

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Vlastita implementacija: HRCM algoritam

Leon Hegedić, Ivan Terzić

Voditelj: *Mirjana Domazet-Lošo*

Zagreb, lipanj 2024.

SADRŽAJ

1. Uvod	1
2. Opis algoritma [3]	2
2.1. Kompresija	2
2.2. Izvlačenje informacija o sekvencama	2
2.3. Pronalazak podudarnih informacija	3
2.4. Kodiranje	5
2.5. Dekompresija	5
3. Primjer izvođenja	7
4. Testiranje i analiza	11
5. Zaključak	12
6. Literatura	13

1. Uvod

U području bioinformatike čest je rad s dugačkim nizovima znakova. Neka taj dugi niz znakova predstavlja niz baza A, C, G i T u DNA molekuli. Ta sekvenca baza određuje genetsku informaciju koja je cijelom svojom duljinom bitna [1], tj. ne smije se izostaviti niti jedna baza u zapisu baze. Ako se uzme u obzir da se tim nizom može predstaviti i cijeli genom pojedine vrste koji može biti dug od 50 000 pa sve do preko 100 000 000 000 znakova, odnosno baza (za primjer, genom čovjeka sastoji se od 3 000 000 000 baza). [2] Ako se uzme u obzir da se slova u današnjim računalima pohranjuju s jednim bajtom, odnosno 8 bitova, neki niz baza dug 1 000 000 znakova zauzeo bi 1MB, a 100 000 000 gotovo 100MB, a to nije ni približno dovoljno za pohranu genoma na primjer kraljevnjaka. Jasno je da je potreban efikasan način za pohranu tih niza baza zbog samog prijenosa, preuzimanja, ali i pohrane na uređajima. Ovaj se rad bavi upravo takvim algoritmom: HRCM algoritmom. Algoritam radi kompresiju datoteka genoma pomoću referentnog niza baza te pohranjuje, s manjim memorijskim zauzećem, inicijalnu datoteku genoma u drugom obliku sa svim potpunim informacijama. Rad se bazira na proučavanju algoritma, opisu teoretskih pojmova nužnih za razumijevanje algoritma, opisu samog algoritma te dokumentiraju testiranja algoritma.

2. Opis algoritma [3]

HRCM algoritam sastoji se od dvije glavne faze: kompresija i dekompresija. U fazi kompresije se sekvence spremaju u datoteke sažetog oblika (manjeg memorijskog zauzeća) koje sadržavaju potpunu informaciju o sekvenci, dakle ne gube se informacije ni o malim slovima, ni specijalnim znakovima niti „X” znakovima. U fazi dekompresije se ti pohranjeni podatci vraćaju u svoj izvorni oblik, kako je primljen na ulazu u algoritam.

2.1. Kompresija

Algoritam na svoj ulaz u fazi kompresije prima referentnu datoteku genoma (referentnu sekvencu) i jednu ili više sekvenci koje se trebaju komprimirati, sve datoteke u FASTA formatu. Sekvence za komprimiranje komprimiraju se pomoću referentnog genoma. Sama faza kompresije algoritma dijeli se u 3 glavna koraka: izvlačenje informacija sekvenci, pronalazak podudarnih informacija u nizovima te kodiranje informacija o sekvencama.

2.2. Izvlačenje informacija o sekvencama

U prvom koraku kompresije, na ulazu samog algoritma, nalaze se datoteke sekvenci namijenjene za kompresiju te referentna sekvenca u FASTA formatu. To znači da je u prvom redu identifikator genoma te u svakom redu ispod niz znakova fiksne duljine za svaki red koji se sastoji od osnovnih baza, malih slova, koje označavaju nesigurnost kod očitavanja, slova N i slova X ili procijepa -. Za svaku se sekvencu pohranjuju sljedeće informacije: identifikator, sekvenca osnovnih baza (A, C, G i T), informacije o pozicijama malih slova, informacije o slovima N, informacije o ostalim znakovima te broj koji označava duljinu retka. Za referentnu sekvencu pohranjuju se samo sekvenca osnovnih baza i informacije o malim slovima, ona ne mora pamtit i informacije o ostalim slovima, itd. Sam proces obavlja se na sljedeći način: obzirom na FASTA format,

prva linija u datoteci je tzv. identifikator, on se izvlači i sprema kao identifikator pojedine datoteke. Nakon toga se, za prvu liniju koja sadrži niz baza, gleda njena duljina te pohranjuje kao duljina retka. Nakon toga se u nizu baza redom bilježe pozicije malih slova te potom i N slova i posebnih slova. Obzirom da je očekivano da će algoritam raditi s izuzetno dugim nizovima znakova, pohrana informacija o malim i N slovima se radi pomoću pohrane relativnog indeksa (udaljenosti) od prošle pojave malog ili N slova (naravno, prošla pojava malog slova se gleda od prošle pojave malog slova, itd.) i pohrane duljine tog niza znakova. Tako se izbjegava pohrana velikih brojeva kao pozicija, što bi bilo očekivano za slučaj kad bi se koristili apsolutni indeksi u nizu znakova. Nakon pohrane tih informacija, znakovi se spremaju u jedan niz znakova, brišu se posebni znakovi i N znakovi te se dobije čist niz znakova A, C, G ili T, odnosno baza. Za referentu sekvencu postupak je analogan, osim što se spremaju samo podatci potrebni za taj niz, dakle niz baza i podaci o malim slovima. Iako se informacije ne spremaju, također se brišu N znakovi i specijalni znakovi. Ovako zapisani nizovi znakova ulaze u drugu fazu kompresije: pronalazak podudarnih informacija.

2.3. Pronalazak podudarnih informacija

U drugoj fazi kompresije, pronalasku podudarnih informacija, kao ulaz primamo nizove znakova zapisane kako je opisano u prošlom odlomku, točno 1 referentan niz znakova i barem jednu sekvencu koja je namijenjena za kompresiju. U ovoj fazi cilj je pronaći podudarnosti između referentne sekvence i sekvenci koje se komprimiraju kako bi se što više smanjilo memorijsko zauzeće zapisa. Ova se faza sastoji od pronalaska podudaranja u nizu baza te u malim slovima. Ukoliko postoji više od jedne sekvence koje se trebaju komprimirati, algoritam koristi i tzv. podudaranje drugog reda (engl. second-level matching). Prvo se u ovoj fazi radi pronalazak podudaranja u sekvencama pojedine sekvence za kompresiju i referentne sekvence. Za to, potrebno je napraviti hash tablicu pojedinih k-mera u referentnoj sekvenci. Ovaj postupak funkcionira kao posmični prozor koji ide nad nizom te onda izračunava sažetak, odnosno hash vrijednost za pojedini k-mer. A, C, G i T se kodiraju kao 0, 1, 2 i 3, time se svako slovo može kodirati s 2 bita jer su 00, 01, 10, 11 vrijednosti koje predstavljaju te brojeve u oblicima bitova. Izračunava se hash vrijednost za prvi k-mer u prozoru na način da se gleda vrijednost k-mera tako da su vrijednosti znaka koji je prije na značajnijem mjestu, to se ostvaruje posmakom za dva bita ulijevo kad se dodaju znakovi. Kreiranje hash tablica odvija se pomoću dva polja: H i L, koje se prije početka kreacije tablice moraju inicijalizirati. H tablica je veličine mogućih hash vrijednosti što

ovisi o duljini k-mera, a L tablica veličine svih mogućih indeksa k-mera u sekvenci, tj. obadvije vrijednosti ovise o duljini k-mera, ali se računaju po različitim formulama. Nakon izračuna sažetka k-mera, recimo da je njegova vrijednost v, njegov indeks se pohranjuje u polje H pod indeksom v, a originalna vrijednost koja je bila u H se sprema u L na indeksu i. Time efektivno polje H drži zadnji indeks pojave k-mera s određenim sažetkom, a L „pokazivač“ na prethodni indeks čime to polje efektivno postaje lista s pokazivačima na prethodni indeks k-mera s istom vrijednosti sažetka. Time se na efektivan način kreira hash struktura koja će znatno ubrzati proces pronalaženja najduljih podnizova koji se podudaraju među dvije sekvence znakova. U procesu pronalaženja najduljih podnizova koji se podudaraju, za svaki se k-mer neke sekvence za kompresiju računa sažetak na isti način, s posmakom, kao i kod izrade hash tablice. Nakon izračuna sažetka se gleda u tablicu H, ako je zapis ekvivalentan onom kod inicijalizacije, u ovom slučaju -1, očito je po procesu izrade hash tablice da onda ne postoji u referentnoj sekvenci k-mer, odnosno podniz, koji se podudara s danim podnizom. Stoga se taj dio sekvence, taj znak, sprema u informacije o znakovima koji se ne podudaraju. U suprotnom, ako postoji indeks u tablici H, postoji bar jedan k-mer koji se podudara između dva niza i onda se ulazi u proces pronalaska najduljeg podudaranja. Tijekom ovog se procesa prolazi kroz sve relevantne k-mere pomoću polja L te se gleda za koju pojavu k-mera se može naći najdulji zajednički niz između referentne sekvence i one koja se komprimira. Informacije se spremaju kao trojka: pozicija (opet relativna u odnosu na prošlu poziciju), duljina tog najvećeg pronađenog podudaranja i niz znakova koji se ne podudaraju. Kada algoritam za kompresiju primi više datoteka, automatski prelazi u način batch kompresije. Ovdje se rezultati prve razine podudaranja koriste za daljnje podudaranje u drugoj razini, kako bi se dodatno smanjili zahtjevi na memorijsko zauzeće korištenjem više sekvenci za pronalazak podudaranja. Podudaranje drugog reda, kao i ono prvog reda, također koristi hash podudaranje, ali umjesto načina koji je implementiran u prvom redu (izvlačenje sažetka iz kodiranih baza) koristi složenije entitete koji uključuju poziciju, duljinu i nepodudarnosti, odnosno izlaz iz podudaranja prve razine, kompresija se izvodi po nizu struktura podataka za podudarnosti iz podudaranja prve razine. Prije tog mora se provesti pretraga najduljih zajedničkih nizova u svim predanim sekvencama za kompresiju. Kako bi se limitirala potrošnja memorije, može se postaviti postotak druge razine podudaranja, p, što znači da se p% sekvenci koje treba komprimirati koristi kao referentne sekvence druge razine. Prvo se stvara hash tablica podataka o podudaranju, outputa iz first-level matchinga. U kreaciju sažetka uključuju se sva tri elementa pojedinog objekta toga izlaza, dakle pozicija, duljina i svaku nepodudarnu bazu. Veliki prosti broj se koristi u množenju tih podataka

za sažetak. U slučaju da postoji konflikt, kao i u podudaranju prvog reda, koristi se odvojeno ulančavanje s H i L listama. Kod faze pretrage, nakon kreacije hash tablice, se na identičan način kao kod podudaranja prvog reda, naravno koristeći ispravnu hash funkciju, računaju sažeci i traži se identičan zapis u hash tablici. Međutim, identična hash vrijednost ovdje ne znači i identičan entitet jer preslikavanje između entiteta i hash vrijednosti u ovom slučaju nije jednoznačno za razliku od prve razine. Kada se pronade identična hash vrijednost, potrebno je provjeriti jesu li entiteti identični na razini samog entiteta, ne samo hash vrijednosti. Ako jesu isti, pretraga se nastavlja kao kod prve razine uzastopno dok se ne pronade nepodudarni entitet, a zatim se bilježe ID sekvence, pozicija i duljina podudarnog segmenta. Nakon pretrage svih referentnih sekvenci, najduži podudarni podniz uzima se za zamjenu i sprema se, ponovo, trojka ID sekvence, pozicija, duljina. Ako se identičan entitet ne pronade nakon pretrage svih referentnih sekvenci, segment se sprema kao nepodudaran segment. Neovisno o podudaranjima prvog i drugog reda odvija se treća glavna faza drugog dijela kompresije: podudaranje informacija o malim slovima. Na ovaj se način jednostavno smanjuje prostor za pohranu podataka na način da, ako za sekvencu za određen zapis o malim slovima taj zapis već postoji u zapisu o malim slovima referentne sekvence, pohranjuje se lokacija tog zapisa, inače ako ne postoji ostaje inicijalan zapis.

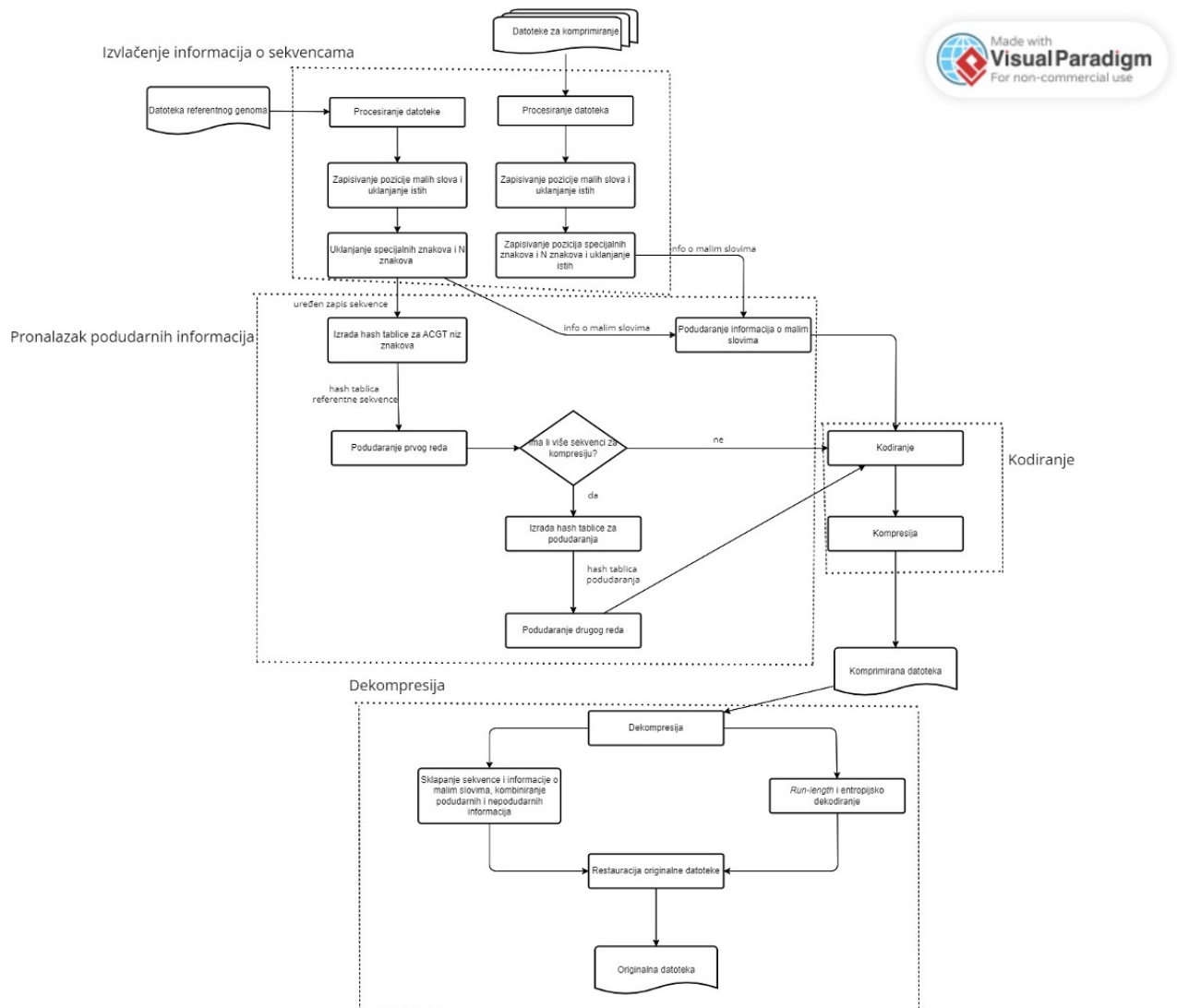
2.4. Kodiranje

S prethodno opisanim podacima kao izlazima iz faza podudaranja, bilo nizova, bilo malih slova, ulazi se u fazu kodiranja, same kompresije. Duljine linija i identifikatori se kodiraju s tzv. run-length encodingom, specijalni znakovi s entropijskim kodiranjem, a pozicije se kodiraju inkrementalnim kodiranjem. Sve se tri informacije kodiraju i komprimiraju u datoteku arhive.

2.5. Dekompresija

Dekompresija je jednostavniji postupak od kompresije te je u ovom slučaju samo reverzan postupak. Arhiva se prvo dekomprimira, te se s algoritmima opisanim u kodiranju dekodiraju pojedine informacije. Informacije o sekvencama i malim slovima se na reverzan način također rekonstruiraju u originalne vrijednosti koristeći i podudarne i nepodudarne informacije te se onda na taj način vraćaju u originalne datoteke genoma. Ovdje se kod obrnute faze podudaranja ne moraju koristiti hash vrijednosti jer su one

u fazi kompresije služile samo za pronalazak bržeg podudaranja, te tablice ovdje nisu bitne.



Slika 2.1: Odabir lokacije za niz AAC - prikaz po uzoru na [2]

3. Primjer izvođenja

Način rada implementiranog algoritma najbolje se vidi provođenjem na jednostavnom primjeru. Neka postoje 3 datoteke: Refseq, T1 i T2 gdje je Refseq datoteka referentne sekvence, a T1 i T2 datoteke za kompresiju. Neka je sadržaj datoteka sljedeći:

- Refseq:
Refseq
ACGAGTTCCTTCCggNnNnnXAC
CCCAAAGGGAAAttTaTccCtGAg

- T1:
T1
ACGAGCGCCCTTCCggAnNnAnXAC
CGCAaCGGGAAAttTaTGGCtGAg

- T2:
T2
ACGAGTTCCTTCCggNnNnnXAC
CGCAaCGGGAAAttTaTGGCtGAg

Datoteke ulaze u proces obrade gdje se one čitaju te obrađuju. Nakon toga slijedi zapisivanje i obrada pozicija malih slova te obrada ostalih znakova. Sukladno opisu algoritma, rezultat je sljedeći:

- Referentna sekvenca:
 - Sekvenca: ACGAGTTCCTTCCGGACCCCAAAGGGAAATTTATCCCT-GAG
 - Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (15, 3), (1, 1), (1, 2), (1, 1), (2, 1)
- T1:
 - Identifikator:

Sekvenca: ACGAGCGCCCTTCCGGAAACCGCAACGGGAAATTTTATGG-CTGAG

Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (7, 1), (7, 3), (1, 1), (4, 1), (2, 1)

Informacije o N: (16, 6)

Informacije o posebnim znakovima: (22, X)

• T2:

– Identifikator: T2

Sekvenca: ACGAGTTCCCTTCCGGACCGCAACGGGAAATTTTATGGCT-GAG

Informacije o malim slovima: (14, 2), (1, 1), (1, 3), (7, 1), (7, 3), (1, 1), (4, 1), (2, 1)

Informacije o N: (16, 6)

Informacije o posebnim znakovima: (22, X)

Na ovom se primjeru vide sve navedene tvrdnje vezano za ovaj dio. Referentna sekvenca sadrži manje informacija, obadviije varijante ekstrakcije sadrže sekvencu i informacije o malim slovima dok identifikator, info o N znakovima i posebnim znakovima imaju samo sekvence za kompresiju. Također je očito da se ti znakovi i brišu iz sekvence te da se radi o apsolutnim indeksima. Npr. (14, 2) označava da se prvi niz malih slova duljine dva mala slova nalazi 14 znakova od početka, sljedeći par (1, 1) označava da se onda jedno mjesto od zadnje pojave malog slova (14+2) nalazi jedno malo slovo itd. Ovako obrađeni podatci idu u fazu pronalaska najduljeg podudaranja, ali kako bi se to moglo provesti, prvo se mora napraviti hash tablica nad referentnom sekvencom. U ovom su primjeru k-meri veličine 2, dakle imamo dva slova koja kodiramo sa po dva bita, kako je opisano u prošlom poglavlju. Sukladno tome, imamo 4 moguća bita i 16 mogućih hash vrijednosti, stoga će polje H biti veličine 16 kako bi moglo zapisati zadnju pojavu svake hash vrijednosti pojedinog k-mera. Recimo, prvi k-mer bio bi AC, drugi CG, itd. . . Te bi se redom računale hash vrijednosti i zapisivale u H, a prethodna vrijednost od H išla bi u listu L kao pokazivač na prethodni indeks k-mera tog hash-a. Duljina polja L je broj k-mera, dakle duljinaSekvence – duljinaK-mera + 1. Vrijednosti polja H i L zapisane su ispod. Za primjer, uzmimo u obzir hash na indeksu 15, s vrijednosti 32. Ta nam informacija govori da je zadnja pojava k-mera čiji je sažetak 15 na indeksu k-mera 32. Ako se prati lista L, isti se taj k-mer nalazi na indeksima 32, 31, 30, 10, i 5. Također se može vidjeti da jedino za k-mer koji bi imao

hash vrijednost 6 ne postoji zapis u H polju, time jednostavno vidimo da taj k-mer nije pronađen u ulaznoj sekvenci. Tako je efikasno stvorena struktura podataka hash tablice za podudaranje prvog reda. 0 H: 28 1 H: 20 2 H: 40 3 H: 33 4 H: 16 5 H: 37 6 H: -1 7 H: 35 8 H: 41 9 H: 13 10 H: 25 11 H: 39 12 H: 34 13 H: 38 14 H: 4 15 H: 32

0 L: -1 1 L: -1 2 L: -1 3 L: -1 4 L: -1 5 L: -1 6 L: -1 7 L: -1 8 L: 7 9 L: -1 10 L: 5 11 L: 6 12 L: 8 13 L: 1 14 L: -1 15 L: 2 16 L: 0 17 L: 12 18 L: 17 19 L: 18 20 L: -1 21 L: -1 22 L: 21 23 L: 3 24 L: 14 25 L: 24 26 L: 15 27 L: 22 28 L: 27 29 L: -1 30 L: 10 31 L: 30 32 L: 31 33 L: -1 34 L: 29 35 L: 11 36 L: 19 37 L: 36 38 L: 9 39 L: -1 40 L: 26 41 L: 23

U nastavku algoritma, u podudaranju prvog reda sudjeluju i T1 i T2, svaki zasebno. Provodi se izračun vrijednosti sažetka svakog k-mera, kao i kod izrade hash tablice te se onda gleda postoji li zapis za taj k-mer u hash tablici. Ako ne, prvi se znak sprema u podatke o nepodudarnim informacijama, ako da ulazi se u proces pronalaska najduljeg podudaranja. Hash tablica izrađena u prethodnom koraku ovdje služi samo za brži pronalazak podudarnog k-mera bez da se pretražuje znak po znak. Za T1 niz izlaza iz podudaranja prvog reda je: (0, 5), (2, 10, 'CG'), (-1, 3, 'A'), (1, 3, 'G'), (1, 12, 'C'), (2, 5, 'GG'), A za T2: (0, 19), (1, 3, 'G'), (1, 12, 'C'), (2, 5, 'GG'). Za podsjetnik, kako radi ova faza, pogledajmo sekvence Refseq i T1: ACGAGTTCCCTTC-CGGACCCCAAAGGGAAATTTTATCCCTGAG ACGAGCGCCCTTCCGGAAC-CGCAACGGGAAATTTTATGGCTGAG Kod prvog podudaranja se računa sažetak za k-mer „AC“. Izračunom se dolazi do Hash vrijednosti 4 i toga da je zadnji k-mer indeksa 16 za ovaj sažetak iz liste H. Prati se lista L i uspoređuje se za svaku pojavu tog k-mera koliko je dug niz podudaranja. Po L, to su k-meri 16 i 0. Obzirom da se na indeksu 16 podudara samo taj k-mer, očito je da će onaj na indeksu 0 dati dulje podudaranje od 5 znakova. Tako se radi za svaki k-mer te je tu od značajne vrijednosti hash tablica koja omogućava brzi pristup svim pozicijama (indeksima) k-mera koji imaju isti sažetak. U sljedećem koraku, algoritam nije mogao pronaći podudaranje za CG te je onda zapisan u nepodudarne informacije i priložen uz sljedeću trojku. Algoritam se nastavlja dalje tom logikom do kraja. Pošto dolazi do kompresije više datoteka nastavlja se na podudaranje drugog reda. T1 i T2, s fokusom na specifične segmente i obradu umetanja, brisanja i nepodudarnosti. Za pozicije od 0-5, postoji točno podudaranje, predstavljeno identičnim znakovima u obje sekvence (označeno kao xxxxx radi jednostavnosti). Pomicanjem na pozicije od 5-19, poravnanje obrađuje razne modifikacije. Nepodudarnost na poziciji 3 je prikazana gdje obje sekvence imaju 'G' (T1: xxxxxG, T2: xxxxxG). Umetanje na poziciji 10 u T1 dodaje 'CG', što dovodi do razlike (T1: xxxxxGxCG, T2: xxxxxGx). Nadalje, brisanje 'A' na poziciji 3 u T1 rezultira produženom sekvencom (T1: xxxxxGxCGxx, T2: xxxxxGxxxx). Ko-

načno kombinirano poravnanje, prikazano u pojednostavljenom obliku, integrira ove promjene, ističući rafinirane sekvence s tim modifikacijama (T1: xxxxxGxCGxx..., T2: xxxxxGxxxx...). Ovaj postupak ilustrira detaljan korak lokalnog poravnanja u drugoj razini podudaranja HRCF algoritma, osiguravajući da su finije razlike sekvenci precizno predstavljene. Nakon ove faze, slijedi podudaranje informacija o malim slovima, kao podsjetnik priložene su informacije o malim slovima za Refseq i T1: (14, 2), (1, 1), (1, 3), (15, 3), (1, 1), (1, 2), (1, 1), (2, 1) (14, 2), (1, 1), (1, 3), (7, 1), (7, 3), (1, 1), (4, 1), (2, 1). U ovoj je fazi cilj pronaći podudarnost informacija malih slova. Koristi se indeksiranje od 1 za redni broj te 0 ako ne postoji zapis o Za T1, izlaz je sljedeći: Mismatched lowercase information: (7, 1), (7, 3), (4, 1), (2, 1) Matched lowercase information: 1, 2, 5, 7, 0, 0, 7, 0, 0 Vidimo za koje je indekse pronađen ekvivalentan zapis u referentnoj sekvenci te za koje nije, ti za koje nije ostaju u obliku kakvom jesu. Za T2 izlaz je: Mismatched lowercase information: (7, 1), (7, 3), (4, 1), (2, 1) Matched lowercase information: 1, 2, 3, 0, 0, 5, 0, 0.

4. Testiranje i analiza

U sklopu ovog projekta odlučili smo obaviti testiranje na nekoliko datoteka. Prva izvođenja smo odradili na datotekama R1.fa, T1.fa, T2.fa, T3.fa, T4.fa i T5.fa. Ove datoteke s vrlo jednostavnim parovima baza navedene su kao primjer u samom radu. Sljedeće testiranje smo izveli s datotekama X_ch1.fa i Y_ch1.fa. Ove datoteke služe kao dobar primjer za rad s malim slovima, specijalnim znakovima i N-ovima. Na kraju smo izveli testiranje s jednim manjim dijelom genoma Escherichia coli spremljenim u datoteci my-test.fna i kompresirali mnogo veći genom Escherichia coli, spremljen u GCF_000005845.2_ASM584v2_genomic.fna, koji sadrži oko 4 milijuna parova baza. Vremenski rezultati su vidljivi dolje u tablici. (Napomena: u tablici je cijeli genom Escherichia coli označen kao "Escherichia" kako ne bi bilo preklapanja.)

Kompresije	Naša implementacija [ms]	Originalna implementacija [ms]
T1 s R1	683.966	621.606
T1-T5 s R1	690.684	626.481
Y_chr1 s X_chr1	677.672	621.907
Escherichia s my-test	5424.945	3322.084

Tablica 4.1: Usporedba srednjih vremena za pojedini oblik kompresije između naše i originalne implementacije

Kao što je vidljivo iz rezultata u tablici, naša implementacija je u prosjeku manje od dvostruko sporija od originalne implementacije. Ovaj rezultat je ohrabrujući kada se uzme u obzir da smo većinom radili s dinamičkom memorijom.

5. Zaključak

Unatoč tome što je naša implementacija HRCM algoritma u prosjeku manje od dvostruko sporija od originalne implementacije, postizemo ohrabrujuće rezultate. Ovo je značajno jer smo većinom radili s dinamičkom memorijom, što može utjecati na brzinu izvođenja algoritma. Iako postoje prostori za poboljšanje u performansama, naši rezultati sugeriraju da je HRCM algoritam i dalje konkurentan izbor za obradu i kompresiju podataka. Dodatno istraživanje i optimizacija mogli bi dodatno unaprijediti performanse algoritma u budućnosti.

6. Literatura

- [1] F. Nositelji kolegija Bioinformatika 1, »Uvodno predavanje,« FER, [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/Bioinformatika_1%20-%20Uvodno_2023_2024.pdf. [Pokušaj pristupa 23. May 2024.].
- [2] N. k. Bioinformatika, »Dinamičko programiranje,« FER, [Mrežno]. Available: https://www.fer.unizg.hr/_download/repository/Bioinformatika_%201_2-3_predavanje_Dinami%C4%8Dko_programiranje.pdf. [Pokušaj pristupa 23. May 2024.].
- [3] Y. J. K. L. S. L. J. H. R. W. Haichang Yao, »HRCM: An Efficient Hybrid Referential Compression Method for Genomic Big Data,« 19 November 2019. [Mrežno]. Available: <https://www.hindawi.com/journals/bmri/2019/3108950/>. [Pokušaj pristupa 23. May 2024.].