SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT IZ BIOINFORMATIKE

Layout faza OLC (Overlap Layout Consensus) sastavljanja genoma

Nino Rašić
Bojana Savić
Josip Vinković

Voditelj: Mile Šikić

Zagreb, siječanj, 2016

SADRŽAJ

UVOD

U molekulama DNA je pohranjena sva informacija potrebna za izgradnju i funkcioniranje nekog organizma. Iz tog razloga je jako bitno znati točan nukleotidni slijed od kojeg je sastavljena neka DNA sekvenca. Ovo je, uz trenutnu tehnologiju, dosta veliki problem. Naime, DNA molekule su jako velike. Čak i vrlo jednostavni organizmi, kao što je npr. bakterija *Escherichia Coli*, imaju DNA sekvence duge više od 4 milijuna baznih parova, dok se kod složenijih organizama, kao što je npr. čovjek, veličina mjeri u milijardama baznih parova. Uz trenutnu tehnologiju, moguće je u jednom komadu odrediti sekvencu bitno manjim fragmentima (red veličine je 10^3). Zbog ovoga se originalni genom mora razbiti na puno malih fragmenata koji se onda sekvenciraju, te se onda originalni DNA slijed mora na neki način rekonstruirati iz sljedova fragmenata. Ovo je proces koji se zove "sastavljanje genoma" (Genome assembly) **(3)**. Danas najpopularniji pristup sastavljanju genoma bazira se na teoriji grafova, i u ovom radu je konstruiran algoritam koji na tom principu pokušava sastaviti ove male fragmente u originalni DNA slijed.

Danas najpopularniji način generiranja malih DNA fragmenata u svrhu određivanja originalnog genoma (originalnog DNA slijeda) zove se *whole genome shotgun* (WGS) sekvenciranje **(3)**. U ovom pristupu se prvo veći broj cijelih genomskih DNA razbije na puno malih fragmenata koji se onda individualno sekvenciraju. Iz ovih sekvenci se onda pokuša odrediti originalna sekvenca. Medjutim, u ovom procesu sastavljanja genoma javljaju se brojne poteškoće, kao što je npr. gubitak nekih fragmenata tijekom cijelog procesa što dovodi do rupa (*gaps*), dolazi do grešaka u samom sekvenciranju fragmenata, te postojanje fragmenata u kojima se nalaze ponavljajući nukleotidi (npr fragmenti koji sadrže dugačke (AT)* ponavljajuće parove, što je česta pojava kod eukariotskih genoma). Za sve navedene probleme, potrebno je onda osmisliti algoritam kako ih riješiti. Tako se npr. problem grešaka u sekvenciranju genoma rješava algoritmom generiranja konsenzusa, problem gubitka fragmenta rješava se upotrebom velikog broja kopija originalnih DNA sljedova, čime se smanjuje vjerojatnost nastajanja rupa, dok se visoko ponavljajuće sekvence mogu detektirati i sortirati iz veće pokrivenosti ovih dijelova fragmentima od ostalih segmenata originalne genomske DNA **(1)**.

Postoji više algoritama kojima se provodi sastavljanje genoma. Ovdje će biti dan kratki osvrt na "Overlap Layout Consensus" (OLC) metodu zato što je predmet ovog rada upravo ovaj Layout dio navedenog algoritma.

OVERLAP LAYOUT CONSENSUS METODA SASTAVLJANJA GENOMA

Ovo je popularna metoda za *de novo* (dakle kad ne postoji neki referentni genom) sastavljanje genoma. Sastoji se od 3 faze: **Overlap**, **Layout** i **Consensus**. Ovjdje će biti dan samo kratki osvrt na Overlap i Consensus fazu, dok će Layout faza biti bolje obrađena pošto je ona predmet ovog rada.

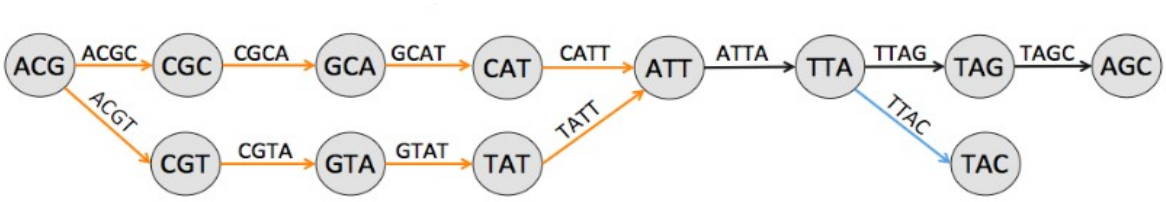
OVERLAP FAZA

Ovo je faza u kojoj se generira inicijalni graf od svih fragmenata nastalih raspadom originalne genomske DNA. Algoritam koji provodi overlap fazu u biti uzima fragment po fragment te ga uspoređuje sa svim drugim fragmentima te se pokušava naći mjesto preklapanja. S time da se prilikom preklapanja uspoređivanje vrši i s originalnim sljedovima i s njihovim komplementima pošto iz samog određivanja DNA slijeda u ovakvim slučajevima ne možemo biti sigurni koji od komplementarnih DNA lanaca ih je stvorio. Već na prvi pogled se da vidjeti da je ovo problem kvadratne složenosti, što, uzevši u obzir veličinu genomskih DNA u odnosu na veličinu fragmenta, može vremenski biti jako zahtjevno čak i za najbrža današnja računala ako se koristi neka od varijanata Needelman Wunschovog algoritma za poravnavanje sljedova. Iz ovog razloga su osmišljeni algoritmi za jako brzu usporedbu sljedova. Jedan od takvih algoritama se danas npr. koristi u NCBI-ijevom BLAST-u. Ovakvim algoritmima je moguće izvršiti milijune usporedaba fragmenata u jednoj sekundi **(3)**.

Nakon što se detektiraju sva preklapanja među fragmentima, konstruira se inicijalni graf. U tom grafu fragmenti predstavljaju vrhove grafa. Dva vrha se povežu bridom ako između odgovarajučih fragmenata postoji preklapanje.

LAYOUT FAZA

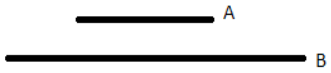
U ovoj fazi se zapravo pokušava konstruirati originalni genom od fragmenata. Rezultat ove faze bio bi, prema tome, niz vrhova čijim se sljednim spajanjem sekvenci (obračajući pritom pažnju na preklapanja) dobiva originalna genomska sekvenca. Ovo je, gledano kroz teoriju grafova, problem nalaženja najdužeg puta kroz graf. Ovo je NP-kompletan problem zato što treba rekurzivno proći kroz sve vrhove, svim mogućim putevima da se odredi koji je najdulji. Pošto se ovdje radi o jako velikim grafovima, ovo često puta nije praktički prikladno odrediti. Zato se nad inicijalnim grafom, dobivenim iz overlap faze, radi niz transformacija koje na neki način provode kompresiju grafa tako da se dobije graf prikladne veličine iz kojega se onda može puno lakše detektirati najduži put **(3)**. Postoje tri glavna načina na koje se vrši kompresija: izbacivanje stringova koji su sadržani u nekom drugom stringu, uklanjanje tranzitivnih bridova i formiranje "contiga" **(1)**. Osim navedene tri transformacije grafa, mogu se provesti još neke dodatne transformacije kao što je npr. odredjivanje mjesta alternativnih puteva (slika 1) te izbor jednog od njih kao osnovnog, uklanjanje vrhova koji "strše" i sl. Algoritam napisan za ovaj rad provodi samo ove tri osnovne transformacije grafa pa će onda samo one stoga biti detaljnije opisane.



Slika 1. Graf prikazuje alternativne puteve (narančasto) i stršeći vrh (plavo). preuzeto sa **(3)**

1) Izbacivanje stringova koji su sadržani u nekom drugom stringu

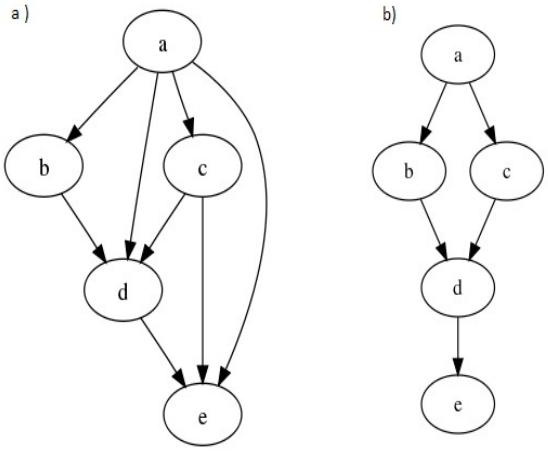
Srž ovog koraka lijepo je prikazana na slici 2. Dakle, ako se neki fragment A preklapa sa drugim fragmentom B na način da je duljina preklapanja jednaka duljini fragmenta A, onda se kaže da je fragment A sadržan u fragmentu B, i vrh koji predstavlja fragment A (i naravno svi bridovi incidentni sa njime) se može pobrisati iz grafa. Ovo rezultira brisanjem jako velikog broja inicijalnih vrhova zato jer je većina fragmenata (pogotovo onih malih) sadržana u nekom drugom (većem) fragmentu.



Slika 2. String A je sadržan u stringu B

2) Uklanjanje tranzitivnih bridova

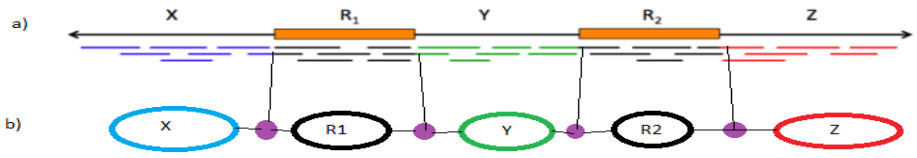
Primjer tranzitivnog brida prikazan je na slici 3. Dakle, gledano samo kroz prizmu grafova, ako imamo tri vrha A, B i C, i ako postoje veze izmedju vrhova A i B, B i C i A i C, onda je brid izmedju A i C tranzitivan i može se ukloniti iz grafa. Dakle, čak i s uklonjenim bridom može se doći do vrha C ako se polazi iz vrha A. Ovo uklanjanje tranzitivnih bridova se jako lijepo uklapa u ovu priču sa sastavljanjem genoma iz fragmenta. Naime može se desiti situacija da postoji preklapanje izmedju vrhova A i C iako njihove sekvence u originalnoj genomskoj DNA nisu susjedne, i medju ovim fragmentima postoji razmak. Fragment B bi u ovom slučaju predstavljao taj razmak i onda je uklanjanje ovog brida logično pošto se ne radi o pravim susjedima. Ipak, postoje još neki dodatni kriteriji za definitivno proglašenje nekog brida tranzitivnim u biološkom smislu osim ovog prethodno navedenog. Jedan od glavnih dodatnih kriterija dan je na duljinu bridova. Obično se uzima da je brid A-C tranzitivan ako mu je duljina veća od zbroja duljina bridova A-B i B-C. Još se dodatno može gledati i orijentacija bridova (ovisno o radu overlap algoritma) **(1)**.



Slika 3. originalni graf (a), isti graf nakon uklanjanja tranzitivnih bridova a-e i a-d (b).

3) Formiranje "contiga"

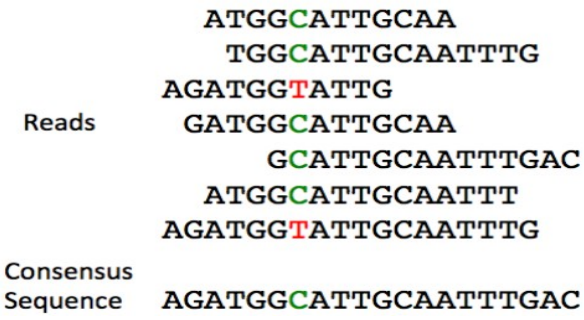
Contig čini kolekcija fragmenata koji se jasno međusobno preklapaju i referenciraju se na isti kontinuirani slijed u originalnoj genomskoj DNA. Contig, gledano sa stajališta grafa, predstavlja podgraf unutar kojega su članovi međusobno povezani velikim brojem veza, a koji je sa ostatkom grafa povezan samo preko dva čvora koji se onda nazivaju "početak" i "kraj" contig-a. Početak i kraj contiga je povezan preko vrha koji se zove "čvor" sa drugim contizima u grafu (slika 4). Svi vrhovi koji pripadaju istom contigu se mogu komprimirati u jedan vrh i na taj način se originalni graf još dodatno jako smanji i dovede u veličinu pogodnu za provedbu nekog računalno zahtjevnog algoritma za određivanje najdužeg puta **(3)**.



Slika 4. Prikazuje fragmente koji čine contige. Svaki contig označen je drugom bojom (a) Preuzeto sa **(3)**. Na slici b prikazani su contizi medjusobno povezani sa fragmentima čvorištima (prikazani ljubičastom bojom).

CONSENSUS FAZA

Nakon layout faze, počevši od prvog contiga, uzimaju se vrhovi i fragmenti se poravnaju prema dobivenom grafu. Ovo rezultira time da se svi fragmenti koji se preklapaju medjusobno poravnaju te se onda za svako mjesto unutar nekog poravnanja može odrediti koja se baza najčešće pojavljuje. Za onu koja je najčešća se uzima da je "consensus" baza i upisuje se na odgovarajuće mjesto poravnanja (slika 5). Nakon ovog koraka, kao rezultat se dobije rekonstruirana genomska DNA **(3)**.



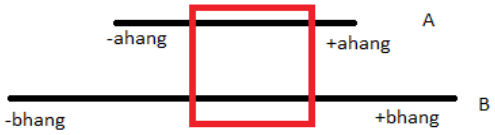
Slika 5. Poravnavanje fragmenata i formiranje konsenzus sekvence. Preuzeto sa **(3)**.

IMPLEMENTACIJA LAYOUT FAZE OLC-a

Kao što je prethodno opisano, layout faza OLC-a se sastoji od 3 glavna dijela - izbacivanje stringova koji su sadržani u nekom drugom stringu, uklanjanje tranzitivnih bridova i formiranje "contiga". Ulaz u program su 2 datoteke. Jedna datoteka opisuje sva moguća preklapanja medju vrhovima i zapisana je u .mhap formatu (5). U ovoj datoteci se nalaze sve potrebne informacije o preklapanju: duljina i identiteti oba fragmenta, mjesto početka i kraja preklapanja na prvom fragmentu, mjesto početka i kraja preklapanja na drugom fragmentu, te za oba fragmenta da li se preklapaju originalni fragmenti ili njihovi komplementi i informacije o kvaliteti preklapanja. U drugoj datoteci se nalazi lista sekvenci fragmenata i njihovih identiteta. Nakon učitavanja ovih datoteka u odgovarajuće strukture tipa unordered_map u c++-u kojima se zapravo formira inicijalni graf, pokreće se prva faza layout algoritma.

1) Izbacivanje stringova koji su sadržani u nekom drugom stringu

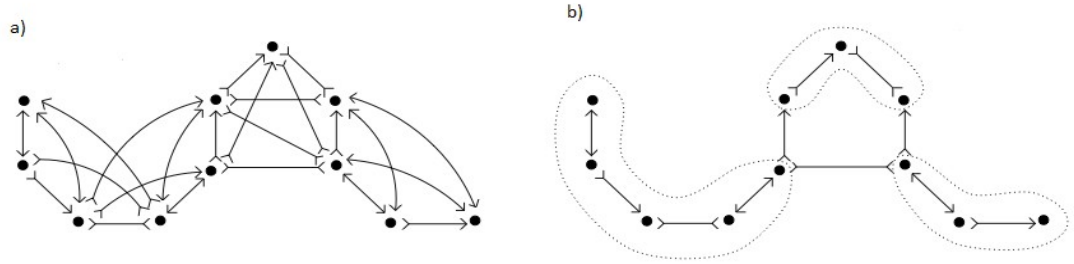
Pošto mjesta preklapanja nisu "potpuna", to jest i s jedne i s druge strane mjesta preklopa postoje "slobodne" baze (slika 6), mora se malo modificirati mehanizam detekcije preklapajućih fragmenata od prethodno izloženoga. Implementirani mehanizam to radi na sljedeći način: imamo dva fragmenta A i B. Prvo se odredi duljina slobodnog dijela fragmenta A sa lijeve strane mjesta preklapanja (- ahang), pa onda duljina slobodnog dijela fragmenta A sa desne strane mjesta preklapanja (+ ahang). Potom se učini isto i za fragment B pa se, analogno, dobije -bhang i + bhang. Ako je - ahang > - bhang && + ahang > + bhang onda je fragment B sadržan u fragmentu A. Ako vrijedi -ahang < - bhang && + ahang < +bhang onda je fragment A sadržan u fragmentu B. Ova provjera se radi za sve vrhove i njihove susjede u inicijalnom grafu. Jednom kad se svi sadržani fragmenti odrede, onda se oni, i svi njihovi bridovi izbace iz originalnog grafa.



Slika 6. Dva fragmenta od kojih je fragment A sadržan u fragmentu B. Ahang i bhang označavaju slobodne krajeve fragmenata A i B.

2) Uklanjanje tranzitivnih bridova

Uklanjanje tranzitivnih bridova provedeno je algoritmom koji je opisan u (3). Algoritam se u biti sastoji od toga da se ide po svim vrhovima, gledaju se njegovi susjedi i susjedi tih susjeda i ako postoji tranzitivna veza, ona se označi. Uklanjanje tranzitivnih bridova se provodi tek nakon što se prođe kroz sve vrhove i označe svi tranzitivni bridovi. Ovo se mora napraviti na ovakav način zato što neki brid B može biti tranzitivan preko nekog drugog tranzitivnog brida A, pa ako se ukloni brid A prije negoli se provede provjera nad bridom B, brid B će ispasti lažno netranzitivan. Kako to izgleda na jednom jednostavnom primjeru prikazano je na slici 7.



Slika 7. Graf prije uklanjanja tranzitivnih bridova (a). Isti graf nakon uklanjanja tranzitivnih bridova (b). Preuzeto sa (4)

3) Formiranje contiga

Ovaj dio programa se može podijeliti u 3 etape od kojih su prve dvije tijesno povezane: pronalaženje vrhova čvorova, razvrstavanje vrhova u contige, te generiranje sekvence contiga.

Prvo se pronadju svi potencijalni čvorovi. To se obavlja na sljedeći način: prolazi se kroz sve vrhove, i za svaki vrh se uspoređuju mjesta preklapanja na testiranom vrhu svih parova njegovih susjeda. Ako se mjesta preklapanja podudaraju ili ako su u ispitivanom paru oba člana orijentirana svojim slobodnim krajem na istu stranu, onda taj dotični par susjeda pripada istom contigu. Formiraju se dvije liste: jedna za lijevi i druga za desni contig. Ako je za par susjeda detektirano da pripadaju istom contigu, a slobodni krajevi im nisu orijentirani na istu stranu, onda testirani vrh sigurno nije čvor i algoritam detekcije prelazi na testiranje idućeg vrha. Inače se par vrhova stavlja u odgovarajuću listu ovisno o orijentaciji svojeg slobodnog kraja. Ako se na kraju te dvije liste razlikuju onda je testirani vrh čvor, inače nije.

Nakon što se detektiraju svi potencijalni čvorovi, provede se algoritam grupiranja fragmenata u contige. On se provodi na sljedeći način: izabere se neki vrh koji nije čvor i stavi se u novi contig. Zatim se stavljaju rekurzivno svi njegovi susjedi ako nisu označeni kao čvorovi i to se vrti sve dok ima vrhova koji se mogu priključiti contigu. Kad contig više ne može rasti, formira se novi contig i od preostalih vrhova se ponovi cijeli postupak. Ovo se vrti sve dok ima nesortiranih vrhova.

Nakon što se formiraju svi contizi, treba ukloniti lažne čvorove. Naime, odluka kojom se neki vrh označio kao čvor, donesena je samo iz lokalne perspektive tog vrha (slika 8.). Njegovi susjedi mogu biti povezani u isti contig preko nekog drugog vrha koji nije susjed testiranom čvoru zato jer je možda brid koji ih je originalno povezivao uklonjen zbog tranzitivnosti. Medjutim, ovo ne predstavlja veliki problem, jer su takvi lažni čvorovi susjedni samo jednom contigu dok su stvarni čvorovi susjedi sa bar dva contiga. Pa ako za neki čvor ustanovimo da je susjed samo jednom contigu, onda ga samo naknadno pridružimo svim ostalim njegovim konstituentima.

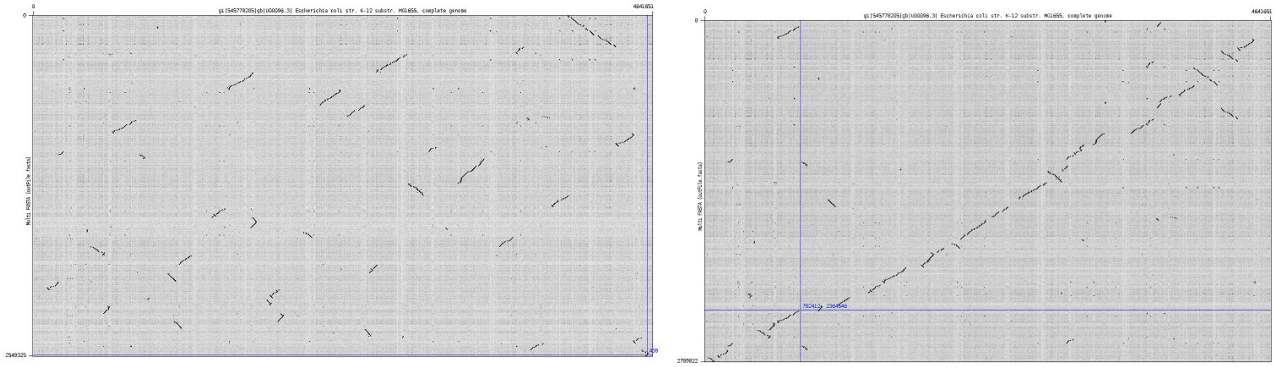
Slika 8. Na slici a) prikazan je pravi fragment čvor (crno). Fragmenti koji pripadaju različitim contizima označeni su različitim bojama. Slika b) prikazuje lažni čvor. Fragment označen ljubičasto nije susjed lažnom čvoru (crno), pa gledano iz perspektive samo lažnog čvora on izgleda kao pravi.

Nakon što se formiraju contizi kreće određivanje njegove sekvence. Ovdje nastane problem pošto nisu uklonjene sve nejasnoće. Najveći problem predstavljaju prethodno spomenuta mjesta alternativnih puteva i "stršeće" sekvence. Zbog ovoga, čisti algoritam dodavanja fragmenata sljedno po principu "dodaj fragment, produži sve njegove susjede vodeći računa o mjestima preklapanja i ponavljaj dok možeš" ne daje dobre rezultate. Zato je osmišljen jedan malo modificirani algoritam koji počinje od nekog vrha, i dodaje samo jednog susjeda na kraj i onda to rekurzivno ponavlja dok je moguće pronaći susjeda koji nije već sortiran. Kad više ne može rasti, uzme idući nesortirani fragment i ponavlja cijeli postupak dok se svi fragmenti ne sortiraju u ovakve nakupine koje se zovu "Chunk"-ovi. Ovo dakle nije idealan postupak, ali se ipak pokazalo da na kraju daje smislene rezultate i formira se manji broj velikih "Chunkova" od kojih se onda može rekonstruirati originalni contig (i na kraju cijeli genom).

REZULTATI

Test na ispravnost sekvenci chunkova

Rad programa testiran je na ukupnom genomu *Escherichie Coli* koji se sastoji od više od 4 milijuna baza. Ulazni graf imao je više od 16 tisuća vrhova (fragmenata) i preko 700k bridova. Na kraju je program generirao oko 300 Chunk-ova od kojih ih je 20-tak bilo velikih i oni su pokrivali oko 3 milijuna baza referentnog *E. Coli* genoma. Usporedba je izvršena programom Gepard (6). Rezultati su prikazani na slici 9.



Slika 9. Lijevo - prvih 30 najvećih Chunk-ova koje je generirao algoritam. Prikazani u random rasporedu. Desno - 25 ručno izabranih i poredanih Chunkova.

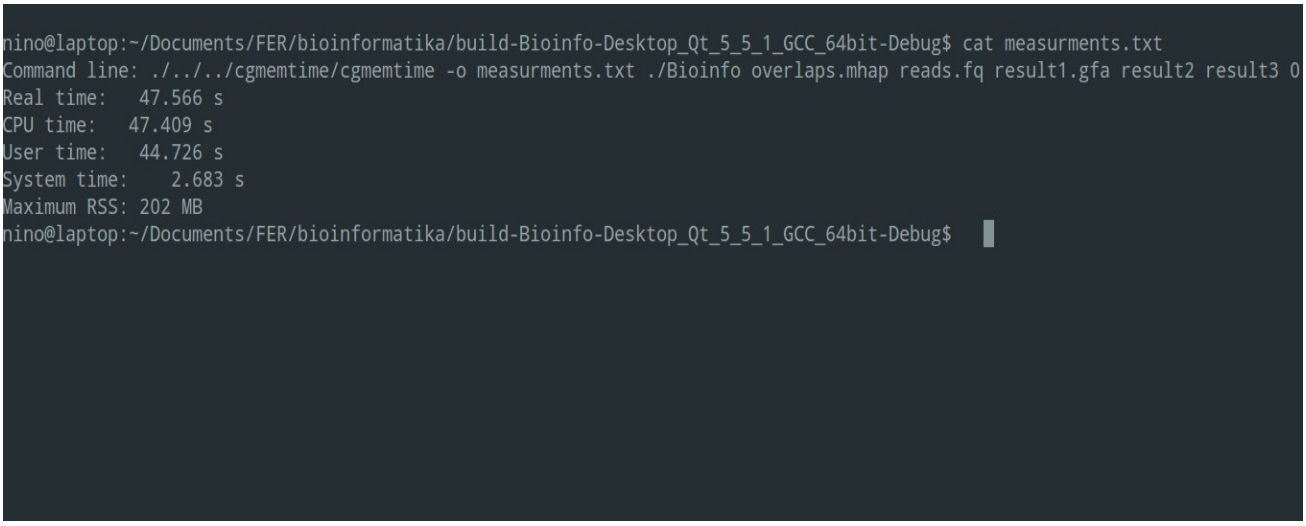
Test trajanja izvođenja programa

U tablici je prikazano trajanje izvođenja pojedinih dijelova programa te vremenski udijeli pojedinih komponenata u vremenu izvođenja istog.

Dio programa	Trajanje u sekundama	Vremenski udio u izvođenju programa
Učitavanje podataka	8.3724	26,94 %
ContainedEdge remover	0.9492	3,05 %
TransitiveEdge remover	0.8819	2,84 %
Chunker	19.6482	63,23 %
Ispis rezultata u datoteke	1.2207	3,93 %
Ostali manji izračuni i ispisi	0.0034	0,01 %
UKUPNO	**31.0758**	**100%**

Iz tablice se može vidjeti da je najzatjevnije učitavanje podataka jer se radi o velikim podacima koji se učitavaju s HDD-a koji radi na 5400 o/min. Također se vidi da je chunker najzahtjevniji dio programa zbog tog što se u njemu zapravo izvršava generiranje sekvence pojedinog chunka. Kako se radi o velikim stringovima za svaki pojedini fragment koji se učitavaju u polje, tu opet ima dosta pristupa memoriji što se odražava i na samo vrijeme izvođenja.

Test mjerenja utrošene memorije



ZAKLJUČAK

Ovdje izloženi algoritam za generiranje layout-a pokazao se kao relativno uspješan. Iako se nije uspjelo dobiti jedan cijeli string koji bi odgovarao originalnom početnom genomu, od 16k malih fragmenata je postignuto da se ipak dobije 25 većih od kojih je bilo moguće ručno sastaviti veliki dio genoma. Problemi koji su nastali vjerojatno potječu od alternativnih puteva i stršećih vrhova na kojima algoritam za generiranje sekvence zapinje. Ovaj problem bi se dao ukloniti detekcijom i uklanjanjem stršećih vrhova, te detekcijom alternativnih puteva i izborom samo jednoga od njih da slučajno algoritam ne završi u ciklusu. Iako bi se ovim poboljšanjima dobili veći fragmenti, ipak nije isključeno da ne bi opet došlo do nekih problema u grafu, jer vjerojatno nisu svi tranzitivni bridovi uklonjeni, to jako ovisi o upotrebljenim parametrima. Opisanim kompresijama grafa, dobivenog na temelju stvarnih podataka, se ne dobivaju lijepi "školski" primjerci kompresiranog grafa na kojima bi ovako opisan algoritam mogao generirati sekvencu cijelog contiga (i na kraju genoma). Zapravo je dosta upitno da li je moguće samo na temelju grafa, bez da se ide na skupi NP kompletan algoritam pronalaženja najdužeg puta, konstruirati algoritam koji bi od ovakvih "stvarnih" primjera uspio generirati potpune sekvence ispravno. Još jedan problem je i što ulazni podaci često puta nisu baš idealni i to onda još dodatno unosi problema u cijelu priču. Medjutim, mnogi od ovih problema dali bi se ukloniti ako bi prilikom izbora idućeg susjeda kojim se produžuje sekvenca bila izvršena neka vrsta provjere poravnavanjem sljedova da se detektira da li se string zaista produljuje u ispravnom smjeru ili se počeo vraćati nazad zato jer je npr. krenuo nazad drugim krakom alternativnog puta i onda naletio na fragment koji je nedostajao (bio gap) u proslom putu. Druga opcija, koja bi izbjegla skupo računanje poravnanja sljedova, a kojom bi se ipak algoritam dao malo poboljšati, je da se prilikom izbora idućeg susjeda pretraži graf do određene dubine i izabere najbolji put (onaj koji je najduži). Ovime ne bi, naravno, bile uklonjene sve nedoumice ali za očekivati je da bi chunkovi bili još veći nego što su sada (prosječno pokrivaju oko 200k baza referentnog genoma).

REFERENCE

(1) Eugene W. Myers, Toward Simplifying and Accurately Formulating Gene Assembly, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.37.9658&rep=rep1&type=pdf, datum zadnjeg pristupa: 8.1.2016.

(2) Eugene W. Myers, The fragment assembly string graph, Bioinformatics, Volume 21

(3) D. Leland Taylor, PHAST (Phage Assembly Suite and Tutorial): A Web-Based Genom Assembly Teaching Tool, Davidsons College, 2012, http://gcat.davidson.edu/phast/docs/Thesis_PHAST_LelandTaylor.pdf, datum zadnjeg pristupa: 8.1.2016

(4) PacBio Genome Assembly, Schatz Lab, Simons Center for Quantitavi Biology, http://schatzlab.cshl.edu/teaching/2013/pacbioasm.shtml, datum zadnjeg pristupa: 8.1.2016, datum zadnje promjene: 8.9.2015

(5) MHAP format, http://mhap.readthedocs.org/en/latest/quickstart.html, datum zadnjeg pristupa: 12.1.2016,

(6) ICB (Institute of Computational Biology), program Gepard: Krumsiek J, Arnold R, Rattei T. Gepard: A rapid and sensitive tool for creating dotplots on genome scale. Bioinformatics 2007; http://www.helmholtz-muenchen.de/icb/software/gepard/index.html, datum zadnjeg pristupa 12.1.2016