SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

BIOINFORMATIKA

Računanje optimalnog praga za q-gram filtre

Antun Flaš, Manuela Kajkara, Marija Kaselj

Zagreb, siječanj, 2016.

Sadržaj

[1. Uvod 1](#_Toc440575586)

[2. Osnovni pojmovi 1](#_Toc440575587)

[2.1. Nepotpuni -gram 1](#_Toc440575588)

[2.2. Prag 2](#_Toc440575589)

[2.2.1. Optimalan prag 2](#_Toc440575590)

[2.2.2. Skup Q 3](#_Toc440575591)

[3. Implementacija 4](#_Toc440575592)

[3.1. Struktura pohrane međurezultata 4](#_Toc440575593)

[3.2. Generiranje uzoraka 6](#_Toc440575594)

[3.2.1. Generiranje svih kombinacija uzoraka 6](#_Toc440575595)

[3.2.2. Generiranje -uzoraka pomoću -uzoraka 6](#_Toc440575596)

[4. Testiranje 7](#_Toc440575597)

[4.1. Zauzeće memorije 7](#_Toc440575598)

[4.2. Vrijeme izvođenja 9](#_Toc440575599)

[5. Zaključak 11](#_Toc440575600)

[6. Literatura 12](#_Toc440575601)

[7. Sažetak 13](#_Toc440575602)

# Uvod

Pretraživanje teksta bitan je dio informacijsko komunikacijske infrastrukture. Do danas je najpopularnije pretraživanje teksta uspoređivanjem kontinuiranog niza znakova, dok je  pretraživanje nepotpunog niza znakova zanemareno. Prve rezultate iz tog području dali su Stefan Burkhardt i Juha Kärkäinen na čijem radu se i temelji ovaj seminar.

U nastavku je dan kratak pregled teorije vezan uz -grame i algoritam izračuna optimalnog praga za -gram filtre uz jednostavan primjer, te opis vlastite implementacije navedenog.

# Osnovni pojmovi

Za dani uzorak , niz znakova i udaljenost između dva niza potrebno je pronaći sve podnizove znakova koji zadovoljavaju navedene uvjete. Kako bi se taj proces ubrzao radi se filtriranje. Filtriranje je algoritam koji provjerava tekst prema uvjetima filtriranja te potom odbacuje višak teksta ostavljajući samo moguća slaganja kako bi se provjerili algoritmom uspoređivanja teksta. Mnogi filtri koriste -grame, podnizove duljine .

Sličnost između niza znakova definiramo pragom koji predstavlja minimalni broj -grama koje mogući podniz niza treba imati s uzorkom .

Kako bi se filtriranje ubrzalo i kako bi se moglo koristiti mnogo različitih oblika uzoraka (eng. *shapes*) koriste se nepotpuni -grami.

## Nepotpuni -gram

Skup definira uzorak filtriranja. Sastoji se od brojeva koji predstavljaju indekse na kojima provjeravamo postojanje uzorka u podnizu.

Uz definirani skup i poziciju :

te niz znakova:

Za , -gram na poziciji *i* u nizu S je gdje je

-gram definiramo i kao -gram.

*Primjer:*

Za uzorak je # # \_ \_ #\_#, =4, =7 te se radi o (4,7)-gramu.

## Prag

Prag  predstavlja -gram sličnost sa uzorkom .

Previsoki prag može rezultirati gubitcima*,* no s višim pragom se povećava i učinkovitost filtriranja. Iz tih razloga veoma je bitno postići optimalan prag.

### Optimalan prag

Optimalan prag je najveći prag kod kojeg ne dolazi do gubljenja podataka odnosno najmanja -gram sličnost bilo koja dva niza znakova duljine  i udaljenosti .

Moguće ga je izračunati iscrpnom pretragom svih kombinacija razlika koristeći (\*) no to je vrlo skupa operacija za velike vrijednosti i . Postupak se bitno ubrza korištenjem dinamičkog programiranja.

#### Rekurzivan izračun optimalnog praga

Prema [1], definiramo:

Za *I* kao skup cjelobrojnih numeričkih vrijednosti vrijedi sljedeća notacija:

kao , isto tako kao ,

je 1 ako je *cond* istinit odnosno 0 ako je *cond* neistinit.

Vrijedi:

Za , i , definiramo optimalan prag kao:

Skup sadrži indekse podudaranja poslijednjih pozicije.

**Dokaz:**

Ako je , je uvijek 0 jer niz znakova koji je kraći od duljine grama ne može ga niti sadržavati.

Vrijednost se može izračunati iz uzimajući u obzir sljedeće mogućnosti:

1. ili ovisno o tome da li postoji podudaranje na poziciji *i* (da li skup sadrži ).
2. Prag je ili jednak ako ne postoji nova razlika, ili +1 ako postoji nova razlika.
3. Skup M\* se mora podudarat sa skupom M na s-2 pozicije  
   .

Uvjeti kombinirani s minimizacijom rezultiraju ispravnim izračunom optimalnog praga.

#### Izračun optimalnog praga dinamičkim programiranjem

Izračun započinjemo računanjem za sve pogreške , za sve postojeće skupove koji se podudaraju na zadnjih pozicija te za . Od svih dobivenih vrijednost za biramo najmanji .

Kako bi se ubrzao izračun optimalnog praga vrijednosti su pohranjene u polju. Za svaku moguću pogrešku *j* između 0 i *k* sadrži za sve skupove čiji broj pogrešaka je između 0 i *j*. Prema tome veličina tog polja je:

Ažuriranje polja s vrijednostima na vrijednosti za potrebno je formulu za izračun pozivati za svaki element polja. Rezultat se dobiva u konstantnom vremenu jer su moguće samo dvije opcije čije su vrijednosti već izračunate i pohranjene u polju.

### Skup Q

Prema [1], učinkovito generiranje novih skupova postže se na sljedeći način. Prvo je potrebno podijeliti (*q-1,s*) uzorke s pozitivnim pragom u skupine kojima je jedina razlika element na predzadnjoj poziciji ().

i pripadaju istom skupu ako .

Novi skup je . Veličine *q* je jer sadrži sve elemente iz skupa i jedan dodatan element iz .

# Implementacija

Za zadane vrijednosti , , i izračun optimalnog praga temelji se na dinamičkom programiranju i strukturi za pohranu međurezultata.

Prag se izračunava za generirane -uzorke te se od svih izračunatih pragova uzima maksimalni.

Pri samom izračunu praga za određeni uzorak predstavljen setom potrebno je ažurirati strukturu puta za svaku duljinu od do . Tako se prag za duljinu računa pomoću vrijednosti pragova dobivenih za duljinu , potom se vrijednosti za duljinu računaju pomoću vrijednosti za , povećavajući tako sve do .

Izračun se vrši prema formuli za izračun praga uz čitanje svih vrijednosti iz pomoćne strukture čime nema potrebe za rekurzijom.

U nastavku je prikazan pseudokod opisanog postupka.

Pseudokod 1 – Izračun optimalnog praga dinamičkim programiranjem

result = 0  
shapes = generate(q,s)  
for shape in shapes  
 shapeThreshold = MAX  
 for i in range (s, m)  
 for key in thresholds  
 for position in thresholds[key]  
 M, j = convert(M, position)  
 threshold = findThreshold(i, j, M, shape)  
 thresholds[key][position] = threshold  
 shapeThreshold = min(threshold, shapeThreshold)  
 end for  
 end for  
 end for  
 result = max(shapeThreshold, result)  
end for

## Struktura pohrane međurezultata

Prilikom izračuna praga za mjeru udaljenosti , potrebno je spremati vrijednosti izračuna za sve vrijednosti greške do uključivo te odgovarajuće vrijednosti skupa koji predstavlja podudaranja na zadnjih pozicija. Za najveći mogući broj pogrešaka , broj mogućih vrijednosti skupa jest:

Odnosno, na zadnjih pozicija može biti 0 pogrešaka do pogreška. Skupovi za pogrešaka u sebi sadrže moguće skupove pridružene pogreškama < .

Ako uzmemo primjer =4, =2 dobivamo sljedeće vrijednosti skupa po svim mogućim vrijednostima pogreške:

Tabela 1 – Sve moguće vrijednosti skupa za pogrešaka

|  |  |  |
| --- | --- | --- |
| **Broj pogrešaka** | **Skup** | **Binarni zapis skupa** |
| 0 | {1, 2, 3} | 111 |
| 1 | {1, 2, 3} | 111 |
| {1, 2} | 110 |
| {1, 3} | 101 |
| {2, 3} | 011 |
| 2 | {1, 2, 3} | 111 |
| {1,2} | 110 |
| {1, 3} | 101 |
| {2, 3} | 011 |
| {1} | 100 |
| {2} | 010 |
| {3} | 001 |

Postupak računanja optimalnog praga za zadanu duljinu niza uključuje ažuriranje ovih vrijednosti za svaku duljinu od do . Tako se prag za duljinu računa pomoću vrijednosti pragova dobivenih za duljinu , potom se vrijednosti za duljinu računaju pomoću vrijednosti za , povećavajući tako sve do .

Potrebno je sve mogućnosti prikazane u tablici spremiti u odgovarajuću strukturu kako bi se lako moglo doći do vrijednosti za zadani broj pogrešaka i skup .  
Kao odabranu strukturu koristili smo mapu koja kao ključ sadrži binarnu reprezentaciju skupa , a kao vrijednost polje izračunatih pragova za svaki broj pogrešaka za koji je skup validan. Pri tome je na prvom mjestu u polju zapisan odgovarajući broj mogućih pogrešaka što predstavlja i duljinu polja. Ostali elementi polja predstavljaju pragove za broj pogrešaka poštivajući zahtjev da prva vrijednost odgovara broju pogrešaka te svaka sljedeća vrijednosti za broj pogrešaka koji je za 1 manji od prethodnog. Ovo je bitno kako bi se jednostavno moglo doći do pozicije u polju koja odgovara vrijednosti za broj pogrešaka , a računa se na sljedeći način:

1 + )

Gdje je indikatorska funkcija čija je vrijednost jednaka 1 ako je ≡T, a funkcija koja skup pretvara u binarni zapis.

Struktura za gore navedeni primjer prema tome izgleda ovako:

|  |
| --- |
| 111: [3, (2), (1), (0)]  110: [2, (2), (1)]  101: [2, (2), (1)]  011: [2, (2), (1)]  100: [1, (2)]  010: [1, (2)]  001: [1, (2)] |

Izraz () predstavlja vrijednost praga za broj pogrešaka i skup kojem je pridružen.

Konačni rezultat, nakon što je struktura ažurirana za duljinu , pronalazi se tako da se za svaku vrijednost ključa pogleda izračunata vrijednost praga za točno pogrešaka i od svih vrijednosti uzme se minimalna.

## Generiranje uzoraka

Za zadanu duljinu uzorka i raspon uzorka , potrebno je generirati moguće vrijednosti (, )-uzorka za izračun praga. Uzorci su predstavljeni skupom .

### Generiranje svih kombinacija uzoraka

Uzorci su izračunati variranjem svih mogućih kombinacija binarnog niza duljine za zadane vrijednosti i . Krajnje pozicije uzorka nije potrebno uključiti u izračun kombinacija jer su one uvijek uključene u skup. Od svih kombinacija uzimaju se samo one čiji je broj jedinica jednak te se indeksi tih jedinica u generiranom uzorku pomaknutih za 1 dodaju u skup. Konačan skup dobije se unijom dobivenog skupa sa pozicijama 0 i .

Primjerice, za =4 i =3, moguće su dvije kombinacije:

= {0, 1, 3}

= {0, 2, 3}

### Generiranje -uzoraka pomoću -uzoraka

Izračun -uzoraka pomoću -uzoraka rekurzivan je postupak koji se poziva umanjujući q sve dok vrijednost ne postane 3. Za =3 uzorci se računaju generiranjem svih mogućih kombinacija. Za sve veće vrijednosti -uzorci se generiraju kombinacijom -uzoraka iz prethodnog koraka za koje je izračunata pozitivna vrijednost praga.

Ti uzorci se odvoje u grupe na način da jednoj grupi pripadaju uzorci koji se razlikuju samo na poziciji. Unijom svakog para uzorka iz grupe dobiju se -uzorci jer se svaki par grupe razlikuje samo na jednoj poziciji.

# Testiranje

Testiranje je provedeno za broj razlika , duljinu niza te za . Zbog značajnog usporenja izvođenja programa povećanjem *q*  i *s*, prikazani su rezultati mjerenja za .

## Zauzeće memorije

Grafikon 1 - Mjerenje zauzeća memorije

Iz Grafikona 1 vidljivo je postojanje rasta zauzete memorije. Taj rast bolje je uočljiv u grafikonima 2 i 3.

Grafikon 2 – Prikaz zauzeća memorije za duljinu uzorka = 2 i raspon uzoraka € [2, 30]

Grafikon 3 - Prikaz zauzeća memorije za duljinu uzorka = 3 i raspon uzoraka € [3, 25]

## Vrijeme izvođenja

Grafikon 4 – Prikaz vremena izvođenja programa

Grafikon 4 pokazuje da vrijeme izvršavanje programa raste linearno do neke vrijednosti raspona nakon čega počinje eksponencijalno rasti. U grafikonima 5 i 6 izdvojen je prikaz rasta za =2 odnosno =3.

Grafikon 5 – Prikaz vremena računanja optimalnog praga za duljinu uzorka =2 i raspon uzoraka € [2, 30]

Grafikon 6 - Prikaz vremena računanja optimalnog praga za duljinu uzorka =3 i raspon uzoraka € [3, 25]

# Zaključak

Rad je dao kratak uvid u izračun optimalnog praga za -gram filtre. Testiranje je potvrdilo očekivanja, izračun dinamičkim programiranjem mnogostruko je brži od rekurzivnog.

Generiranje skupa uzoraka pomoću prethodnih skupova s pozitivnim pragom rezultiralo je iako točnim, dosta sporijim izračunima za nevelike vrijednosti raspona i duljine uzorka . S porastom raspona i duljine uzorka očekuje se veće ubrzanje s navedenim generiranjem.

Ovo interesantno područje još nije pretjerano istraženo te se autori nadaju da će ovaj rad pobuditi zanimanje čitatelja.

# Literatura

[1] Burkhardt, Stefan i Kärkkäinen, Juha. *Better Filtering with Gapped q-Grams*. (2003): 1001-1020 Preuzeto: 16.10.2015., Poveznica: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.5942&rep=rep1&type=pdf>

# Sažetak

Kako bi se ubrzao proces pretraživanja teksta prvo se obavlja filtriranje, a jedno od mogućih je -gram filtriranje. -gram je podniz duljine . Optimalan prag je najveći prag kod kojeg ne dolazi do gubitka podataka. Računanje optimalnog praga q-gram algoritma moguće je postići na dva načina, rekurzivnim izračunom te dinamičkim programiranjem. Implementacija oba načina napravljena je u C++ programu. Testiranjem je utvrđeno da oscilacije u zauzeću memorije postoje, no nisu pretjerano izražene, dok se vrijeme izvođenja vidno povećalo za velike broj uzoraka koje je potrebno ispitati s povećanjem raspona .