SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Bioinformatika

OLC postupak sastavljanja genoma
Problem 3: Konsenzus
Realigner

Lipovac Igor, Tošić Franjo, Vulić Nives

Voditelj: *doc. dr. sc. Mile Šikić*

Zagreb, Siječanj, 2014.

Sadržaj

1. Uvod

Sastavljanje, mapiranje i analiziranje nukleotida genoma jedno je od osnovnih područja proučavanja bioinformatike. Sastavljanje genoma je proces u kojem se iz velikog broja kratkih očitanja DNA sekvence sastavlja sekvenca koja bi trebala reprezentativno predstavljati originalnu DNA sekvencu. Jedan od postupaka kojim se dobivaju početna kratka očitanja na temelju kojih se vrši sastavljanje zove se "shotgun" sekvenciranje. Tim postupkom dobije se veliki broj kratkih preklapajućih očitanja koja se zatim različitim programima i algoritmima sastavljaju u konačnu sekvencu. Dva različita pristupa sastavljanju genoma su de-novo i mapirajući pristup. Glavna razlika je u tome što de-novo postupci koriste kratka očitanja dobivena primjerice "shotgun" postupkom te iz njih, bez znanja o referentnom genomu, pokušavaju sastaviti sekvencu koja će biti reprezentacija tog genoma dok mapirajući pristupi sastavljaju genom iz očitanja s obzirom na neki već postojeći referentni genom.

OLC algoritam spada u takozvanu "novu generaciju" postupaka za sastavljanje sekvenci. OLC je de-novo postupak koji se sastoji od 3 faze - (1) *overlap*; (2) *layout* i (3) *consensus*. U prvoj fazi (*overlap*) kratka očitanja se poravnavaju te se traže preklapanja između očitanja te se stvara graf preklapanja očitanja, druga faza (*layout*) pojednostavljuje graf preklapanja i daje točniji raspored očitanja koja se preklapaju, konačno u posljednjoj fazi (*consensus*) iz rasporeda očitanja se određuje konsenzus - niz koji će reprezentativno predstavljati sastavljeni dio sekvence. Upravo tom posljednjom fazom bavi se ovaj rad te će u sljedećim odlomcima biti opisan algoritam te implementacija algoritma iskorištena za stvaranje konsenzusa. Rad algoritma će biti ilustriran na jednostavnom primjeru. Prikazati će se dobiveni rezultati te dati osvrt na iste u kratkom zaključku.

2. Opis zadanog problema

Potrebno je ostvariti postupak kojim će se generirati sekvenca koja predstavlja konsenzus iz približnog rasporeda očitanja koji je dobiven nakon druge faze nekog programa koji vrši OLC sastavljanje. Konačni konsenzus uvelike ovisi o prve dvije faze OLC postupka. Postoji nekoliko različitih pristupa ovome problemu. Do točnog konsenzusa mogli bismo doći poravnavajući sva očitanja algoritmima dinamičkog programiranja, no problem s tim je složenost koja kod takvog postupka raste eksponencijalno s brojem očitanja. Postoji niz pristupa koji pokušavaju smanjiti spomenutu složenost ili riješiti problem na neki drugi način. Neki od tih pristupa su probabilistički - generiranje Bayesovog modela ili modela skrivenih Markovljevih lanaca koji će najvjerojatnije predstavljati početnu sekvencu iz koje su generirana očitanja. Značajni su progresivni postupci - jedan način rada je da u jednom koraku poravnaju 2 odabrana očitanja i iz njih generiraju jedan niz i time smanjuju broj očitanja koja je potrebno poravnati sve dok ne dođu do konačnog niza. Algoritam koji je implementiran u sklopu ovog projekta koristi iterativni pristup poboljšanja početnog poravnanja očitanja. Taj algoritam predstavili su Anson i Myers u [1].

2.1 Opis algoritma

Cilj implementiranog algoritma (*ReAligner*, kako su ga nazvali autori), je kroz niz iteracija poboljšati početno poravnanje nizova očitanja koje je dobiveno iz podataka o preklapanju istih. Pretpostavlja se da je početno poravnanje nizova temeljeno na izlazu *layout* faze OLC postupka globalno točno, ali nije lokalno optimalno. Također, algoritam se temelji na tvrdnji autora da podaci o preklapanju očitanja dobiveni iz prethodnih faza rijetko imaju više od 10% pogreške u estimaciji pozicije preklapanja između dva očitanja.

```
cctggt-acgta-cact-tgt
     tcacgtatccctctgttaga
         gta-ccctctgttagaaagctcacgt
                              ctcacttagttctctg-t
                               tcactt-gttctgtg-tag
                                    t-gttccgtgctagtagcta
-------------------------------------------------------
CCTGGTCACGTA-CCCTCTGTTAGAAAGCTCACTT-GTTCTGTG-TAGTAGCTA
```

Slika 1. Primjer početnog poravnanja te konsenzus dobiven iz tog poravnanja

ReAligner algoritam temeljen je na round-robin paradigmi. Sastoji se od serije iteracija unutar koje se ponavljaju koraci popravljanja početnog poravnanja. U određenom koraku ukupno višenizno poravnanje se particionira na 2 skupa. Ta dva sada odvojena skupa se međusobno poravnaju i zatim se spajaju ovisno o rezultatima poravnanja. Postoje 3 bitne stvari koje određuju ovaj algoritam, a to su način particioniranja skupova, metoda uspoređivanja skupova te konačno funkcija ocjene konsenzusa/konačnog poravnanja pomoću koje će se odrediti koje je optimalno međusobno poravnanje 2 skupa. Pripadnost baze konsenzusu određujemo na temelju "majority vote" tehnike po promatranom stupcu za određenu poziciju te na temelju kvalitete očitanja za svaku bazu ako su očitanja dana u FASTQ formatu.

Način particioniranja skupova

Naša postupak radi tako da početno poravnanje particioniramo tako da u svakom koraku odabiremo očitanje na *k*-tom mjestu, izdvajamo ga iz višeniznog poravnanja te njega poravnavamo s konsenzusom podporavnanja koje nastaje izdvajanjem spomenutog očitanja. Rezultat tog procesa bi bilo novo poravnanje iz kojeg nastaje konsenzus s boljom ocjenom. Recimo da je B podporavnanje poravnanja A. B je takav skup poravnatih očitanja koji se dobije uklanjanjem *k*-tog očitanja te zatim uklanjanjem svih stupaca u novom skupu poravnatih očitanja koji se sastoje samo od praznina ('-', *dash*) nakon uklanjanja *k*-tog očitanja.

Koraci algoritma

1. Ponavljaj
2. za neki *k* od 1 do ukupnog broja očitanja {
3. - izdvoji *k*-to očitanje iz početnog poravnanja A, dobijemo B <- (A bez *k*)
4. - poravnaj izdvojeno očitanje s konsenzusom podporavnanja B tako da je to poravnanje optimalno s obzirom na funkciju ocjene
5. - vrati *k*-to očitanje u skup poravnatih očitanja A s obzirom na rezultate prošlog koraka
6. }
7. sve dok se iznos funkcije ocjene konsenzusa novodobivenog poboljšanog skupa A **ne smanjuje** nakon izvođenja koraka 2. - 6.

Opisani koraci implementiranog algoritma uvelike se drže postupka opisanog u referenciranom radu [1] te uzimaju u obzir savjete i opisana rješenja za probleme za koje autori navode da su susreli u svojoj implementaciji. Koraci od 2. do 6. su koraci unutar jedne iteracije implementiranog algoritma. U svakoj implementaciji pamtimo najbolje moguće višenizno poravnanje (skup poravnatih očitanja), s obzirom na funkciju ocjene konsenzusa tog poravnanja, dobiveno kroz seriju od *k* poravnavanja na opisani način. Ukoliko se iznos funkcije ocjene nije smanjio nakon serije poravnavanja.

Bitno je naglasiti da u izgradnji konsenzusnog niza **koristimo podatke o kvaliteti očitanja ukoliko nam je to dostupno** (ulazna datoteka je FASTQ formata).

Funkcije ocjene

Standardna funkcija ocjene konsenzusa poravnanja A, koju u konačnici i pokušavamo minimizirati u postupku opisanom u prošlom poglavlju glasi:

$$\delta(a_1, a_2, \ldots, a_k) = \min_{x \in \Sigma \cup \{-\}} \; | \{a_i : a_i \neq x\}$$

gdje je $a_{1\text{-}n}$ baza u k-tom stupcu poravnanja, x znak konsenzusa koji promatramo, a on je iz skupa Σ koji se sastoji od oznaka baza A, C, G i T te unije tog skupa s prazninom $-$.

U implementaciji algoritma koristimo dvije različite funkcije ocjene poravnavanja izdvojenog očitanja i konsenzusa podporavnanja. Recimo da je *S* očitanje izdvojeno iz početnog poravnanja i da je *B* podporavnanje dobiveno izdvajanjem *S*. Konsenzus označimo s *c.* Bitno je napomenuti da se konsenzus sastoji od skupa baza koje se ovisno o poravnanju mogu nalaziti na toj poziciji u konačnom nizu, primjerice ukoliko u postupku određivanja konsenzusa odlučimo da C i G mogu biti na poziciji *i* u konačnom nizu onda je *c(i)* = [C, G]. Sa *x* označimo niz promatranih simbola iz niza *S*, a sa *X* niz svih skupova simbola konsenzusa poravnanja (u implementaciji klasa *Metasymbol*). S ~ označimo trenutno poravnanje između dvije particije. Prva funkcija ocjene poravnanja tada glasi:

$$\delta_c(x, X) = \; \delta_c(S \sim \mathrm{B}) = \; \begin{cases} 0 \; ako \; je \; x \in c(X) \\ 1 \; ina\check{c}e \end{cases}$$

Efektivno izlaz ove funkcije je izlaz uspoređivanja *S* s konsenzusom *B* po standardnoj *edit-distance* shemi. Iz toga dobivamo da je konačna funkcija δ za konsenzus poravnanja u koje smo vratili izdvojeni niz upravo:

$$\delta(A') = \delta(B) + \delta_c(S{\sim}\mathrm{B})$$

te autori predloženog algoritma u svom radu dokazuju da je iznos te funkcije uvijek manji ili jednak iznosu funkcije početnog poravnanja $\delta(A)$.

Druga funkcija ocjene predložena je jer ponekad koristeći samo prvu funkciju za *pairwise* poravnanje ne dobivamo optimalan iznos funkcije ocjene samog konačnog konsenzusa budući da ne uzimamo u obzir sve baze koje se nalaze u B u stupcu koji promatramo pri izračunu jednog simbola konsenzusa. Uzmemo $a_1, .., a_n$ kao simbole stupca koji promatramo, x kao simbol iz S te tada ta funkcija glasi:

$$\delta_a(x, (a_1, .., a_n)) = \begin{cases} \frac{\{a_i : a_i \neq x\}}{n} \; ako \; je \; n{>}0 \\ 0 \; ako \; je \; n{=}0 \end{cases}$$

Ona pak nema svojstvo koje je objašnjeno pri zbrajanju za dobivanje $\delta(A')$ te se konačnu u samoj implementaciji koristi funkcija za pairwise ocjenu:

$$\delta_{a+c} = 0.5\,\delta_a + 0.5\delta_c$$

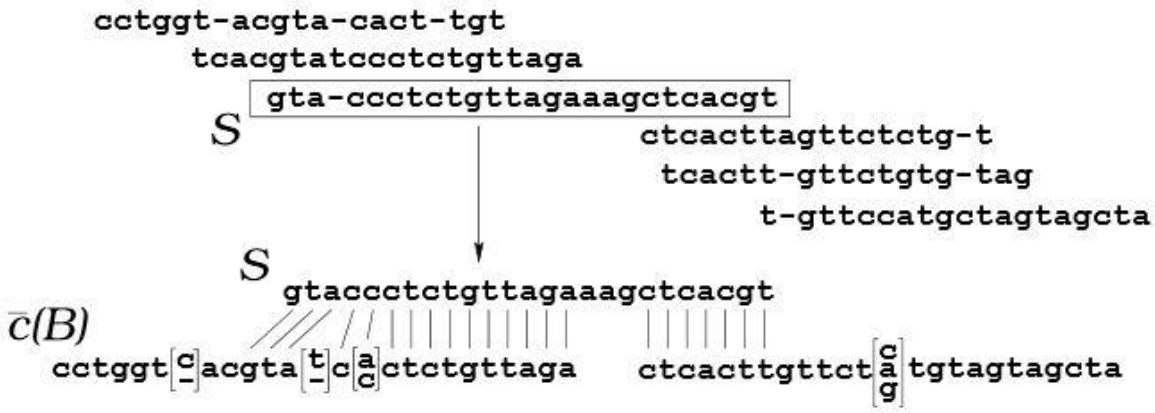
Ocjena konsenzusa novog poravnanja tada je:

$$\delta(A') = \delta(B) + \delta_{a+c}(S{\sim}\mathrm{B})$$

Ubrzanje postupka poravnavanja te poštivanje početnog rasporeda unutar granica ε

U algoritmu radimo samo lokalno popravljanje početnog poravnanja tako da izdvajamo pojedino očitanje i uspoređujemo ga s ostatkom. Pred nas je postavljen zahtjev da poštujemo raspored preklapanja očitanja dobiven iz *layout* faze. Efektivno taj uvjet ispunjavamo tako da pojasno ograničimo matricu udaljenosti između 2 niza koja poravnavamo u svakom koraku (za poravnavanje 2 niza koristimo Needleman - Wunsch algoritam) tako da taj pojas uzima uz obzir zadani $\boldsymbol{\varepsilon}$**.** Time postižemo memorijsko i vremensko ograničavanje na razine koje su prihvatljive i omogućavaju da se program dovoljno brzo izvodi.

2.2 Jednostavan primjer izvođenja postupka



Slika 2. Primjer rada jednog round-robin koraka unutar neke iteracije algoritma, preuzet iz [1]

Recimo da imamo očitanja 3 očitanja: ACCTG, ACTTG, ACT-T. Koristit ćemo samo prvu funkciju ocjene u prikazu. Početni layout izgleda primjerice ovako:

ACCTG
ACTTG
ACT-T

A[CA]CTTGT
Score = 4

Odabiremo prvo očitanje ACCTG i izdvojimo ga, konsenzus ostatka je A[CA][TC]TGT. Najbolje novo poravnanje po funkciji ocjene je:

ACCTG
ACTTG
ACT-T

ACCTTGT
Score = 3

Izdvajanjem drugog očitanja neće se mijenjati layout. Konačno izdvojimo posljednje očitanje i poravnamo s konsenzusom koji je AC[CT]TG i dobivamo:

ACCTG
ACTTG
ACT - T

ACCTTGT
Score = 3

I tako dok god može smanjivati Score kroz iteracije, što u ovom slučaju više ne može.

2.3 Ulazni podaci

Sada ulazimo u tehničke aspekte same programske implementacije i prikazujemo oblik ulaznih podataka koje prima naš program te na temelju kojih generira svoje rezultate. Za rad programa potrebni su rezultati dobiveni iz prijašnje 2 faze OLC postupka. Budući da radimo samo treći dio - konsenzus - morali smo te rezultate preuzeti iz nekog drugog OLC assemblera. Program koji smo koristili je Minimus, koji se nalazi u AMOS skupu programskih alata [2]. Minimus radi na temelju ulaznog FASTA niza očitanja. Nakon layout/unitig faze Minimus izbacuje rezultate u datoteku .afg formata kojeg definira i koristi AMOS. Prikaz te datoteke slijedi:

```
{LAY
{TLE
clr:0,338
off:0
src:1
}
{TLE
clr:0,1883
off:139
src:2
}
{TLE
clr:0,3250
off:396
src:3
}
{TLE
clr:0,604
off:450
src:4
}
```

U ovom prikazu ključna riječ `LAY` određuje početak novog rasporeda (layouta) očitanja za određeni contig. `TLE` je ključna riječ koja označava početak skupa podataka za pojedino očitanje. Indeks očitanja je predstavljen brojem koji dolazi nakon `src`, pomak očitanja od početne baze contiga prikazan je s `off` dok `clr` prikazuje početnu i krajnju iskoristivu bazu. Eventualno ako su brojevi u `clr` silazno posloženi onda je naše očitanje obrnuto u prikazu layouta.

Ulazna očitanja se pak predaju u FASTA ili FASTQ formatu. Ukoliko su generirana ReadSim programom, indeks očitanja parsiramo iz opisa, inače se oslanjamo da su očitanja posložena po redoslijedu.

Program prima sljedeće ulazne parametre: prvi je uvijek FASTA ili FASTQ datoteka s očitanjima, drugi je .afg datoteka s prikazanim layout formatom. Opcionalno programu se mogu postaviti 2 zastavice od kojih prva određuje parametar epsilon koji efektivno utječe na širinu pojasa pretraživanja u matrici dinamičkog programiranja te on predstavlja regulaciju da nam program pri popravljanju poravnanja prati početnu pretpostavku da su preklapanja definirana layoutom globalno ispravna s razinom greške do epsilon posto te parametar koji određuje gornji broj iteracija koje će program izvesti.

Primjer pozivanja programa:

```
java -jar realigner.jar test_2/readsInput2.fasta test_2/layouts2.afg -e0.1 -i10
```

2.4 Izlazni podaci

Program generira niz konsenzusa, po jedan za svaki LAY objekt prezentiran u ulaznoj .afg datoteci. Ti nizovi se ispisuju u datoteku *consensus.fasta* u FASTA formatu gdje se u redu označenom s > postavi src određenog LAY objekta te se u redovima ispod njega nalaze baze konsenzusa. Druga datoteka, *consensus_profile.txt*, u koju se također ispisuju rezutati je obična tekstualna i, iako je na prvi pogled i ona u FASTA formatu, ona za simbole baza konsenzusa koji su ostali neodređeni ispiše sve moguće kombinacije simbola za tu poziciju, primjerice ako na određenoj poziciji mogu doći baze A i C ona će ispisati [AC].

3. Prikaz rezultata

Eksperimentalne rezultate smo odlučili prikazati na dva načina. Prvi od njih koristi skripte za ocjenjivanje izlaznih FASTA datoteka koje su dostupne na [3]. Usporedili smo izlaz našeg programa i izlaz Minimusa/AMOS-a za iste ulazne podatke te dobili statističku analizu koja će biti prikazana u nastavku. Za drugu analizu smo odlučili vidjeti koliko je naš postupak popravljanja početnog poravnanja poboljšao odlučivost u fazi izgradnje konsenzusa. Naime, ukoliko ne možemo razlučiti koja baza se nalazi na određenoj poziciji mi postavimo simbol *n* za tu poziciju. Drugi postupak analize efektivno se svodi na prebrajanje *n* simbola u konsenzusu prije i poslije izvođenja ReAligner algoritma.

Dodatno bi bilo jako dobro kada bi se rezultati mogli usporediti s izlaznim rezultatima AMOS-a i samim referentnim genomom na temelju sličnosti, ali nažalost nismo našli pogodnu metodu kojom bismo izveli takvu analizu budući da usporedba s AMOS-om ne govori koliko smo dobri s obzirom na referentni genom, jer i Minimus ima određenu grešku u svom radu, a s druge strane usporedbu s referentnim genomom je nemoguće izvesti budući da je naš izlaz razlomljen u contige koji zahtjevaju daljnje faze obrade da bi se došlo do niza koji se može uspoređivati s referentim nizom.

Konačno dati ćemo osvrt na memorijske performanse te brzinu izvođenja programa.

Statistička analiza i usporedba s Minimusom

Provodimo je na testnim primjerima generiranim na podnizu (ili cijelom nizu) koji predstavlja genom E. coli te su očitanja generirana pomoću programa ReadSim iz referentnog niza. Svi testni primjeri, osim jednog za kojeg su očitanja dostupna na stranici [5], dostupni su u repozitoriju gdje se nalazi i implementacija programa [4]. Testovi su provedeni na Minimusovim izlazima za primjere u folderu test_1, test_2 i test_3 na repozitoriju. Slijedi ispis statističkih podataka dobiven iz [3], skripta analyzerCore.py:

Prvi testni primjer - test_1 folder

Realigner

```
[AN:] Getting basic statistics for contigs...
[AN:] Total number of contigs:  1
[AN:] Number of contigs bigger then size25000B: 1
[AN:] Total length of all contigs:  34798
[AN:] Maximum contig size:  34798
```

```
[AN:] Minimum contig size:  34798
[AN:] Average contig length:  34798.0
[AN:] Median contig length:  34798
[AN:] E-size with threshold 0:  34798.0
[AN:] N25 measure:  34798
[AN:] N50 measure:  34798
[AN:] N56 measure:  34798
[AN:] N75 measure:  34798
[AN:] Ratio between median and E-size:  1.0
[AN:] Ratio between n50 and E-size:  1.0
[AN:] Ratio between contig and read length:  463.973333333
[1, 1, 34798, 34798, 34798, 34798.0, 34798, 34798.0, 34798, 34798, 34798, 34798, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Minimus

```
[AN:] Total number of contigs:  1
[AN:] Number of contigs bigger then size25000B: 1
[AN:] Total length of all contigs:  34798
[AN:] Maximum contig size:  34798
[AN:] Minimum contig size:  34798
[AN:] Average contig length:  34798.0
[AN:] Median contig length:  34798
[AN:] E-size with threshold 0:  34798.0
[AN:] N25 measure:  34798
[AN:] N50 measure:  34798
[AN:] N56 measure:  34798
[AN:] N75 measure:  34798
[AN:] Ratio between median and E-size:  1.0
[AN:] Ratio between n50 and E-size:  1.0
[AN:] Ratio between contig and read length:  463.973333333
[1, 1, 34798, 34798, 34798, 34798.0, 34798, 34798.0, 34798, 34798, 34798, 34798, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Drugi testni primjer - test 2 folder

Realigner

```
[AN:] Total number of contigs:  6
[AN:] Number of contigs bigger then size25000B: 0
[AN:] Total length of all contigs:  9864
[AN:] Maximum contig size:  3502
[AN:] Minimum contig size:  147
[AN:] Average contig length:  1644.0
[AN:] Median contig length:  1562
[AN:] E-size with threshold 0:  2649.09671533
[AN:] N25 measure:  3502
[AN:] N50 measure:  2685
[AN:] N56 measure:  2685
[AN:] N75 measure:  2454
[AN:] Ratio between median and E-size:  1.69596460648
[AN:] Ratio between n50 and E-size:  0.986628199377
[AN:] Ratio between contig and read length:  21.92
[6, 0, 9864, 3502, 147, 1644.0, 1562, 2649.0967153284673, 3502, 2685, 2685, 2454, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Minimus

```
[AN:] Total number of contigs:  6
[AN:] Number of contigs bigger then size25000B: 0
[AN:] Total length of all contigs:  9864
[AN:] Maximum contig size:  3492
[AN:] Minimum contig size:  147
[AN:] Average contig length:  1644.0
```

```
[AN:] Median contig length:  1563
[AN:] E-size with threshold 0:  2644.22283049
[AN:] N25 measure:  3492
[AN:] N50 measure:  2690
[AN:] N56 measure:  2690
[AN:] N75 measure:  2451
[AN:] Ratio between median and E-size:  1.69176124792
[AN:] Ratio between n50 and E-size:  0.982982464868
[AN:] Ratio between contig and read length:  21.92
[6, 0, 9864, 3492, 147, 1644.0, 1563, 2644.2228304947284, 3492, 2690, 2690, 2451, -
1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Treći testni primjer - test_3 folder

Realigner

```
[AN:] Total number of contigs:  2
[AN:] Number of contigs bigger then size25000B: 0
[AN:] Total length of all contigs:  251
[AN:] Maximum contig size:  161
[AN:] Minimum contig size:  90
[AN:] Average contig length:  125.5
[AN:] Median contig length:  125
[AN:] E-size with threshold 0:  135.541832669
[AN:] N25 measure:  161
[AN:] N50 measure:  161
[AN:] N56 measure:  161
[AN:] N75 measure:  90
[AN:] Ratio between median and E-size:  1.08433466135
[AN:] Ratio between n50 and E-size:  0.841874737077
[AN:] Ratio between contig and read length:  1.67333333333
[2, 0, 251, 161, 90, 125.5, 125, 135.5418326693227, 161, 161, 161, 90, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Minimus

```
[AN:] Getting basic statistics for contigs...
[AN:] Total number of contigs:  2
[AN:] Number of contigs bigger then size25000B: 0
[AN:] Total length of all contigs:  251
[AN:] Maximum contig size:  161
[AN:] Minimum contig size:  90
[AN:] Average contig length:  125.5
[AN:] Median contig length:  125
[AN:] E-size with threshold 0:  135.541832669
[AN:] N25 measure:  161
[AN:] N50 measure:  161
[AN:] N56 measure:  161
[AN:] N75 measure:  90
[AN:] Ratio between median and E-size:  1.08433466135
[AN:] Ratio between n50 and E-size:  0.841874737077
[AN:] Ratio between contig and read length:  1.67333333333
[2, 0, 251, 161, 90, 125.5, 125, 135.5418326693227, 161, 161, 161, 90, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1]
```

Dodatno pojašnjenje ovih podataka pogledati na [3]. Isto tako moramo naglasiti da, iako po statistici dajemo slične podatke, se Minimus ipak čini uspješniji u kreiranju konsenzusa na temelju odlučivosti pri odabiru baze koja ulazi u konsenzus u pojedinoj poziciji. Više o tome u idućem poglavlju.

Analiza odlučivosti pri biranju baze konsenzusa

Eksperimenti su provedeni na 3 testna primjera dostupna u repozitoriju [4]. Na svakom od njih provedeno je izračunavanje konsenzusa prije i poslije izvođenja ReAlignera. Provođenje ReAlignera, čak i kroz malen broj iteracija, daje gotovo uvijek barem 50 posto manje pozicija koje su označene simbolom n (one za koje ne možemo odlučiti koja je prava baza). Taj rezultat se višestruko poboljšava ukoliko uz očitanja imamo i podatke o kvaliteti očitanja za pojedinu bazu tako da se broj označenih s *n* smanji na prilično zanemariv broj za dobar početni layout. Očitanja s kvalitetom rađena su na podskupu očitanja iz [5]. Slijedi primjer jednog kraćeg konsenzusa bez i sa popravljanjem pomoću ReAlignera gdje vidimo odnos broja pozicija označenih s *n*:

```
>2_BEZ_REALIGNERA
CGTGAAACGGGACGTGAACTGGAGCTGGCGGATATTGAAATTGAACCTGTGCTGCCCGCA
GAGnTTnAnnnCnnnGGnnnnTnnnnnCnnTTnTnnnGnnnnTnTnnCAnnnCnnnnnnn
TCTnTnnnnCGCGCGnGnnGnnnnCCCGTGATGAAGGAAAAGTTTTGCGCTATGTTGGCA
ATATTGATGAAGATGGCGTCTGCCGCGTGAAGATTGCCGAAGTGGATGGTAATGATCCGC
TGTTCAAAGTGAAAAATGGCGAAAACGCCCTGGCCTTCTATAGCCACTATTATCAnCCGn
CCGCnGnTGnTGCGCGGAnATGGTGnGGGnAAnGACGTTnCAGCTGCnnGGGTnTTTTnn
nnnTTnnnnnnnnnCCCTCTCATGGAAGTTAGGnnnnnAnnnTGnTAAAnTTTnnnCCCC
CGnCTAnTGCCAnTATGAGCGTCGGGTTTGATGTGCTCGGGGCGGCGGTGACACCTGTTG
ATGGTGnnnnnnnCnnnGAnnTnnnnnCnnnnnTGAGGCGGCAGAGACATTCAGTCnnAn
nAnCnnnGnnnnnTTGCCGATAAGCTGCCGTCAGAACCACGGGAAAATATCGTTTATCAG
TGCTGGGAGCGTTTTTGnCAGGAACTGGGTAAGCAAATTCCAGTGGCGATGACCCTGGAA
AAGAATATGCC
>2_NAKON_REALIGNERA
CGTGAAACGGGACGTGAACTGGAGCTGGCGGATATTGAAATTGAACCTGTGCTGCCCGCA
GAGCTTTAACGCCGAGGGTGATGTTGCCGCTTTTATGGCGAATCTGTCACAACTCGACGA
TCTCTTTGCCGCGCGCGTGGCGAAGGCCCGTGATGAAGGAAAAGTTTTGCGCTATGTTGG
CAATATTGATGAAGGTGGCGTCTGCCGCGTGAnTGCCnnnGnnGAnGnnnATGnTCCnnT
nTCCAnTGAnAAAAAGnGAAAAAnCGCnnAnnCnCTCTAGCCACTCTTATCAGCCGCTGC
nGTTGGTACnGnGCGGATATGGTGCGGGCAATGACGTTACAGCTGCCGGTGTCTTTGCTG
ATCTGCTACGTACCCCTnnnnnGAAGTTAGGAGTCTGACATGGTTAAAGTTTATGCCCCG
GCTTCCAGTGCCAATATGAGCGTCGGGTTTGATGTGCTCGGGGCGGCGGTGACACCTGTT
GATGGTGCATTGCTCGGAGATGTAGTCACGGTTGAGGCGGCAGAGACATTCAGTCTnACA
ACnnCGnnCGCnnnGCCGATAAGCTGCCGTCAGAACCACGGGAAAATATCGTTTATCAGT
GCTGGGAGCGTTTTTGnCAGGAACTGGGTAAGCAAATTCCAGTGGCGATGACCCTGGAAA
AGAATATGCC
```

Analiza performansi

Ocjena memorijske složenosti programa ovisi o koraku poravnavanja dva niza Needleman-Wunsch algoritmom. Budući da koristimo pojasno pretraživanje matrice, ako s L označimo dužinu najvećeg niza i sa eps označimo mogućnost pogreške izraženu u broju simbola niza, ta memorijska složenost je ograničena na matricu cjelobrojnih vrijednosti u složenosti *O(L x (L+2eps))*. Jedina druga stvar koja bitno utječe na potrošnju memorije je veličina ulaznih datoteka. Naime, u našem postupku zbog brzine i efikasnosti sva očitanja, i kvalitete istih ako su prisutne, moramo držati u radnoj memoriji u obliku raznih struktura podataka koje u konačnici zauzmu otprilike 1.5 veličine ulazne datoteke te nam se gornja ocjena memorijske složenosti može uvećati za pribrojnik M koji predstavlja veličinu ulaznih datoteka predstavljenih strukturama koje koristi program. Dobivamo *O(M + L * (L+2eps))*. Kroz postupke profiliranja programa najveća je memorijska zauzetost bila od 1.2 GB pri izračunu konsenzusa za primjer iz [5]. Moramo napomenuti da su ulazne datoteke težile više od 900mb zajedno.

Što se brzine izvođenja tiče ona ovisi o više faktora. Algoritam je iterativan, u svakoj iteraciji se izvodi *k* poravnavanja (k je broj ulaznih očitanja) čije trajanje ovisi o prosječnoj duljini nizova koji se poravnavaju - označimo tu ovisnost sa *l*; za svaki od *n* contiga/layouta se provodi maksimalno *i* iteracija. Približna ocjena vremenske složenosti daje *O(n * i * k * l)*. Program inače u vrlo prihvatljivom (rekli bismo brzo) vremenu radi za manje ulazne datoteke s kraćim očitanjima, dok je za obrađivanje najvećeg testnog primjera bilo potrebno oko 15 minuta budući da nam je Minimus u layout fazi izgenerirao 54000 contiga koje je trebalo poravnati. Taj broj je još umanjen s obzirom na to da je Minimus jako loše izveo prve dvije faze te su podaci za našu fazu bili dosta netočni i skraćeni, što primjećujemo iz toga da nisu uzeta u obzir velika većina očitanja te iz velikog broja contiga/layouta.

4. Zaključak

OLC postupak sastavljanja genoma je jedan veoma zanimljiv i poticajan problem, kako iz perspektive biologa tako i programera. Implementirali smo program koji na temelju prve dvije faze daje konsenzus u OLC postupku. Susreli smo se s drugim programima koji se bave istim problemom te koriste različite pristupe rješavanju istoga. Uočili smo njihove dobre i loše strane te uvidjeli kompleksnost problema koji se rješava. Vjerujemo da su dobiveni rezultati nesavršeni te da se daljnjim budućim radom mogu poboljšati. Neki od prijedloga za budući rad bi bili revidiranje funkcije ocjene te same njene implementacije te osmišljavanje i implementacija boljeg načina razlučivanja (ukomponirati probabilistički model?) baze koja će se pojaviti u samom konsenzusu. Također potrebno bi bilo bolje analizirati same izlazne podatke uz pomoć referentnog genoma te na taj način doći do boljeg razumijevanja problema s kojima se susreće naš ostvareni algoritam.

5. Literatura i reference

[1] *ReAligner: A Program for Refining DNA Sequence Multi-Alignments*
Eric L. Anson i Eugene W. Mzers. Journal of Computational Biology.
Fall 1997, 4(3): 369-383.

[2] *Minimus: a fast, lightweight genome assembler*
Daniel D Sommer**,** Arthur L Delcher**,** Steven L Salzberg i Mihai Pop
Implementacija: http://sourceforge.net/projects/readsim/files/latest/download

[3] https://github.com/vzupanovic/skripte

[4] https://github.com/IgorLipovac/bioinfo-consensus

[5] http://download.clcbio.com/testdata/paeruginosa-reads.zip