

SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5464

**Algoritmi ažuriranja i pretraživanja  
raspodijeljenih višedimenzionalnih  
podataka primjenom podjele prostora na  
regije**

*Dario Sindičić*

Zagreb, lipanj, 2018

Zagreb, 13. ožujka 2018.

## ZAVRŠNI ZADATAK br. 5464

Pristupnik: **Dario Sindičić (0036490887)**  
Studij: Računarstvo  
Modul: Računarska znanost

Zadatak: **Algoritmi ažuriranja i pretraživanja raspodijeljenih višedimenzionalnih podataka primjenom podjele prostora na regije**

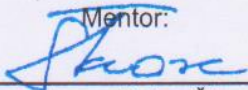
### Opis zadatka:

Proučiti i opisati algoritme i strukture podataka za upravljanje višedimenzionalnim podacima. Proučiti i opisati osnovni algoritam sortiranja jednostavnih podataka zasnovan na podjeli podatkovnog raspona na regije (engl. bucket sort). Po uzoru na taj algoritam, programski ostvariti algoritme za ažuriranje i pretraživanje raspodijeljenih višedimenzionalnih podataka primjenom podjele prostora na regije (engl. buckets). Ostvareni algoritmi trebaju omogućiti ažuriranje pojedinačnih višedimenzionalnih podataka te pretraživanje i dohvat skupa višedimenzionalnih podataka koji se nalaze unutar zadanog područja pretrage. Ispitati radna svojstva algoritama s obzirom na veličinu raspodijeljene računalne okoline, razlučivost podjele prostora na regije, veličinu područja pretrage te različite odnose učestalosti operacije ažuriranja i operacije pretraživanja. Usporediti radna svojstva ostvarenih algoritama u odnosu na algoritme zasnovane na binarnim stablima. Opisati algoritme, njihovo programsko ostvarenje i rezultate ispitivanja. Navesti korištenu literaturu i primljenu pomoć.

Zadatak uručen pristupniku: 16. ožujka 2018.

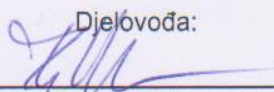
Rok za predaju rada: 15. lipnja 2018.

Mentor:



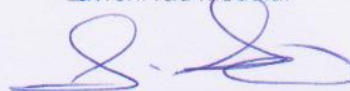
Doc. dr. sc. Dejan Škvorc

Djelovođa:



Doc. dr. sc. Tomislav Hrkać

Predsjednik odbora za  
završni rad modula:



Prof. dr. sc. Siniša Srbljić

Zahvaljujem mentoru izv. prof. dr. sc. Dejanu Škvorcu na pomoći pri izradi ovoga rada te zahvaljujem svojoj obitelji, prijateljima i profesorima na podršci tijekom cijelog školovanja.

## Sadržaj

1.	Uvod.....	7
2.	Programska paradigma Map&Reduce .....	9
2.1	Praktične primjene problema.....	10
3.	Binarno stablo .....	11
3.1	Binarno stablo pretraživanja.....	11
3.2	Prikaz strukture.....	12
3.3	Složenosti uobičajenih operacija .....	14
3.4	Upravljanje višedimenzionalnim podacima .....	15
3.5	Pseudokod .....	15
4.	Podjela prostora na regije.....	17
4.1	Sortiranje podjelom na regije .....	17
4.1.1	Vremenska složenost.....	18
4.1.2	Memorijska složenost .....	18
4.1.3	Pseudokod .....	18
4.2	Prikaz strukture.....	19
4.3	Složenosti uobičajenih operacija .....	21
4.4	Upravljanje višedimenzionalnim podacima .....	21
4.5	Pseudokod .....	22
5.	Sustav za mjerenje radnih svojstva algoritama za ažuriranje i pretraživanje raspodijeljenih višedimenzionalnih podataka .....	24
5.1	Opis sustava.....	24
5.2	Upit.....	25
5.3	Pomak.....	26
6.	Mjerenje radnih svojstva i analiza rezultata.....	28
6.1	Opis usporedba .....	28
6.2	Parametri mjerenja .....	29
6.3	Utjecaj broja regija .....	30
6.4	Utjecaj početnog raspršenja.....	32
6.5	Utjecaj broja točaka.....	34
6.6	Utjecaj veličine pomaka točaka.....	37
6.7	Utjecaj broja računala.....	40
7.	Usporedba radnih svojstva binarnog stabla i podjele prostora na regije.....	44
7.1	Usporedba po broju točaka.....	44
7.2	Usporedba po veličini pomaka točaka.....	44

7.3	Usporedba po broju računala.....	44
8.	Zaključak.....	46
9.	Literatura.....	47

## Popis slika

Slika 1.	Map&Reduce .....	9
Slika 2.	Prikaz binarnog stabla.....	11
Slika 3.	Prikaz mrežni i brojčanih čvorova .....	13
Slika 4.	Raspodjela i poveznice binarnih stabla.....	14
Slika 5.	Prikaz nekoliko regija .....	17
Slika 6.	Struktura liste pretinaca podataka.....	20
Slika 7.	Prikaz upita i odgovora na njega.....	25
Slika 8.	Razmjena krivo raspoređenih točaka .....	26
Slika 9.	Korisničko sučelje.....	29
Slika 10.	Vrijeme upita uz promjenu broj regija .....	30
Slika 11.	Vrijeme upita uz promjenu većeg broj regija.....	31
Slika 12.	Vrijeme promjene vrijednosti točaka uz promjenu broj regija .....	31
Slika 13.	Rezultat trajanja upita kod slučajnog raspršenja.....	32
Slika 14.	Rezultat trajanja upita kod realnog raspršenja .....	33
Slika 15.	Vrijeme promjene točaka kod realnog raspršenja.....	33
Slika 16.	Vrijeme promjene točaka kod slučajnog raspršenja.....	34
Slika 17.	Vrijeme upita uz promjenu broja regija, broj točaka = 5 milijuna.....	35
Slika 18.	Vrijeme promjene točaka uz promjenu broja regija, broj točaka = 5 milijuna .....	35
Slika 19.	Vrijeme upita uz promjenu broja regija, broj točaka = 18 milijuna.....	36
Slika 20.	Vrijeme promjene točaka uz promjenu broja regija, broj točaka = 18 milijuna .....	36
Slika 21.	Vrijeme upita i promjena kod binarnog stabla u slučaju 5 i 18 milijuna točaka .....	37
Slika 22.	Rezultat trajanja upita kod malih pomaka .....	38
Slika 23.	Rezultat trajanja upita kod velikih pomaka .....	38

Slika 24. Vrijeme promjene točaka kod velikog pomaka .....	39
Slika 25. Vrijeme promjene točaka kod malih pomaka .....	39
Slika 26. Varijacija veličine pomaka kod binarnog stabla .....	40
Slika 27. Performanse sustava kod malih pomaka – Lista pretinaca podataka.....	41
Slika 28. Prikaz performansi sustava kod velikih pomaka – Lista pretinaca podataka .....	41
Slika 29. Prikaz performansi sustava kod malih pomaka – Binarno stablo .....	42
Slika 30. Prikaz performansi sustava kod velikih pomaka – Binarno stablo .....	42
Slika 31. Usporedba vremena na upit za dvije strukture uz promjenu broja računala uključenih u obradu.....	45

# 1. Uvod

U današnjem svijetu količina podataka raste velikom brzinom. Podaci mogu biti diskretni kao na primjer podaci o vremenskoj pojavi s vrijednostima sunčano i kiša, ili kontinuirani kao primjerice količina padalina izražena u milimetrima. Također podaci se stvaraju i ažuriraju velikom brzinom te je potrebno odgovoriti na takve zahtjeve. Osim jednodimenzionalnih koriste se i višedimenzionalni podaci kao što su zemljopisne koordinate, različiti geometrijski oblici ili podaci pomoću kojih se prikazuju ovisnosti kao vremenske prognoze ili kretanja burzovnih indeksa. U toj velikoj količini podataka potrebno je razviti alate koji će uspješno riješiti neke od sljedećih problema. Potrebno je ažurirati višedimenzionalne podatke te upit nad podacima treba biti obrađen u što kraćem roku.

Višedimenzionalni podaci se sastoje od niza uređenih  $n$ -torki pri čemu  $N$  predstavlja dimenzionalnost podataka. Skup višedimenzionalnih podataka potrebno je moći brzo i jednostavno pretraživati i ažurirati kao pojedinačne članove ili sa nekog određenog podskupa. S obzirom na velike količine podataka često sve podatke nije moguće držati u radnoj memoriji nego se spremaju na disk. Kako bi se riješio taj problem, odlučeno je da bi se gore navedena svojstva najbolje mogla zadovoljiti ako se u obradu uključi veliki broj računala te ako se primijeni metoda Map&Reduce kako bi se svakom računalu dao lako obradivi zahtjev na obradu.

Ovaj rad bavi se usporedbom radnih svojstava dvije strukture za rukovanje višedimenzionalnim podacima: binarnog stabla i liste pretinaca podataka (engl. *bucket structure*). Radna svojstva binarnog stabla i liste pretinaca podataka ispitana su za različite karakteristike podataka i sustava za upravljanje tim podacima kao što su broj računala uključen u obradu, veličina promjene podataka i frekvencija upita. Višedimenzionalni podaci korišteni u pokusima su dvodimenzionalni koji predstavljaju geografske lokacije, tj. koordinate svijeta.

U drugom poglavlju predstaviti će se programska paradigma Map&Reduce koja omogućava paralelno izvođenje zadataka. U trećem i četvrtom poglavlju bit će predstavljane dvije strukture podataka koje su korištene u pokusima, binarno stablo te lista pretinaca podataka. U petom poglavlju predstaviti će se sustav za mjerenje

i usporedbu njihovih radnih svojstava. Rezultati koje su postigle dvije navedene strukture nad različitim promjenama u sustavu i usporedba njihovih radnih svojstva prikazana je u šestom i sedmom poglavlju te se u osmom poglavlju iznosi zaključak rada.



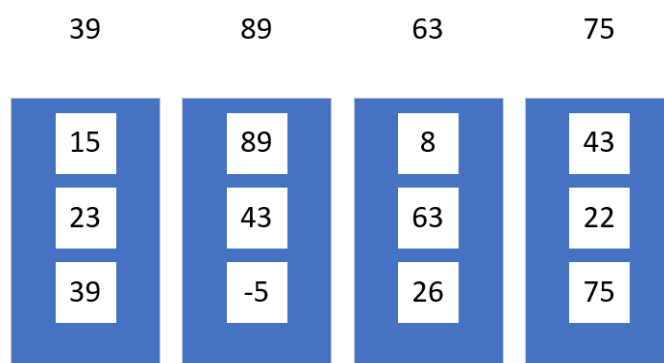
## 2. Programska paradigma Map&Reduce

Map&Reduce je programska paradigma koja omogućava paralelnu obradu određenog problema. Ideja je da se problem razloži na dva ili više manjih jednostavnijih problema koji su međusobno neovisni. Kada se jednostavniji problemi riješe, njihovi rezultati se spajaju te se dobiva rješenje glavnog problema.

Najjednostavniji je primjer traženja najvećeg broja u nizu. Primjerice, tada niz od 12 brojeva biva podijeljen na četiri manja niza kao što se vidi na slici 1. U ta četiri niza traži se koji je broj najveći. Kombinacijom dobivenih rezultata manjih problema dobiva se rezultat glavnog problema, a on je u donjem slučaju 89.

$$\max(\max(\{a_0, a_1, \dots\}), \max(\{b_0, b_1, \dots\}), \dots) \quad (1)$$

Izrazom 1 prikazano je rješenje gore navedenog problema, ali tako da je prvotni niz podijeljen na više nizova, ovdje nazvani a, b i tako dalje. Od svakog podniza traži se najveći broj preko funkcije max te se od svih najveći brojeva podijeljenih nizova opet traži najveći broj. Takav postupak skupljanja znanja o manjim nizovima se zove „reduce“, dok se sama podjela na njih zove „map“.



Slika 1. Map&Reduce

Paradigma Map&Reduce primjenjiva je na problem ažuriranja i pretraživanja raspodijeljenih višedimenzionalnih podataka jer omogućava brzi odgovor na upit te promjenom strukture podataka može se postići brzo ažuriranje. Problem se sastoji od dva jednostavna. Mora se moći pretražiti niz n-torki te se moraju jednostavno

napraviti promjene vrijednosti n-torki. Napraviti će se mala ograda te će se zadati minimalna i maksimalna vrijednost svake vrijednosti n-torke. Ako na raspolaganju postoji N računala, svakom računalu će se uniformno dodijeliti jedna regija za koju će on biti nadležan. Svako računalo će biti zaduženo za obradu upita i promjena nad svojom regijom. Pošto je problem N-dimenzionalni, svakom računalu bit će dodijeljeno N regija, po svaka za jednu dimenziju.

Ograničenje ovakvog pristupa je da se upit mora moći rastaviti po komponentama, kao na primjer:

$$\forall_{(x,y)} (20 \leq x \leq 150) \wedge (15 \leq y \leq 30)$$

ali također su mogući i složeniji upiti kao ovaj na primjer:

$$\forall_{(x,y)} ((x - x_0)^2 + (y - y_0)^2) \leq 5^2$$

Takvim upitima ovaj se rad neće baviti te će se za sve upite smatrati da se mogu rastaviti po komponentama. Iako se i takvi upiti mogu svesti na jednostavnije promjenom koordinatnog sustava kao na primjer u sferni koordinatni sustav za gornji primjer.

## 2.1 Praktične primjene problema

Ažuriranje i pretraživanje višedimenzionalnih podataka je u praksi vrlo primjenjivo. Od geografskih lokacija koje se prikazuju kao uređeni parovi, dodavanjem treće koordinate koja označuje dubinu pa sve do genetskih algoritama gdje se vrlo često susreću podaci velikih dimenzija te su potrebni alati ažuriranja i pretrage kako bi se dobila raspršenija nova generacija.

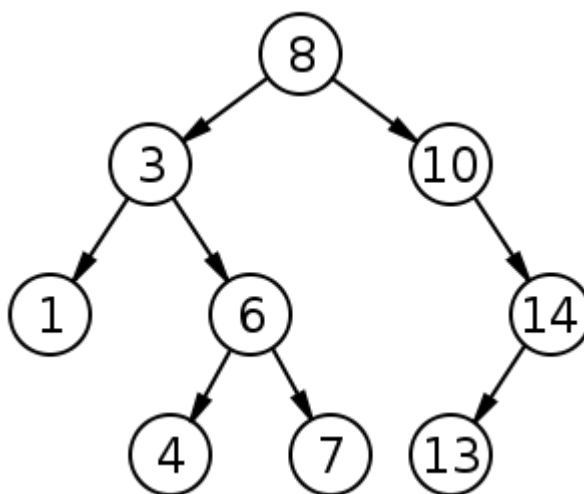
Taxi služba koja ima veliku flotu automobila koji se stalno kreću treba moći brzo na poziv kupca odgovoriti koji se taksi nalazi najbliže njegovoj lokaciji. Primjena se nalazi i u marketingu kada imamo korisnike sa svojim mobitelima koji se stalno kreću i dućan koji želi da kada netko dođe 100 metara od njega da mu se pošalje notifikacija o određenoj akciji ili promociji.

### 3. Binarno stablo

U ovom poglavlju predstaviti će se binarno stablo pretraživanja i njegova osnovna svojstva. Predstaviti će se struktura koja se sastoji od više binarnih stabala povezanih čvorovima međusobno. Ona će se rasporediti na više računala te će svako računalo biti zaduženo za jedno stablo. Predstaviti će se rješenje problema višedimenzionalnih podataka te će biti prikazan graf strukture kao i njezin pseudokod.

#### 3.1 Binarno stablo pretraživanja

Binarno stablo pretraživanja (engl. *Binary Search Tree*) [8] je specijalna vrsta binarnog stabla u kojemu za svaki čvor vrijedi da su sve vrijednosti lijevog podstabla manje ili jednake od njegovog i čije su vrijednosti desnog podstabla veće ili jednake od njegovog. Primjer binarnog stabla pretraživanja prikazan je na slici 2.



Slika 2. Prikaz binarnog stabla

Složenosti ove strukture su vrlo poznate te su prikazane u tablici 1.

*Tablica 1. Složenosti operacija binarnog stabla*

	Prosječan slučaj	Najgori slučaj
Memorija	$O(n)$	$O(n)$
Traženje	$O(\log n)$	$O(n)$
Dodavanje	$O(\log n)$	$O(n)$
Brisanje	$O(\log n)$	$O(n)$

Običnim dodavanjem novih čvorova, binarno stablo bi izgubilo na performansama. Ako se dodavaju vrijednosti samo manje od korijena, stablo više nije balansirano i ne može odgovarati na upite u gore navedenim složenostima. Stoga se kod dodavanja svakog novog čvora vrši balansiranje binarnog stabla kako bi se osigurala gore navedena složenosti te kako dubina stabla ne bi prešla  $\log(N)$ . Balansiranje je jednostavno i brzo kada je broj čvorova mali, ali složenost raste kada stablo ima veliki broj čvorova. Implementacija koja se koristila u radu se oslonila na implementaciju Red-Black tree iz razreda TreeSet [5] koji je dio standardne Java biblioteke.

## 3.2 Prikaz strukture

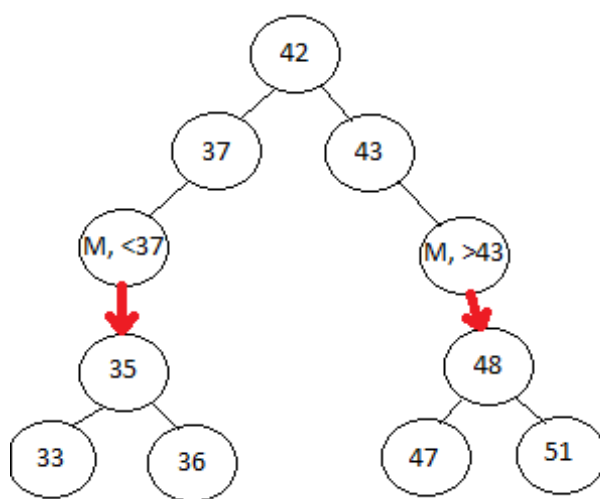
Struktura koja će se koristiti je binarno stablo traženja, uz male modifikacije. Struktura se mora „razapeti” kroz više računala te će se uvesti dvije vrste čvorova.

Vrste čvorova

1. Brojčani čvor – čvor koji predstavlja vrijednost koja je dodana u stablo, običan čvor
2. Mrežni čvor – čvor koji vodi na sljedeće računalo, nema vrijednost samu po sebi

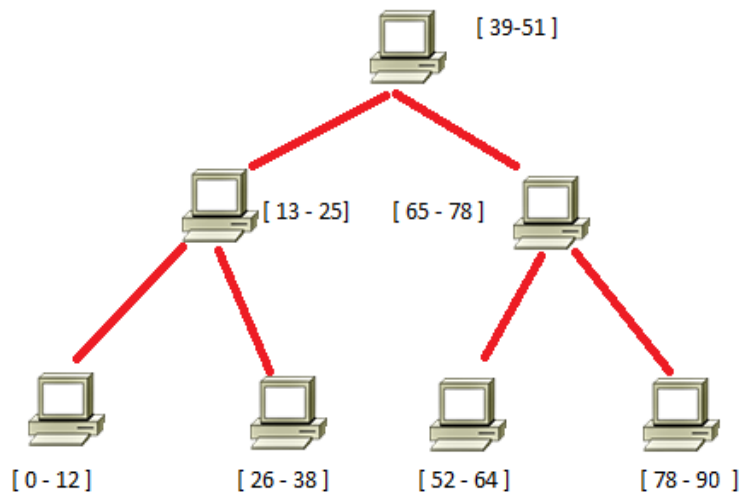
Mrežni čvor je dodan kako bi se struktura mogla povezati preko više računala. Svakom računalu je dodijeljena jedna regija nad kojom je on nadležan. Mrežni čvor se sastoji zapravo od IP adrese te porta nad koji se treba propagirati upit dalje ako upit nije razriješen.

Svako računalo čija regija ima i lijeve i desne susjede imam dva mrežna čvora, za svaku regiju po jedno. Ako je regija rubna, binarno stablo ima samo jedan mrežni čvor. Ako mrežni čvor vodi na lijevu regiju, njegova vrijednost je maksimalna vrijednost lijeve regije. Ako vodi na desnu regiju, njegova vrijednost je minimalna vrijednost desne regije.



*Slika 3. Prikaz mrežni i brojčanih čvorova*

Na slici 3 je prikazana struktura sastavljena od 3 binarna stabla povezana mrežnim čvorovima. Ako se traže svi brojevi između 36 i 47, započet će se upitom nad prvim stablom. Tada upit zahvaća brojčane i mrežne čvorove. Nad dobivenim mrežnim čvorovima paralelno se izvode upiti te se dobiveni brojčani čvorovi dodaju nad već prethodno dobivene u gornjem stablu. Kada u rezultatu više nema mrežnih čvorova, upit je gotov. Rezultat upita je {36, 37, 42, 43, 47}. Na slici 4. vidljiva je raspodjela regija na više računala te njihova povezanost u obliku binarnog stabla. Vršno računalo, s regijom od 39 do 51, zaprima sve upite te ih prosljeđuje dublje u strukturu, ako se u upitu našao mrežni čvor. Ako se u upitu našao mrežni čvor, od računala na kojeg vodi taj mrežni čvor se traži da izvede upit za svoje podstablo na isti način kao što ga je izveo i njegov roditelj te da mu tada vrati rezultate upita.



*Slika 4. Raspodjela i poveznice binarnih stabla*

U ovakvoj strukturi umjesto puno malih binarnih stabala postoji jedno veliko te se svi upiti izvoditi nad njime. Ovakva struktura nije pogodna ako dođe do kvara nekog od računala, tada cijelo podstablo nije dostupno za upite ili promjene, ali time se ovaj rad ne bavi.

### 3.3 Složenosti uobičajenih operacija

Složenosti uobičajenih operacija ne razlikuju se puno od klasičnih složenosti kod binarnog stabla pretraživanja. Struktura se ne razlikuje previše od klasičnog balansirano binarnog stabla, ali u ovom stablu balansiranje nije moguće provesti u cjelini. Moguće je balansirati samo dio koji je vidljiv samo na jednom računalu pa će zbog toga dubina stabla biti veća od  $\log_2(N)$ , gdje je  $N$  broj čvorova u stablu.

$N_1$  – broj čvorova u jednom računalu

$N_2$  – broj računala u stablu

$$\log_2 N_1 * \log_2 N_2 =$$

$$\log_2(N_1^{\log_2 N_2}) \quad (2)$$

Izrazom 2 raspisana je složenost pretrage vrijednosti u takvoj strukturi.  $N_1$  predstavlja broj čvorova u jednom računalu, dok je  $N_2$  broj računala u stablu. Složenost pretraga na jednom stablu jednaka izrazu 2 kada se u njega za  $N_2$  uvrsti 1. Ako broj čvorova u strukturi raste linearnom brzinom, složenost strukture rasti će logaritamskom brzinom.

Ako su podaci savršeno raspoređeni, onda imamo istu složenost kao i kod klasičnog balansiranog binarnog stabla. Već za male promjene složenost je veća, ali ju logaritamska funkcija drži stabilnom.

### 3.4 Upravljanje višedimenzionalnim podacima

Za uspješno upravljanje s višedimenzionalnim podacima potrebno je imati  $N$  binarnih stabla, gdje je  $N$  broj dimenzija naših podataka. Svaki upit bit će podijeljen na upite po komponentama i svaki taj upit će se primijeniti na pojedino binarno stablo te će se na kraju napraviti presjek dobiveni upita kako bi se dobio konačni rezultat.

### 3.5 Pseudokod

Prikazan je pseudokod funkcija članica razreda `BinaryTree` koji je naslijeđen od `TreeSet<Node>` razreda koji mu je dao funkcionalnost binarnog stabla.

```
public void updateNumberNode(NumberNode node, double newValue)
{
    this.remove(node);
    node.setValue(newValue);
    this.add(node);
}

public void deleteNumberNode(NumberNode node)
{
    this.remove(node);
    cache.add(node);
}

public void addNumberNode(double newValue, int dot)
{

```

```

        add(new NumberNode(newValue, dot));
    }
    public void addNetworkNode(String address, double value, Orientation
orientation) {
        add(new NetworkNode(value, address, orientation));
    }
    public SortedSet<Node> query(double min, double max)
    {
        SortedSet<Node> sortedMap = this.subSet(
                                new NumberNode(min, -2),
                                new NumberNode(max, -2)
                                );
        return sortedMap;
    }

```

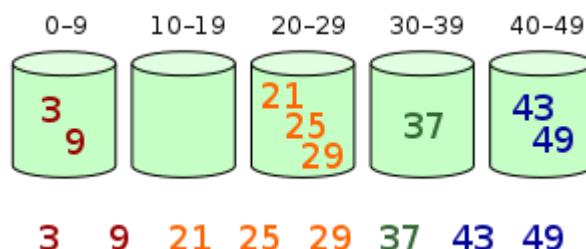


## 4. Podjela prostora na regije

U četvrtom poglavlju predstaviti će se algoritam podjele na regije te njihovog sortiranja (engl. *bucket sort*) kao i njegove prednosti i mane. On je poslužio kao inspiracija za listu pretinaca podataka (engl. *bucket structure*) koja će se koristiti u daljnjim poglavljima. Također je predstavljen način kako je riješen problem više dimenzionalnih podataka te je predstavljen graf strukture kao i njegov pseudokod.

### 4.1 Sortiranje podjelom na regije

Sortiranje podjelom na regije (engl. *bucket sort*) [9] je algoritam sortiranja brojeva koji brojeve raspoređuje u N regija (engl. *bucket*). Svaka regija se potom zasebno sortira koristeći uobičajeni algoritam sortiranja ili ponovno algoritam sortiranja podjelom na regije. Kada su sortirane sve regije dovoljno je proći po svim regijama i pokupiti brojeve u sortiranom redoslijedu kako bi se dobio sortirani niz. Na slici 5. prikazan je niz brojeva te su oni podijeljeni u 6 regija, od kojih je svaka regija veličine 10. Na taj se način dobiva više nizova manjih duljina što ubrzava postupak sortiranja.



Slika 5. Prikaz nekoliko regija

Jedna regija može biti polje, jednostruko ili dvostruko povezana lista, balansirano binarno stablo ili čak tablica raspršenog adresiranja. Odabir strukture ovisi naravno o broju dodavanja i micanja iz strukture.

Sortiranje podjelom na regije je slično mnogim drugim algoritmima sortiranja kao što je sortiranje prebrojavanjem (engl. *counting sort*) uz uvjet da je svaka regija veličine 1. Sortiranje s podjelom na dvije regije polja se može gledati kao poboljšana verzija „brzog sortiranja” (engl. *quick sort*) gdje je stožerni element (engl. *pivot*)

uvijek median niza te tada nije potrebna zamjena brojeva s lijeve i desne strane niza.

#### 4.1.1 Vremenska složenost

*Tablica 2. Složenost algoritma podjelom na regije*

Najgori slučaj	$O(N^2)$
Najbolji slučaj	$\Omega(N + K)$
Prosječan slučaj	$\Theta(N + K)$

Najgori slučaj predstavlja situaciju kada postoji samo jedna regija te se tada svi brojevi ubacuju u jednu regiju. Najbolji slučaj je kada su brojevi savršeno raspoređeni po regijama tako da sve regije imaju jednak broj članova.

#### 4.1.2 Memorijska složenost

Memorijska složenost ovisi o odabir kojega će tipa biti regija, ali se općenito računa kao veličina polja + broj pokazivača na regije.

#### 4.1.3 Pseudokod

```
function bucketSort(array, n)
    buckets = new array of n empty lists
    for i = 0 to (length(array) - 1) do
        insert array[i] into buckets[ getBucket(array[i],n) ]
    for i = 0 to n - 1 do
        sort(buckets[i])
    return concatenation of buckets[0], ... , buckets[n-1]
```

Funkcija `getBucket` može biti implementirana da uniformno određuje regije ili po nekoj drugoj distribuciji ovisno o podacima koji stižu. U gore navedenom primjeru na slici 5, uzeto je da `getBuckets` vraća `array[i] DIV 10`.

## 4.2 Prikaz strukture

Lista pretinaca podataka koja će se koristiti u programu inspirirana je algoritmom sortiranja po regijama te funkcionira na vrlo sličan način. Kao što je prije navedeno odabir kojega će tipa biti regija može igrati veliku ulogu u cjelokupnoj složenosti pa će se u idućem odlomku navesti neke mogućnosti s njihovim vrlinama i manama.

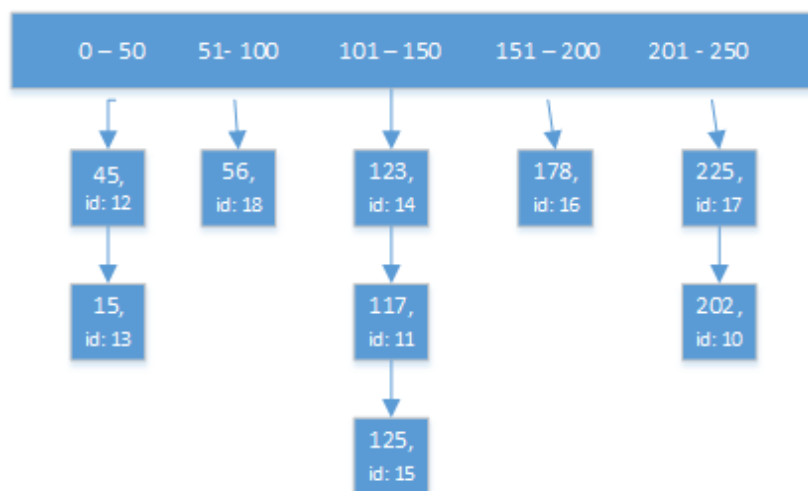
*Tablica 3. Prikaz vrlina i mana različitih vrsta implementacija regija*

Naziv strukture	Vrlina	Mana
Polje	Dodavanje elemenata	Brisanje elemenata, pretraga
Jednostruko-povezana lista	Dodavanje, brisanje elemenata	Pretraga elemenata
Binarno balansirano stablo	Pretraga elemenata, dodavanje i brisanje	balansiranje u slučaju velikog broja modifikacija
Tablica raspršenog adresiranja	Lako dodavanje i brisanje elemenata	Memorijska potrošnja, nužan „rehash“ i za male promjene

Jednostavno polje vrlo brzo dodaje nove elemente, ali brisanje ili pretraga su linearne složenosti. Binarno balansirano stablo ima mogućnost brze pretrage, kao i

brisanja ili dodavanja elemenata, ali balansiranje postaje veliki problem u slučaju velikog broja elemenata.

Za implementaciju regije uzeta je jednostruka povezana lista zato što je očekivan veliki broj dodavanja i izbacivanja izvan strukture. Uz uvjet da ćemo uz dovoljan broj regija osigurati da nam broj vrijednosti u regiji bude mali, nisu nam potrebne napredne strukture podataka. Ona je također pokazala najbolje performansama u sustavu naspram drugih struktura. Njezina struktura može se vidjeti na slici 6.



Slika 6. Struktura liste pretinaca podataka

Ovisno o broju regija napravi se polje regija. Za svaki broj, njegova regija se odredi tako da se za strukturu zna minimalna vrijednost koja se može dodati kao i maksimalna te broj regija. Uz pretpostavku da će vrijednosti biti uniformno distribuirane koristimo navedenu formulu.

$$\begin{aligned}
 &getBucket(value, minValue, maxValue, numOfBuckets) \\
 &= \frac{(value - minValue)}{(maxValue - minValue) * numOfBuckets} \quad (3)
 \end{aligned}$$

Izraz 3 određuje formulu pomoću koje se može odrediti kojoj regiji pripada određena vrijednost, ako je poznata minimalna i maksimalna moguća vrijednost te sam broj regija na raspolaganju.

U ovom slučaju točke se u regiju dodaju jedna po jedna bez nekog određenog poretka te kada tražimo nešto u njemu trebamo ga pretražiti cijelog. Ukoliko bi nam veličina regije bila jako velika, onda bi binarno balansirano stablo bilo bolje rješenje, ali ako imamo dovoljno veliki broj regija možemo pretpostaviti da će broj vrijednosti u regiji biti dovoljno mali.

### 4.3 Složenosti uobičajenih operacija

U slučaju odabira polja kao regije, složenosti su sljedeće:

- dodavanje nove vrijednosti  $O(1)$
- brisanje neke vrijednosti  $O(M)$
- dohvat neke vrijednosti  $O(M)$

gdje je  $M$  prosječan broj vrijednosti u strukturi.

### 4.4 Upravljanje višedimenzionalnim podacima

Podatak je predstavljen kao  $n$ -torka vrijednosti. Kako bi se takvi podaci obrađivali jednostavno, brzo i paralelno umjesto jedne kompleksne strukture podataka odlučeno je koristiti njih  $N$ , ovisno o dimenziji podataka. Postojat će  $N$  lista pretinaca, od kojih će svaka biti zadužena za jednu dimenziju. Tako će se upiti moći podijeliti na upite zasebne po dimenzijama.

Npr. upit može tražiti koordinate za čiji  $x$  vrijedi da je između 44 i 52, te za čiji  $y$  vrijedi da je između 12 i 17. Glavni upit se razdvaja na dva manja upita za svaku koordinatu te će takvi upiti ići prema strukturi. Rezultat glavnog upita dobije se tako da se napravi presjek rezultata svih upita koje smo napravili. Rezultati upita su identifikatori  $N$ -dimenzionalnih podataka, stoga identifikatori koji se nađu u svim upitima će zadovoljavati sve upite i oni će biti rješenje glavnog upita.

## 4.5 Pseudokod

Ovdje je prikazan pseudokod funkcija članica klase BucketStructure.

```
public int getBucket(double value)
{
    int val = (int)((value - this.minValue) / sizePer-Bucket);
    return val;
}

public void add(double newValue, int dot)
{
    int newBucket = getBucket(newValue);
    add(newValue, dot, newBucket);
}

public void update(Pair old, double newValue)
{
    int oldBucket = getBucket(old.value);
    int newBucket = getBucket(newValue);

    if(newBucket != oldBucket)
    {
        this.buckets[oldBucket].remove(old);
        old.setValue(newValue);
        this.buckets[newBucket].add(old);
    } else {
        int index = this.buckets[oldBucket].indexOf(old);
        this.buckets[oldBucket].get(index).setValue(newValue);
    }
}

public List<Integer> query(double min, double max)
{
    int firstBucket = getBucket(min);
    int lastBucket = getBucket(max);
    List<Integer> result = new ArrayList<>();
    for(int i=firstBucket; i<=lastBucket; ++i)
    {
        for(Integer id : buckets[i])
        {
            result.add(id);
        }
    }
}
```

```
    }  
    return result;  
}
```

## **5. Sustav za mjerenje radnih svojstva algoritama za ažuriranje i pretraživanje raspodijeljenih višedimenzionalnih podataka**

U petom poglavlju se predstavlja kako je projektiran cjelokupan sustav s više računala. Predstavit će se njegova struktura te zadaće svih računala u njemu. Predstavit će se opis naredbi u sustav te će se dijagramima predstaviti odnosi računala u sustavu kao i njihova interakcija.

### **5.1 Opis sustava**

Jedan od uvjeta sustava je bila da se zadaća sustava raspodjeli na više malih, lakše rješivih zadataka kako bi oni mogli biti riješeni na više računala. Računala će komunicirati preko TCP protokola te će tako dijeliti sve potrebne podatke.

Računala se mogu podijeliti u dvije grupe. Prva grupa se sastoji od jednog računala koje će upravljati svima ostalima te druge grupe u kojoj se nalaze radnici koji će obrađivati zahtjeve dobivene od glavnog računala.

Zadaće glavnog računala su:

1. određivanje regija sa svako računalo
2. određuje kakvu će strukturu računala koristiti
3. davanje početnih pozicija točaka svakom računalu
4. šalje upite na računala
5. šalje oznaku da se točke mogu promijeniti

Zadaće ostalih računala su:

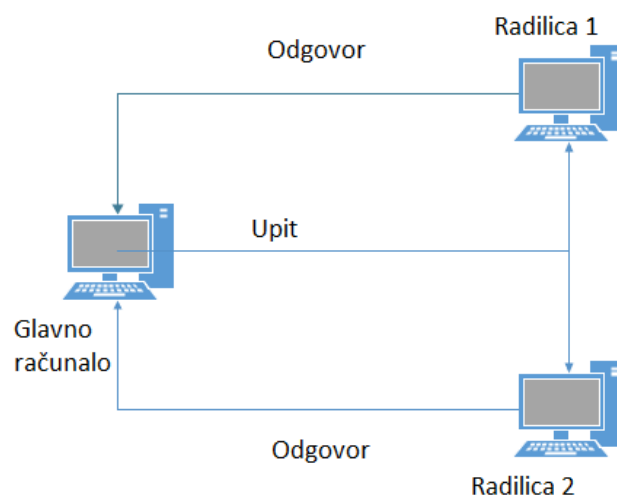
1. obrađivanje zahtjeva i slanje odgovora
2. mijenja vrijednosti točaka kada za to dobije zahtjev



Regije koje su dodijeljene radnicima od glavnog računala su stalne za cijelo vrijeme računanja. Veličine regija bi se mogle mijenjati ako se u nekoj regiji pojavi veliki broj točaka te bi ona tada zbog brzine moglo neki broj točaka preseliti nekom drugom računalu, ali to u ovom radu nećemo razmatrati.

## 5.2 Upit

Upit se šalje s glavnog računala na radnika. On se sastoji od područja za koje se upit šalje, broja komponente po kojoj se traži te ključan broj po kojem će radnici prepoznati da se radi o upitu. Na slici 7 prikazan je odnos glavnog računala i radilica kada se koristi lista pretinaca. Tada glavno računalo šalje upit direktno samo onim računalima (radilicama) čije regije imaju presjek s upitom. Kod binarnog stabla, upit se šalje na početni čvor koje prosljeđuje upite dalje drugim radilicama kao što je prikazano slikom 4.



*Slika 7. Prikaz upita i odgovora na njega*

Radnici, kojih može biti više, nakon što prime upit trebaju prebrojati za koliko točaka vrijedi taj upit te kao odgovor poslati broj točaka za koji to vrijedi te njihove identifikatore.

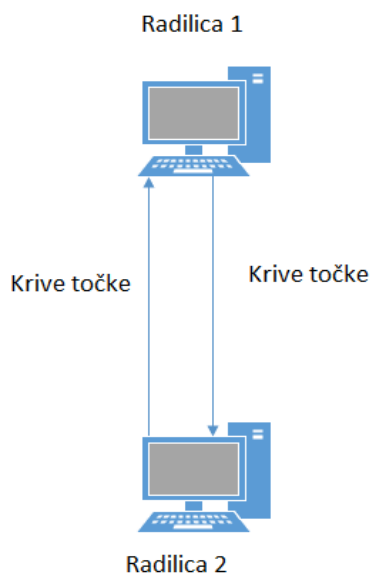
## 5.3 Pomak

Zahtjev za pomak znači da glavno računalo traži od svakog računala posebno da promijeni vrijednosti svojih točaka. Taj pomak može biti mali ili veliki. Svaki pomak ima minimalni i maksimalni postotni pomak. Postotni pomak je u odnosu na veličinu komponente.

*Tablica 4. Vrste korištenih pomaka u pokusima*

	Minimalni pomak	Maksimalni pomak
Mali pomak	0%	0.5%
Veliki pomak	40%	50%

Kada računalo primi takav zahtjev ono na svim svojim točkama radi promjenu vrijednosti i kao takve ih pamti za daljnje upite. Ako točka u promjenama svojih vrijednosti ispadne iz regije koja je određena trenutnom računalu ona se stavlja u listu točaka koja nisu raspoređene te će se ona morati preko mreže poslati računalu koje je nadležno za njezinu regiju.



*Slika 8. Razmjena krivo raspoređenih točaka*

Slanje točaka koje su postavljene u listu onih koje nisu raspoređene se odvija kada svako računalo od glavnog računala dobije znak da može početi slati krivo raspoređene točke preko mreže. Na slici 8. vidljivo je da u razmjeni točaka ne sudjeluje glavno računalo već samo radilice. Radilice tada neovisno jedne o drugima međusobno razmjenu točke kako bi ih stavili u točnu regiju. Regije koje su određene na početku ostaju konstantne za vrijeme izvođenja pa sama računala znaju kome točno trebaju poslati krive točke ako ih uopće i imaju, što znači da se slanje krivih točaka odvija u jednom koraku.

## 6. Mjerenje radnih svojstva i analiza rezultata

U šestom poglavlju predstaviti će se rezultati usporedbe dvije prethodno predstavljene strukture, binarnog stabla te liste pretinaca podataka. Sustav će biti podvrgnut raznih modifikacijama parametara, od različitog broja regija, različitih početnih raspršenja, broja točaka te mnogih drugih.

### 6.1 Opis usporedba

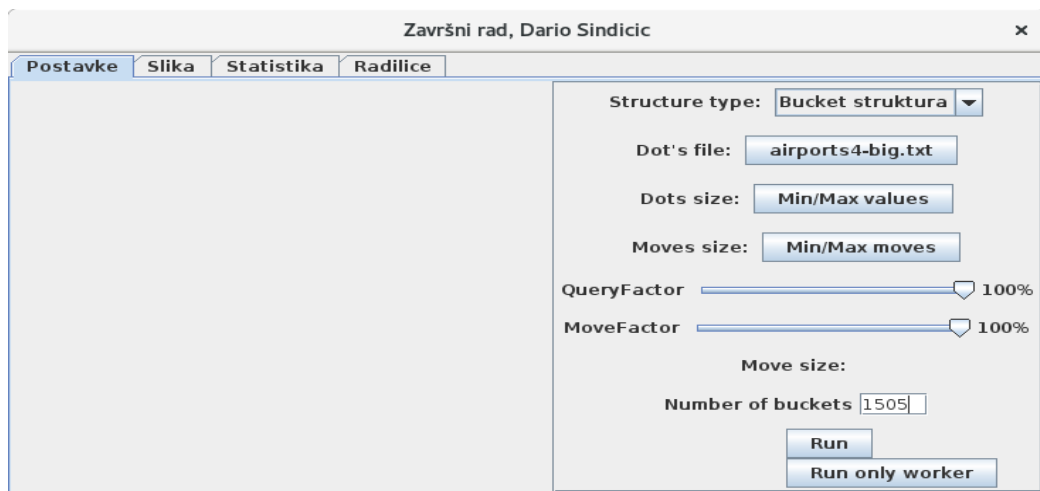
Svaki sustav u kojem postoje N-dimenzionalni podaci je specifičan sam za sebe. Podaci mogu biti raspršeni ili centrirani oko više točaka. Upiti na sustav mogu biti rijetki pa sve do vrlo čestih. Vrijednosti podataka mogu biti statičke ili se mogu mijenjati, a i sama promjena podataka može varirati od vrlo male pa sve velikih nedeterminističkih promjena. Broj računala koji obrađuje zahtjeve može biti veliki ili samo jedno računalo može biti zaduženo za cijeli sustav.

To su sve parametri kojima su bile podvrgnute gore dvije navedene strukture podataka kako bi se vidjelo koja pokazuje bolje performanse u navedenim specifičnim slučajevima. Pokusi su bili izvođeni nekoliko puta, u računalnim učionicama Fakulteta elektrotehnike i računarstva. Za vrijeme izvođenja pokusa, vatro zid je bio ugašen jer su računala razmjenjivala podatke preko TCP veze. Brzina veze je bila standardna 100 Mbit/s, a karakteristike računala u učionici su bile:

- CPU – 2.8 GHz, dvije jezgre
- RAM – 4 GB
- Windows 10

Sustav je bio programiran u programskom jeziku Java, verzije 9. [2], [3]. Na slici 9 nalazi se korisničko sučelje preko kojeg su se podešavali različiti parametri

sustava, dodavali radnici te se dobivala slika raspršenja točaka. On se pokrenuo na glavom računalu te je bio odgovoran za karakteristike sustava.



*Slika 9. Korisničko sučelje*

## 6.2 Parametri mjerenja

Parametri koji će se mijenjati su:

- broj računala
- broj točaka
- vrsta početnog raspršenja
- tip strukture
- veličina pomaka točke

U pokusima su bili korišteni dvodimenzionalni podaci. Njihov broj je bio variran od nekoliko milijuna pa do desetina milijuna.

Vrsta početnog raspršenja označuje način biranja početne pozicije točaka. U ovom slučaju uzete su dvije vrste:

1. uniformno biranje početne pozicije točke
2. uzimanje realnih podataka, u ovom slučaju su se uzimale geografske lokacije svjetskih aerodroma

Veličina pomaka točke je bila specificirana u poglavlju 4.1.2, tj. postoji mali i veliki pomak.

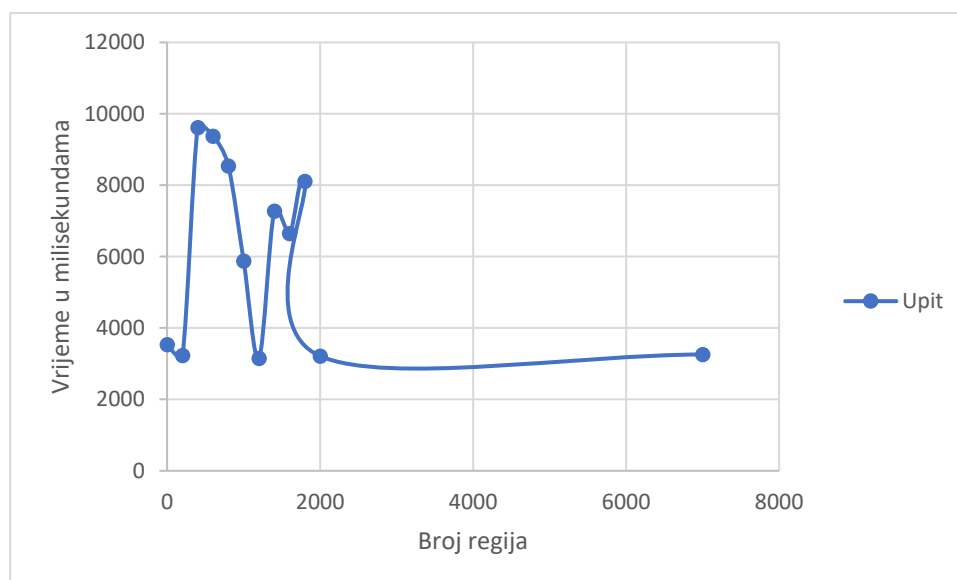
Broj računala nad kojima je sustav bio izvođen je varirao od samo jednog pa do njih 20.

Kod varijacije većeg broja računala, uvode se dva pojma, lokalna i globalna promjena. Lokalna promjena znači promjena vrijednosti točaka na samom računalu, dok globalna promjena znači slanje točaka preko mreže u njihove regije jer su promjenom izašli iz trenutne.

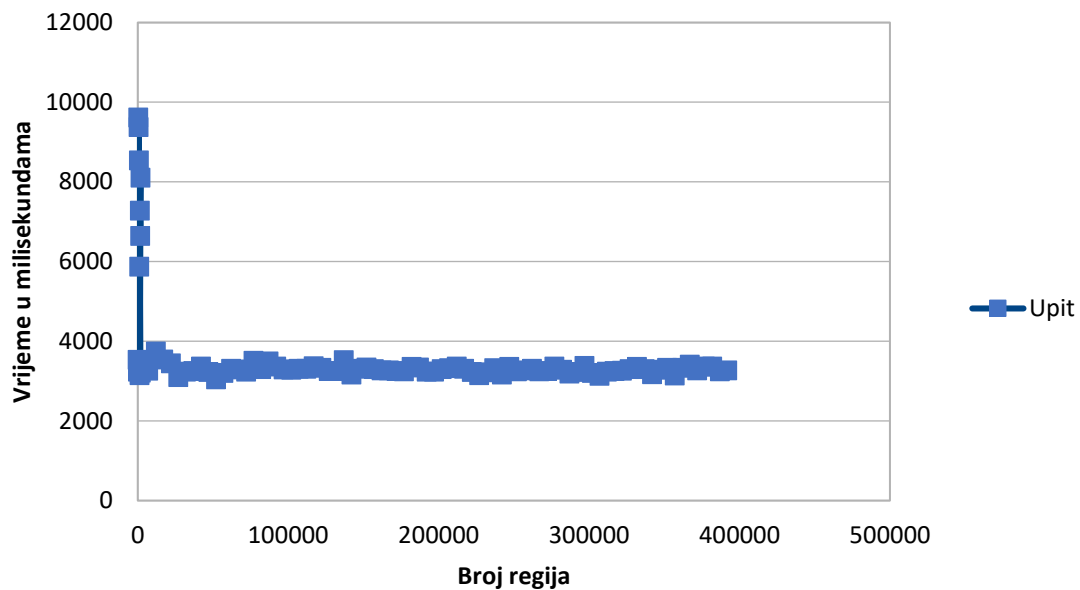
### 6.3 Utjecaj broja regija

Varijacija broja regija znači da će se mijenjati broj regija ili pretinaca u listi pretinaca. Pokus će se odvijati na jednom računalu uz realno raspršenje, maleni pomak te 18 milijuna točaka.

Na slici 10 prikazan je graf koliko je trajala obrada i odgovor na upit uz promjenu broja regija dok je na slici 11 prikazano koliko je trajala obrada i odgovor na upit uz puno veću promjenu regija.

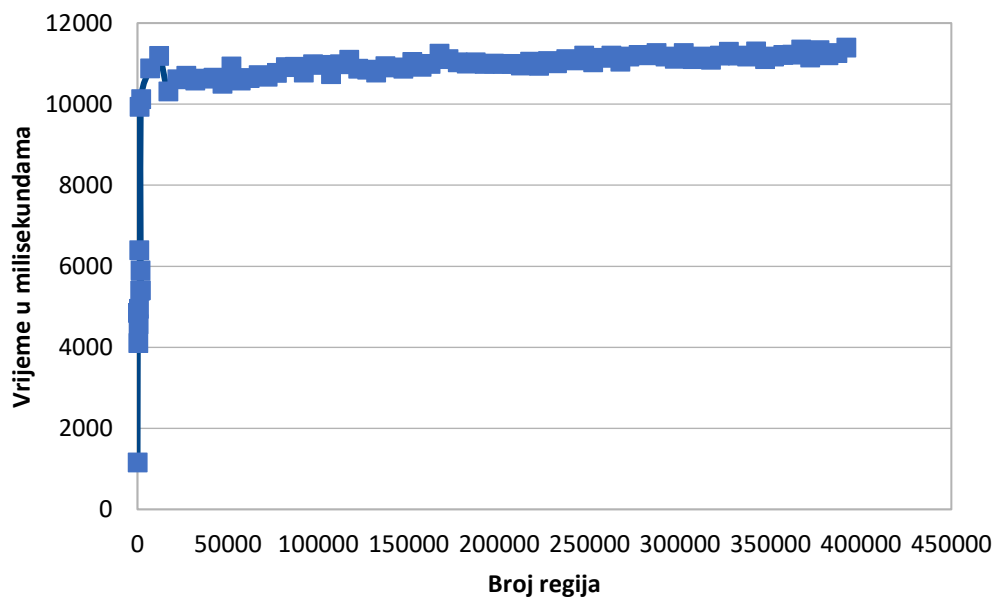


Slika 10. Vrijeme upita uz promjenu broj regija



*Slika 11. Vrijeme upita uz promjenu većeg broj regija*

Veliko povećanje broja regija nije rezultiralo velikim smanjenjem vremena upita, dok se kod malih povećanja broja regije sa slike 11 vidi da se kod povećanja broja regija smanjuje i vrijeme upita, ali samo do 2000 regija.

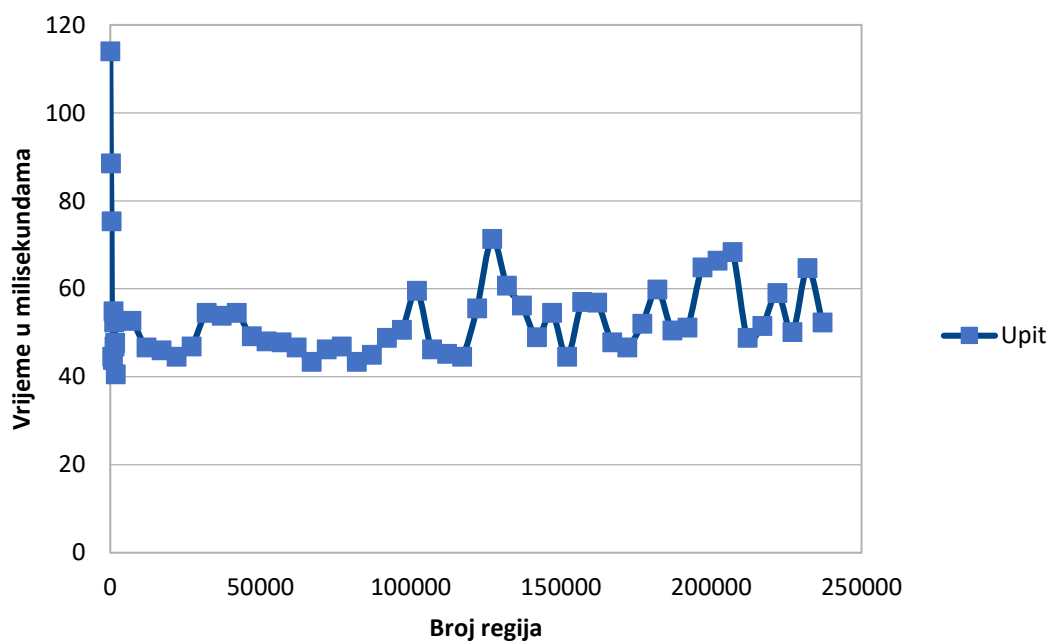


*Slika 12. Vrijeme promjene vrijednosti točaka uz promjenu broj regija*

Vrijeme promjene vrijednosti točaka raste kroz porast broja regija kao što je vidljivo sa slike 12. Razlog tomu je što povećanje broja regija, zahtjeva češću promjenu točke iz regije u regiju. No to vrijeme se stabilizira već oko 2000 regija.

## 6.4 Utjecaj početnog raspršenja

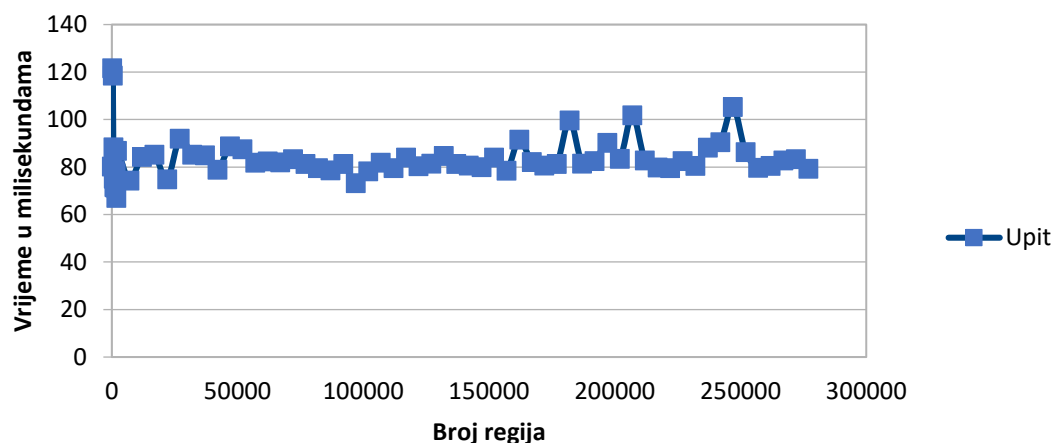
U ovom pokusu gledalo se kakvog utjecaja ima početno raspršenje točaka. U obzir su uzeta dva slučaja, slučajno odabrane točke te su uzete geografske lokacije svjetskih aerodroma što se u ovom slučaju zove realno raspršenje.



*Slika 13. Rezultat trajanja upita kod slučajnog raspršenja*

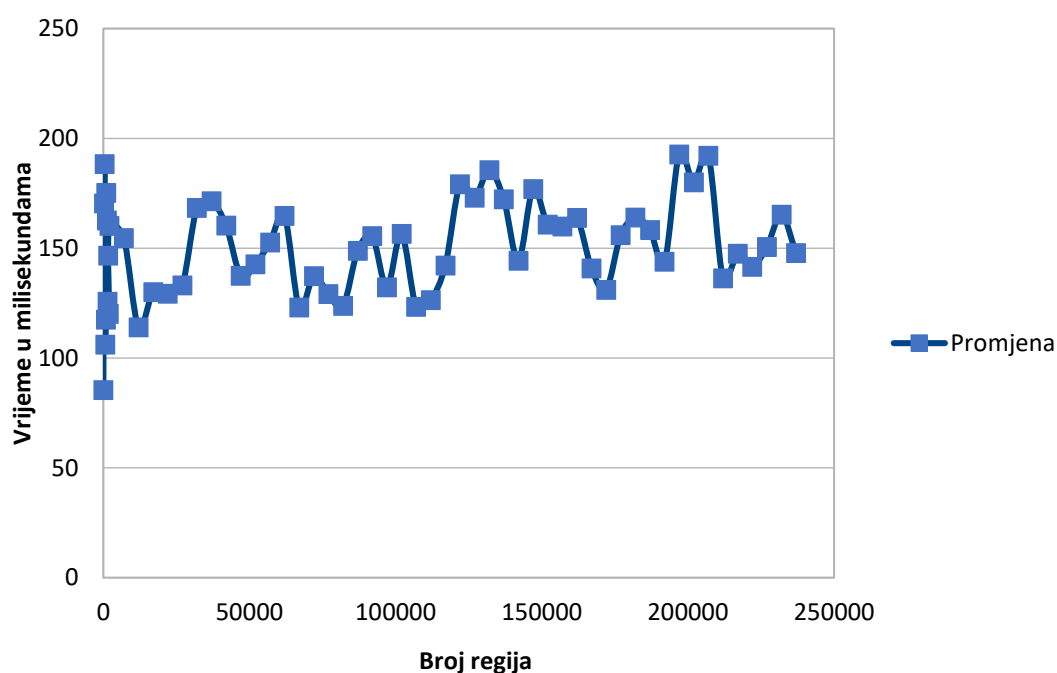
Kod slučajnog raspršenja vidljivo je da i u ovom slučaju uz značajno povećanje broja regija, vrijeme upita nije dodatno palo te se ustalilo već kod 20 000 regija.





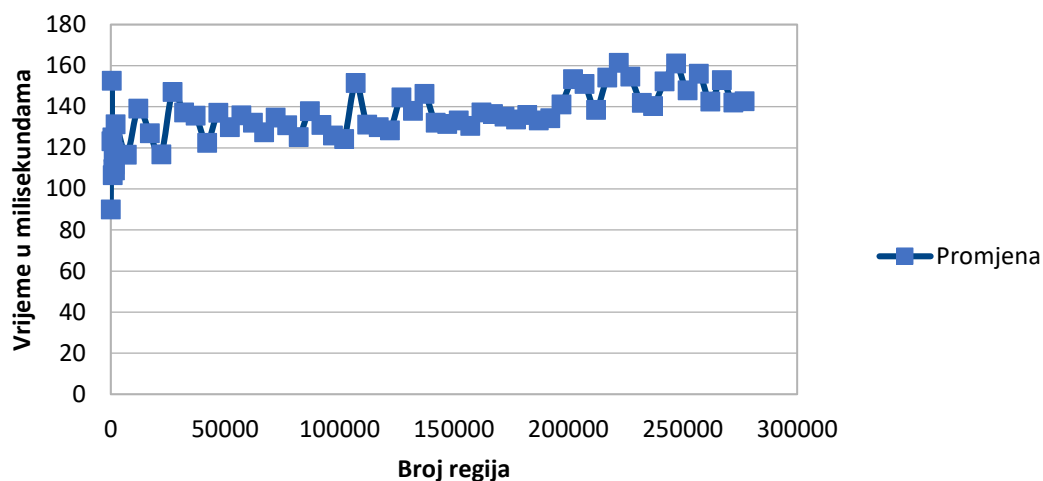
*Slika 14. Rezultat trajanja upita kod realnog raspršenja*

Kod realnog raspršenja vidljive su slabije performanse sustava što je moguće objasniti time što u tom slučaju nisu sve regije jednako popunjene što dovodi do povećanja vremenske složenosti.



*Slika 15. Vrijeme promjene točaka kod realnog raspršenja*

Sa slike 15. vrijeme promjene točaka kod realnog raspršenja raste do 7000 regija da bi se tada počelo smanjivati i biti konstantno za broj regija veći od 20 000.



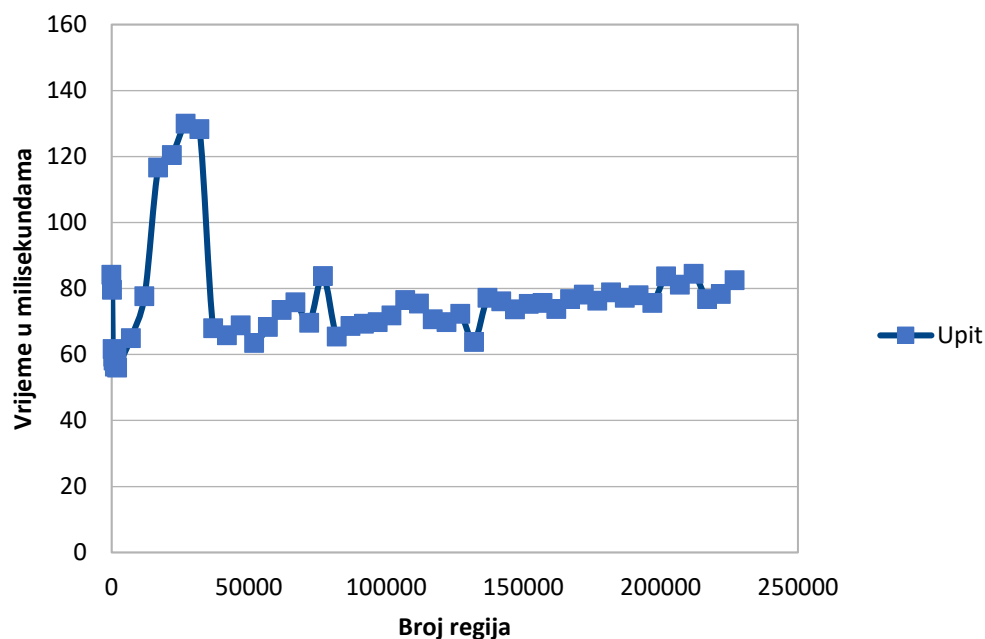
*Slika 16. Vrijeme promjene točaka kod slučajnog raspršenja*

Sa slike 16 vidi se da je vrijeme promjena kod slučajnog raspršenja je manje nego kod realnog raspršenja. Glavni uzrok tomu je što regije nisu podjednako zastupljene kod realnog raspršenja. Kod slučajnog raspršenja, broj točaka u regiji je manji te je i njihovo micanje i dodavanje puno brže.

Binarno stablo u ovom slučaju nije pokazalo nikakve razlike. Realno ili slučajno raspršenje ne mijenja performanse stabla, ali ono je pokazalo lošije rezultate od liste pretinaca.

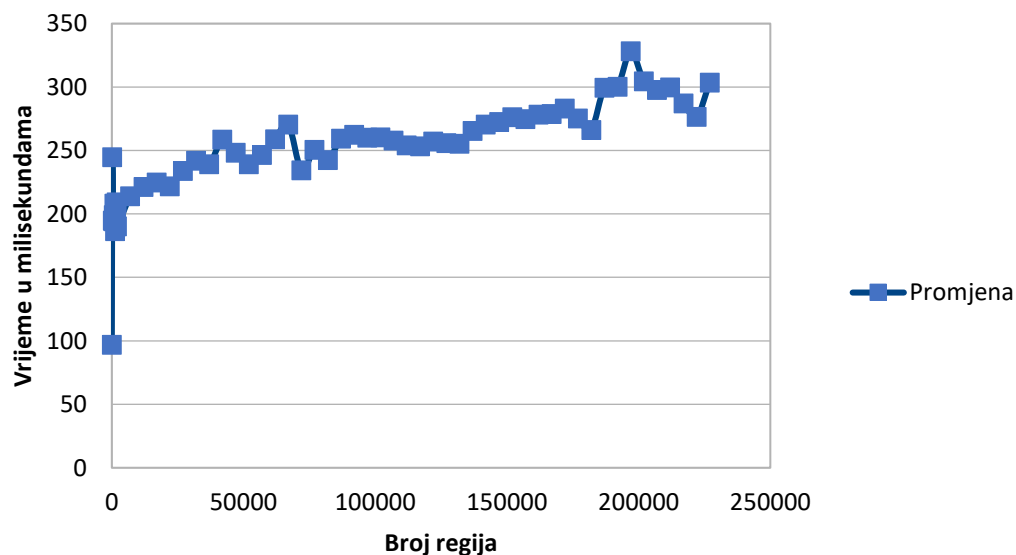
## 6.5 Utjecaj broja točaka

U ovom pokusu razmatrao se broj točaka te kako njihov broj utječe na performanse sustava. Pokus se izvodio na jednom računalu, uz realno raspršenje te mali pomak.



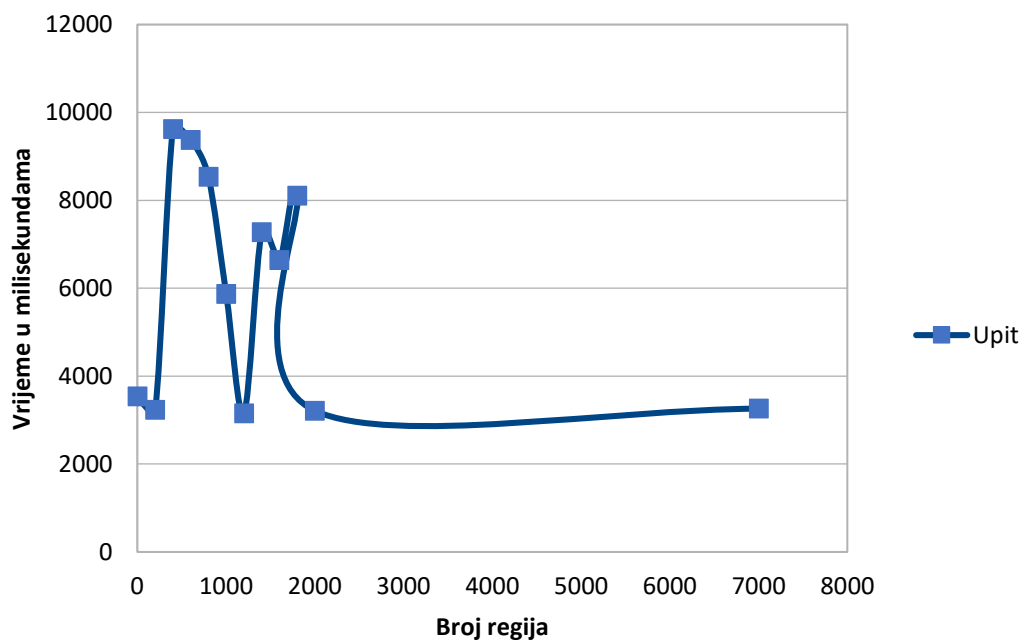
*Slika 17. Vrijeme upita uz promjenu broja regija, broj točaka = 5 milijuna*

Lista pretinaca podataka s 5 milijuna točka, na slici 17 pokazuje da se nakon 50 000 regija vrijeme upita ne mijenja.



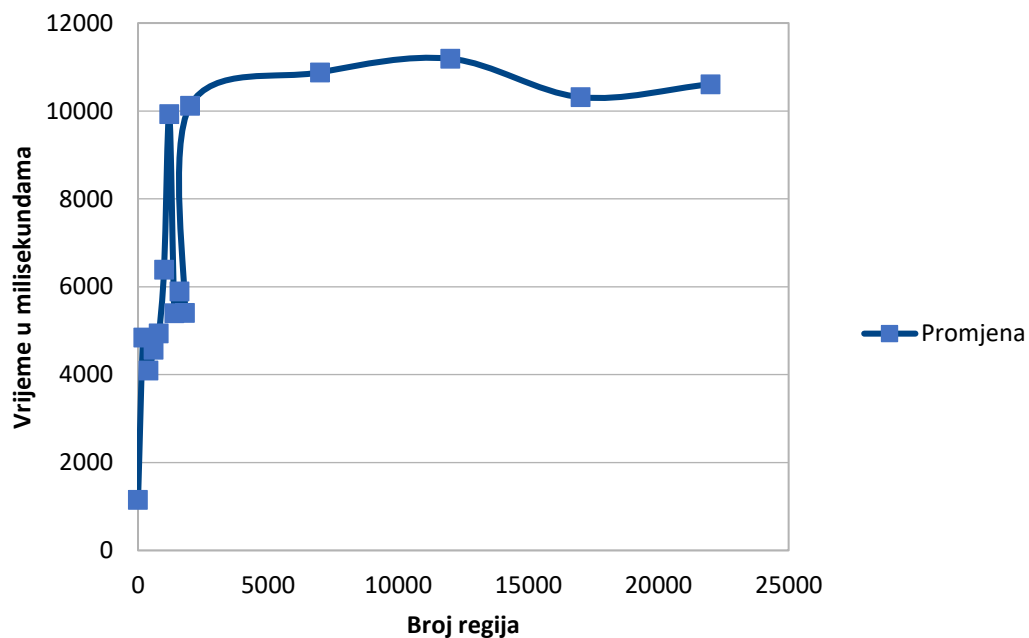
*Slika 18. Vrijeme promjene točaka uz promjenu broja regija, broj točaka = 5 milijuna*

Slika 18 pokazuje da vrijeme promjene raste kroz povećanje broja regija jer raste vjerojatnost promjene regije za pojedinu točku.



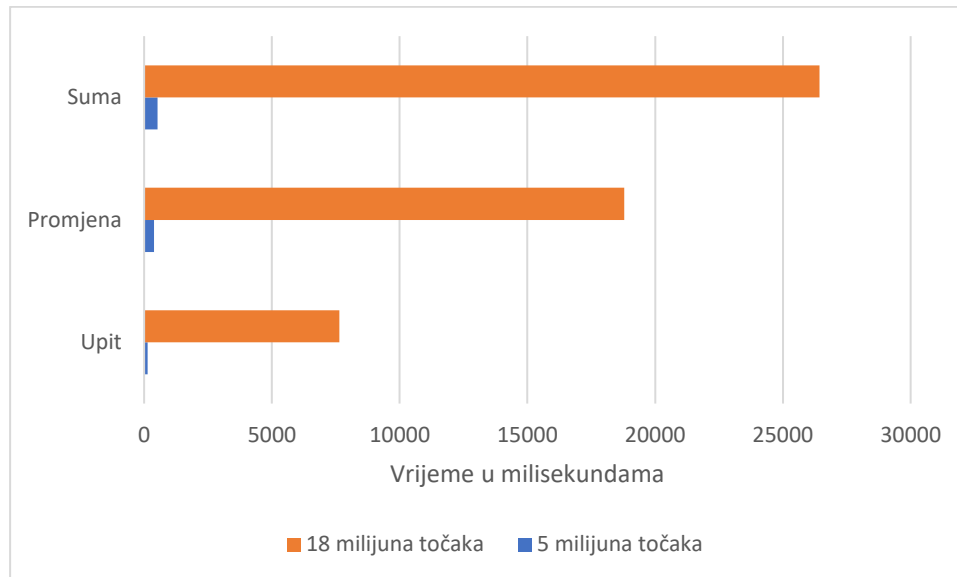
Slika 19. Vrijeme upita uz promjenu broja regija, broj točaka = 18 milijuna

Na slici 19 vidimo performanse kada u sustav stavimo 18 milijuna točaka. Vrijeme upita oscilira do 2000 regija da bi se nakon toga ustalilo.



Slika 20. Vrijeme promjene točaka uz promjenu broja regija, broj točaka = 18 milijuna

Vrijeme promjena raste kao i kod primjera s 5 milijuna točaka s povećanjem broja regija. Performanse sustava s 5 milijuna točaka su 50 puta bolje nego kod 18 milijuna točaka te možemo zaključiti da performanse sustava slabe puno brže s povećanjem broja točaka.

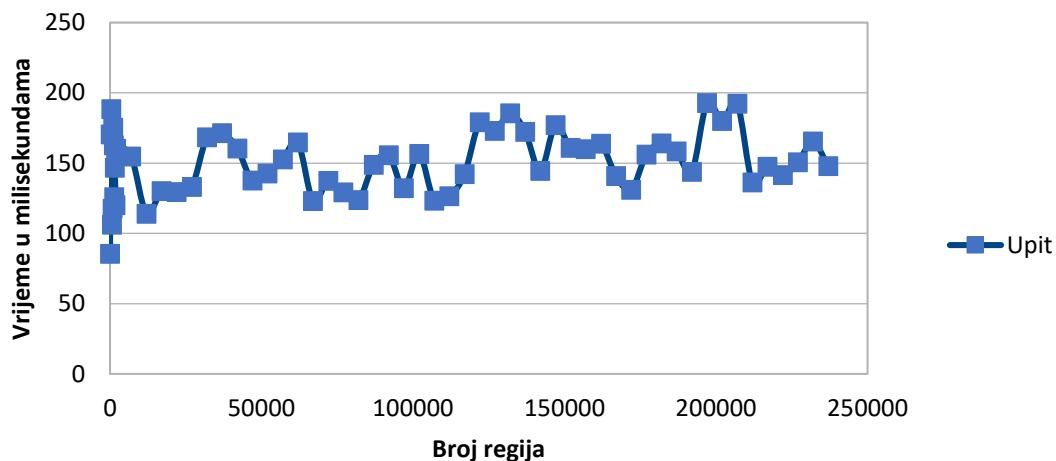


*Slika 21. Vrijeme upita i promjena kod binarnog stabla u slučaju 5 i 18 milijuna točaka*

Na slici 21. vidljivo je da je sustav puno sporiji s 18 milijuna točaka te odnos vremena nije linearno zavisn s brojem točaka. Glavni uzroku tomu je veći broj točaka u binarnom stablu te tada balansiranje traje puno duže.

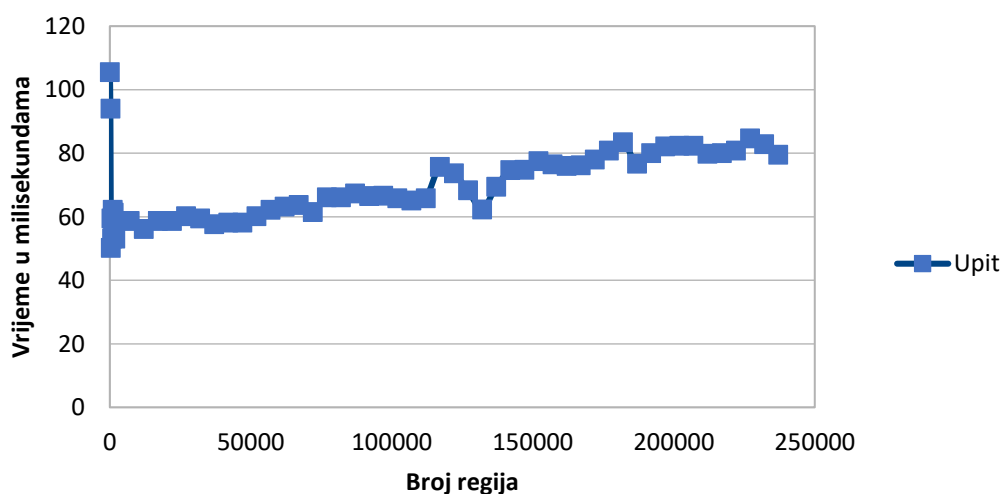
## 6.6 Utjecaj veličine pomaka točaka

U ovom pokusu varirana je bila veličina pomaka točaka. Pokus se izvodio na jednom računalu, uz realno raspršenje te 5 milijuna točaka.



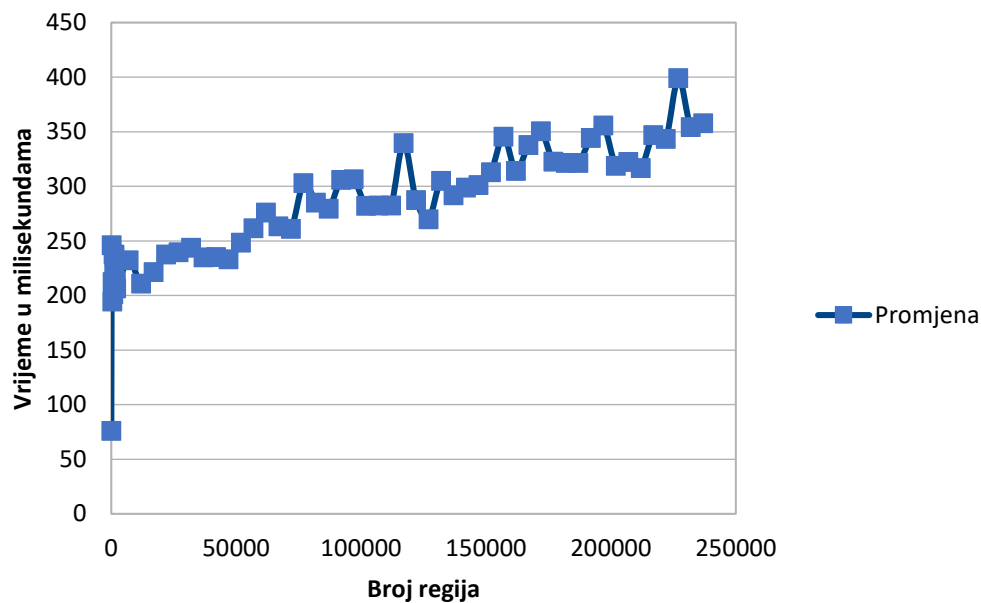
*Slika 22. Rezultat trajanja upita kod malih pomaka*

Na slici 22 vidimo varijaciju broja regija uz male pomake te promatramo vrijeme obrade upita. Kod malih promjena struktura se bitno ne mijenja stoga je i vrijeme bilo konstantno.



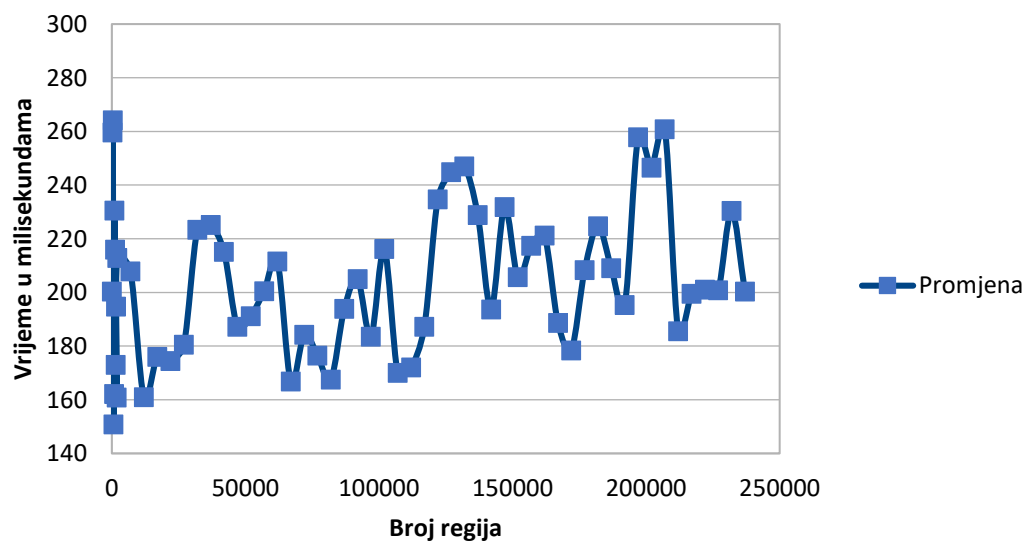
*Slika 23. Rezultat trajanja upita kod velikih pomaka*

Na slici 23. vidi se vrijeme upita kod velikih pomaka te ono izrazito raste uz povećanje broja regija. U tome pokazuje drugačija svojstva nego kod malih promjena. Razlog tomu su puno veće promjene na strukturi te podaci neće biti slijedno zapisani. Tada raste vjerojatnost da će se za podatak morati ići u memoriju što usporava cijeli proces.



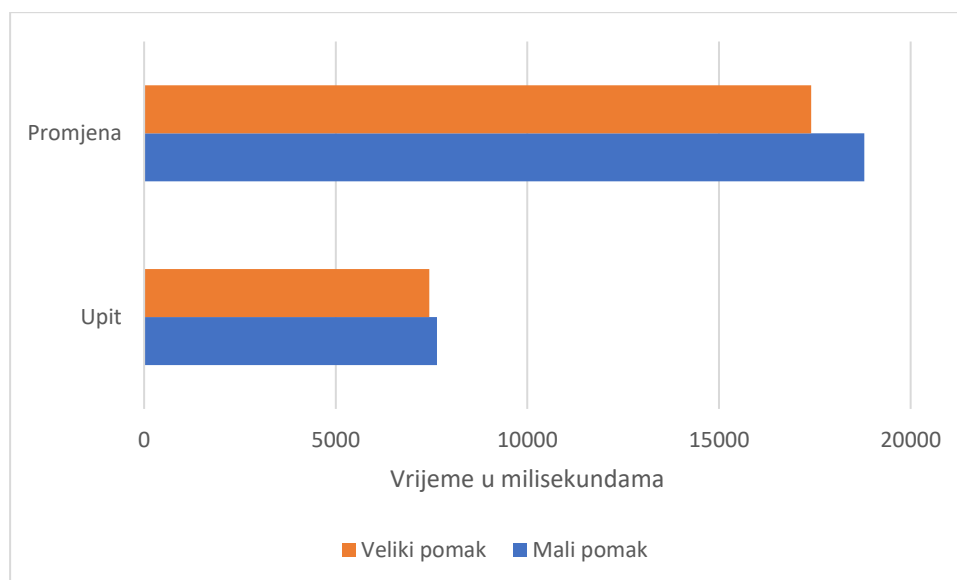
*Slika 24. Vrijeme promjene točaka kod velikog pomaka*

Na slici 24. vidimo graf vremena promjena točaka kod velikog promjena. Kao što je i prije navedeno, zbog velikih promjena na strukturi raste i vrijeme potrebno za obradu.



*Slika 25. Vrijeme promjene točaka kod malih pomaka*

Slika 25 pokazuje da kada imamo male promjena, vrijeme promjene točaka je otprilike konstantno te se pokazuje da povećanje broja regija ne utječe značajno na taj atribut.



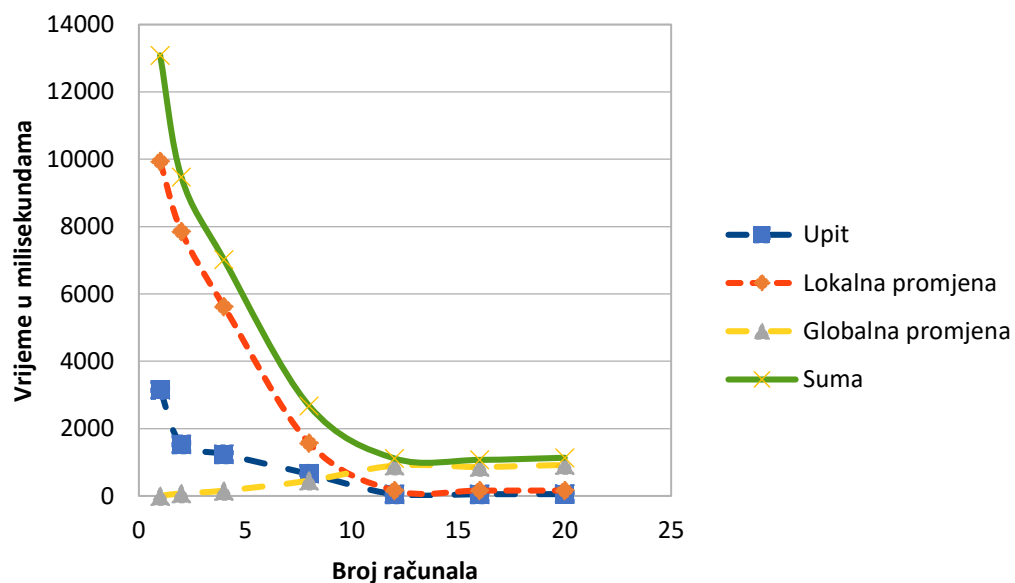
*Slika 26. Varijacija veličine pomaka kod binarnog stabla*

Za razliku kod binarnog stabla, ovdje se ne vide značajne razlike u odabiru malog i velikog pomaka, kao što je vidljivo sa slike 26. Razlog tomu je što već kod male promjene može doći do promjene strukture stabla te njegovog balansiranja, dok se kod liste pretinaca na malim promjenama može dogoditi da točka ne napusti regiju.

## 6.7 Utjecaj broja računala

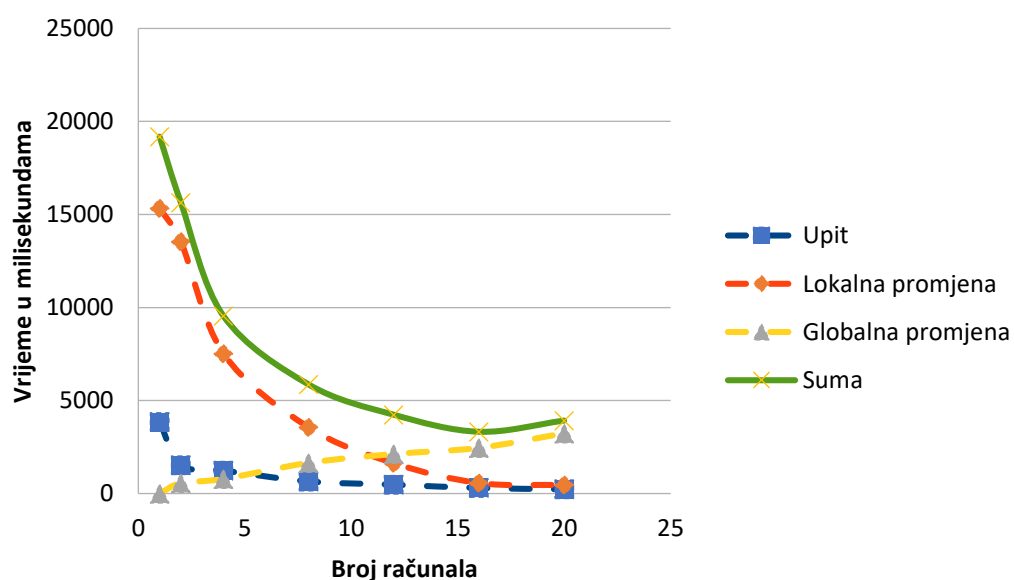
U ovom pokusu je bio variran broj računala koji će biti uključeni u obradu, tj. koliko će radilica obrađivati zadane operacije. Također bit će i varirana veličina pomaka kod varijacije vrijednosti točaka.





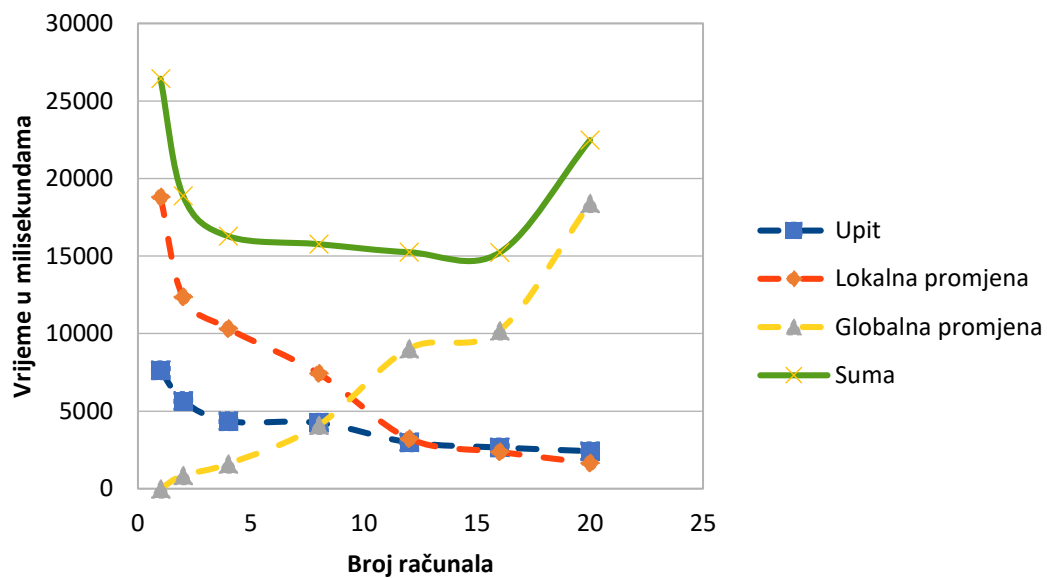
*Slika 27. Performanse sustava kod malih pomaka – Lista pretinaca podataka*

Na slici 27 prikaze su performanse liste pretinaca podataka kod malih pomaka. Vrijeme odgovora na upit te lokalna promjena su pale vrlo brzo te su već počele stagnirati kada se broj radilica povećao na 12. Globalna promjena je rasla s povećanjem broja radilica zbog smanjenja regija na jednom računalu.

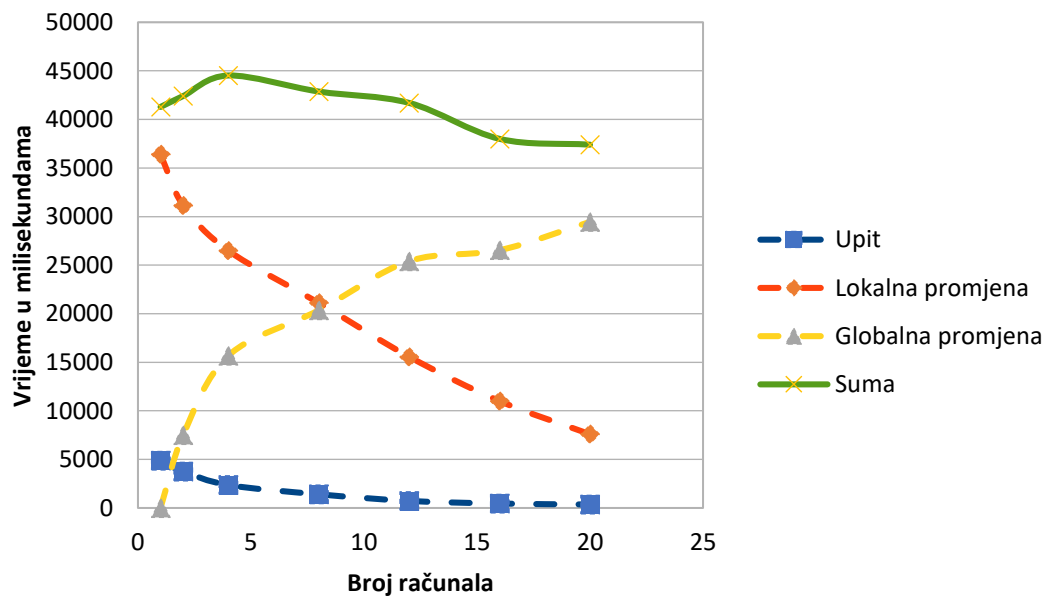


*Slika 28. Prikaz performansi sustava kod velikih pomaka – Lista pretinaca podataka*

Kod velikih promjena, na slici 28. vidljivo je da globalna promjena ima veliki utjecaj na ukupan rezultat. Kod velikih promjena, veća je vjerojatnost da će se točka morati slati preko mreže drugom računalu što bitno usporava proces.



Slika 29. Prikaz performansi sustava kod malih pomaka – Binarno stablo



Slika 30. Prikaz performansi sustava kod velikih pomaka – Binarno stablo

Binarno stablo karakteristično je po brzom odgovoru na upit što se pokazalo u ovom slučaju istinito. Isto kao i kod prethodne strukture, lokalna promjena je također padala s povećanjem broja radilica dok je globalna promjena rasla kao što je vidljivo sa slike 29. Binarno stablo pokazalo je da kada se posao rasporedi na veliki broj računala, to postaje biti kontraproduktivno te se ukupno vrijeme odziva počinje povećavati.

Kod veliki promjena, na slici 30 vidljivo je da su performanse sustava dodatno pale zbog globalne promjene.

## **7. Usporedba radnih svojstva binarnog stabla i podjele prostora na regije**

U ovome poglavlju predstaviti će se rezultati iz prošlog poglavlja te će se istaknuti kada je bolje koristiti listu pretinaca, a kada binarno stablo. U obzir će se uzimati frekvencije učestalosti upita i promjena koje dolaze na sustav.

### **7.1 Usporedba po broju točaka**

Veće razlike u usporedbi po broju točaka se nisu vidjele. Obje strukture su logično pokazale pad performansi porastom broja točaka što je utjecalo ne samo na povećanje vremena upita nego i na povećanje vremena promjene zbog porasti količine podataka koje struktura mora promijeniti i održavati.

Binarno stablo je pokazalo bolje performanse upita od liste pretinaca, kod slučaja 18 milijuna točaka, kada je broj regija bio između 400 i 800. Ukoliko bi frekvencija upita bila izrazito velika, a broj promjena izrazito mali onda bi binarno stablo u tom slučaju bio bolja opcija.

### **7.2 Usporedba po veličini pomaka točaka**

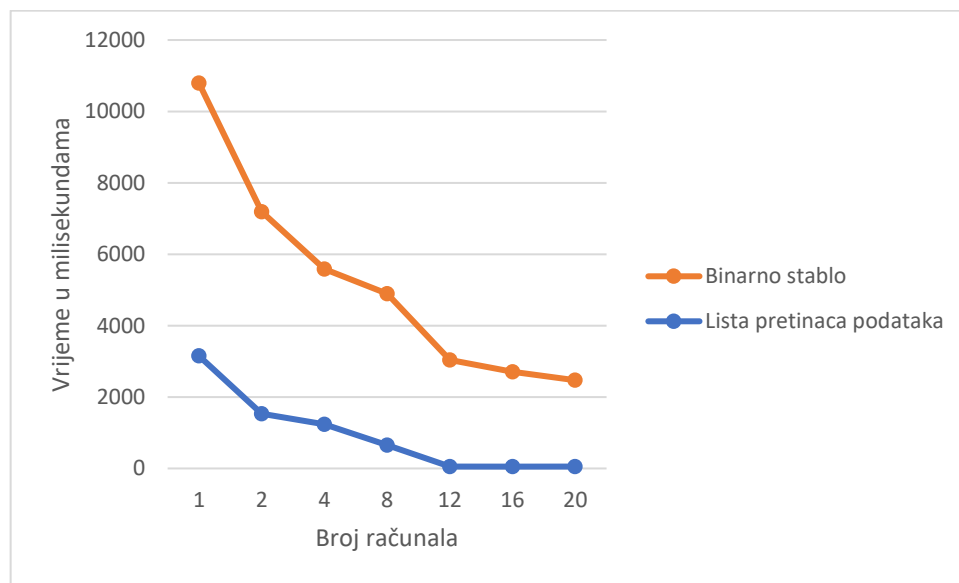
Veća razlika između struktura se u ovom slučaju ne vidi. Obje strukture pokazuju pad performansi kada je veličina pomaka izrazito velika. Vrijeme upita se tada postepeno povećava kako se točke sve više i više miču. Rezultat toga je sve veći i veći broj cache promašaja [1]. Sukladno s dobivenim rezultatima, da bi sustav zadovoljavao dane zahtjeve, frekvencija promjena mora biti mala.

### **7.3 Usporedba po broju računala**

Obje strukture pokazuju rast performansi ako je veći broj računala uključen u obradu. To se pokazalo neovisno o frekvenciji upita, promjene ili veličini pomaka. Iako i tu je postojala granica, [6]. Ako se u obradu uključio veći broj računala, vjerojatnost globalne promjene ili slanja krivih točaka je postala veća te su se one

slale preko mreže. Vrijeme upita je padalo s povećanjem broja računala, ali je promjena rasla s njima. Obrat se događa kada se u igru uvedu 20 računala. Tada veličina regije postaje još manja te se uz velike promjene skoro svaka točka mora slati ispravnom računalu preko mreže što usporava cijeli proces.

Lista pretinaca podataka je pokazala puno bolje performanse zbog direktnog upita, dok je binarno stablo trebalo pitati slijedno više računala da dođe do rezultata, rezultat toga je prikazan na slici 31. Neovisno o frekvencijama upita ili promjena lista pretinaca se pokazala kao bolje rješenje.



*Slika 31. Usporedba vremena na upit za dvije strukture uz promjenu broja računala uključenih u obradu*

## 8. Zaključak

Podaci velike dimenzije zahtijevaju drugačiji pristup ako se želi da se operacije nad njima izvode brže. Umjesto kompleksnih struktura kao K-D stabla [7], odabrane su jednostavnije strukture s jednostavnom implementacijom. Razlog tomu bio je podrška sustave s velikim frekvencijama promjena, a kod kompleksnijih strukturi dosta vremena bi se trošilo samo na njihovo balansiranje pri čemu bi performanse pale. U ovom radu odlučeno je da će se usporediti dvije jednostavne strukture, binarnog stabla te liste pretinaca podataka inspirirane algoritmom sortiranja s podjelom na regije. Također u obradu se uključilo više računala koja su bila povezana putem TCP/IP veze, te su se njihovi poslovi podijelili tako da budu nezavisni kako bi značajke Map&Reduce principa izašle na vidjelo.

Implementacija binarnog stabla oslonila se na implementaciju Red-Black stabla kao djela standardne Java biblioteke, dok se za listu pretinaca koristila jednostruko povezana lista zbog mogućih čestih promjena u sustavu.

Rezultati mjerenja su očekivani, lista pretinaca podataka je pokazivala bolje performanse, ali ne u svim slučajevima. U tim slučajevima je binarno stablo imalo bolje performanse u upitima kada je njihova frekvencija bila visoka, a frekvencija promjena mala, ali mnogo lošije u promjeni vrijednosti. Kada se uvelo više računala, lista pretinaca je pokazivala još bolje rezultate, što može zahvaliti direktnom upitu koji ide odmah na sva računala zadužena za te regije. Dok kod binarnog stabla upit ne ide direktno. Upit se prvo šalje na prvo računalo u stablu te ga ono propagira drugima.

Naravno smanjenjem broja regija, ili pretinaca liste pretinaca, pada i njezina brzina, te binarno stablo puno brže nađe željene točke dok lista pretinaca mora proći po svim točkama u regiji, što može biti veliki broj.

## 9. Literatura

- [1] Percival, Colin. "Cache missing for fun and profit." (2005)
- [2] Java, <https://java.com>
- [3] Eclipse, <http://www.eclipse.org/>
- [4] Corwin, Edward, and Antonette Logar. "Sorting in linear time-variations on the bucket sort." *Journal of computing sciences in colleges* 20.1 (2004): 197-202.
- [5] TreeSet (Java SE 9 & JDK 9),  
<https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>
- [6] Hill, Mark D., and Michael R. Marty. "Amdahl's law in the multicore era." *Computer* 41.7 (2008).
- [7] Radošević, Stefan. „Izrada K-D stabla za pronalazak najbližeg susjeda u 3D oblaku točaka“, (2016), završni rad, Sveučilište Josipa Jurja Strossmayera u Osijeku  
Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek
- [8] Binary tree - Wikipedia, [https://en.wikipedia.org/wiki/Binary\\_tree](https://en.wikipedia.org/wiki/Binary_tree)
- [9] Bucket sort - Wikipedia, [https://en.wikipedia.org/wiki/Bucket\\_sort](https://en.wikipedia.org/wiki/Bucket_sort)
- [10] Binary search tree - Wikipedia,  
[https://en.wikipedia.org/wiki/Binary\\_search\\_tree](https://en.wikipedia.org/wiki/Binary_search_tree)

# Algoritmi ažuriranja i pretraživanja raspodijeljenih višedimenzionalnih podataka primjenom podjele prostora na regije

## Sažetak

U ovom radu napravljena je usporedba dviju struktura podataka, binarnog stabla i liste pretinaca podataka, na problemu ažuriranja i pretraživanja višedimenzionalnih podataka primjenom podjele prostora na regije. Opisano je binarno stablo i sortiranje podjelom na regije te su analizirane njihove složenosti. Opisan je sustav za mjerenje i usporedbu radnih svojstva s podjelom prostora na regije nad kojim su se izvodila mjerenja. Binarno stablo i lista pretinaca podataka su međusobno uspoređene na različitim karakteristikama sustava kao što su učestalost promjene, broj računala uključen u obradu, broj točaka u sustavu, veličina pomaka i broj regija. Prikazani su rezultati mjerenja i izvedeni su zaključci o primjenjivosti pojedine strukture u ovisnosti o karakteristikama podataka i sustava.

## Ključne riječi

višedimenzionalni podaci, binarno stablo, podjela na regije



# Insert and Search Algorithms over Distributed Multidimensional Data Using Bucket-based Scattering

## **Summary**

In this thesis, the performance of two data structures, binary tree and bucket structure, against the insertion and searching of the multidimensional data is presented. Binary tree and bucket sort algorithm are described and their complexity is analyzed. A distributed system designed for performance measurement is presented. Binary tree and bucket structure were compared with each other on different system's characteristics, such as frequency of updates, the number of computers used for data processing, number of points in system, scale of movement, and number of regions. The results are presented and the conclusions about the usability of particular data structure with respect to the given data and system's characteristics are given.

## **Keywords**

multidimensional data, binary tree, bucket-based scattering