SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA 1

Seminar

Smanjenje premještanja u Cuckoo filtru

Iva Bokšić
Karolina Mirković
Voditelj: *Mirjana Domazet-Lošo*

Zagreb, lipanj 2022.

SADRŽAJ

1. Uvod

Pohrana velikog broja podataka te pretraživanje podskupova u podatcima predstavljali su problem dugi niz godina. S napretkom tehnologije kapacitet memorije za pohranjivanje narastao je i do nekoliko TB. Time je prividno riješen problem pohrane podataka, ali je ostao neriješen problem pretraživanja. Da bi se pronašao određeni podskup u podatcima moralo bi se redom proći po cijeloj dužini pohranjenoga podatka što predstavlja vremenski vrlo zahtjevan posao. U nadi da se riješe problemi i pohrane i pretraživanje podataka javljaju se brojni algoritmi među kojima je i Cuckoo filter, koji je nastao iz prethodnog Bloom filtra kao njegova poboljšana verzija. Cuckoo filter pokazao je značajne prednosti u pretraživanju podskupova te brisanju podataka u usporedbi s Bloom filtrom iz kojeg je nastao. Iako je poboljšana verzija i sam Cuckoo filter pati od nekih problema. Kako bi pohranio podatke Cuckoo filter koristi takozvane pretince ili košare (*engl.* bucket) koje odabire metodom slučajnog odabira što može dovesti do njihove prepunjenosti te velikog broja premještanja čime mu se performanse drastično smanjuju.

Tema ovog seminara se temelji na implementaciji Cuckoo filtera i pokušaju poboljšanja njegovih performansi pri premještanju podataka. U drugom poglavlju objašnjen je princip rada orginalnog Cuckoo filtra te njegove poboljšane verzije koja smanjuje broj premještanja. Treće poglavlje razmatra rezultate koji su dobiveni pri analizi performansa izgrađenih Cuckoo filtera. Četvrto poglavlje donosi zaključak je li moguće izgraditi uspješan Cuckoo filter s manjim brojem premještanja i jesu li se performanse drastično promijenile. Popis literature korištene pri izradi rada se nalazi u petom poglavlju. Šesto poglavlje predstavlja sažetak cijelog seminara.

2. Cuckoo filter

Cuckoo filter (**CF**) predstavlja vjerojatnosnu strukturu podataka koja se koristi pri pohrani podataka te ispitivanju postoji li se u podatku određeni podskup. Prilikom ispitivanja moguće je postojanje lažno pozitivnih podudaranja odnosno moguće je da upit vrati da se podskup nalazi u podatku iako se on ne nalazi. S druge strane nije moguće postojanje lažno negativnih rezultata upita što znači da je nemoguće da podskup postoji u podatkovnoj strukturi, a da upit vrati da ga nije pronašao. Cuckoo filter odmogućava i brisanje podataka iz svoje stukture što ga razlikuje od Bloom filtra iz kojeg je sam i nastao. Iako može vratiti lažno pozitivne rezultate prostorna učinkovitost pohranjivanja podataka te brzina pronalaska podskupova mu omogućuju dobre performanse. Glavne četiri prednosti Cuckoo filtra su:

• podržano dinamičko dodavanje i uklanjanje stavki
• pružanje veće izvedbe pretraživanja
• laka implementacija za razliku od alternativa
• korištenje manje prostora od Bloom filtra ako je stopa lažno pozitivnih manja od 3%.

2.1. Podatkovna stuktura Cuckoo filtra

Cuckoo filter je dvodimenzionalna podatkovna stuktura koja se sastoji od košara ili pretinaca u koje se može unjeti unaprijed definiran broj podataka. Stuktura se može predočiti tablicom u kojoj redci predstavljaju košare, a broj stupaca odgovara zadanom broju podataka koje se u njega mogu pohraniti. CF zapravo ne pohranjuje izvorne podatke već koristi hash funkciju kako bi izračunao "otisak prsta" (*engl.* fingerprint) te također koristi hash funkciju kao bi odabrao broj pretinca odnosno poziciju unutar tablice u koju će podatak pohraniti. Prilikom konstruiranja CF veličina otiska određena je ciljanom stopom lažno pozitivnih rezultata. Manje stope lažno pozitivnih zahtijevaju dulje otiske jer je time moguće postići da više podataka ima različiti otisak.

2.2. Hash funkcija

Pri implementaciji hash-a korištena je softverska knjižnica OpenSSL koja se koristi za osiguravanje komunikacije preko računalnih mreža od prisluškivanja ili potrebe za identifikacijom stranke na drugom kraju. [1] OpenSSL na jednostavan način omogućava implementaciju algoritma SHA-1 (engl. Secure Hash Algorithm 1) koji ulaz pretvara u 160-bitnu vrijednost. U ovom projektu korišteno je samo 64 bitova koji su podijeljeni na dva dijela od 32 bita. Prvi dio je korišten kao prva hash vrijednost koju algoritam vraća, a druga vrijednost je korištena kao tzv. "otisak prsta" (engl. fingerprint) koji je objašnjen u prethodnom potpoglavlju. Drugi hash kojeg funkcija vraća računa se kao XOR vrijednost prvog hash-a i "otiska prsta".

2.3. Umetanje podataka u Cuckoo filter

Kako bi se pohranio podatak u CF najprije je potrebno izračunati njegov otisak i broj pretinca pomoću hash funkcije. Svakom podatku dodijeljena su dva moguća pretinca u koje ih je moguće pohraniti. Kako bi se dobili brojevi pretinaca potrebno izračunati hash vrijednost od samog podatka x kako bi se dobila jedna pozicija te tu istu hash vrijednost provući kroz XOR operator zajedno s hash vrijednosti otiska podatka kako bi se dobila alternativna pozicija za spremanje podatka:

$$h_1(x) = hash(x) \tag{2.1}$$

$$h_2(x) = h_1(x) \oplus hash(otisak\ od\ x) \tag{2.2}$$

Hashiranje otiska podatka osigurava da se pozicije pretinaca nalaze na različitim mjestima unutar tablice čime se smanjuje broj kolizija prilikom umetanja te se poboljšava iskorištenost tablice. Kada se otisak ne bi hashirao alternativna pozicija pretinca nalazila bi se samo na maloj udaljenosti od orginalne pozicije. Važno svojstvo koje osigurava XOR operator iz izraza **(2.2)** je to da se alternativna pozicija uvijek može izračunati ako trenutnu poziciju provučemo kroz operator ekskluzivno ili zajedno sa otiskom podatka. Tada vrijede jednakosti:

$$h_1(x) = h_2(x) \oplus hash(otisak\ od\ x) \tag{2.3}$$

$$h_2(x) = h_1(x) \oplus hash(otisak\ od\ x) \tag{2.4}$$

Kako brojevi pretinaca ne bi prelazili unaprijed definiran ukupan broj pretinaca m, potrebno je za vrijednosti h_1 i h_2 izračunati modulo m da se dobiju konačne pozicije pretinaca u tablici.

$$pozicija_1(x) = h_1(x) \mod m \tag{2.5}$$

$$pozicija_2(x) = h_2(x) \mod m \tag{2.6}$$

Za umetanje se koriste samo informacije unutar tablice koja sadrži otiske podataka što znači da se prave vrijednosti podatka ne koriste niti ih je potrebno dodato spremati. Moguće je da se pojave dva različita podatka *x* i *y* koja imaju isti otisak. Zato je potrebno prije umetanja provjeriti postoji li taj otisak već u tablici. Ako postoji struktura tablice se ne mijenja, a funkcija umetanja vraća informaciju da je podatak pohranjen. S obzirom na popunjenost pretinaca na pozicijama $pozicija_1$ i $pozicija_2$ razlikuju se tri varijante umetanja podataka.

2.3.1. Oba pretinca su slobodna

Najjednostavniju varijantu predstavlja slučaj kada su oba pretinca slobodna odnosno imaju praznih mjesta u koje se može pohraniti otisak podatka. U tom slučaju slučajnim odabirom se bira koji će to pretinac biti. U pretinac se pohranjuje novi podatak te funkcija umetanja vraća informaciju da je novi podatak uspješno pohranjen.

2.3.2. Jedan pretinac je slobodan

U slučaju da samo jedan pretinac ima slobodnih mjesta za pohranu on se odabire i u njega se smješta otisak podatka. Funkcija umetanja također vraća informaciju da je novi podatak uspješno pohranjen.

2.3.3. Oba pretinca su popunjena

Najsloženiji slučaj je kada su oba pretinca popunjena. Tada se slučajnim odabirom izabire jedan od dva puna pretinca te iz njega izbacuje jedan otisak te se na njegovo mjesto pohranjuje novi otisak. Za izbačeni otisak računa se alternativna pozicija na koju se on može spremiti prema izrazu:

$$h_i(r) = pozicija(x) \oplus hash(otisak\ od\ r) \tag{2.7}$$

gdje *x* predstavlja novi podatak koji je potrebno pohraniti u podatkovnu strukturu, a *r* označava podatak koji je izbačen iz pretinca kako bi se *x* u njega spremio. Pozicija na koju se *r* treba pokušati spremiti računa se prema izrazu **(2.6)**.

$$pozicija(r) = h_i(r) \mod m \tag{2.8}$$

Ako pretinac na izračunatoj poziciji *pozicija(r)* nije pun podatak *r* se pohranjuje u njega i algoritam završava. U slučaju da je pun algoritam nastavlja petlju u kojoj izbacuje jedan podatak iz pretinca računa njegovu alternativnu poziciju i pokušava ga smjestiti u pretinac novoizračunate pozicije. Ovisno je li pretinac novoizračunate pozicije slobodan ili ne algoritam se prekida ili nastavlja pretragu sve dok ne pronađe slobodni pretinac. Ovakav sistem nažalost ima i negativnu stranu, a to je da može zapeti u beskonačnoj petlji gdje se podaci redom izbaciju iz pretinaca, a sve alternativne pozicije pretinaca su pune. Kako bi se to spriječilo potrebno je definirati unaprijed zadani broj koji se označava kao **MNK** te predstavlja najveći broj dozvoljenih premještanja podataka unutar stukture da bi se novi podatak pohranio. Ako se dođe do MNK vrijdnosti, zadnje izbačeni podatak iz pretinca se izbacuje iz cijele podatkovne strukture CF-a te funkcija umetanja vraća informaciju da podatak nije uspješno umetnut.

2.4. Pretraga podataka u Cuckoo filtru

Kako bi se provjerilo je li podatak sadržan u tablici potrebno je izračunati njegov otisak pomoću hash funkcije te moguće pozicije pozicija_1 i pozicija_2 prema izrazima **(2.5)** i **(2.6)**. Potom je potrebno proći po pretincima na tim pozicijama i usporediti tamo spremljene otiske s otiskom samog podatka. Ako se otisak nalazi znači da je podatak već spremljen.

2.5. Smanjenje premještanja podataka

Orginalni Cuckoo filter je implementiran da prilikom umetanja ako su oba pretinca slobodna slučajnim odabirom odabire u koji će pretinac pohraniti otisak. To može dovesti do toga da se određeni pretinac češće odabire i time prije popuni. Posljedica bržeg popunjavanja pojedinih pretinaca je veći broj premještanja podataka u slučaju da su oba pretinca popunjena. Kako bi se smanjio ukupan broj premještanja, odnosno slučaj da su oba pretinca puna napravljena je poboljšana implementacija Cuckoo filra koja pri unosu otiska bira pretinac koji ima manje otisaka pohranjenih. Time se pre-

tinci ravnomjerno popunjavaju te je slučaj popunjenosti oba pretinca rjeđi te je ukupno premještanje otisaka rjeđe. Kako bi se usporedile performanse rada orginalnog CF i poboljšane verzije napravljeni su testovi umetanja i pretraživanja podataka po strukturi filtra.

2.6. Uporaba Cuckoo filtra za traženje podnizova

Cuckoo filter pronašao je veliku primjenu pri obradi i analizi velikog skupa podataka. Stvarni zapis podatka nije moguće vratiti iz njegovog otiska, ali često to nije niti potrebno. Kod velikog skupa podataka često je zanimljivo tražiti postoje li manji podskupovi podataka u orginalnom podatku ili se pokušavaju usporediti dva velika skupa podataka tako da im se odaberu slučajni nizovi određene duljine na istim lokacijama i usporede primjenom CF i otisaka u pretincima. Vrlo često se ne uspoređuju cijeli skupovi podataka već se odredi broj podnizova na slučajnim pozicijama koji se mora podudarati i onda se sa određenom vjerojatnošću može tvrditi da su skupovi isti. Bioinformatika je znanosti koja usko povezuje biologiju i računarstvo te često obrađuje različite genome živih bića. Kako se genomi sastoje i od nekoliko milijuna nukleotida koriste se različite strukture podataka kako bi se pohranile informacije o genomu na što bolji način među kojima je i Cuckoo filter.

2.6.1. Escherichia coli genom

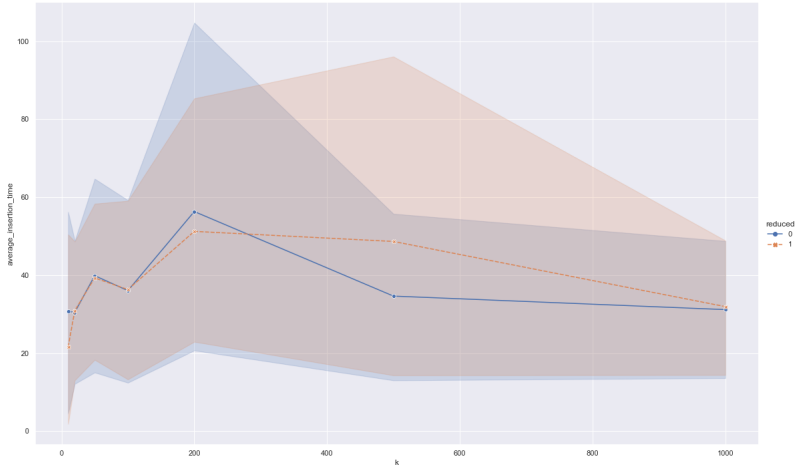
Kako bi se istestirala implementirana i poboljšana verzija Cuckoo filtra korišten je genom Escherichie coli. Escherichie coli je često korištena pri istraživanjima zbog svoje jednostavnosti strukture i genetskog materijala te su brojni radovi napisani na tu temu. Pri testiranju algoritama korišteni su takozvani "**k-mer**i" koji predstavljaju podnizove duljine k sadržane unutar biološke sekvence genoma E. coli.

3. Rezultati

Kako bi se opravdala teza da poboljšani CF s smanjenim brojem premještanja (CF-SBP) ima bolje performanse od orginalnog CF napravljena su brojna testiranja nad genomom E. coli koja će se detaljno objasnit. Zastavica "*reduced*" s vrijednošću 1 predstavlja poboljšanu verziju CF te u grafovima ta verzija prikazana narančastom bojom. Zastavica "*reduced*" s vrijednošću 0 predstavlja orginalnu verziju CF-a koji je u grafovima prikazan s plavom bojom.

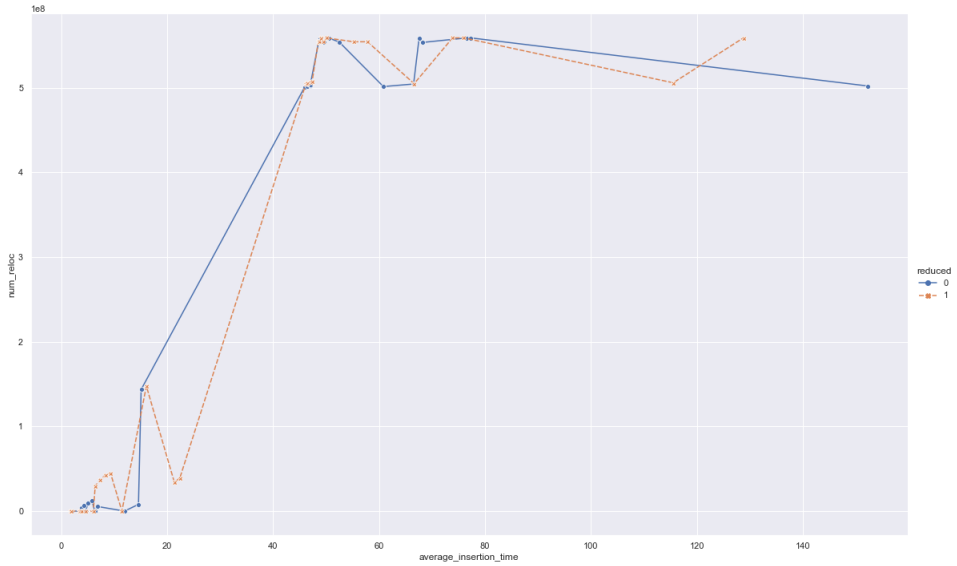
3.1. Rezultati umetanja k-mera

Graf **3.1** prikazuje da se za male vrijednosti duljine k-mera prosječno vrijeme unosa ne razlikuje za pojedine verzije CF-a. S porastom duljine CF verzija sa smanjenim brojem premještaja prosječno duže pohranjuje podatke. To se može objasniti činjenicom da su pretinci jednako popunjeni te ih je potrebno cijele pretražiti kako bi se ustvrdilo radi li se o otisku koji je već pohranjen što rezultira dužim trajanjem za razliku od CF-a gdje je moguća situacija da je jedan pretinac već popunjen te se otisak sprema u nepopunjeni.



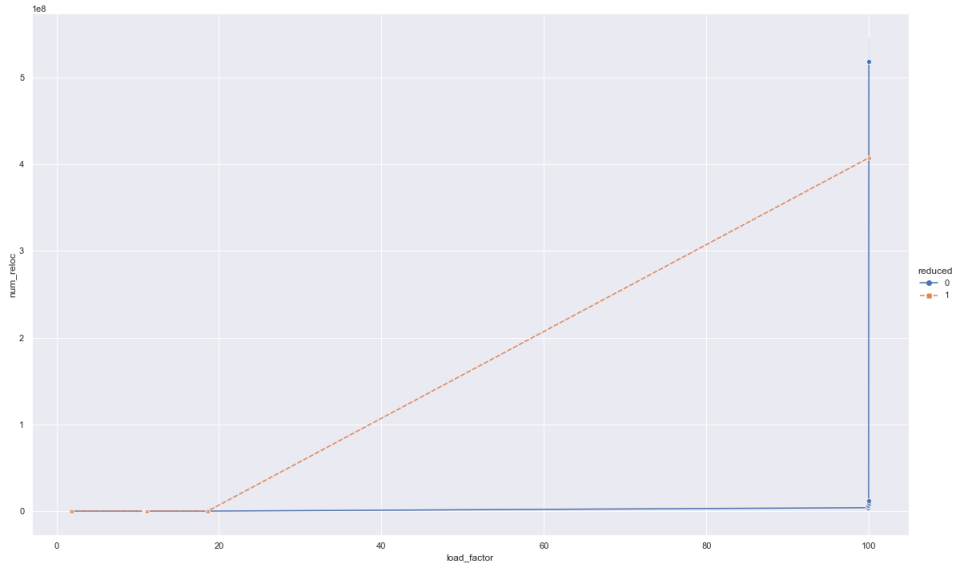
Slika 3.1: Prosječno vrijeme unosa podataka u ovisnosti s duljinom k-mera

Graf **3.2** prikazuje s porastom broja premještanja raste i prosječno vrijeme potrebno za pohranu k-mera. U većem dijelu grafa linija CF-SBP se nalazi ispod linije orginalnog CF-a što znači da za jednako prosječno vrijeme CF-SBP napravi manji broj premještaja.



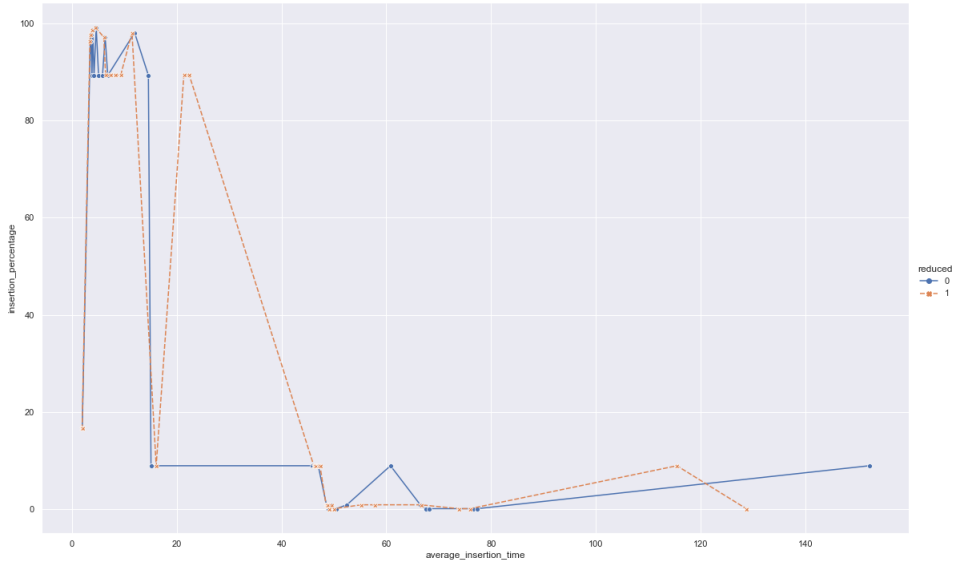
Slika 3.2: Prosječno vrijeme unosa podataka u ovisnosti o broju premještaja

Graf **3.3** prikazuje ovisnost popunjenosti tablice o ukupnom broju premještanja. Vidi se da s porastom broja premještanja raste i popunjenost tablice. Također je vidljivo da je postignut rezultat jednake popunjenosti 100%, ali CF-SBP je to postigao s manjim brojem premještanja.



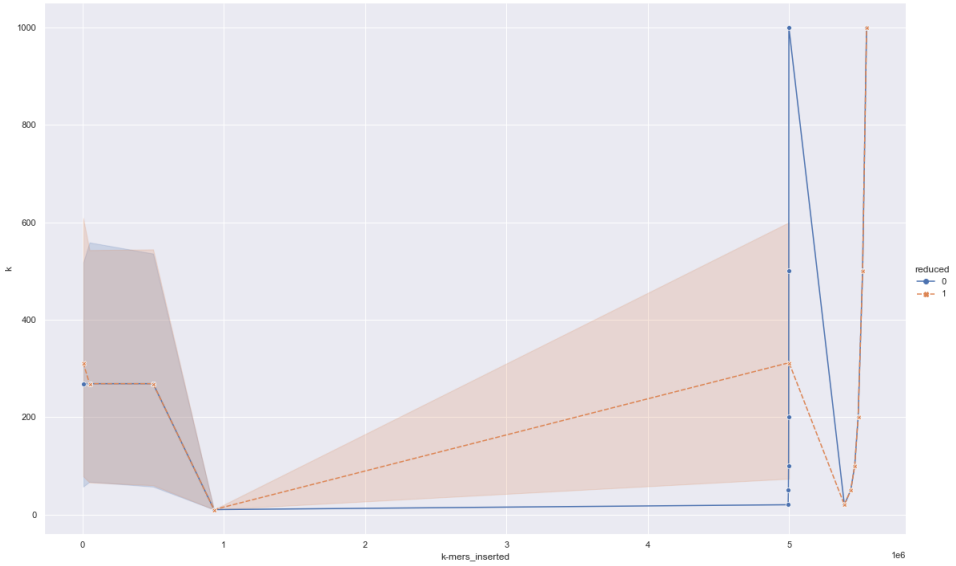
Slika 3.3: Postotak popunjenosti Cuckoo tablice u ovisnosti o broju premještanja

Graf **3.4** prikazuje da CF-SBP u većini slučajeva za jednako prosječno vrijeme postiže veći postotak pohranjenih k-mera.



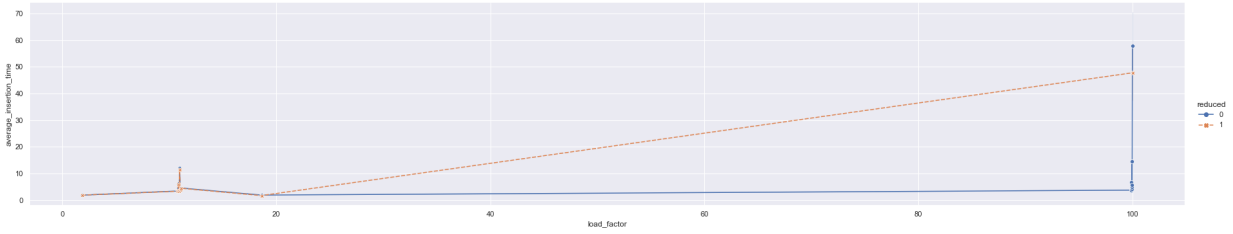
Slika 3.4: Postotak pohranjenih k-mera u ovisnosti o prosječnom vremenu pohranjivanja

Iz grafa **3.5** može se vidjeti da CF-SBP za jednaki broj pohranjenih podataka pohranjuje k-mere sa većom duljinom.



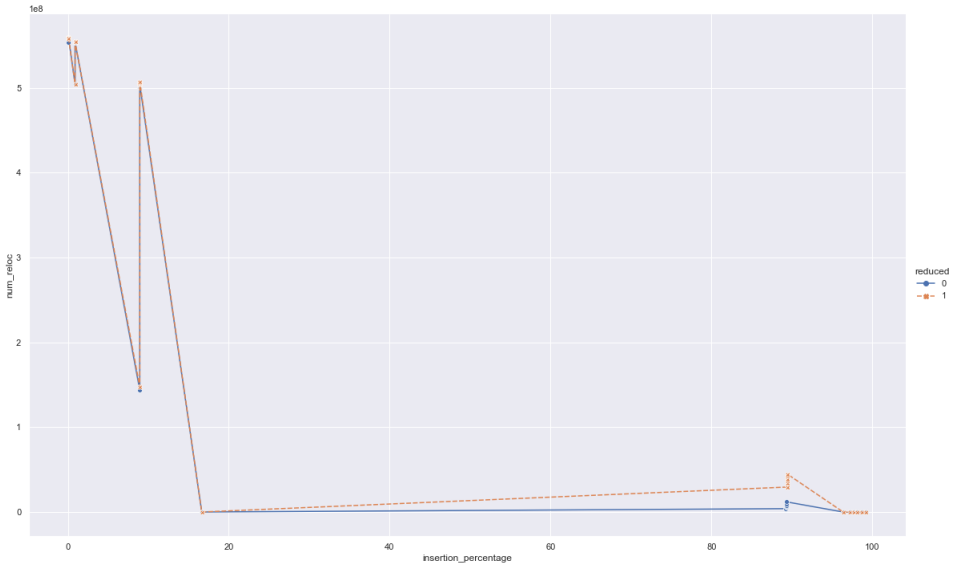
Slika 3.5: Duljina k-mera u ovisnosti o broju pohranjenih k-mera

Graf **3.7** prikazuje da za manju popunjenost tablice prosječno vrijeme pohranjivanja je jednako, da bi maksimalnu popunjenost od 100 % CF-SBP postigao za 12 ms brže od CF-a.



Slika 3.6: Popunjenost Cuckoo tablice u ovisnosti o prosječnom vremenu pohranjivanja

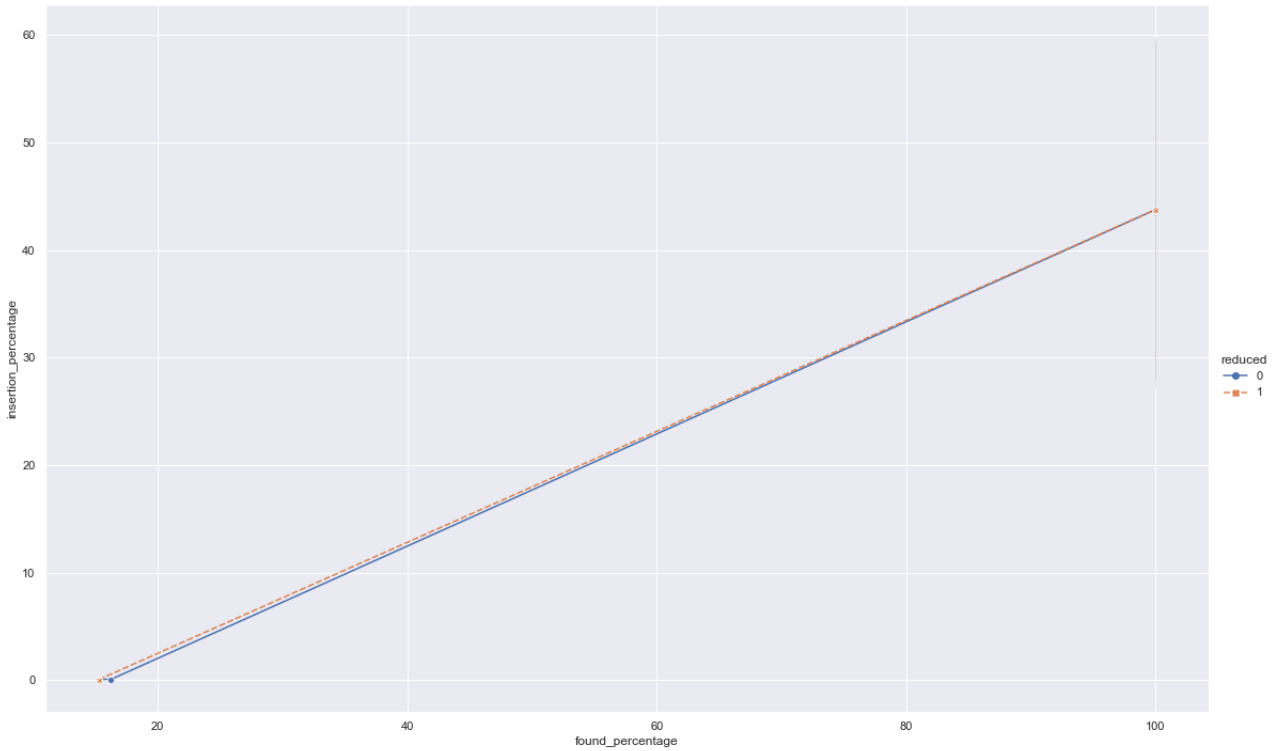
Graf **3.8** prikazuje da s padom broja premještaja se poveća broj umetnutih k-mera u tablici. Nema značajne razlike između verzija CF-a.



Slika 3.7: Broj premještanja k-mera u ovisnosti o postotku pohranjenih k-mera

3.2. Rezultati pretrage k-mera

Pri pretraživanju k-mera u ovisnosti o vezriji CF-a ne može se uočiti značajna razlika. S povećanjem postotka umetnutih k-mera poveća se i broj pronađenih k-mera.

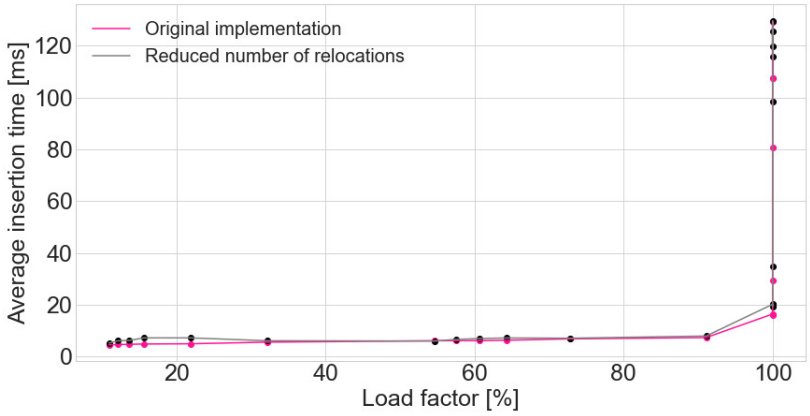


Slika 3.8: Postotak unesenih k-mera u ovisnosti o postotku pronađenih k-mera

3.3. Istraživanje rezultata za k-mer fiksirane duljine

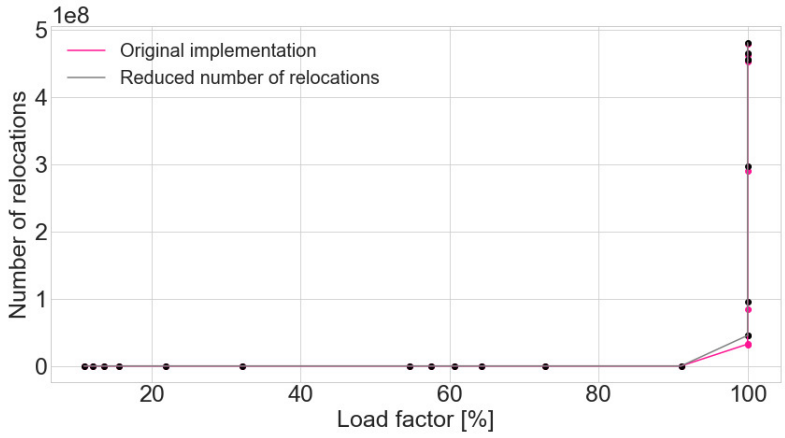
Provedeno je istraživanja nad algoritmima koristeći k-mer fiksirane duljine iznosa 100. U ovisnosti o veličini tablice odnosno broja pretinaca uspoređene su neke od performansi.

Graf **3.9** prikazuje da je prosječno vrijeme za pohranu podatka u većini slučajeva jednako za obe implementacije. Na nekim mjestima uočava se da CF-SBP duže vremena pohranjuje što se može objasniti činjenicom da je ispitivanje popunjenosti pretinaca vremenski složenija radnja od slučajnog odabira pretinca.



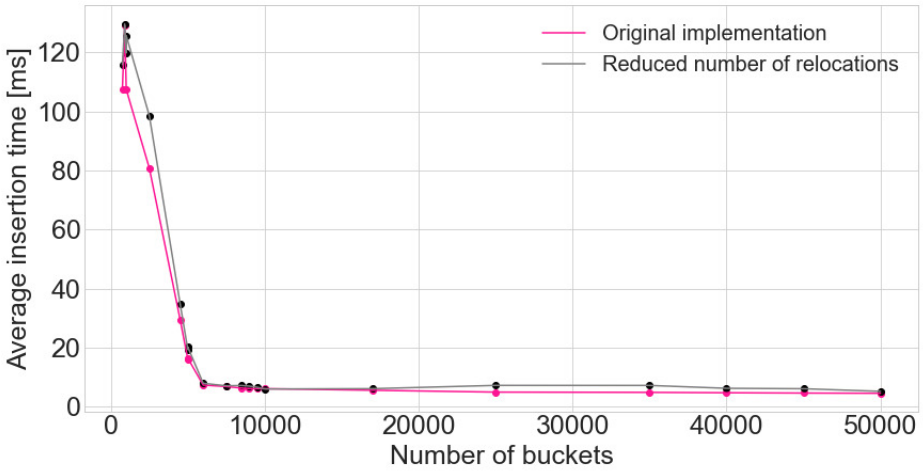
Slika 3.9: Popunjenost Cuckoo tablice u ovisnosti o prosječnom vremenu pohranjivanja

Iz grafa **3.10** uočava se da je popunjenost tablice ostvarena uz jednak broj relokacija oba algoritma.



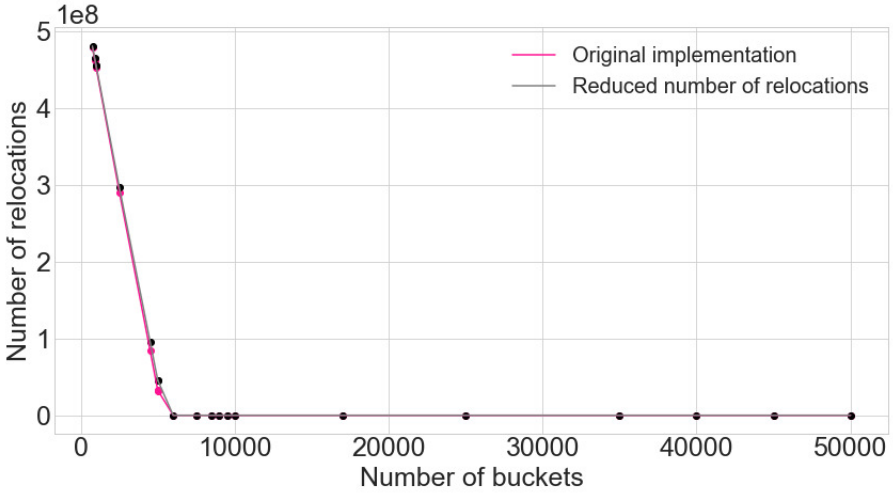
Slika 3.10: Postotak popunjenosti Cuckoo tablice u ovisnosti o broju premještanja

Na nekim mjestima na grafu **3.11** uočava se da CF-SBP duže vremena pohranjuje što se može objasniti činjenicom da je ispitivanje popunjenosti pretinaca vremenski složenija radnja od slučajnog odabira pretinca.



Slika 3.11: Broj pretinaca u ovisnosti o prosječnom vremenu pohranjivanja

Graf **3.12** prikazuje da je broj relokacija gotovo jednak neovisno o ukupnom broju pretinaca te se smanjuje što je više pretinaca.



Slika 3.12: Broj pretinaca u ovisnosti o broju relokacija

Ovo istraživanje najmjerno je izabrano i stavljeno u rad jer prikazuje slučaj kada uz veličinu k-mera od iznosa 100 nije postignut značajan poboljšani performans poboljšane implementacije Cuckoo filtra.

4. Zaključak

U sklopu ovog projekta istražen je problem pohrane i pretraživanja skupa podataka. Obrađena je tema CF (engl. Cuckoo Filter) filtera te je izrađena verzija koja smanjuje ukupan broj premještanja. Analizirane su performanse Cuckoo filtera koje su detaljno opisane u 3. poglavlju. Može se zaključiti da je verzija sa smanjenim brojem premještanja pokazala da za jednako prosječno vrijeme pohrane k-mera broj premještanja je manji. Također uspjela je postići jednaku popunjenost tablice za manji broj premještaja. Za jedanko prosječno vrijeme CF-SBP postiže veći postotak pohranjenosti k-mera. Također je maksimalnu popunjenost od 100% CF-SBP postigao za 12 ms brže od CF-a. Može se zaključiti da Cuckoo filter sa smanjenim brojem premještaja postiže bolje performanse od običnog Cuckoo filtra.

5. Literatura

[1] https://en.wikipedia.org/wiki/OpenSSL

[2] *The Power of Better Choice: Reducing Relocations in Cuckoo Filter*
https://ieeexplore.ieee.org/document/8885169
Wang et al. 2019.

[3] *Cuckoo Filter: Better Than Bloom*
https://www.cs.cmu.edu/ binfan/papers/login_cuckoofilter.pdf
Fan et al. 2013.

[4] *Cuckoo Filter: Practically Better Than Bloom*
http://www.cs.cmu.edu/
Fan et al. 2014

6. Sažetak

U sklopu ovog projekta istražen je problem pohrane i pretraživanja skupa podataka. Obrađena je tema CF (engl. Cuckoo Filter) filtera. Prvo je opisan algoritam, zatim je objasnjena implementacija. Na kraju su prikazane i analizirane performanse izradenog filtera.