SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Poboljšanje Bloom filtra korištenjem k-mer Bloom filtra

Magdalena Halusek i Katarina Prgeša
Voditelj: *Mirjana Domazet-Lošo*

Zagreb, siječanj 2020.

SADRŽAJ

1. Uvod

Bloom filtar je podatkovna struktura koja memorijski efikasno određuje nalazi li se traženi element u određenom setu ili ne [1]. Glavni nedostatak Bloom filtra je što odgovor na upit postojanja elementa u setu može biti ili da se definitivno ne nalazi u setu ili da se možda nalazi. Drugim riječima, Bloom filtar ne može dati pozitivan odgovor s potpunom sigurnošću. Razlog tome je što se za unošenje elemenata u Bloom filtar i za njihovo traženje koriste hash funkcije. Prilikom unošenja i traženja elemenata, hash funkcije vraćaju indekse koji se kod unošenja koriste za postavljanje '1' na pozicijama koje odgovaraju dobivenim indeksima. Kod provjere postojanja elementa u nizu, gledaju se vrijednosti na dobivenim indeksima. Ako je na jednom od dobivenih indeksa postavljena '0', tada Bloom filtar vraća negativni odgovor na postojanje elementa u nizu, a ako je na svakom od indeksa vrijednost '1', tada se u nizu nalazi traženi element ili neki drugi element koji bi dao istu kombinaciju indeksa. Mjera kojom se izražava nesigurnost pozitivnog odgovora koji daje Bloom filtar naziva se lažno-pozitivna mjera te će se u nastavku koristiti kratica FPR koja dolazi od engleskog naziva (*False Positive Rate*). U nastavku će biti prikazano kako smanjiti FPR koji daje Bloom filtar korištenjem k-mera, gdje k-mer označava jedan niz od k elemenata (A, C, G i T) [3]. Pošto se u bioinformatici radi s velikom količinom podataka, u interesu je smanjiti memorijsko opterećenje te će u nastavku biti opisano i kako se korištenjem k-mera mogu poboljšati memorijske performase.

2. Smanjenje FPR-a korištenjem k-mera

Kod korištenja k-mera, ulazni niz se obrađuje na način da se od početka sekvence uzme jedan podniz duljine k (jedan k-mer) i stavi se u Bloom filtar. Sljedeći podniz se dobije na način da se od početka ulaznog niza pomakne za jedan znak prema kraju niza i stavi se u Bloom filtar. Bloom filtar se tako puni sve dok se ne dođe do kraja ulaznog niza. Glavna ideja kod smanjenja FPR-a stoji iza činjenice da se u Bloom filtru nalaze odabrani k-mer te k-mer koji mu neposredno prethodi i k-mer koji ga neposredno slijedi [2]. U daljnjem tekstu, k-mer koji neposredno prethodi nekom drugom k-meru će se zvati lijevi susjed, a k-mer koji neposredno slijedi isti k-mer će se zvati desni susjed. Dakle, kao dodatna provjera postojanje odabranog k-mera u Bloom filtru svakako može poslužiti i provjera postojanja desnog ili lijevog susjeda promatranog k-mera. Navedena provjera se može podijeliti na dva pristupa:

- jednostrani k-mer Bloom filtar
- dvostrani k-mer Bloom filtar.

2.1. Jednostrani k-mer Bloom filtar

Jednostrani k-mer Bloom filtar se temelji na provjeri koja mora rezultirati pozitivnim odgovorom u barem 2 slučaja:

- promatrani k-mer se nalazi u Bloom filtru
- jedan od susjeda promatranog k-mera se nalazi u Bloom filtru

Za potrebe navedene provjere, potrebno je rekonstruirati sve moguće susjede promatranog k-mera te za svaki od njih posebno provjeriti njihovo pojavljivanje u Bloom filtru. Lijevi susjedi se mogu dobiti dodavanjem jedne baze na početak promatranog k-mera te oduzimanjem jedne baze koja se nalazi na posljednjem indeksu promatranog

k-mera. Desni susjedi se mogu dobiti oduzimanjem jedne baze na početku promatranog k-mera i dodavanjem jedne baze na kraj k-mera.
Primjer 1. Ako je odabran 8-mer, čija se prisutnost u Bloom filtru želi provjeriti, AACCTTGG, njegovi lijevi susjedi bi bili:

- AAACCTTG
- CAACCTTG
- GAACCTTG
- TAACCTTG

Desni susjedi navedenog 8-mera bi bili:

- ACCTTGGA
- ACCTTGGC
- ACCTTGGG
- ACCTTGGT

Opisani postupak se može prikazati algoritmom 1.

Algoritam 1 Jednostrani k-mer Bloom filtar [2]

```
1: function ONE_SIDED_KBF_CONTAINS(query)
2:     if BF.CONTAINS(query) then
3:         return CONTAINS_SET(neighbour_set(query))
4:     return false
5:
6: function CONTAINS_SET(set)
7:     for kmer ∈ set do
8:         if BF.CONTAINS(kmer) then
9:             return true
```

2.2. Dvostrani k-mer Bloom filtar

Za razliku od jednostranog k-mer Bloom filtra, dvostrani k-mer Bloom filtar mora rezultirati pozitivnim odgovorom u 3 slučaja:

- promatrani k-mer se nalazi u Bloom filtru
- lijevi susjed k-mera se nalazi u Bloom filtru
- desni susjed k-mera se nalazi u Bloom filtru

Pošto kod dvostranog k-mer Bloom filtra oba susjeda promatranog k-mera moraju biti prisutni, dvostrani k-mer Bloom filtar ima jednu iznimku, a to je u slučaju kada se promatrani k-mer nalazi na rubu sekvence. Takvi k-meri se nazivaju rubni k-meri te moraju biti spremljeni zasebno. Ako se radi o iznimci, dvostrani k-mer Bloom filtar mora dati pozitivne odgovore u sljedeća 3 slučaja:

- promatrani k-mer se nalazi u Bloom filtru
- jedan od susjeda se nalazi u Bloom filtru
- promatrani k-mer je rubni k-mer

Opisani postupak se može prikazati algoritmom 2.

Algoritam 2 Dvostrani k-mer Bloom filtar [2]

```
1: function TWO_SIDED_KBF_CONTAINS(query)
2:     contains_left ← CONTAINS_SET(left_neighbour_set(query))
3:     contains_right ← CONTAINS_SET(right_neighbour_set(query))
4:     if contains_left == 1 and contains_right == 1 then
5:         return true
6:     if contains_left == 1 or contains_right == 1 then
7:         if EDGE_K-MER_SET.CONTAINS(query) then
8:             return true
9:     return false
```

3. Princip preklapanja sekvenci za smanjenje k-mer setova

Korištenje k-mera u obradi velike količine podataka može biti memorijski zahtjevno te se zbog toga radi na smanjenju broja k-mera koji moraju biti spremljeni u Bloom filtar. Jedan od načina na koji se to može postići je korištenje svojstva preklapanja sekvenci. Prije objašnjenja navedenog načina, potrebno je definirati sljedeće oznake:

- U - set k-mera koje treba pohraniti
- k - jedan k-mer iz U
- L_k - set k-mera koji se u U nalaze prije k
- R_k - set k-mera koji se u U nalaze nakon k

Smanjenje broja k-mera koje treba pohraniti se postiže na način:
Ako sigurno postoji jedan k-mer u L_k i jedan k-mer u R_k, tada se može potvrditi postojanje k-mera k iz prisutnosti $v \in L_k$ i $w \in R_k$, bez potrebe za pohranom k.
Primjer 2. Ako je promatrani 8-mer k = AACCTTGG te njegovi susjedi v = TAACCTTG i w = ACCTTGGA, tada se do željenog k-mera k može doći kombinacijom v i w. Kako bi se stvarno moglo kombinacijom v i w doći do k postoji jedan uvjet za koji je potrebno definirati dodatne oznake:

- P_{vk} - skup pozicija lijevih susjeda v
- A_{kw} - skup pozicija desnih susjeda w
- $S_k(v, w)$ - udaljenost između odgovarajućih pozicija lijevih i desnih susjeda
- s - broj k-merova koje je dopušteno preskočiti

$S_k(v, w)$ je opisan s 3.1.

$$S_k(v, w) = \{i_w - i_v \mid i_v \in P_{vk}, i_w \in A_{kw}\} \tag{3.1}$$

Definiranjem potrebnih oznaka uvjet se može opisati s 3.2.

$$min(S_k(v, w)) \leq s \tag{3.2}$$

Navedeni uvjet se može rastaviti na dva problema.

Problem 1: Opušteni problem smanjenja k-mer setova (algoritam 3)

Iz zadanog skupa k-mera U treba pronaći mali podskup $K \subset U$ za sve $k \in U$, bilo da je $k \in K$ ili postoje $v \in K \cap L_k$ i $w \in K \cap R_k$ za koje vrijedi $min(S_k(v, w)) \leq s$.

Problem 2: Strogi problem smanjenja k-mer setova (algoritam 4)

Iz zadanog skupa k-mera U treba pronaći mali podskup $K \subset U$ za sve $k \in U$, bilo da je $k \in K$ ili postoje $v \in K \cap L_k$ i $w \in K \cap R_k$ za koje vrijedi $min(S_k(v, w)) = s$.

Algoritam 3 Opušteni problem smanjenja k-mer setova [2]

```
1:  function DECIDE_PRESENT(query, contains_left, contains_right)
2:      if contains_left == 1 and contains_right == 1 then
3:          return true
4:      if contains_left == 1 or contains_right == 1 then
5:          if EDGE_K-MER_SET.CONTAINS(query) then
6:              return true
7:      return false
8:
9:  function RELAXED_CONTAINS_NEIGHBOURS(query, l_dist, r_dist)
10:     contains_left ← CONTAINS_SET( ⋃_{i≤l_dist} s_distant_left_neighbour_set(query, i))
11:     contains_right ← CONTAINS_SET( ⋃_{i≤r_dist} s_distant_right_neighbour_set(query, i))
12:     return DECIDE_PRESENT(query, contains_left, contains_right)
13:
14: function RELAXED_CONTAINS(query, s)
15:     if BF.CONTAINS(query) then
16:         if RELAXED_CONTAINS_NEIGHBOURS(query, s, s) then
17:             return true
18:     else
19:         for i ← 0 to s − 1 do
20:             if RELAXED_CONTAINS_NEIGHBOURS(query, i, s − (i + 1)) then
21:                 return true
22:     return false
```

Algoritam 4 Strogi problem smanjenja k-mer setova [2]

```
1: function STRICT_CONTAINS_NEIGHBOURS(query, l_dist, r_dist)
2:     contains_left ← CONTAINS_SET(s_distant_left_neighbour_set(query, l_dist))
3:     contains_right ← CONTAINS_SET(s_distant_right_neighbour_set(query, r_dist))
4:     return DECIDE_PRESENT(query, contains_left, contains_right)
5:
6: function STRICT_CONTAINS(query, s)
7:     if BF.CONTAINS(query) then
8:         if STRICT_CONTAINS_NEIGHBOURS(query, s, s) then
9:             return true
10:    for i ← 0 to s − 1 do
11:        if STRICT_CONTAINS_NEIGHBOURS(query, i, s − (i + 1)) then
12:            return true
13:    return false
```

Kao rješenja na navedene probleme navode se tri pristupa:

- *Pristup 1: Pristup najboljeg podudaranja indeksa* - Podskup K se gradi pohlepno tako da se uzima svaki s-ti k-mer iz ulaznog niza U od točno određenog indeksa i_r. Indeks i_r se određuje tako da je najveće preklapanje k-mera s dosadašnjim podskupom K. Ulazni niz je dobiven očitavanjima i s je proizvoljan.
- *Pristup 2: Pristup podudarajućeg seta* - Podskup K se gradi pohlepno tako da se za svaki k-mer iz ulaznog niza U zapamti podskup lijevih susjeda L_k i desnih susjeda R_k k-mera k te se u K pohrane k-meri koji se nalaze u najviše setova R_k i L_k. Ulazni niz je dobiven očitavanjima i $s = 1$.
- *Pristup 3: Pristup jedinstvene sekvence* - Podskup K se gradi tako da se iz ulaznog niza uzme svaki s-ti k-mer počevši od nulte pozicije. Ovaj pristup se koristi kod slučaja gdje je ulazni niz poznat (cijeli genom) te je s proizvoljan.

4. Usporedba rezultata

Rezultati prikazani u ovom poglavlju, dobiveni su testiranjem nad genomom Pseudomonas aeruginosa. Prilikom implementacije jednostranog i dvostranog k-mer Bloom filtra, ulazna datoteka se je parsirala tako da se iz datoteke čitala linija po linija te se svaki k-mer stavljao u Bloom filtar. Kod dvostranog k-mer Bloom filtra su se dodatno spremali i rubni k-meri u poseban Bloom filtar. Odabrani parametri nad kojima se je testiralo su:

- $k = 20$
- $s = 1$

Pošto je testiranje provedeno nas cijelim genomom, parsiranje ulazne datoteke za slučaj smanjenja zauzeća memorije za Bloom filtar je izvedeno na način da se iz ulaznog niza uzeo svaki s-ti k-mer, počevši od nultog indeksa. Savaka implementacija se je testirala nad istim skupom podataka koji se sastoji od milijun k-mera. Skup k-mera je dobiven tako da se iz ulazne datoteke nasumično izabralo milijun k-mera i od njih se 3/4 mutiralo na način da se nasumično izabrala jedna baza u pojedinom k-meru i zamijenila se s drugom nasumično izabranom bazom.

Tablica 4.1: Vremensko izvođenje implementacija

Skup podataka	1-kBf[ms]	2-kBf[ms]	HS[ms]	BF[ms]
P. aeruginosa				
E. coli				

Tablica 4.2: Vremensko izvođenje implementacija autora prijedloga

Skup podataka	1-kBf[ms]	2-kBf[ms]	HS[ms]	BF[ms]
P. aeruginosa				
E. coli				

Tablica 4.3: Memorijsko zauzeće

Skup podataka	1-kBf[MB]	2-kBf[MB]	HS[MB]	BF[MB]
P. aeruginosa				
E. coli				

Tablica 4.4: Memorijsko zauzeće autora prijedloga

Skup podataka	1-kBf[MB]	2-kBf[MB]	HS[ms]	BF[MB]
P. aeruginosa				
E. coli				

5. Zaključak

Ovim radom se je pokazalo kako se korištenjem k-mera može postići bolji FPR, ali i kako smanjiti memorijsko zauzeće koje je potrebno za Bloom filtar. Također, pokazano je kako se odjednom ne može postići bolji FPR i manji Bloom filtar. Takav rezultat je očekivan jer smanjenjem Bloom filtra se smanjuje količina indeksa bit vektora koja se može koristiti. Time se bit vektor brže popuni jedinicama što uzrokuje veću vjerojatnost preklapanja indeksa novih elemenata s elementima za koje je potrebno provjeriti prisutnost u Bloom filtru.

6. Literatura

[1] *Bloom Filters by Example*. URL https://llimllib.github.io/bloomfilter-tutorial/.

[2] Improving bloom filter performance on sequence data using k-mer bloom filters. 2016. URL https://www.researchgate.net/publication/309827965_Improving_Bloom_Filter_Performance_on_Sequence_Data_Using_k-mer_Bloom_Filters.

[3] *k-mer Counting, part I: Introduction*, 2018. URL https://bioinfologics.github.io/post/2018/09/17/k-mer-counting-part-i-introduction/.