ne ispituje cijeli graf već samo gleda najkraće bridove. U suprotnom bi se složenost cijelog algoritma poprilično povećala.

Potom još maknemo kontige koji sadrže samo jedno očitanje jer oni ne nose nekakvu informaciju koja nam je bitna za generiranje grafa. Oni su u većini jer nam ostaje samo skup kontiga koji imaju uparena očitanja. Primjerice, od 200 kontiga, nakon micanja onih sa samo jednim očitanjem, ostaje oko 60 kontiga.

Drugi algoritam, iako složeniji, daje bolje rezultate što je i za očekivati jer se radi o kombiniranju vrhova i bridova s ciljem pronalaska najdužeg kontiga.



Slika 6. Graf na izlazu iz faze razmještaja

Ova slika predstavlja tri kontiga koja su dobivena iz početnog grafa primjenom sastavnih koraka faze razmještaja.

2.3 Faza konsenzusa

Ovo je posljednja faza OLC pristupa. Kad dođemo do ovog dijela, graf je već reduciran u velike skafolde - nizove kontiga za koje su određeni međusobna pozicija i udaljenost. Idealno bi bilo da se dobije jedan jako veliki skafold koji bi bio predstavljen jednim čvorom koji bi pak nastao sažimanjem drugih čvorova u prethodnim fazama.

Konsenzus svih očitanja se izračunava kombiniranjem svih skafolda počevši od krajnje lijevog očitanja svakog skafolda. Praznine u genomu se mogu i dalje pojaviti ako ne postoji dovoljno parova u fazi konsenzusa. U tom slučaju, rezultat bi bio fragmentirani genom sastavljen od višestrukih skafolda budući da ne postoji način da te praznine popune, odnosno skafoldi sjedine.

Cilj je u ovom koraku odrediti smještaj pojedinih baza u genomu. Budući da je genom presekvencioniran, događa se situacija u kojoj postoji višestruki izbor, tj. potrebno je konsenzusom odrediti koje očitanje treba odrediti nukleotid na takvim mjestima. Kad postignemo konsenzus, postupak staje i genom je sastavljen.

3. Podaci

3.1 MHAP format

Ulaz u programski kod predstavljen je datotekom s MHAP preklapanjima. MHAP je kratica za *MinHash Alignment Process* i predstavlja jedan od novijih pristupa u postupku sastavljanja genoma. Zbog načina na koji su podaci prikazani, moguće je predočiti velik prostor preklapanja. Lako je čitljiv, stoga omogućava brže pronalaženje grešaka.

Izlaz kojeg daje MHAP poprilično nalikuje M4 BLASR-ovom formatu. Uopćeni prikaz jednog retka u datoteci s MHAP preklapanjima dan je sljedećim izrazom:

[A ID] [B ID] [Jaccard score] [# shared min-mers] [0=A fwd, 1=A rc] [A start] [A end] [A length] [0=B fwd, 1=B rc] [B start] [B end] [B length]

Primjer izlaza bi bio:

155 11 87.83225 206 0 69 1693 1704 0 1208 2831 5871

155 27 87.11507 159 0 455 1678 1704 0 0 1225 1862

Ovdje je vidljivo da preklapanje postoji između sekvenci 155 i 11 te 155 i 27. Uzevši u obzir prvi primjer, vidljivo je da su i sekvenca 155 i sekvenca 11 u uobičajenom redoslijedu, tj. nisu komplementirane. Treći broj predstavlja iznos Jaccardovog koeficijenta kao poznate mjere u statistici za uspoređivanje sličnosti i razlika između skupova za uzorkovanje, a poprima vrijednosti između 0 i 1, odnosno 0 i 100. Značenje preostalih brojeva je lako protumačiti na temelju izraza pisanog podebljanim slovima.

Bitno je naglasiti da granice koje su naznačene, a koje su uključive, nisu u potpunosti točne jer nema potpunog poravnanja. Preklapanje koje označava Jaccardov index je poprilično veliko što samo znači da je MHAP označio sekvence koje se podudaraju, a stvarni položaji preklapanja su unutar određene greške. Ovakav skup podataka je dosta pogodan za određivanje međusobnog položaja očitanja i gradnju grafa u fazi razmještaja.

Indeksiranje očitanja kreće od 1, a ne od 0.

3.2 GFA format

GFA je kratica za *Graph Format Assembly*. Puno je lakše prikazati i eksplicitno opisati poveznice između krajeva segmenata umjesto opisivanja grafa korištenjem lukova koji su usmjereni u dva smjera. Ta činjenica je razlog zašto se ovaj format koristi u mnogim prilikama, a osobito je čest u radovima koji se bave sastavljanjem genoma i prikazivanjem dobivenog grafa.

Isječak iz jedne datoteke u GFA formatu izgleda ovako:

H VN:Z:1.0

S 1 2 CGATGCAA *

L 2 3 5M

S 3 4 TGCAAAGTAC *

L 3 6 0M

S 5 6 TGCAACGTATAGACTTGTCAC * RC:i:4

L 6 8 1M1D2M1S

S 7 8 GCATATA *

L 7 9 0M

S 9 10 CGATGATA *

S 11 12 ATGA *

C 9 11 2 4M

Iz gornjeg primjera je jasno da se radi o grafu 1:2 >----> 3:4; 5:6 >----> 3:4;

5:6 >----< 7:8 <---->9:10. pri čemu je iz ispisa baza jasno da se 11:12 nalaze sadržani u 9:10. Slova prva u svakom retku imaju svoje značenje. H označava zaglavlje, L je poveznica koja se sastoji od oznaka dvaju krajeva i CIGAR-a koji opisuje poravnanje preklapanja promatrajući prvi kraj ciljane sekvence. CIGAR opisuje simetrično preklapanje (npr. 5M), praznine (10N) i sl. Oznaka C predstavlja onaj slijed baza koji je već sadržan u drugom slijedu (eng. *containment*).

4. Rezultati

Implementacija koju smo ostvarili već je opisana u prethodnim poglavljima. Kao što je i navedeno, koristili smo dva pristupa u izračunavanju kontiga. Može se pretpostaviti da će algoritam koji krene od pojedinačnog čvora i samo pokušava izgraditi kontige trajati trostruko kraće nego drugi algoritam. Budući da drugi nastoji naći najdulji kontig uparivanjem očitanja, očigledno je da će u slučaju njegova korištenja vremenska složenost biti na uštrb kvalitete rješenja. Prilkom pokretanja implementacije u programskom jeziku C++, jednostavniji algoritam je dao rješenje za 25 sekundi, dok je složenijem trebalo 70 sekundi.

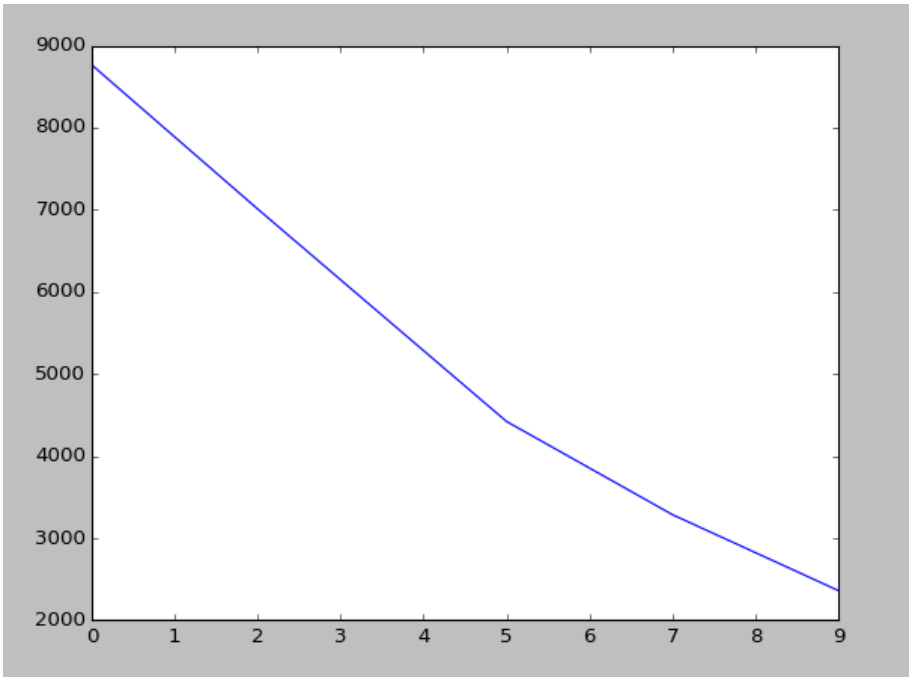
Radi se o problemu koji zahtijeva rad s velikim brojem baza. Iz tog razloga datoteke koje predstavljaju ulaz i izlaz su veličine reda MB. Očigledno je da će i memorijsko zauzeće biti prilično. Prilikom određivanja utroška memorije za rad s bakterijom *Echerichiom coli* duljine 5 600 000 baza, utvrđeno je da ono iznosi 55.6 MB.

U prethodnim poglavljima je spomenut parametar *percentMargin*. Isti se koristi prilikom izbacivanja već sadržanih očitanja u trenutku kad se treba odrediti je li preklapanje ono što zovemo „*containment*“. Tablica u nastavku predstavlja ovisnost broja bridova koji ostanu nakon uklanjanja već sadržanih bridova o ovom parametru.

percentMargin	Broj bridova koji ostaju u grafu
0	8765
2	7010
5	4422
7	3287
9	2362

Tablica 1. Ovisnost broja zadržanih bridova o parametru *percentMargin*

Očito je da broj bridova opada povećanjem *percentMargina* što je vidljivo iz sljdećeg grafa:



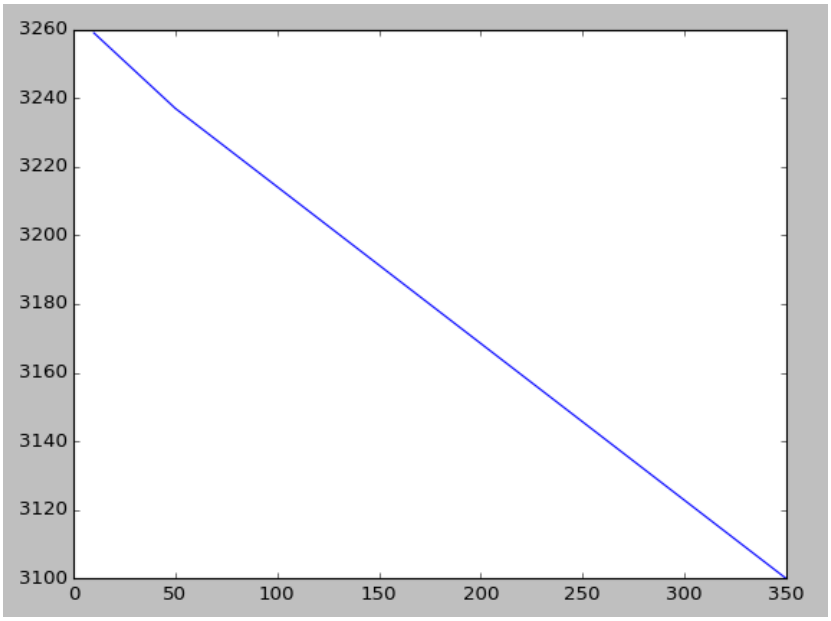
Slika 7. Graf ovisnosti broja zadržanih bridova o parametru *percentMargin*

Kao što je razmatrano za ovaj parametar, tako možemo prikazati rezultate za parametar FUZZ. Prije uklanjanja tranzitivnih, bridova je u grafu ukupnoo 6574.

FUZZ	Broj bridova nakon uklanjanja tranzitivnih
10	3259
50	3237
350	3100

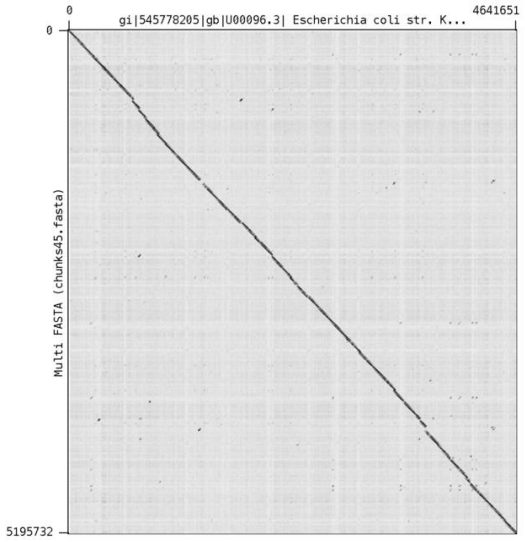
Tablica 2. Ovisnost broja zadržanih bridova o parametru FUZZ nakon provedenog koraka uklanjanja tranzitivnih bridova

Iz prikaza je vidljivo kako za različite vrijednosti FUZZ-a, broj bridova koji ostanu nakon uklanjanja tranzitivnih bridova ne varira previše. Razlika je svega nekoliko desetaka bridova.



Slika 8. Graf ovisnosti broja zadržanih bridova o parametru FUZZ nakon provedenog koraka uklanjanja tranzitivnih bridova

Konačni prikaz koji se dobije primjenom implementiranog algoritma izgleda poprilično dobro. Kontige koje smo dobili poredali smo na način da smo izgradili sekvencu DNA koja je u Gepardu prikazana pod kutom od 45º. Najduži kontig ima oko 1 400 000 baza, a kontige smo rezali jer smo ih željeli prikazati u Gepardu kako bi se i inače dobilo da se provode sve faze. Rezanje nije bilo potrebno jer *Echerihia coli* koja je i prikazana u svim ovim primjerima zapravo ima cirkularni genom.

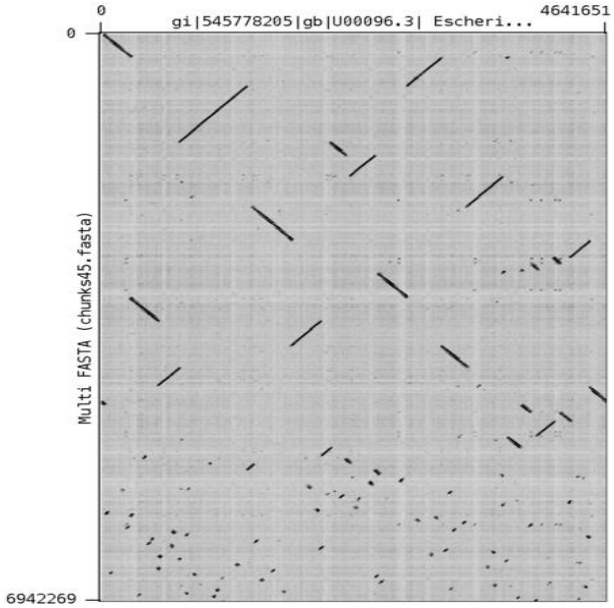


Slika 9. Poravnanje dobiveno pomoću implementacije

Kako je i vidljivo, postoje određene praznine što je i bilo za očekivati jer je potrebno još provesti i fazu konsenzusa.

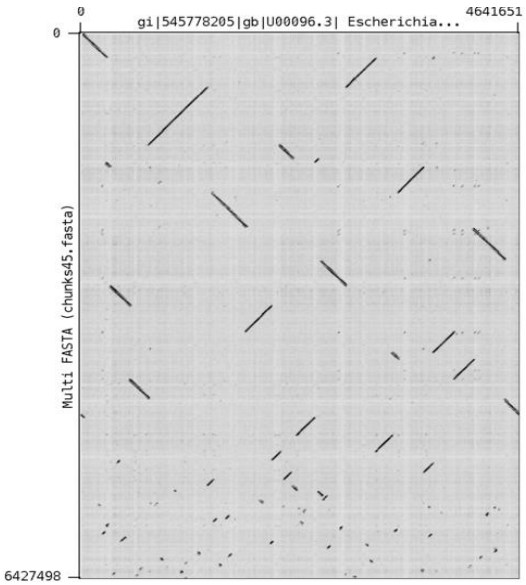
Zanimljivo je još pogledati kako izgleda genom kad koristimo različite vrijednosti *percentMargina*:

- *percentMargin* = 0:



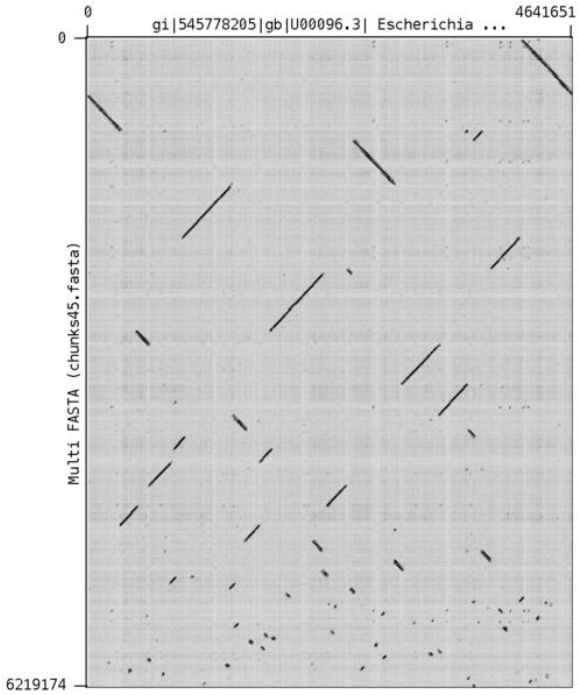
Slika 10. Kontizi za *percentMargin* = 0

- *percentMargin* = 2:



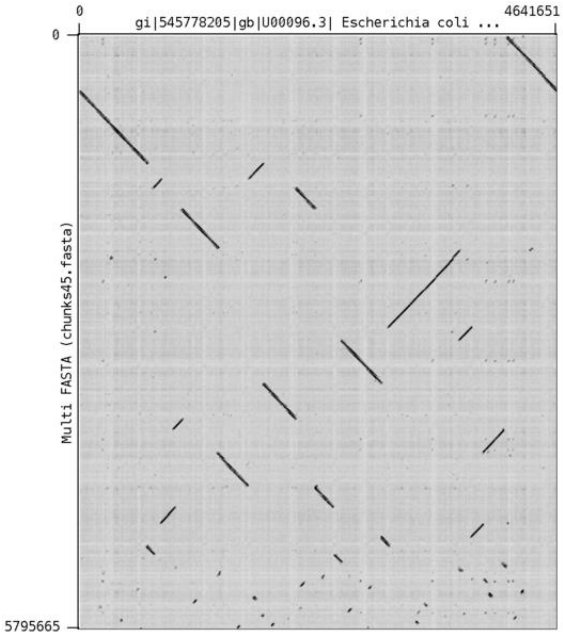
Slika 11. Kontizi za *percentMargin* = 2

- *percentMargin* = 5:



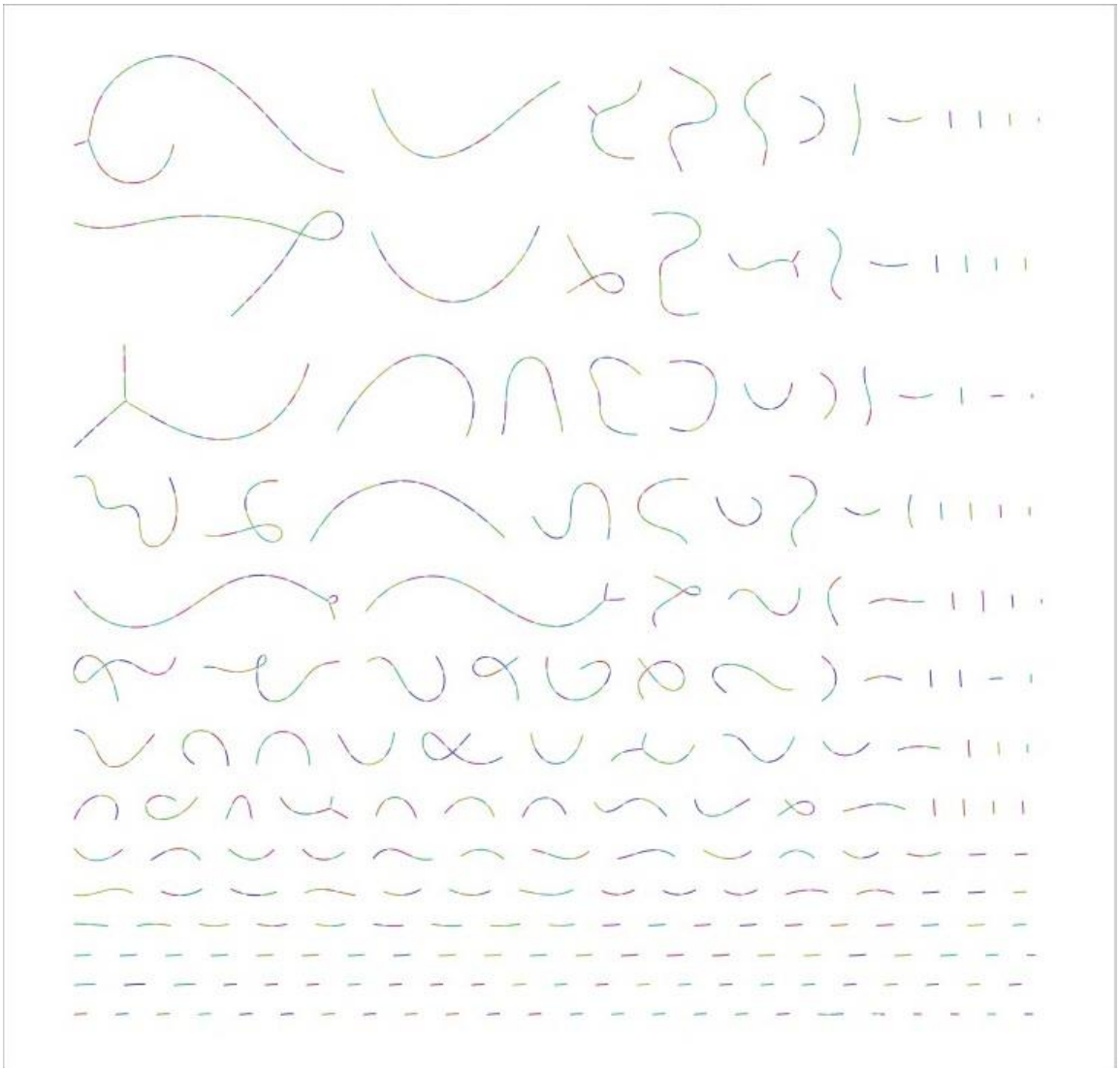
Slika 12. Kontizi za *percentMargin* = 5

- *percentMargin* = 7:



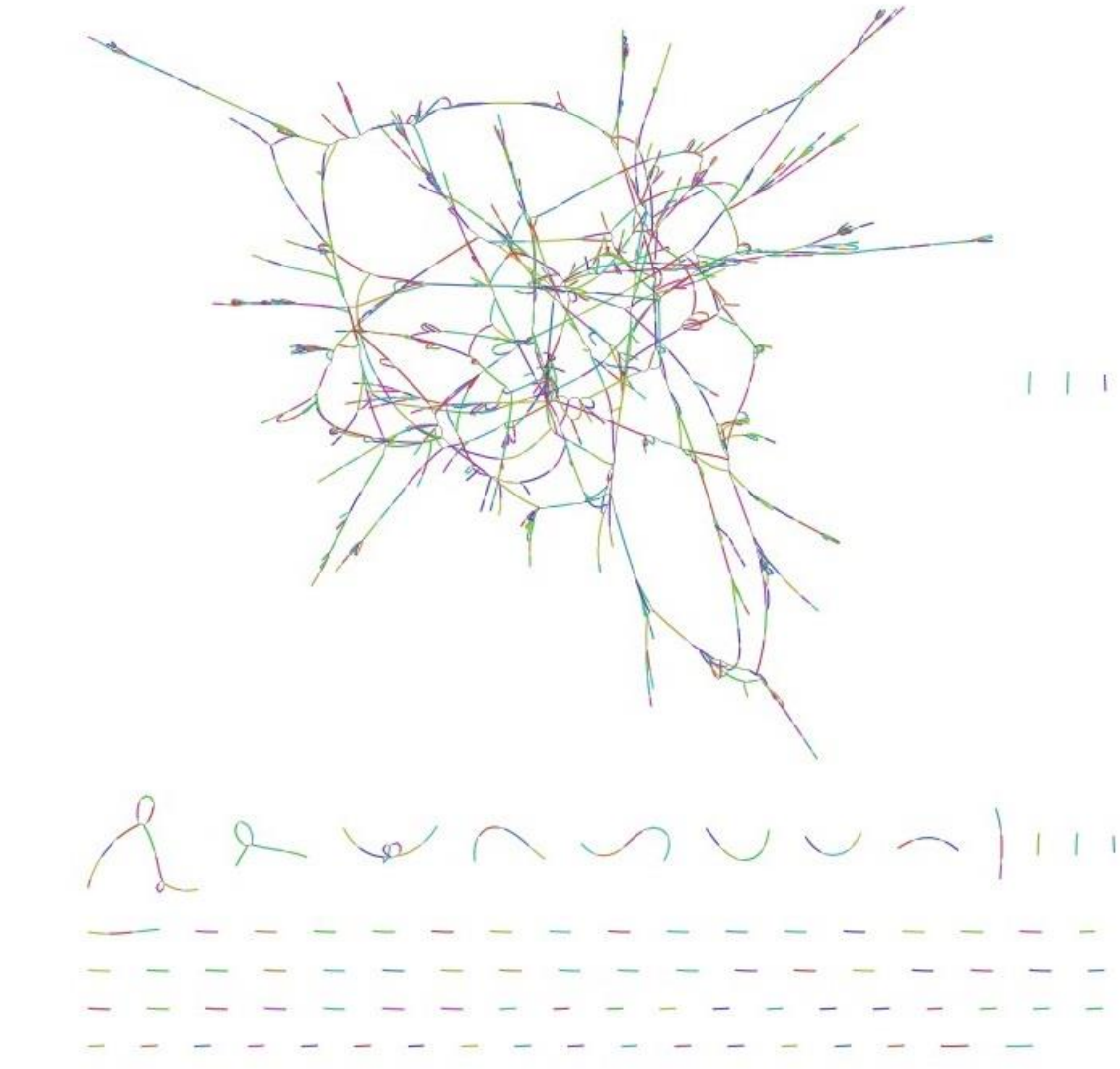
Slika 13. Kontizi za *percentMargin* = 7

Još je zgodno pogledati kako sastavljeni genom izgleda u alatu Bandage. On crta graf dobiven na izlazu iz faze razmještaja. Taj graf se zadaje u GFA formatu. Kontizi koje smo dobili izgledaju ovako:



Slika 14. Kontizi dobiveni implementacijom i prikazani pomoću alata Bandage

Njihova povezanost je vidljiva kad su nacrtani svi zajedno.



Slika 15. Međusobna povezanost kontiga prikazana u alatu Bandage

5. Literatura

[1] Myers, E. W. (2005). The fragment assembly string graph. Bioinformatics, 21(suppl 2), ii79-ii85.

[2] Myers, E. W. (1995). Toward simplifying and accurately formulating fragment assembly. Journal of Computational Biology, 2(2), 275-290.

[3] Huang, X. *et al.* (2003) PCAP: A whole-genome assembly program. *Genome res.,* **13**, 2164-2170.

[4] Narzisi, G., Mishra, B. (2011). Comparing de novo genome assembly: The long and short of it, PLoS ONE

[5] Pavlović, D. Evaluacija metode za ispravljanje pogrešaka kod dugačkih očitanja, Završni rad, FER, 2013.

6. Sažetak

U ovom radu je prikazan postupak sastavljanja genoma *de novo* pristupom. Najprije je dan kratki uvod u područje s obrazloženjem problematike i algoritama koji se koriste kako bi se postigao što bolji rezultat. Naglasak je na algoritmu za sastavljanje genoma koji pripada skupini algoritama temeljenih na grafu. Radi se o OLC pristupu,točnije samo njegovoj drugoj fazi iako su i druge faze opisane.

Za ulaz je korišten MHAP format koji prikazuje preklapanja kao početnu točku faze razmještaja. Pokazan je slijed koraka koje treba proći kako bi se dobili kontizi koji su zadovoljavajuće duljine i koji, posloženi na odgovarajući način budući da provodimo samo ovu fazu, daju poprilično lijepe sekvence DNA. Posebna pozornost je pridana prikazu rezultata koji su bolji, odnosno lošiji ovisno o vrijednosti postavljenih parametara. Primjetna je razlika između dobivenih prikaza u alatu Gepard, ne samo u ovisnosti o navedenim parametrima već i u ovisnosti o primijenjenom algoritmu za izgradnju kontiga. Isti su pobliže objašnjeni u sklopu faze generiranja kontiga.

Kao što je i navedeno, parametri itekako utječu na kvalitetu generiranih rješenja. Postoje određeni načini da se ona poboljša, ponajprije kombinacijom različih algoritama koji mogu pridonijeti dobivanju boljih i većih kontiga, idealno jednog. Potrebno je provesti dodatne simulacije na novim testnim primjerima i napraviti mjerenja koja će pomoći da se postupak sastavljanja genoma dovede na optimalnu razinu. To se odnosi i na poboljšanje rješenja, ali i smanjenje zauzeća memorije i vremenske složenosti.