SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

PROJEKT

**High-speed and high-ratio referential genome compression**

Marko Bago, Anton Filipović, Matija Križanec

Voditelj: Mirjana Domazet-Lošo

Zagreb, siječanj , 2020.

**Sadržaj**

[1. Uvod 3](#_Toc29850120)

[2. HiRGC metoda 4](#_Toc29850121)

[2.1 Predprocesiranje 4](#_Toc29850122)

[2.2 Greedy matching algoritam 6](#_Toc29850123)

[2.3 Naknadna obrada 8](#_Toc29850124)

[2.4 Dekompresija 8](#_Toc29850125)

[3. Testiranje i rezultati 9](#_Toc29850126)

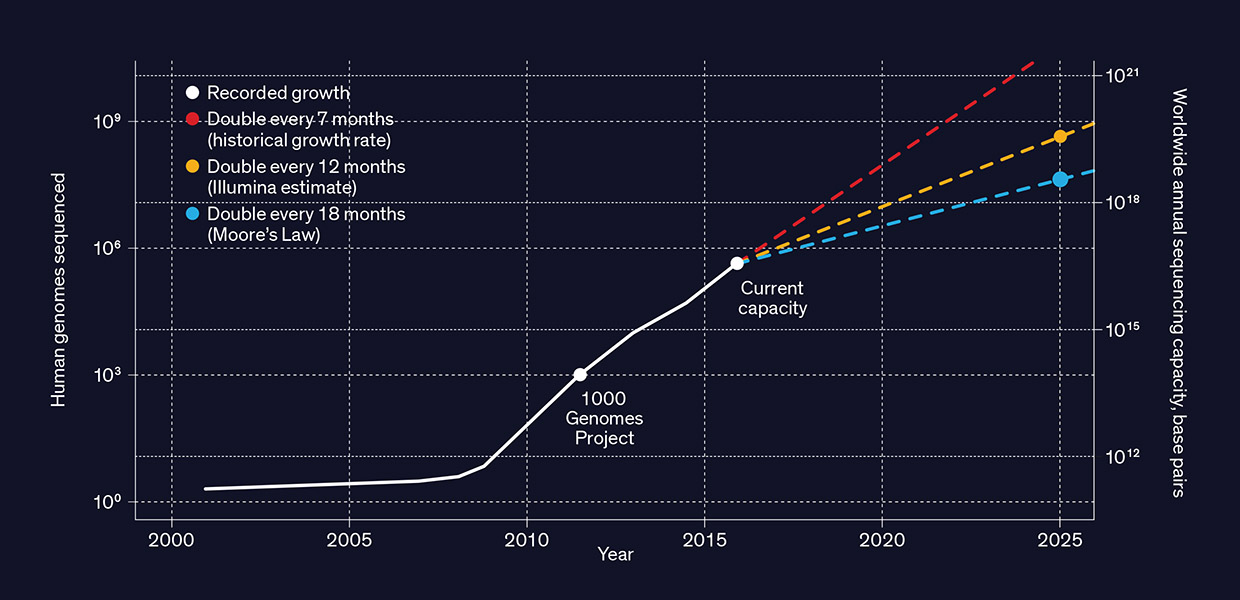
[4. Opis implementacije 10](#_Toc29850127)

[5. Zaključak 11](#_Toc29850128)

[6. Literatura 12](#_Toc29850129)

# Uvod

Kako metode sekvenciranja genoma postaju sve efikasnije te potreba za njima sve brže raste tako raste i potreba za što efikasnijom pohranom takvih podataka (Slika 1). Tradicionalne metode kompresije genoma ne postižu dovoljno dobru stopu kompresije zbog specifičnosti DNA sekvenci poput male abecede te čestih ponavljanja nizova. To bi zbog sve veće obilnosti sekvenciranih genoma moglo predstavljati problem za njihov prijenos i pohranu na računala pošto jedna FASTA datoteka koja sadrži zapis genoma može doseći veličinu čak i do 100 GB. Moguće rješenje za ovaj problem jest korištenje metoda koje se temelje na korištenju referentnog genoma te spremaju samo razlike između dva iznimno slična genoma (ciljnog i referentnog).



*Izvor: Stephens ZD, Lee SY, Faghri F, Campbell RH, Zhai C, Efron MJ, et al.(2015) PLoS Biol 13(7)*

*Slika 1 –* ***Rast sekvenciranja ljudskog genoma –*** *Od prve publikacije ljudske sekvence genoma 2001. godine postoji nagli porast u broju sekvenciranih ljudskih genoma*

Cilj ovog projekta jest implementacija jedne takve metode, HiRGC (High-speed and high-ratio referential genome compression) koja postiže stope kompresije i do 200 puta. Testirali smo ju na više različitih genoma i mjerili stopu kompresije te vremensku kompleksnost algoritma.

# HiRGC metoda

Metoda koristi referentni genom kako bi spremila razlike između njega i ciljnog genoma spremajući pozicije i dužinu zajedničkih podnizova genoma. Motivacija za ovakav postupak jest činjenica da su genomi različitih jedinki iste vrste iznimno slični, npr. sličnost ljudskih genoma jest >99% (Lander et al., 2001). HiRGC metodu možemo podijeliti u 3 koraka: predprocesiranje, pohlepno sparivanje (eng. Greedy matching) te naknadna obrada.

## Predprocesiranje

Algoritam na ulaz dobiva dvije datoteke: ciljnu i referentnu datoteku u FASTA tekstualnom formatu. FASTA datoteka sadrži identifikator genoma te slijed sekvenci genoma. U koraku predprocesiranja iz ciljne se datoteke izvlače pomoćne informacije poput identifikatora, pozicije i duljine sekvenci genoma označenih slovima G={A,T,G,C}, pozicije i duljine sekvenci genoma označenih malim slovima g={a,t,g,c} i slovom N te druga slova i njihove pozicije. U nastavku je detaljno opisana procedura predprocesiranja:

1. Spremi identifikator iz FASTA datoteke kao znakovni niz
2. Čitaj sekvencu genoma liniju po liniju te spremaj pozicije kraja svake linije u polje newline\_indices
3. Ako naiđeš na intervale sekvence koji sadrže mala slova spremi njihove pozicije u polje lower\_sq\_begin\_indices te njihove duljine u polje lower\_sq\_lengths. Usporedno pretvori mala slova u velika.
4. Ako naiđeš na nizove slova N spremi njihove pozicije u polje n\_sq\_begin\_indices te njihove duljine u polje n\_sq\_lengths. Usporedno makni sva slova N iz sekvence.
5. Ako naiđeš na veliko slovo koje ne spada u skup G={A,C,G,T} U 'N' spremi njegovu poziciju u polje other\_char\_indices te njegovu vrijednost u polje other\_chars te izbaci to slovo iz sekvence.

Time dobivamo pomoćne informacije koje će se koristiti u dekompresiji te ciljnu datoteku genoma koja sadrži samo slova iz skupa G. Iz referentnog genoma također izvučemo samo slova iz skupa G tako da slova koja se već nalaze u G prepišemo, mala slova pretvorimo u velika, a sva ostala velika slova zanemarimo. Time smo znatno smanjili abecedu koju koristi datoteka te možemo primijeniti enkodiranje oblika:

**'A' → 0, 'C' → 1, 'G' → 2, 'T' → 3**

Nadalje takav cijelobrojni niz spremamo kao k-torke s cijelobrojnim vrijednostima veličine (2 \* k) bita koje, ako nam je dan cijelobrojni niz sa vrijednostima u(i) = {0,1,2,3}, računamo kao:

(1)

gdje je i ∈ [1, n – k + 1] tj. postoji n – k + 1 preklapajućih k-torki koje će pretstavljati unose u hash tablicu.

Pseudokod za punjenje hash tablice s nizom brojeva iz referentnog genoma r(i):

1: Sve vrijednosti h(i) su inicijaliziraj na -1

2: **od** *i* = 1 **do** (*n*−*k*+1)

3: Izračunaj k-torku iz referentne sekvence:

4: Ažuriraj hash tablicu: *p*(*i*)=*h*(), *h*()=*i*.

gdje je *p(\*)* polje unosa u tablicu, *h(\*)* polje pokazivača na headere, a *s* veličina hash tablice. U sljedećem primjeru prikazana je jedna iteracija ažuriranja hash tablice:

*Primjer 1. (k=4 niz 1103...):*

*(i = 0, j = 0) V = 0000 0000 + r(0+0)\*40 = 0000 0000 + 0000 0001 = 0000 0001*

*(i = 0, j = 1) V = 0000 0001 + r(0+1)\*41 = 0000 0001 + 0000 0100 = 0000 0101*

*(i = 0, j = 2) V = 0000 0101 + r(0+2)\*42 = 0000 1001 + 0000 0000 = 0000 0101*

*(i = 0, j = 3) V = 0000 0101 + r(0+3)\*43 = 0000 0101 + 1100 0000 = 1100 0101*

*Dobili smo vrijednost k-torke: V=1100 0101 te ažuriramo hash tablicu tako da postavimo vrijednosti p(i=0) = h(V mod s) i h(V mod s) = i = 0. Dakle polje p spaja sve unose koji imaju jednaku hash vrijednost u jednu povezanu listu.*

## Greedy matching algoritam

U hash tablici, greedy matching algoritam pronalazi maksimalno podudarajući niz počevši od početne pozicije 0 ciljne sekvence brojeva. Recimo da algoritam obrađuje i-tu k-torku u ciljnoj sekvenci. Prvo će izračunati vrijednost k-torke Vi te pronaći odgovarajuću „košaru“ iz hash tablice referentne sekvence kojoj pripadaju unosi sa hash vrijednosti (Vi mod s). Ako u njoj ne postoji niti jedan unos, i-ta k-torka u ciljnoj sekvencise ne poklapa se sa referentnom te se preskače do (i + 1)-e k-torke. Ako unosi postoje pretražuje k-torke u „košari“ i gleda poklapaju li se u k-torkama brojevi iz ciljne i referentne sekvence. Dokle god se poklapaju proširuje se najduži poklapajući niz. Ako je algoritam pri pretraživanju „košare“ naišao na više jednako dugih poklapajućih nizova uzima se onaj prvi. U izlaznu datoteku zapisuje se pozicija i dužina poklapajućeg niza. Niz između posljednjeg preklapajućeg niza i novopronađenog jest nepoklapajući niz koji se također zapisuje u izlaznu datoteku.

*Primjer 2. :*

*Ako su nam dani ciljni i referentni niz:*

*Referentni: 0032302230103002021*

*Ciljni: 003210223012130020022*

*Matching algoritam daje rezultat: M = [(1,4),(1),(6,6),(21),(14,5),(0),(7,2)], gdje su poklapajući nizovi reprezentirani parom (pozicija pokl. niza, duljina pokl. niza), a nepoklapajući niz kao (nepoklapajući niz).*

Pseudokod Greedy matching algoritma:

1: **Ponavljaj** do kraja ciljne sekvence:

2: Izračunaj k-torku iz ciljne sekvence:

3:  indeks referentne k-torke = h(())

4: **Ponavljaj** dok postoje k-torke u dobivenoj hash “košari”

5: Izračunaj u koliko se znakova podudaraju ciljna i referentna k-torka

6: **ako** se poklapaju u više od k znakova i više od veličine prethodnog najdužeg poklapajućeg niza: pronađen novi poklapajući niz

7: Nađi nepoklapajući niz: znakovi između zadnjeg poklapajućeg niza novopronađenog poklapajućeg niza

8: Preostali znakovi se tretiraju kao nepoklapajući niz

**Izlaz u datoteku:** indeksi početaka poklapajućih nizova i njihove duljine te nepoklapajući nizovi

## Naknadna obrada

Kako bi dodatno komprimirali podatke koristimo run-length encoding, RLE, algoritam (Held i Marshall, 1991) kako bi kompaktnije zapisali pozicije krajeva linija u ciljnom genomu. Ovaj algoritam koristan je kada imamo jako puno ponavljajućih znakova koje trebamo spremiti što je u našem slučaju istina jer se u većini datoteka genoma koje mi koristimo dužina linija ne mijenja. Tada bi se polje koje sadrži duljine linija (npr. polje [100, 100, 100, 100, .. ] dužine 2000) moglo zapisati pomoću RLE kao: (broj koji se ponavlja, broj ponavljanja) ili u danom slučaju kao: (100, 2000).

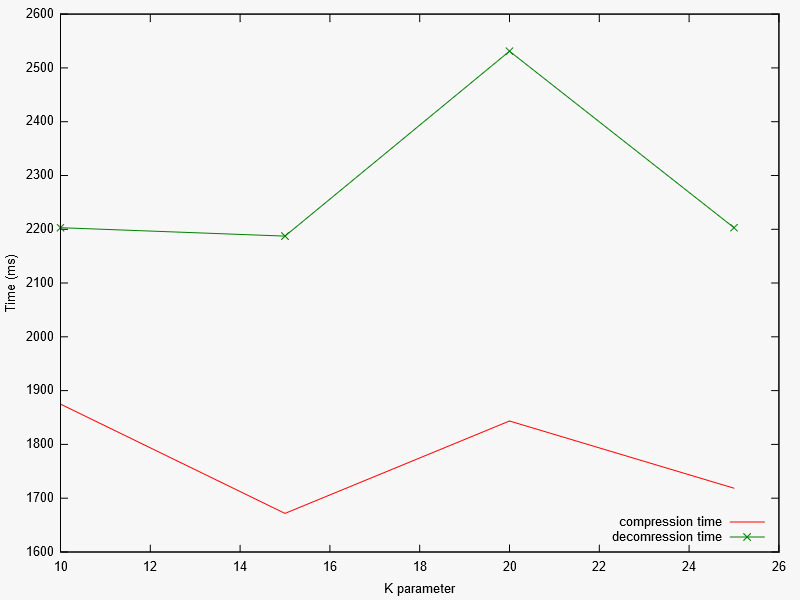
Izlaz cijelog postupka tj. pomoćni podaci, pozicije poklapajućih nizova i nepoklapajući nizovi spremaju se u ASCII tekstualnu datoteku koja se još naknadno kompresira PPMd (Prediction by partial matching data) kompresijskim algoritmom. Za to koristimo gotovu implementaciju PPMd algoritma u sklopu programa 7zip (<http://www.7-zip.org/>), a također smo koristili i algoritam LZMA2 te primjetili da u mnogim slučajevima ostvarujemo bolje stope kompresije.

## Dekompresija

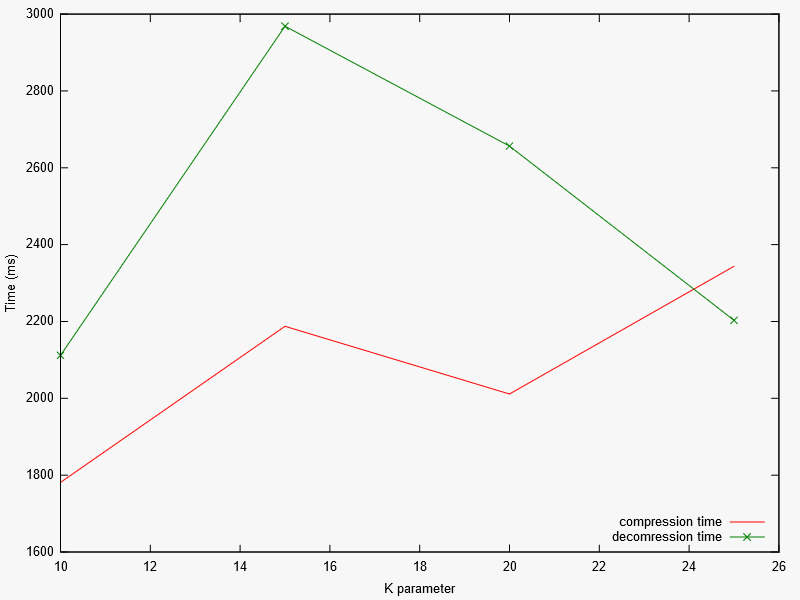
Pošto su sve faze algoritma: PPMd, run-length encoding i greedy matching reverzibilne, dekompresija se zapravo vrši kao reverzija kompresijskog dijela algoritma.

# Testiranje i rezultati

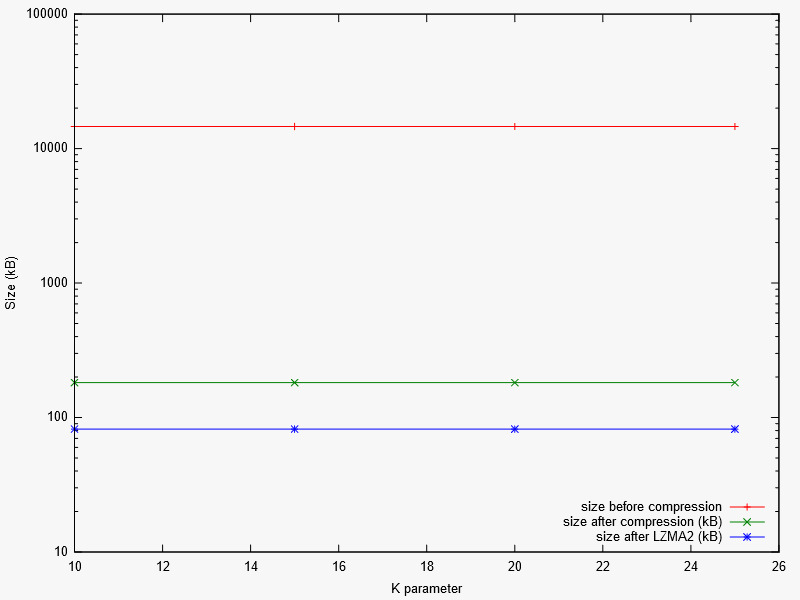
Testiranje smo proveli na 2 različite vrste kromosoma, kromosomima vrste *C.elegans* te *H.sapiens*. Kod vrste *C.elegans* testiranje smo proveli na dva različita para kromosoma, dok smo kod *H.sapiens* testiranje proveli nad jednim parom kromosoma. Tokom testiranja nad kromosomima vrste *C.elegans* varirali smo parametar algoritma k (veličina n-torke vrijednosti prilikom postupka sažimanja za tablicu sažimanja) od vrijednosti 10 do vrijednosti 25 s veličinom koraka 5. Tokom variranja parametra k mjerili smo vrijeme koje je algoritmu potrebno za kompresiju i vrijeme potrebno za dekompresiju te veličinu izlazne datoteke nakon kompresije. Dobivene rezultate prikazujemo grafovimaGraf 1, Graf 2, Graf 3, Graf 4**.**



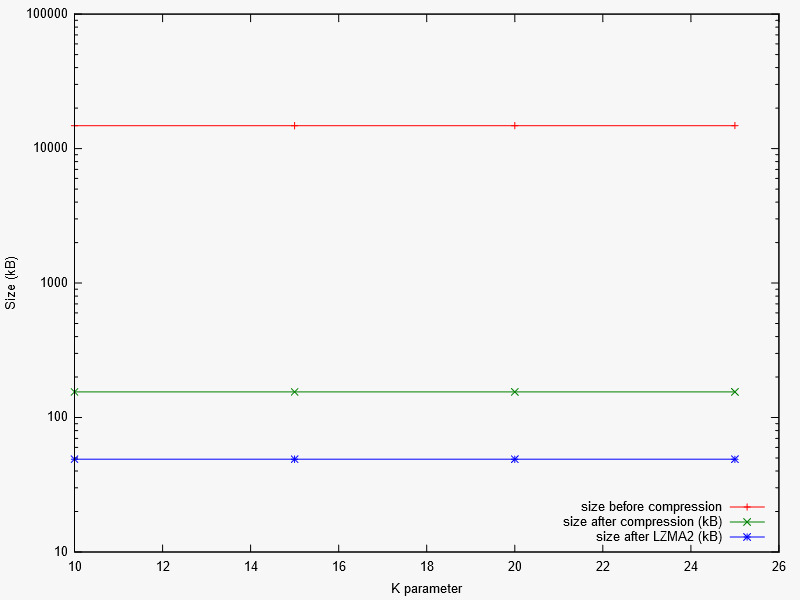
*Graf 1 – prikaz kretanja vremena kompresije i dekompresije u ovisnosti o parametru k nad prvim parom kromosoma vrste C.elegans*



*Graf 2 - prikaz kretanja vremena kompresije i dekompresije u ovisnosti o parametru k nad drugim parom kromosoma vrste C.elegans*



*Graf 3 – prikaz veličine datoteka prije kompresije, nakon kompresije te nakon LZMA2 kompresije u zavisnosti o parametru k nad prvim parom kromosoma vrste C.elegans*



*Graf 4 – prikaz veličine datoteka prije kompresije, nakon kompresije te nakon LZMA2 kompresije u zavisnosti o parametru k nad drugim parom kromosoma vrste C.elegans*

Tokom testiranja nad kromosomima vrste *H.sapiens* također smo pratili vrijeme kompresije, dekompresije te veličinu izlazne datoteke nakon kompresije za naše riješenje te originalnog rješenja autora algoritma. Dobivene rezultate prikazujemo grafovima Graf 5 i Graf 6.

*Graf 5 – prikaz odnosa veličine zapisa kromosoma u različitim fazama algoritma te usporedba s originalnim rješenjem na paru kromosoma vrste H.sapiens*

Target=hg17, reference=hg18

*Graf 6 – usporedba vremena kompresije i dekompresije kromosoma vrste H.sapiens našeg i originalnog rješenja, mjereno nad parom kromosoma hg17 i hg18*

# Opis implementacije

Projekt je implementiran u programskom jeziku C++ 14. Konfiguracija projekta definirana je Cmake alatom za upravljanje C/C++ projektima.

# Zaključak

Problem prevelike količine genomskih podataka koje se trebaju prenositi pohranjivati na računala sve je prisutniji u znanstvenim i medicinskim zajednicama te je sve više potrebna efikasna metoda za kompresiju takvih podataka. Metoda koju smo implementirali u radu postiže kompresiju i do 300 puta te pretstavlja moguće rješenje za ovaj problem. Kroz projekt upoznali smo se sa procesom optimizacije programskog koda, pisanjem kvalitetnijeg C++ koda te rješavanjem memorijskih problema na koje smo nailazili. Također dobili smo priliku raditi sa stvarnim genomskim podacima te s rješavanjem pravog problema u znanstvenoj zajednici.

# Literatura

[1] Liu Y.,Peng H., Wong L., Li J. High-speed and high-ratio referential genome compression. (2017.), Bioinformatics

[2] Pavlichin D., Weissma T. (2018, Aug). The Desperate Quest for Genomic Compression Algorithms. URL: https://spectrum.ieee.org/computing/software/the-desperate-quest-for-genomic-compression-algorithms pristupljeno : 1.12.2020.

[3] Lander,E.S. et al. (2001) Initial sequencing and analysis of the human genome. Nature, 409, 860–921.

[4] Held,G., and Marshall,T. (1991). Data Compression; Techniques and Applications: Hardware and Software Considerations. John Wiley and Sons, Inc., New York.