SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Bioinformatika

Genome assembly
Problem 2: Unitigging

Dokumentacija

Mentor:
doc. dr. sc. Mile Šikić

Studentski tim:
Ivan Damjanović
Mirko Jurić-Kavelj
Josip Marić

Siječanj, 2014.

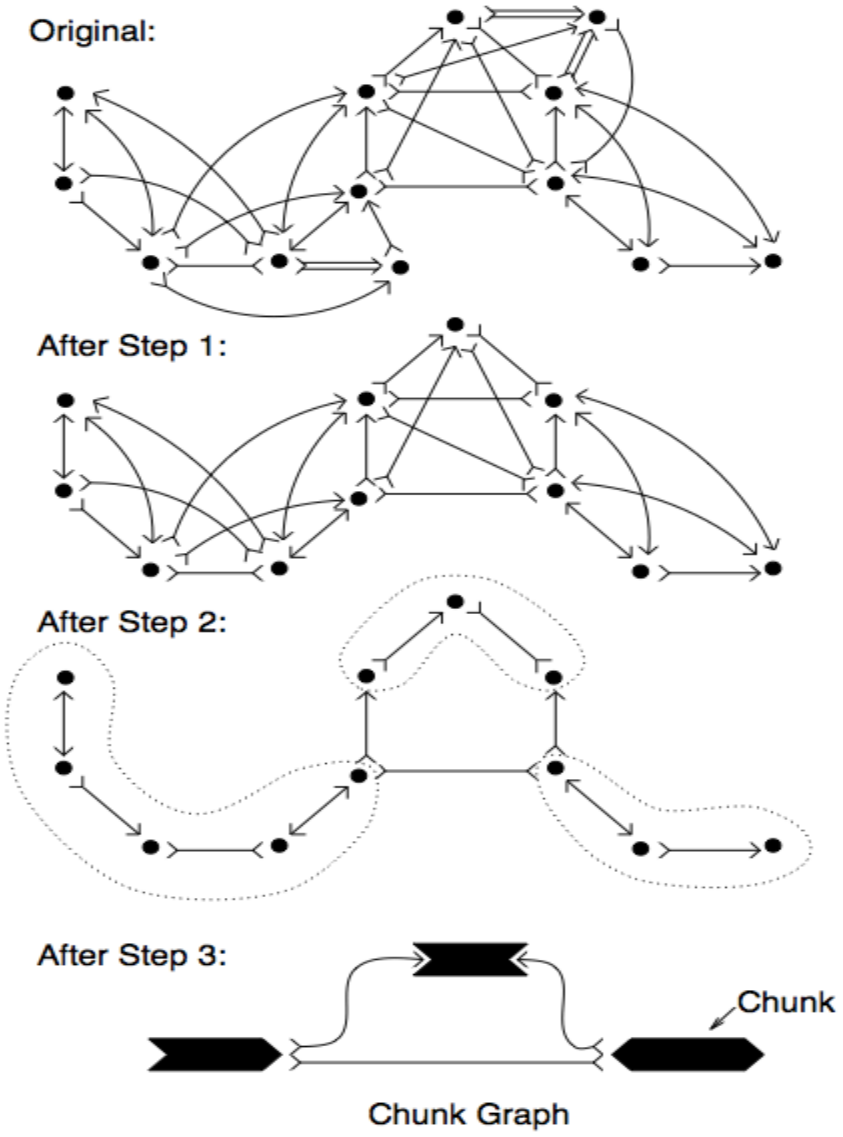
1. Opis zadanog problema

1.1. Uvod u problem

Jedan od glavnih ciljeva bioinformatike je mapiranje i analiziranje sljedova nukleotida DNA. Postupak slaganja genoma započinje sekvenciranjem fragmenata DNA čija prosječna duljina najčešće ne prelazi 2000 nukleotida. Nakon sekvenciranja potrebno je sastaviti fragmente da bi se dobio originalni poredak nukleotida u DNA. Problem se može postaviti kao pronalazak najkraćeg niza znakova (nukleotida) tako da za svaki fragment postoji podniz u tom nizu znakova koji odgovara fragmentu sa postotkom pogreške ε. Postupak koji rješava ovaj problem sastoji se od tri koraka: (1) *Overlapping*, (2) *Unitigging* i (3) *Consensus*. U prvoj fazi pronalaze se preklapanja između fragmenata i generira se graf preklapanja u kojem su vrhovi očitanja, a bridovi povezuju dva vrha ako se ta dva očitanja preklapaju. U drugoj fazi preklapajući graf se pojednostavljuje da bi se moglo pronaći konačno rješenje u trećoj fazi.

U ovom projektu, proučavala se i ostvarila programska implementacija druge faze ovog postupka. Druga faza podijeljena je u tri podfaze koje rezultiraju pojednostavljenim preklapajućim grafom.

Na slici 1 prikazane su tri faze Unitiggin postupka. U prvoj fazi uklanjaju se očitanja tj. vrhovi preklapajućeg grafa koja su sadržana u nekom drugom očitanju , te svi bridovi koji izlaze iz tog vrha. U drugoj fazi uklanjaju se tranzitivne veze među bridovima (tranzitivnost opisana u nastavku), a u trećoj fazi od preostalih vrhova i bridova izgrađuje se *chunk* graf (objašnjeno u nastavku) koji se predaje postupku Consensus.


Original:
After Step 1:
After Step 2:
After Step 3:
Chunk
Chunk Graph

1.2. Opis ulaza

Unitigging faza prima kao ulaz listu preklapanja među očitanjima. Primjer formata jednog preklapanja:

{OVL
adj:N
rds:1,2
scr:0
ahg:139
bhg:1685
}

OVL je ključna riječ koja predstavlja jedno preklapanje među dva očitanja. Parametar *adj* može poprimiti vrijednost iz skupa {N, A, I, O}, te govori kako su orijentirani fragmenti unutar preklapanja. Budući da svaki fragment može biti iz jednog lanca DNA njegova orijentacija može biti u lijevo ili u desno. Dakle, vrijednosti parametra adj imaju sljedeće značenje:

- N - oba fragmenta orijentirana su u desno
- I - prvi fragment orijentiran je u desno, drugi u lijevo
- O - prvi fragment orijentiran je u lijevo, drugi u desno
- A - oba fragmenta orijentirana su u lijevo

Parametar *rds* sadrži identifikatore fragmenata koji se preklapaju.

Parametri *ahg* i *bhg* pokazuju koliko baza u fragmentu ne sudjeluje u preklapanju. *ahg* se odnosi na prvi fragment u preklapanju, a *bhg* na drugi. Ovi parametri mogu poprimiti i negativne vrijednosti ako se drugi fragment preklapa s početkom prvog fragmenta (negativni ahg) ili se prvi fragment proteže dalje od kraja drugog.

Iz ovih parametara slijede sve mogućnosti orijentacije i preklapanja očitanih fragmenata. Prikazati će se sva preklapanja za vrijednost parametra adj=N, a ostala preklapanja slijede analogno.

1. ahg pozitivno, bhg pozitivno

   1: ==> <-- 1685bp -->

   2: <-- 139bp --> ==>

2. ahg pozitivno, bhg negativno

   1: ==>

   2: <-- 139bp--> ==> <-- --1685bp →

3. ahg negativno, bhg pozitivno

   1: <-- -139bp--> ==========> <-- 1685bp →

   2: ==========>

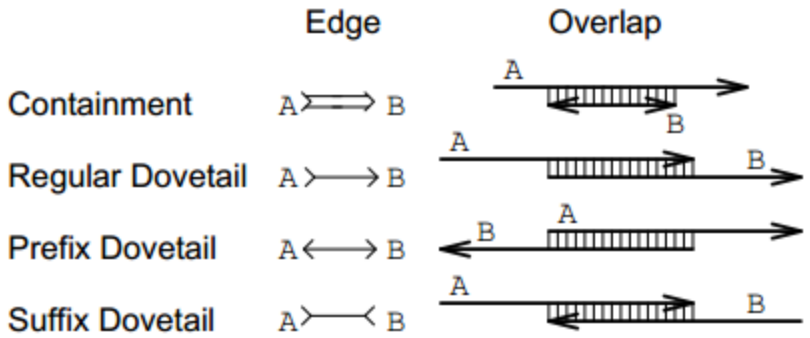
4. ahg negativno, bhg negativno

   1: <-- -139bp--> ==========>

   2: ==========> <-- -1685bp →

1.3. Opis postupka

Iz opisa ulaza vidimo da sva preklapanja možemo podijeliti na dvije glavne skupine: preklapanja u kojima jedan fragment sadrži drugi, te preklapanja u kojima se dva fragmenta preklapaju svojim krajem ili početkom, ali jedan ne sadrži drugi. Na slici 2 su prikazane sve moguće kombinacije preklapanja:



Dakle, prvi korak u postupku (nakon učitanja grafa u memoriju) je uklanjanje bridova koji su tipa *Containment*. Oni se spremaju u posebnu listu da bi se mogli kasnije dodati u ispis.
U drugom koraku nad tako reduciranim grafom provodi se sljedeći postupak: za svaka tri brida koja tvore trokut provjerava se zadovoljavaju li svojstvo tranzitivnosti te se označava brid koji treba biti izbačen iz grafa radi tranzitivne veze (jedan od tri). Nakon toga ponovo se vrši prolaz kroz sve bridove te se uklanjaju oni koji su označeni u prethodnom prolazu. Uvjeti za označavanje brida koji će se ukloniti prikazani su na slici 3.

$$\tau.suf_g \neq \tau'.suf_g$$
$$\pi.suf_f = \tau.suf_f$$
$$\pi.suf_h = \tau'.suf_h$$
$$\tau.hang_f + \tau'.hang_g \in \pi.hang_f \pm (\varepsilon \cdot length(\pi) + \alpha)$$
$$\tau.hang_g + \tau'.hang_h \in \pi.hang_h \pm (\varepsilon \cdot length(\pi) + \alpha)$$

pri čemu su τ, τ’ i π bridovi, a f, g i h su vrhovi, tj. očitanja. τ.suf(g) definira se kao istina ako je preklapajući dio očitanja g u preklapanju τ, njegov sufiks, a neistina inače. τ.hang(g) definira se kao duljina nepreklapajućeg dijela očitanja g u preklapanju τ. ε i α su konstante pri čemu je ε prosječna pogreška očitanja, a α je konstanta koja je tipično 3. Ako su ovi uvjeti zadovoljeni brid π se označava kao onaj koji se treba ukloniti iz grafa.

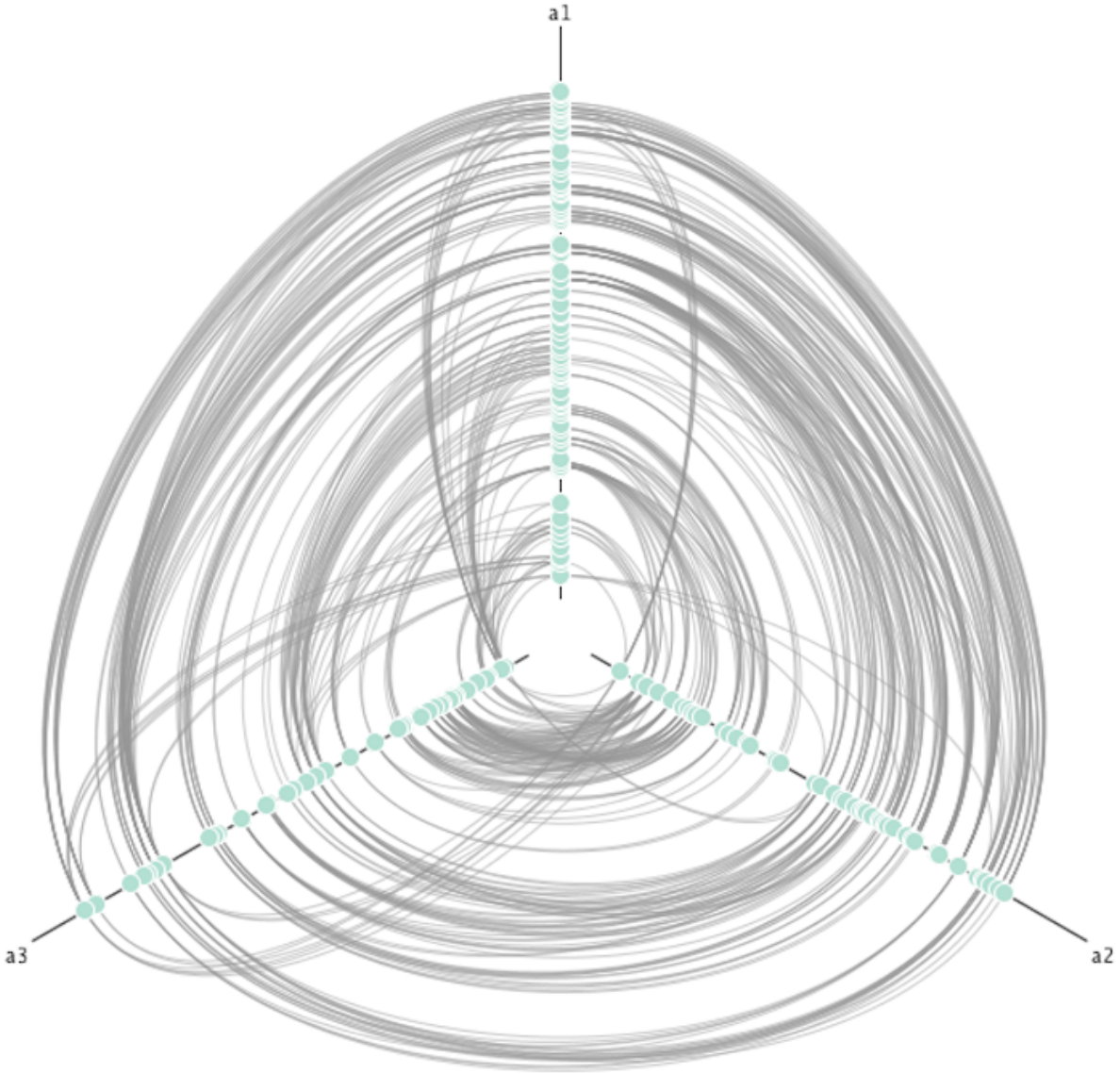
Nakon uklanjanja tranzitivnih bridova u trećem koraku potrebno je graf dobiven drugim korakom transformirati u *chunk* graf. Chunk graf je graf koji za vrhove ima *chunk-ove* umjesto očitanja. Svaki *chunk* je zapravo niz očitanja, a dobije se na sljedeći način:

1. svi vrhovi u grafu nakon drugog koraka pretvore se u *chunk-ove* te se tako dobije početni *chunk* graf.
2. za svaki brid (f spojen sa g vezom π) u grafu provjere se sljedeći uvjeti:
    a. za svaki brid τ spojen s vrhom f vrijedi π.suf(f) != τ.suf(f)
    b. za svaki brid τ spojen s vrhom g vrijedi π.suf(g) != τ.suf(g)
3. ako su uvjeti zadovoljeni, dva *chunk-a* koji su povezani tim bridom spoje se u jedan.

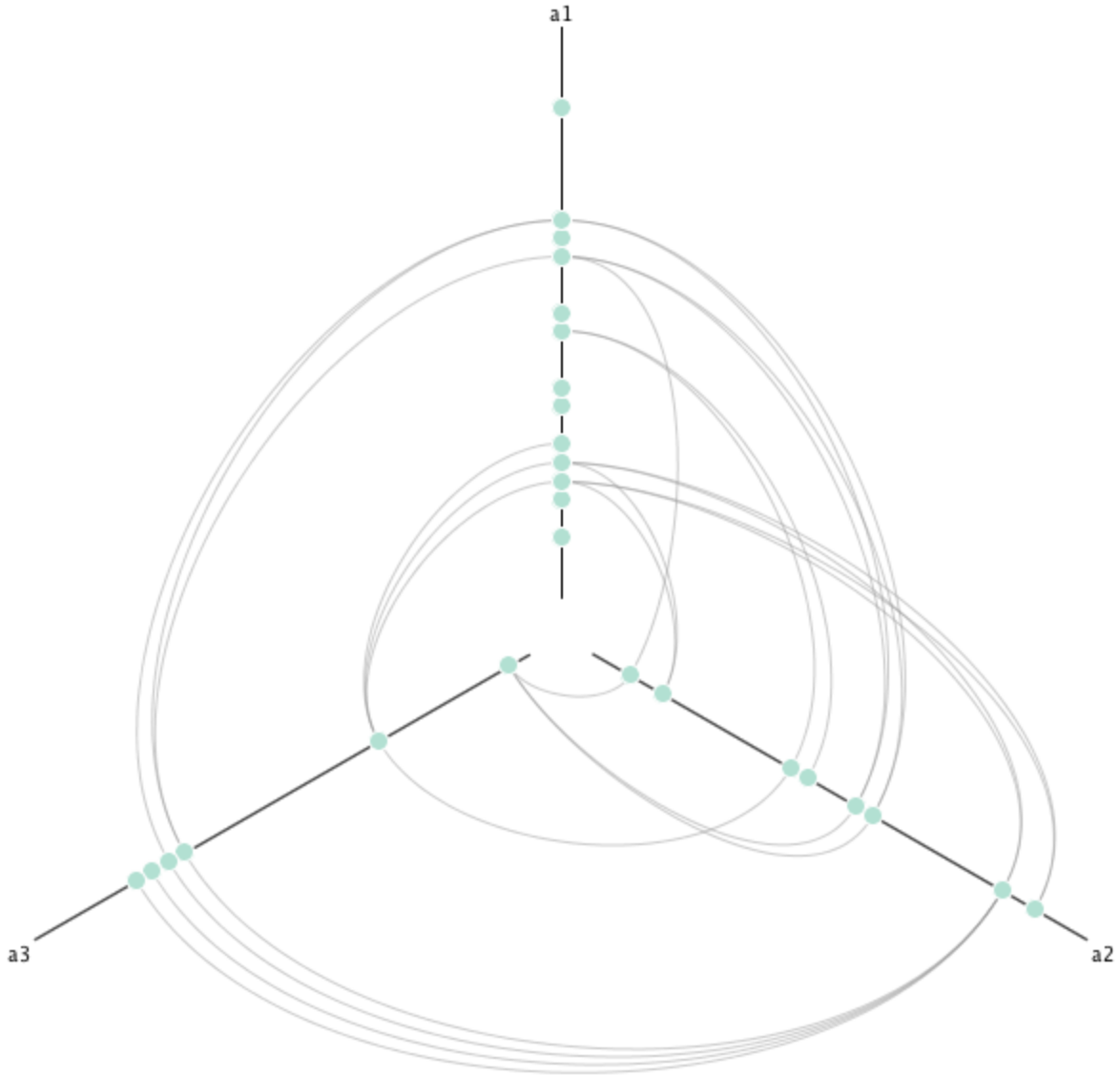
Konačno dobiveni graf nakon trećeg koraka, sastoji se od *chunk-ova* koji su najčešće nepovezani. Takav graf zapisuje se u datoteku kako bi poslužio kao ulaz trećoj fazi, tj. *Consensus*.

2. Primjer problema

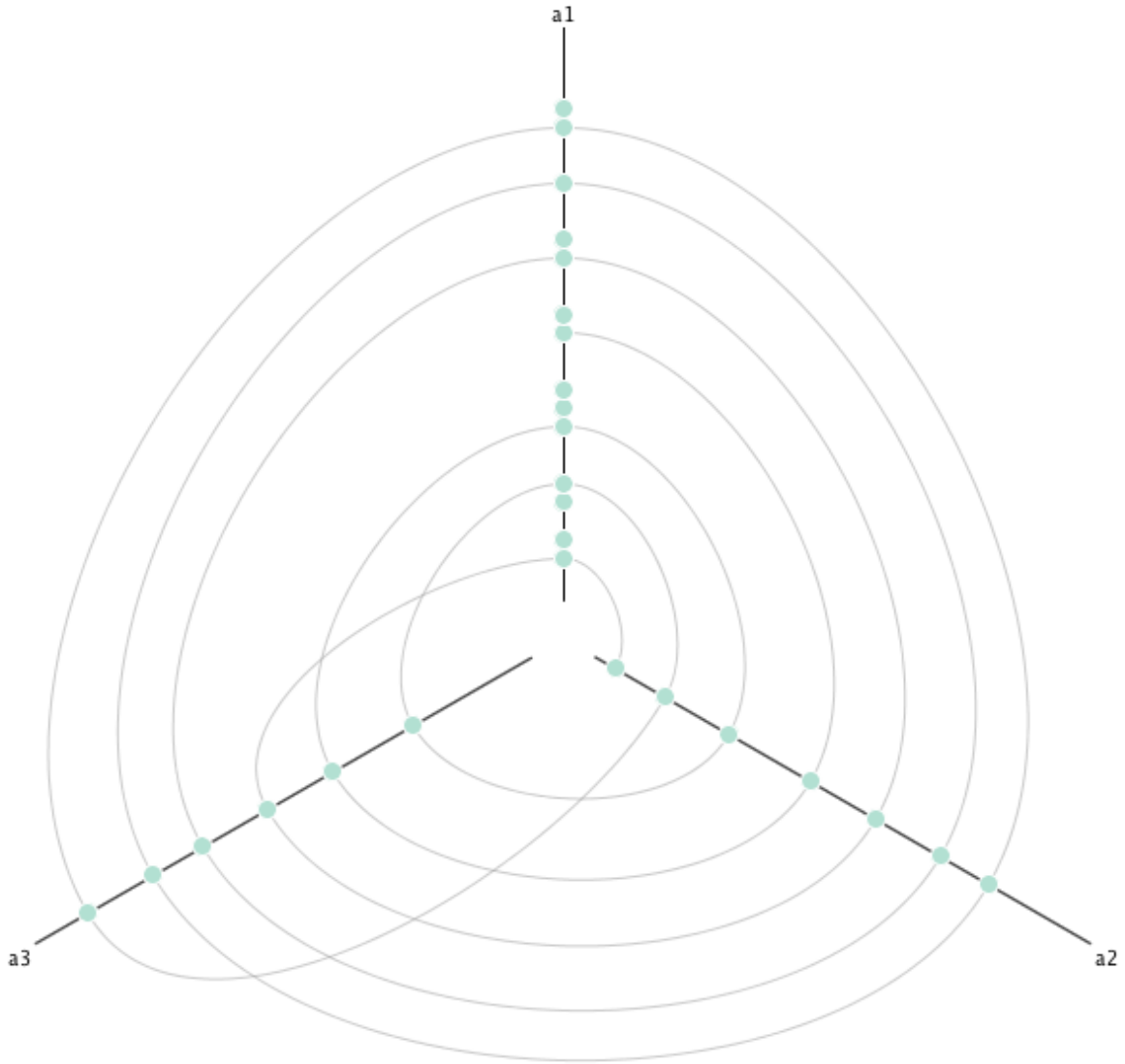
Kao primjer problema nad kojim ćemo prikazati rad algoritma izabran je dio genima bakterije Escerichiae Coli nad kojim je izvršeno 157 očitanja. Alatom Minimus izvršena je overlapping faza koja je izgenerirala 1063 preklapanja među očitanjima. Na slici je prikazan graf u kojem su vrhovi očitanja koja su postavljena na tri osi. Preklapanja među očitanjima su bridovi koji povezuju dva vrha, također prikazani na slici.



Na sljedećoj slici prikazan je graf koji se dobije nakon uklanjanja vrhova tj. očitanja, koji su sadržani u nekom drugom vrhu, odnosno očitanju, tzv. containment vrhovi.



Nakon provedbe drugog koraka, tj. uklanjanja tranzitivnih veza graf poprima svoj konačni oblik koji je prikazan na sljedećoj slici. Sa slike se jasno vidi da je dobiveni graf lanac, tj. svaki vrh je stupnja dva osim početnog i krajnjeg koji su stupnja jedan. Graf je jednostavan i u ovom primjeru dobiveni konačni graf ima samo jedan *chunk.*


a1
a3
a2

3. Prikaz rezultata testiranja

Testiranje je provedeno na 4 testna uzorka u kojima su očitanja uzeta sa bakterije Escerichiae Coli. Očitanja su generirana alatom readsim u pacbio_ec tehnologiji. Primjeri se razlikuju u broju nukleotida genoma s kojeg se uzimaju očitanja, te prosječnoj duljini očitanja. Svaki nukleotid prosječno je pokriven sa 6 očitanja.

1. primjer:
    - duljina genoma: 700000
    - prosječna duljina očitanja: 1000
    - prosjčno vrijeme izvođenja: 356 ms
    - prosječno memorijsko zauzeće: 43,556 KB
    - rezultati testiranja pomoću skripte: https://github.com/vzupanovic/skripte
        - [AN:] Total number of contigs: 165
        - [AN:] Number of contigs bigger then size 25000B: 5
        - [AN:] Total length of all contigs: 1311580
        - [AN:] Maximum contig size: 32005
        - [AN:] Minimum contig size: 0
        - [AN:] Average contig length: 7948.96969697
        - [AN:] Median contig length: 6441
        - [AN:] E-size with threshold 0: 14827.6897467
        - [AN:] N25 measure: 19997
        - [AN:] N50 measure: 13598
        - [AN:] N56 measure: 12594
        - [AN:] N75 measure: 9153
        - [AN:] Ratio between median and E-size: 2.30207883042
        - [AN:] Ratio between n50 and E-size: 1.0904316625
        - [AN:] Ratio between contig and read length: 105.986262626

2. primjer:
    - duljina genoma: 1400000
    - prosječna duljina očitanja: 2000
    - prosjčno vrijeme izvođenja: 502 ms
    - prosječno memorijsko zauzeće: 99,448 KB
3. primjer:
    - duljina genoma: 1400000
    - prosječna duljina očitanja: 4000
    - prosjčno vrijeme izvođenja: 548 ms
    - prosječno memorijsko zauzeće: 19,132 KB
4. primjer:
    - duljina genoma: 2400000
    - prosječna duljina očitanja: 2000
    - prosjčno vrijeme izvođenja: 706 ms

prosječno memorijsko zauzeće: 193 MB

Uz ove primjere, proveli smo testiranje i na primjeru očitanja uzetih sa genoma bakterije Paerignose. Broj učitahin očitanja bio je 21653530, a nakon micanja containment vrhova 21894845. Prosječno memorijsko zauzeće iznosilo je 2.08 GB, a vrijeme izvođenja 2 sata i 49 minuta.

4. Upute za instalaciju i pokretanje

Program je pisan u programskom jeziku Java i nisu korištene nikakve dodatne biblioteke tako da je jedini preduvjet ispravno instaliran i podešen JRE 7. Cjelokupan kod se nalazi na Github repozitoriju na adresi:

https://github.com/JKMirko/bioinformatika-projekt

U korijenskom direktoriju projekta nalazi se i izvršna datoteka *unitig.jar*, koja je generirana iz Eclipse-a, te direktorij *testData/* s testnim podacima potrebnim da bi se program mogao pokrenuti.

4.1. Pokretanje programa

SYNOPSIS

```
$ java -jar unitig.jar path_to_overlaps path_to_reads \
> [-oLayout=outputLayout.afg] [-oOverlaps=outputOverlaps.afg] [-epsilon=0.1] \
> [-alpha=3]
```

Program prima 2 obavezna argumenta i 4 opcionalna argumenta.

Obavezni:

- **path_to_overlaps** - putanja do datoteke koja sadrži informacije o preklapanjima
- **path_to_reads** - putanja do datoteke koja sadrži informacije o očitanjima

Opcionalni:

- **-oLayout** - Određuje putanju do datoteke u kojoj će biti zapisani podatci o *unitig layotu*. Ako datoteka postoji, bit će prebrisana, inače će biti stvorena nova. Ako ovaj argument nije unešen, *unitig layout* će biti ispisan na standardni izlaz.
  Način korištenja: *-oLayout=filepath*
- **-oOverlaps** - Određuje putanju do datoteke u kojoj će biti zapisani podatci o *unitig overlapu*. Ako datoteka postoji, bit će prebrisana, inače će biti stvorena nova. Ako ovaj argument nije unešen, *unitig overlap* će biti ispisan na standardni izlaz.
  Način korištenja: *-oOverlaps=filepath*
- **ε** - Realan broj iz intervala [0,1]. Koristi se u fazi uklanjanja tranzitivnih bridova. Pretpostavljena vrijednost je 0.1.
  Način korištenja: *-epsilon=value*

- **α** - Pozitivna cjelobrojna konstanta. Koristi se u fazi uklanjanja tranzitivnih bridova. Pretpostavljena vrijednost 3.
  Način korištenja: *-alpha=value*

Zainteresiranog čitatelja se upućuje na članak [1] za više informacija o parametrima ε i α.

Primjer korištenja:

```
$ java -jar unitig.jar testData/overlaps.afg testData/reads.2k.10x.fasta \
> -oLayout=outputLayout.afg -oOverlaps=outputOverlaps.afg -epsilon=0.1 \
> -alpha=3
```

5. Zaključak

Bioinformatika je brzo rastuće područje znanosti koje objedinjuje biologiju i računarstvo. Interdisciplinarnost zahtjeva da pojedinac svoju stručnost razvija u više različitih grana znanosti. Često algoritme koji svoju svrhu pronalaze u biologiji implementiraju biolozi koji najčešće nisu dovoljno stručni na području računarstva. Zbog toga trenutno postojeće implementacije i rješenja nekih problema u bioinformatici ostavljaju dosta prostora za poboljšanje. Cilj ovog projekta bio je da se upoznamo s područjem bioinformatike i problemima koji su trenutno aktualni, te da pokušamo naći mogućnost poboljšanja trenutno postojećih rješenja.

Iz provedenih testiranja uočavamo da za genome duljine reda 1000000 nukleotida, algoritam radi vrlo brzo. Za test na bakteriji paerignosa red vremena izvođenja mjeri se u satima, a memorijsko zauzeće u GB. No, budući da algoritam ima još prostora u poboljšavanju brzine izvođenja, a minimus kao alat s kojim se mogu usporediti performanse ne uspijeva obraditi cijeli ulaz zaključujemo da se kvaliteta algoritma može u najmanju ruku nositi sa trenutno postojećim rješenjima.

Kao daljnje mogućnosti poboljšanja ovog postupka sekvenciranja genoma ističemo veću interakciju triju faza postupka. Navodimo potencijalnu mogućnost poboljšanja traženja svih preklapanja u prvoj fazi tako da jednom pronađeni *containment* vrh više ne uspoređujemo sa drugima jer ta informacija nije korisna u daljnim fazama postupka. Također, algoritam ima mogućnost ubrzanja paralelizacijom čija implementacija nije teško ostvariva.

6. Literatura

[1] Toward Simplifying and Accurately Formulating Fragment Assembly, Eugene W. Myers 1995.
[2] http://sourceforge.net/apps/mediawiki/amos/index.php?title=Message_Types