SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2350

Učinkovito sažimanje genoma korištenjem referentnog genoma

Juraj Radanović

Zagreb, svibanj 2021.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 12. ožujka 2021.

DIPLOMSKI ZADATAK br. 2530

Pristupnik:	**Juraj Radanović (0036503660)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	izv. prof. dr. sc. Mirjana Domazet-Lošo

Zadatak: **Učinkovito sažimanje genoma korištenjem referentnog genoma**

Opis zadatka:

U okviru ovoga diplomskog rada potrebno je proučiti problem sažimanja skupa genoma korištenjem referentnog genoma te postojeće metode koje se koriste za taj problem. U sklopu rada potrebno je proučiti i implementirati algoritam SCCG te vlastitu i originalnu implementaciju primijeniti na kompresiju i dekompresiju skupa genoma.

Rok za predaju rada: 28. lipnja 2021.

Zahvaljujem se mentorici doc. dr. sc. Mirjani Domazet-Lošo na vremenu i pomoći u izradi rada

SADRŽAJ

1. Uvod

Određivanje DNK organizma, tj. sekvenciranje je postalo jeftinije i brže nego prije, zahvaljujući novim tehnologijama sekvenciranja. Jedna od svrha sekvenciranja bi bila detekcija abnormalnih DNK u organizmu koja mogu upućivati na bolest. Sekvenciranje je jedan proces, koji dovodi do pohranjivanja DNK sekvence, prijenosa i procesiranja iste. Svaki od ovih procesa zahtjeva specifične implementacije kako bi svaki proces bio što više optimiziran, memorijski i vremenski. U ovome radu se bavim pohranjivanjem genoma, konkretnije njihovom kompresijom. Što se više genoma sekvencira, to je više podataka za pohranu i treba ih moći pohraniti na efikasne načine u smislu memorije i smanjenja gubitaka podataka tijekom dekompresije. Zapis ljudskog genoma je u redovima 10^9 okteta i držanje takvih datoteka je nemoguće. Stoga je potrebno napraviti algoritam koji će efikasno pohraniti taj genom. Postoje općeniti algoritmi za kompresiju, kao npr. 7zip, ali oni su neučinkoviti za pohranu genoma iz razloga što su genomi srodnih vrsta veoma slični. Konkretno, ljudski DNK su 99.9% slični, što znači da postoje puno redundantnih podataka. U ovom radu ću implementirati algoritam koji koristi referentni genom uz ciljni genom koji se komprimira, kako bi se iskoristila neka inherentna svojstva genoma i time ostvario veći stupanj kompresije.

2. Kompresija genoma

Kompresiju uz pomoć referentnog genoma su prvi predložili Brandon et al (2009). Prije toga su se genomi sažimali ne koristeći druge podatke. Kako bi takvi algoritmi bili bolji, često su se koristila statistička svojstva i teorija informacije za bolju kompresiju. BioCompress2 (Grumbach, Tahi, 1994.) koristi LZ77 (Lempel, Ziv, 1977.) algoritam za kompresiju bez gubitaka podataka i statističko kodiranje za detekciju i kodiranje palindroma. DNACompress (Chen et al., 2002.) pronalazi približno slična ponavljanja u sekvenci i kodira ih koristeći LZ77. Ovo su neki od primjera kompresijskih algoritma za genome koji ne koriste referentni genom. Posljedično tome, oni ignoriraju visoku sličnost homolognih organizama što dovodi do toga da nisu efikasni za kompresiju velikih sekvenci na razini gigabajta.

Algoritmi koji koriste referentni genom iskorištavaju tu sličnost, na način da se podniz ciljne sekvence pozicionira na referentnu sekvencu koristeći sekvencijsko podudaranje (uparivanje dvaju podnizova). Time se dugački nizovi u ciljnom genomu mogu sažeto napisati pomoću samo dva broja koji odgovaraju početnoj poziciji uparenih podnizova u referentnom genomu i duljini tog podniza. Jedan od takvih algoritama je GRS (Wang, Zhang, 2011.) koji koristi Unix diff program za traženje sličnosti između dvije sekvence, te se onda na najdužem zajedničkom podnizu koristi Huffmanovo kodiranje. Algoritam radi dobro samo ako su ciljni i referentni genom (sekvence) vrlo slični. GReEn (Pinho et al., 2010.) koristi globalnu hash tablicu (slično kao i u algoritmu u radu) za podnizove sekvence fiksne duljine referentnog genoma. Tada se podnizovi sekvenca ciljnog genoma hashiraju i traže u globalnoj tablici. Ako ciljni podniz postoji u globalnoj tablici, tada se kopiraju samo pozicije i duljine iz referentnog genoma, inače se koristi neki od standardnih statističkih alata za kodiranje. RLZ (Kuruppu et al., 2010.) koristi poboljšana sufiksna polja za traženje najduljeg prefiksa ciljne sekvence u referentnoj, te LZ77 za kompresiju tog prefiksa. ERGC (Saha, Rajasekaran, 2015.) dijeli ciljne i referentne sekvence na podnizove fiksne duljine i stvara hash tablicu za sve referentne segmente. Tada se traži podudaranje segmenta podniza ciljnog i podniza referentnog genoma, engl. *local matching*. Podudarni nizovi

se spremaju u datoteku koja se sažima koristeći delta kodiranje.

Performanse, tj. stupanj kompresije ovih algoritma znatno ovisi o izboru referentnog genoma. Što su ciljni i referentni genom sličniji, to će kompresija biti bolja. Local matching povezuje segmente ciljnog sa segmentima referentnog genoma. Ako sekvence nisu poravnane dovoljno dobro, moguće je da ciljni podnizovi ne će naći svoj par u referentnom segmentu. Algoritam HiRGC (Liu et al., 2017.) je riješio ovaj problem koristeći globalnu hash tablicu za cijelu referentnu sekvencu. U ovome radu ću implementirati SCCG algoritam (Shi et al., 2018.) za kompresiju genoma bez gubitaka u jeziku C++. Algoritam koristi neka svojstva već prethodno navedenih algoritma.

3. SCCG algoritam

Ulaz u algoritam su ciljni genom odnosno sekvenca i referentni genom odnosno sekvenca. Implementirani algoritam sažima datoteke koje su u FASTA formatu. FASTA format je jednostavni format datoteke koji sadrži nukleotidne sekvence (baze A, C, G, T) ili aminokiseline zapisane jednim slovom za svaku bazu. Prvi redak svake FASTA datoteke sadrži opis sekvence, a dalje se nižu redovi baza gdje je svaki red tipično duljine 80 slova. Jednostavni format FASTA datoteke čini ju laganom za parsirati za korištenje u nekom programskom jeziku. Abeceda koja se koristi za zapis sekvence je Σ, a veličina te abecede je $|\Sigma|$. Generalno govoreći, algoritmi za kompresiju su efikasniji, što je $|\Sigma|$ manja, stoga je prvi korak ovog algoritma pretvaranje svih malih slova u velika, čime se efektivno smanji veličina abecede, Ta mala slova su potrebna prilikom dekompresije, kako ne bi bilo gubitaka podataka, iz tog razloga se u privremenu datoteku spremaju pozicije i duljine svih podnizova malih slova, točnije pamti se početna i završna pozicija podniza u cijelom nizu. Algoritam koristi lokalnu hash tablicu za pretraživanje segmenata, kao i globalnu hash tablicu, ukoliko je lokalno pretraživanje loše. Nakon lokalnog i/ili globalnog pretraživanja kada je algoritam gotov, podaci koji su se spremili u datoteku će se dodatno procesirati, pri čemu se koristi tzv. delta kodiranje.

3.1. Lokalno pretraživanje

Lokalno pretraživanje uključuje segmentiranje ciljnog genoma *T* i referentnog genoma *R*, izgradnja lokalne hash tablice H_L, te traženje istih segmenata. Prvi korak je segmentacija obiju sekvenca *R* i *T* na segmente $r_1, r_2, ..., r_i$ i $t_1, t_2, ..., t_j$ gdje su svi segmenti jednake duljine L. Za zadnje segmente r_i i t_j vrijedi da su manji ili jednaki L. Slijedeći korak je izgradnja hash tablice H_L za sve referentne segmente r. Prilikom pretraživanja segmenta, lokalna hash tablica se izgradi za svaki segment, pri čemu se svaki referentni segment dijeli na još manje podnizove zvane *k-merovima*, podnizove segmenta duljine k. Za svaki *k-mer* izračuna se njegov hash koristeći *std::hash<>()* standardne bibli-

oteke te se *k-mer* i njegova početna pozicija u segmentu r spreme pod tim hashom. Moguće je da postoji više istih *k-mera*, pri čemu se svi spremaju pod istim hashom jer imaju različite pozicije unutar segmenta. Broj zapisa u H_L je $L - k + 1$. Važno je napomenuti kako se ovdje koristim strukturu *unordered_map* koja odgovara hash mapi sa konstantnim vremenom pretraživanja, unosa i brisanja elementa. Ovime završava izgradnja lokalne hash mape H_L. Za primjer, niz *ABCCAB* i $k = 2$, H_L će imati sve skupa 5 zapisa, neki pod istim pretincem (ključem):

Pretinac	k-mer	početna pozicija
752165258	AB	[0, 4]
501045426	BC	1
1977225635	CC	2
2010780873	CA	3

Tablica 3.1: Primjer hash tablice H_L

Tek izgradnjom H_L počinje proces lokalnog pretraživanja. Ciljni segment t se dijeli na $L - k + 1$ *k-mera* i za svaki *k-mer* se izračuna hash i traži njegov par u tablici H_L. Ukoliko u H_L ne postoji izračunati hash, proces se nastavlja sa idućim *k-merom*. Ukoliko postoji hash, to znači da se istim *k-mer* pojavio i u ciljnom i u referentnom segmentu. Dodatno se provjerava jesu li *k-merovi* stvarno isti usporedbom slova, kako bi se izbjegao slučaj gdje različiti nizovi daju isti hash. Oba *k-mera* ciljnog i referentnog segmenta se dalje proširuju na način da se uzima slijedeće slovo u segmentima i tako se traži najdulje podudaranje. Ako se *k-mer* ciljnog segmenta t pojavljuje više puta unutar referentnog segmenta r, tada se proširivanje *k-merova* radi na svim referentnim *k-merovima* i uzima se najduži, tj. početna pozicija i duljina unutar referentnog segmenta. Ukoliko postoji više parova koji su se proširili za jednak broj slova, tada se uzima ona početna pozicija referentnog *k-mera* koja je najbliža krajnjoj poziciji referentnog *k-mera* prošlog para, kako bi delta kompresija bila efikasnija. Algoritam lokalnog pretraživanja se može opisati slijedećim pseudokodom:

Algorithm 1: Lokalno pretraživanje

Ulaz: Referentni segment S_r, ciljni segment S_t, veličina *k-mera* k

Rezultat: Pozicije početka i kraja referentnog i ciljnog segmenta

Izgradi lokalnu hash tablicu H_L na S_r;
startRef = 0;
startTar = 0;
endRef = 0;
endTar = 0;
while *i < S_t.size() - k + 1* **do**
    ciljniKmer = S_t[i, i+k];
    if *hash ne postoji u H_L* **then**
        i++;
        continue;
    end
    Pronađi sve referentne *k-merove* k_R iz H_L;
    l_{max} = 0;
    for *svaki k_R == ciljniKmer* **do**
        proširi oba *k-mera* dokle god su isti;
        l = proširenje;
        if $l == l_{max}$ **then**
            if *početna pozicija referentnog k-mera najbliža završnoj poziciji prošlog para* **then**
                startRef = početna pozicija referentnog *k-mera*;
            end
        else
            $l_{max} = l$;
            startRef = početna pozicija referentnog *k-mera*;
        end
    end
    startRef = startPosition;
    startTar = i;
    endRef = startPosition + k + l_{max} - 1;
    endTar = i + k + l_{max} - 1;
    spremi ovu četvorku u listu;
end
vrati listu;

Za $S_R = ABACABAxABA$, $S_T = ABADABAcABA$ i $k = 2$ lokalno pretraživanje će vratiti tri četvorke:

startRef	endRef	startTar	endTar
0	2	0	2
0	2	4	6
0	2	8	10

Tablica 3.2: Primjer lokalnog pretraživanja

Prvi *k-mer* (ciljni) je AB i on ima svog para u hash tablici H_L. Oba *k-mera* (ciljni i referentni) se mogu proširiti za samo jedan znak, a slijedeći znak na kojem se razlikuju se također pamti. H_L ima tri zapisa AB *k-merova* sa pozicijama 0, 4 i 8. Ciljni *k-mer* se proširuje uspoređujući proširenja sa sva tri zapisa iz H_L. Pošto nije bilo većeg proširenja od prvog zapisa, uzima se pozicija prvog referentnog zapisa kao rezultatnog (što će kasnije biti efikasnije prilikom delta kodiranja). Pamte se pozicije u referentnom i ciljnom *k-meru* i proces se nastavlja sa idućim *k-merom* počevši od slijedećeg znaka gdje se *k-merovi* nisu podudarili. Slično se dogodi i za druge *k-merove* AB, ujedno su to i jedini parovi koje lokalno pretraživanje nađe. Ovime smo skratili nizove ABA sa tri znaka na dva znaka (njihove pozicije u referentnom segmentu). Ostala slova će se morati pamtiti.
Za drukčije nizove $S_R = CGGACGC$, $S_T = CGGGGACGGCA$ i $k = 2$ lokalno pretraživanje će dati slijedeće rezultate:

startRef	endRef	startTar	endTar
0	2	0	2
2	5	4	7

Tablica 3.3: Primjer lokalnog pretraživanja

Prvi ciljni *k-mer* CG nađe u H_L svog para. U hash tablici postoje dva takva zapisa, jedan za pozicijom 0, a drugi sa pozicijom 4. Proširivanje se događa sa oba zapisa, gdje se prvi zapis (sa pozicijom 0) može proširiti za jedan znak, a drugi zapis se ne može proširiti. Time je prvi rezultat $S_R[0:2]$ i $S_T[0:2]$. Slijedeći *k-mer* koji se pregledava je GA. Nakon pronalaska svog para u H_L, *k-merovi* se produžuju za dva znaka. Lokalno pretraživanje nastavlja od slijedećeg *k-mera* CA, ali on nema svog para i algoritam se

zaustavlja.
Ako ne postoji niti jedna podudarnost, tada se k smanji. Time se poveća vjerojatnost da će manji ciljni *k-merovi* naći svog para u tablici. Ako i dalje ne postoji podudarnost na segmentu t, tada bi se segment trebao u cijelosti zapisati takak kakav je, bez kompresije. Što je lokalno pretraživanje lošije, to će biti više nekomprimiranih zapisa i time je efikasnost loša. Kao mjera efikasnosti lokalnog pretraživanja koriste se dvije varijable praga T_1 i T_2. Ako je omjer broja nekomprimiranih znakova i duljine segmenta t veći od praga T_1, onda je segment loše podudarnosti sa referentnim segmentom. Ako je broj takvih loših segmenata veći od T_2, tada dvije sekvence nisu dovoljne sličnosti. Kako ne bi dvije takve sekvence zapisali direktno bez kompresije, što nema smisla ionako, koristi se globalno pretraživanje.

3.2. Globalno pretraživanje

Ciljna sekvenca *T* i referentna sekvenca *R* se ne segmentiraju na manje dijelove kao u lokalnom pretraživanju. Ovime se poveća vjerojatnost pronalaženja podnizova *T* u sekvenci *R*. Sam algoritam je skoro isti lokalnom pretraživanju, razlika je u tome što se proširivanje podudarnih *k-merova* ograničava na *m* proširenih slova. Time se pokušava izbjeći preveliko podudaranje između *k-merova* (tj. izbjegava se prevelika razlika između susjednih podudarnih parova), što poboljšava performanse delta kodiranja kasnije. Ukoliko algoritam ne pronađe par *k-merova*, onda se granica *m* makne. Umjesto lokalne hash tablice H_L, izgradi se globalna hash tablica H_G na cijeloj sekvenci *R*. Veličina H_G je znatno veća od H_L i time zauzima puno više memorije. Točnije:

$$|H_L| = L - k + 1$$

$$|H_G| = |R| - k + 1$$

Veličina lokalne tablice najviše ovisi o broju *L*, tj. o veličini segmenta, dok veličina globalne tablice ovisi o veličini referentne sekvence *R*. S obzirom da je $|R|$ često nekoliko redova veličine veće od $|L|$, time je izgradnja i držanje H_G u memoriji puno skuplje. Rezultat pretraživanja je isti kao i u lokalnom pretraživanju, traže se pozicije gdje se sekvence podudaraju.

3.3. Delta kodiranje

Delta kodiranje ili delta kompresija je način komprimiranja podataka, gdje se uzastopni podaci kodiraju u obliku razlika (engl. deltas). Ukoliko su podaci slični, tj. njihove razlike su malo, onda je delta kompresija efikasnija, dakle priroda podataka određuje efikasnost kompresije. Za niz $[0, -1, 1, 4]$ će delta kompresije izgledati $[0, -1, 2, 3]$. Često se ovakvom kompresijom može smanjiti broj različitih znakova, tj. smanji se veličina abecede čime se opet može postići bolja kompresija na način da se koristi manje bitova za prikaz svih znakova.

Delta kodiranje se koristilo u nekim sustavima sa HTTP serverom gdje su se ažurirane web stranice slale kao razlika u inačicama iz razloga što se većina stranica ne mijenja jako brzo i time se smanji promet. Još jedna primjena je delta kopiranje, čime se može brzo kopirati datoteka kojoj je samo dio promijenjen. Sustav za verzioniranje Git koristi delta kodiranje prilikom *git repack* operacije.

Delta kodiranje će se koristiti u SCCG algoritmu u zadnjim koracima nakon lokalnog i globalnog pretraživanja, gdje će se pozicije uzastopnih podudarenih parova pretraživanja komprimirati. Očito ovime, ako su uzastopne pozicije bliže, onda će i delta kompresija biti bolja.

4. Zaključak

Zaključak.

LITERATURA

Delta encoding. URL https://en.wikipedia.org/wiki/Delta_encoding. Pristup: 21.4.2021.

Fasta format. URL https://en.wikipedia.org/wiki/FASTA_format. Pristup: 24.4.2021.

Compression of genomic sequencing data. URL https://en.wikipedia.org/wiki/Compression_of_Genomic_Sequencing_Data. Pristup: 20.4.2021.

M. C. Brandon, D. C. Wallace, i P. Baldi. Data structures and compression algorithms for genomic sequence data. 25(14):1731–1738, 2009.

Dmitir Pavlichin i Tsachy Weissman. The Desperate Quest for Genomic Compression Algorithms, 2018.

Wei Shi, Jianhua Chen, Mao Luo, i Min Chen. High efficiency referential genome compression algorithm. 2018.

Učinkovito sažimanje genoma korištenjem referentnog genoma

Sažetak

Sažetak na hrvatskom jeziku.

Ključne riječi: Ključne riječi, odvojene zarezima.

Title

Abstract

Abstract.

Keywords: Keywords.