SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

ZAVRŠNI RAD br. 5133

# Algoritam staničenja velike baze podataka genoma

Mateo Stjepanović

Zagreb, lipanj 2017.

*Umjesto ove stranice umetnite izvornik Vašeg rada.*
*Da bi ste uklonili ovu stranicu obrišite naredbu* `\izvornik`.

*Zahvaljujem mentorici doc. dr. sc. Mirjana Domazet-Lošo na podršci.*

# SADRŽAJ

# 1. Uvod

Bioinformatika je interdisciplinarna znanost koja se bavi razvojem programa i metoda za interpretiranje bioloških podataka. Spaja matematiku, statistiku, računalnu znanost te druge prirodno matematičke znanosti. U posljednjem desetljeću bioinformatika kao znanost doživaljava veliki porast, te kao takva uspjeva mapirati genome mnogih živih bića.

U navedenom porastu najviše se ističu metode analiziranja genoma, te, spajanjem statističkih analiza i algoritama iz područja računalne znanosti, metode određivanja položaja istih u taksonomskom stablu. Upravo u tom području postoje mnogi alati koji su razvijeni upravo s ciljem poboljšanja točnosti određivanja taksonomskog stabla novoj, nepoznatoj, jedinki.

Za većinu jedinki u uzorku su vrsta, rod i više razine stabla nepoznati. Ako je jedinka potpuno nepoznata tada će se klasificirati kao nova nepoznata vrsta te se neće dalje klasificirati. U velikom broju slučajeva će postojati neke sličnosti sa nekim već određenim vrstama, te je potrebno pronaći te sličnosti kako bi se sjedinka uspješno klasificirala. Za to služe razni algoritmi poravnanja. Jedan od tih je BLAST. Postoje druge metode i programi koji pospješuju učinkovitost BLAST-a uvođenjem metoda strojnog učenja. Iako bolje preciznosti ti programi imaju jednu veliku manu. Vrijeme potrebno za klasificiranje podatka je jako veliko, stoga su više manje neiskoristivi.

Tu na scenu nastupa alat za klasifikaciju metagenoma nazvan Kraken(Derrick E. Wood i Steven L.).

*Kraken is ultrafast and high accurate program for assigning taxonomic labels to metagenomic DNA sequences.*[1]

Za razliku od alata koji su pokušali poboljšati preciznost BLAST algoritma, te time izgubili na brzini, Kraken je jedan od rijetkih alata koji postiže točnost koja premašuje onu u BLAST algoritmu, s time da ne gubi na brzini izvođenja. Rad Krakena se sastoji od toga da se ulazni podatak "razbija" na k-mere, te se minimizer algoritmom traže oni

[1] Wood and Salzberg: **Kraken: ultrafast metagenomic sequence classification using exact alignments.** *Genome Biology* 2914 15:R46

k-meri koji imaju istu vrijednost minimzer-a. Tada se taj podatak uspoređuje s podacima u Kraken-ovoj bazi podataka, s tim da je baza sortirana na način da su podaci sa istim minimizerom smješteni jedan pored drugog, tako da se pretraživanje jako pospješuje i ubrzava. Autori alata su shvatili da Kraken može naići an problem prilikom izvođenja na računalu s ograničenim resursima, točnije na računalu s RAM-om ispod 70GB. Iz tog razloga je razvijena MiniKraken baza podataka koja je sa prvotnih 70GB podataka smanjena na 4GB.

| | HiSeq | | MiSeq | | simBA-5 | |
|---|---|---|---|---|---|---|
| Classifier | Precision | Sensitivity | Precision | Sensitivity | Precision | Sensitivity |
| Megablast | 99.03 | 79.00 | 92.44 | 75.76 | 96.93 | 93.67 |
| NBC | 82.33 | 82.33 | 77.78 | 77.78 | 97.64 | 97.64 |
| PhymmBL | 79.14 | 79.14 | 76.21 | 76.21 | 96.11 | 96.11 |
| PhymmBL65 | 99.13 | 73.95 | 92.47 | 73.03 | 99.08 | 95.45 |
| Kraken | 99.20 | 77.15 | 94.71 | 73.46 | 99.90 | 91.25 |
| Kraken-Q | 99.12 | 76.31 | 94.69 | 70.41 | 99.92 | 89.54 |
| MiniKraken | 99.44 | 66.12 | 97.41 | 67.95 | 99.95 | 65.87 |
| MiniKraken-Q | 99.36 | 65.67 | 97.32 | 65.84 | 99.98 | 65.31 |
| Kraken-GB | 99.51 | 93.75 | 98.48 | 86.23 | 99.48 | 91.13 |

**Tablica 1.1:** Klasifikacija roda za tri metagenoma

# 2. Definiranje problema

Uz već spomenutost rješenja problema korištenja velike količine resursa, nailazi se na problem koji će se pokušati riješiti tijekom ovog rada.

| | HiSeq | | MiSeq | | simBA-5 | |
|---|---|---|---|---|---|---|
| Classifier | Precision | Sensitivity | Precision | Sensitivity | Precision | Sensitivity |
| Kraken | 99.20 | 77.15 | 94.71 | 73.46 | 99.90 | 91.25 |
| MiniKraken | 99.44 | 66.12 | 97.41 | 67.95 | 99.95 | 65.87 |

**Tablica 2.1:** Isječak iz Tablice 1.1

Iako se iz dane tablice vidi da se preciznost korištenjem MiniKraken baze podataka neznatno povećala u odnosu na Kraken bazu podataka. S druge strane osjetljivost jako opada. Osjetljivost predstavlja mjeru koja određuje postotak uočenih lažno negativnig podataka, tj. predstavlja postotak točno kvalificiranih podataka koji su uistinu tako i predstavljeni. S druge strane precizonst predstavlja izbjegavanje lažno pozitivnih podataka, tj. koliko podataka je uočeno da ne pripadaju nekoj skupini, s tim da oni uistinu ne pripadaju toj skupini.

Ideja ovog rada je da se algoritmima sličnim algoritmima straničenja osposobi računalo s ograničenim resursima za rad s bazom podataka čija veličina uvelike nadilazi količinu RAM-a raspoloživog na računalu. Prvotno je potrebno podjeliti bazu podataka na više manjih koje će se moći uspješno učitati u memoriju te pretražiti. Ideja je iskoristiti "kontejnere" k-mera koji su određeni svaki jedinstvenom minimizer vrijednosti, te pomoću binarnog pretraživanja indeksa učitavati samo onu bazu podataka pomoću koje je moguće klasificirati podatak u taksonomsko stablo. Želeći se približit straničenju, predstavit će se samo iteriranje po indeksima te učitavati slijedno baze podataka i pretraživati svaku.

# 3. Strukture podataka i algoritmi u Kraken-u

### 3.0.1. Baza podataka

#### Kreiranje baze podataka

Nastanak baze podataka se događa u nekoliko koraka. Za početak se sa NCBI-a preuzimaju biblioteke koje sadrže trenutno poznate vrste. Budući da pretraživanje baze podataka u Kraken-u osniva na k-merima potrebno je proizvesti k-mere za svaki podataka u bazi podataka. Za to služi alat naziva Jellyfish(Guillaume Marçais i Carl Kingsford) koji računa k-mere zadane duljine 31 ( korisnik također može zadati duljinu k-mera).
Jellyfish je alat za brzo i memorijski povoljno prebrojavanje podnizova u danom nizu. Za potrebe prilagodbe svim resursima za Jellyfish postoje opcije za brzo izračunavanje koje je memorijski zahtjevnije, te ono sporije ali memorijski ne toliko zahtjevno. U kontekstu bioinformatike Jellyfish se koristi za prebrojavanje k-mera u zadanom metagenomu. Sam rad se bazira na hash tablici koja je osposobljena za paralelan rad preko više CPU-a. Hash tablica se sastoji od para (ključ,vrijednost), gdje je ključ zadani k-mer a vrijednost je broj njegovog ponavljanja u metagenomu.

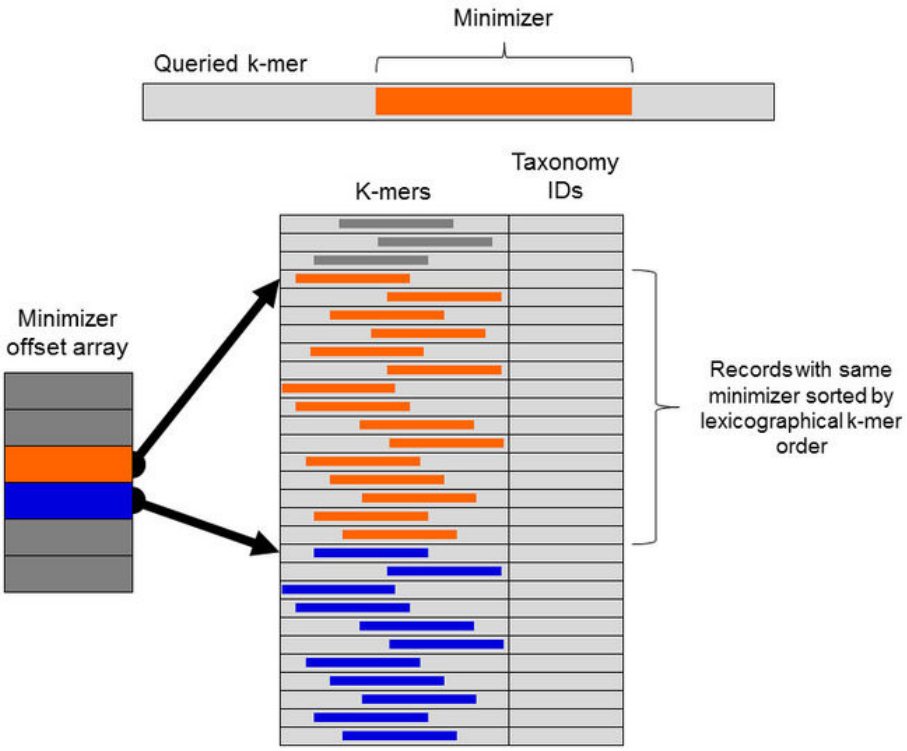
**Tablica 3.1:** Prikaz niza te njegovih k-mera (k = 4)

| | |
|---|---|
| Ulaz: | AGATCGAGTG |
| 4-mer: | AGAT GATC ATCG TCGA CGAG GAGT AGTG |

Nakon rada Jellyfish-a u bazu se spremaju 4 bajtne vrijednosti k-mera te se kreira identifikacijska oznaka za svaki podataka. Tada program kreiranja baze podataka ponovno prolazi kroz istu, te za svaki identifikator podatka izračunava njegov LCA(*Najmanji zajednički predal*), te ga sprema u bazu podataka zajedno s identifikatorom i njegovim k-merima.

### Strukture baze podataka

Gore navedeni proces klasificiranja ulaznog genoma pomoću baze podataka se , već navedeno, bazira na pretraživanju k-mera. Baza podataka se sastaoji od identifikacijskih oznaka svakog podatka u njoj, njihovog izračunatog najmanjeg zajedničkog pretka, te k-mera koje on tvori.




**Slika 3.1:** Prikaze strukture baze podataka

S ciljem poboljšanja brzine pretraživanja, uvodi se algoritam indeksiranja i raspoređivanja poznat pod imenom Minimizer (Roberts M, Hayes W, Hunt B, Mount s, te Yorke J.).

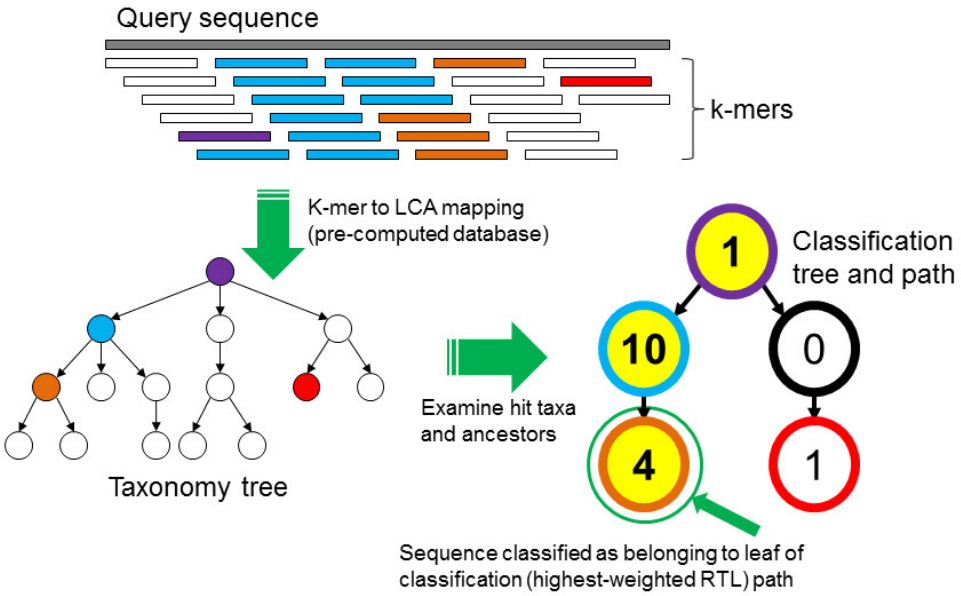
```
ptr <- pokazivač_na_k-mer;
k_ct <- brojač kontejnera;
vrijednosti <- 1ull « (nt * 2);
while brojač je različit od broja kontejnera do
    brojač++;
    dohvati_k-mer(ptr);
    ključ <- izračunaj_ključ();
    b_brojac[ključ]++;
end
b_ofset_ofset[vrijednosti +1];
for i < vrijednosti do
    b_ofset = b_ofset[i-1]+b_brojac[i-1];
end
indeks(b_ofset);
```

**Algoritam 1:** Računanje indeksa kontejnera

Prilikom sortiranja i indeksiranja se stvaraju svojevrsni kontejneri koji sadržavaju one podatke čiji k-meri imaju iste minimizer vrijednosti.Tom idejom se poboljšava pretraživanje na način da se za svaki ulazni podataka računaju njegovi k-meri te njegove minimizer vrijednosti, te se pretragom liste indeksa dohvaća samo onaj kontejner koji sadrži istu minimizer vrijednost.

### 3.0.2. Algoritam klasifikacije podataka

Klasificiranje ulaznog podatka se odvija na više razina. Za početak se za ulazni genom računaju svi njegovi k-meri ( koje korisnik može postaviti, ili biti zadani). K-mere ulaznog skupa ćemo na dalje označavati sa K(S). Zatim se za svaki k-mer kreiraju njegove LCA vrijednosti, tj. traži se onaj podatak u taksonomskom stablu, koji predstavlja prvog pretka tog k-mera. Od svih k-mera se tada kreira stablo koje predstavlja sve moguće klasifikacije za dani genom, a koje su dobivene upravo navedenim algoritmom. Svaka vrijednost u stablu ima izračunatu vrijednost koja predstavlja koliko k-mera se podudara sa tim podatkom. Ako za neki k-mer ne postoji poznati LCA , tada se on neće ni stavljati u navedeno stablo. Način rada koji uvodi tečine za svaki čvor u stablu omogućuje veliku točnost u radu Krakena-a, tako da ako postoji neki put u stablu koji je pogrešan a težine su jako male, on će se zanemariti. Ako postoje dva puta u stablu koja imaju iste težine tada se samo za te podatke ponovno računa LCA te se pokušava odrediti koji put zapravo predstavlja točnu klasifikaciju.



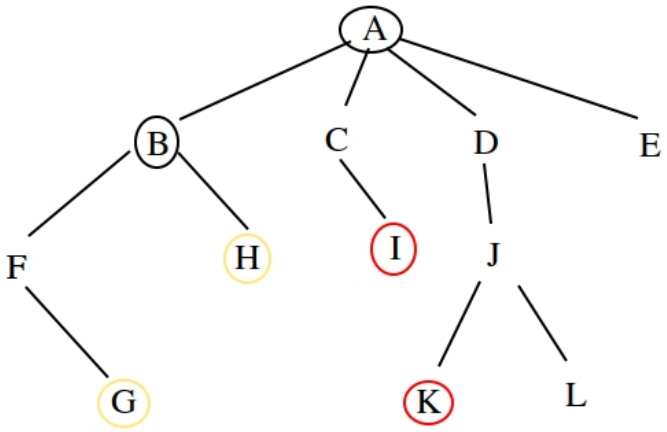

**Slika 3.2:** Prikaza klasifikacije podataka

Iz slike 3.2 se može vidjeti način rada algoritma. Nakon određivanja k-mera u prvom dijelu slike se pokušavaju one klasificirata na LCA pomoću već izvedene baze

podataka. Nakon toga se računaju težine te taj dio je predstavljen u desnom prikazu na slici. Iako postoje dva puta do lista. prihvaća se samo ovaj lijevi jer on ima veće težine, tj. broj k-mera koji se podudaraju s ovim putom je mnogo veći nego na desnom putu.

### 3.0.3. LCA(*Lower Common Ancestor*)

LCA(*Lower common ancestor*) je jedan od osnovnih algoritamskih problema u strukturama stabala.A označava pronalazak čvora u stablu koji je najudaljeniji od korijena stabla, te je zajednički čvorovima *u* i *v*. Prvu ideju i definiciju LCA dali su Alfred Aho, John Hopcroft i Jeffrey Ullman 1973. godine u svom radu *On finding lowest common ancestors in trees*. Zbog svoje složenosti te raznolike primjenjivosti ovaj problem je još uvijek aktualan, te se i danas objavljuje mnogo članaka i radova na tu temu.




**Slika 3.3:** Prikaz stabla i LCA vrijednosti

Na slici 3.3 se nalazi prikaz strukture stabla te su bojama označeni listovi za koje želimo naći najmanjeg zajedničkog pretka (LCA). Crno obojani čvorovi su čvorovi koji su LCA tim dvama listovima. Tako je za listove H i G najmanji zajednički predak čvor B, dok je za listove I i K najmanji zajednički predak A. U radu *On finding lowest common ancestors in trees* 1973. godine je predstavljen sljedeći pseudokod za određi-

vanje najmanjeg zajedničkog pretka.

```
procedure getancestor(u,i)[a]:
if ancestor(u,i-1) = undefined then; then
    ancestor(u,i) <- getancestor(getancestor(u,i-1),i-1);
end
resultis ancestor(u,i);
```

**Algoritam 2:** Dohvaćanje pretka

[a] A.V.Aho, J.E.Hopcroft, J.D.Ullman, **On finding lowest common ancestor in trees** *ACM New York, NY, USA ©1973*

Algoritam 2 predstavlja metodu getancestor(u,i), koja prima čvor *u* te dubinu *i*. Funkcija dohvaća 2î-tog pretka te ga stavlja u niz. *Getancestor* se poziva tijekom traženja najmanjeg zajedničkog pretka

```
procedure find(u,v,i,d)[b]:
if i = 0 then
    resultis ancestor(u,0);
else
    if getancestor(u,i-1)=getancestor(v,i-1) then
        resultis find(u,v,i-1,d);
    else
        resultis find(getancestor(u,i-1),getancestor(v,i-1),min(i-1,|_log(d-2^(i-
          1))_|),d-2^(i-1));

    end
end
```

**Algoritam 3:** Pronalazak LCA

[b] A.V.Aho, J.E.Hopcroft, J.D.Ullman, **On finding lowest common ancestor in trees** *ACM New York, NY, USA ©1973*

Algoritam 3 predstavlja metodu koja pronalazi najmanjeg zajedničkog pretka za ulazne podatke *u* i *v* koji su na dubini *d*. Početne pretpostavke su da je $d > 2\hat{}(i-1)$ te su im 2î-ti preci jednaki ili ne postoje.

# 4. Algoritmi indeksiranja i pretrage

### 4.0.1. Minimizer - algoritam za reduciranje podataka

Usporedba podataka je osnova moderne bioinformatike. Ona se danas temelji na metodi *seed-and-extend*. Za svaki ulazni podatak se uzima samo jedan uzorak, te se on uspoređuje u bazi podataka. Ako se nađe poklapanje tada se taj uzorak proširuje na cijeli podatak te se ponovno uspoređuje. Da bi sve to funkcioniralo potrebno je uzeti uzorak za svaki podskup u jednom genomu, što , za kompleksnije genome, zauzima jako puno memorije.Na ideju rješenja su došli Michael Roberts, Wayne Hayes, Brian R. Hunt, Stephen M. Mount i James A. York u svom radu *Reducing storage requirements for biological sequence comparison*. Koristi se isti način rada, tj.*seed-and-extend*, uz razliku da se ne uzimaju svi uzorci, nego samo mali podskup jedinstvenih, nazvanih "minimizer-i".
Navedeni algoritam sažimanja se koristi i u Kraken bazi podataka, te ju uvelike smanjuje i pospješuje brzinu izvođenja. Prilikom kreiranja MiniKraken baze podataka se koristi isti način rada, uz razliku da se odbacuje samo određeni postotak prvih i zadnjih vrijednosti. Da bi kontejneri u kojima su sadržani podaci s istim minimizer vrijednostima zadržao svojstvo kolekcije, potrebno je da su ti isti podaci normalno distribuirani, što se postiže brisanjem pola bitova prilikom ispitivanja jednakosti. S ciljem pospješenja brzine izvođenja, jer svako računanje minimizer vrijednosti je relativno skupo, prilikom klasifikacije podatka koristi se onaj kontejner koji je korišten prije, te ako k-meri nisu nađeni u njemu, računa se minimizer vrijednost. Ako je minimizer vrijednost jednaka kao u kontjerneru zaključuje se da se podatak ne može klasificirati, inače se pozicionira u dani kontejner te se ponovno vrši pretraga.

Postije dvije vrste minimizer-vrijednosti: unutarnje i vanjske. Svakoj od njih je srž računanja isti, tj. za ulazni niz se računaju svi njegovi k-meri za zadani k, te se iz njih tada bira onaj k-mer koji je leksikografski najmanji.

Slika 4.1 prikazuje način odabira minimizera za ulazni niz. Ulazni niz je podjeljen na 3-mere , a za računanje minimizer-a se koristi prozor od 5 k-mera. Korištenje

| Pozicija: | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| Ulazni niz: | 2 | 3 | 1 | 0 | 3 | 4 | 3 |
| k-meri | 2 | 3 | 1 | | | | |
| | | 3 | 1 | 0 | | | |
| s | | | 1 | 0 | 3 | | |
| podebljanim | | | | **0** | **3** | **4** | |
| minimizerom | | | | | 3 | 4 | 3 |

**Slika 4.1:** K-meri ulaznog niza s minimizer-om

prozora znači da se samo u 5 k-mera odjednom traži minimizer vrijednost. Budući da je *034* leksikografksi najmanji podniz on se uzima kao zadani minimizer. Algoritam minimizer nam garantira da će dva niza koja su ista sadržavati barem jednu istu minimizer vrijednost.

### 4.0.2. Binarno pretraživanje

Za pretraživanje svakog kontejnera se koristi binarno pretraživanje. Binarno pretraživanje je jedan od osnovnih algoritama u računalnoj znanosti. Glavni zadatak algoritma je pronalazak vrijednosti u sortiranom nizu. Osnovna i glavna pretpostavka jest da je niz sortiran, u suprotnom sam algoritam nema smisla.

```
binarno_pretraživanje(niz,cilj):
lo = 1, hi = size(niz);
while lo <= hi do
    mid = lo + (hi - lo) / 2;
    if niz[mid] == cilj then
        return mid;
    else
        if niz[mid] < cilj then
            lo = mid + 1;
        else
            hi = mid + 1;
        end
    end
end
```

**Algoritam 4:** Binarno pretraživanje

Inicijalno je radni prostor binarnog pretraživanja cijeli ulazni niz. Tada se pozicionira na sredinu tog niza, te se ispituje je li vrijednost na sredini jednaka onoj koju mi tražimo. Ako je rezultat potvrdan izlazimo iz petlje te vraćamo poziciju te vrijednosti. U protivnom se ispituje odnos tražene vrijednosti i one koja je na sredini. Ako je vrijednost na sredini manja od tražene radni prostor postavljamo na onu polovicu koja se nalazi desno od indeksa, tj. prelazimo u polovicu koja sadrži veće vrijednosti, u suprotnom se pozicioniramo na manju polovicu. Složenost binarnog pretraživanja je O(logN). Logaritam je sporo rastuća funkcija, stoga je binarno pretraživanje jako efikasan algoritam. Kao primjer za pretragu imenika od milion imena, pomoću binarnog pretraživanja bismo pronašli traženu vrijednost u najviše 21 korak.

# 5. Pseudokod i razrada algoritma

# 6. Analiza učinkovitosti rješenja

# 7. Zaključak

Zaključak.

# LITERATURA

**Algoritam staničenja velike baze podataka genoma**

## Sažetak

Sažetak na hrvatskom jeziku.

**Ključne riječi:** Ključne riječi, odvojene zarezima.

**Title**

## Abstract

Abstract.

**Keywords:** Keywords.