# Implementacija alata za interaktivno traženje i pretraživanje skupova redeskripcija

**Kozjak, Iva**

**SVEUČILIŠTE U ZAGREBU**

**PRIRODOSLOVNO–MATEMATIČKI FAKULTET**

**MATEMATIČKI ODSJEK**

Iva Kozjak

# IMPLEMENTACIJA ALATA ZA INTERAKTIVNO TRAŽENJE I PRETRAŽIVANJE SKUPOVA REDESKRIPCIJA

Diplomski rad

Voditelj rada:
doc. dr. sc. Matej Mihelčić

Zagreb, siječanj, 2023.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

*Zahvaljujem mentoru doc. dr. sc. Mateju Mihelčiću na susretljivosti, dostupnosti, stručnim savjetima i strpljenju tijekom pisanja ovog rada.*
*Posebna zahvala roditeljima, sestri i dečku na podršci i razumijevanju koje mi bezuvjetno pružaju, na vjeri u mene i moje mogućnosti te na motivaciji i pomoći na svakom koraku mog obrazovanja.*
*Veliko hvala prijateljima koji su mi pokazali ljepotu studentskih dana, kolegama koji su mi pomagali kad je bilo potrebno i svima ostalima koji su svjesno ili nesvjesno pripomogli ostvarenju ovog cilja svojom podrškom, savjetima i preporukama.*

# Sadržaj

**Sadržaj** **iv**

**Uvod** **1**

**1 Simboli i pojmovi** **2**
1.1 Uvod . . . 2
1.2 Definicije i primjeri . . . 2
Podaci . . . 3
Pravila (deskripcije) . . . 3
Redeskripcije . . . 4
Skup ograničenja . . . 6
Mjere kvalitete redeskripcije . . . 7

**2 CLUS-RM algoritam** **8**
2.1 Uvod . . . 8
2.2 Pseudokod i opis algoritma . . . 9

**3 Alat InterSet** **11**
3.1 Uvod . . . 11
3.2 Struktura baze podataka . . . 11
3.3 Skupovi podataka . . . 13
3.4 Struktura alata InterSet . . . 14
Pretraživanje skupova redeskripcija na temelju entiteta . . . 14
Pretraživanje skupova redeskripcija na temelju atributa . . . 15
Pretraživanje skupova redeskripcija na temelju mjera kvalitete redeskripcije 15

**4 Razvojno okruženje i korišteni programski alati** **16**
4.1 Uvod . . . 16
4.2 Java Spring . . . 16
Spring Boot . . . 16

4.3 Angular.js . . . . . 17
4.4 IntelliJ IDEA . . . . . 17

**5 Praktični rad** **18**
5.1 CLUS-RM algoritam s uvjetima na entitete . . . . . 18
Pseudokod funkcije . . . . . 20
5.2 Implementacija poslužiteljske aplikacije alata InterSet . . . . . 21
Struktura koda . . . . . 21
Model . . . . . 22
Controller . . . . . 23
Service . . . . . 24
Repository . . . . . 24
5.3 Povezivanje alata InterSet s algoritmom CLUS-RM . . . . . 26
Pozivanje algoritma u Angular-u (promjene u HTML-u) . . . . . 26
Pozivanje algoritma u Angular-u (promjene u JS-u) . . . . . 27
Pozivanje algoritma u Spring Boot-u . . . . . 28
Prikaz dohvaćenih podataka (promjene u JS-u) . . . . . 32
Prikaz dohvaćenih podataka (promjene u HTML-u) . . . . . 33
Spremanje odabranih redeskripcija (promjene u JS-u) . . . . . 33
Spremanje odabranih redeskripcija u Spring Boot-u . . . . . 35

**6 Opis i prikaz novih funkcionalnosti** **38**
6.1 Pokretanje CLUS-RM algoritma . . . . . 38
Uvjeti na entitete . . . . . 38
Uvjeti na atribute . . . . . 40
Dodatne postavke . . . . . 42
6.2 Prikaz i spremanje novih redeskripcija u bazu . . . . . 43
Kartica entiteti . . . . . 43
Kartica atributi . . . . . 46

**Bibliografija** **49**

# Uvod

Traženje redeskripcija je potpodručje dubinske analize podataka te spada u metode otkrivanja znanja iz podataka. Mnogi algoritmi dubinske analize podataka, specifično i traženja redeskripcija, kreiraju veliki broj pravila koja opisuju razne podskupove entiteta. Takvi skupovi se teško koriste u praksi zato što se mora pregledavati jako veliki broj, dosta često redundantnih, pravila. Skupovi su nestrukturirani te je teško uočiti neke kompleksnije poveznice, grupe i pravilnosti koje su važne za razumijevanje problema.

CLUS-RM [18] je algoritam za traženje redeskripcija koji koristi višeciljna prediktivna stabla klasteriranja [14]. Alat InterSet [21] omogućava interaktivno pretraživanje velikih skupova redeskripcija prema njihovim najvažnijim svojstvima kao što su entiteti koje opisuju, atributi koji ih tvore i mjere točnosti.

Glavni nedostatak alata InterSet je što se oslanja na prije izračunatu listu redeskripcija te je cilj ovog rada povezati ga s alatom CLUS-RM korištenjem Java Spring (Boot) okruženja. Na taj će se način konstruirati efektivno interaktivno okruženje za traženje, pregledavanje i duboku analizu skupova redeskripcija.

U prvom poglavlju definiraju se osnovni pojmovi vezani uz redeskripcije te se pobliže objašnjavaju kroz primjere. U drugom i trećem poglavlju ukratko se opisuju: algoritam za traženje redeskripcija (CLUS-RM) i alat za pretraživanje redeskripcija (InterSet). U četvrtom poglavlju prezentira se razvojno okruženje te korišteni programski alati. U petom poglavlju opisuje se implementacija interaktivnog alata InterSet (uz pozivanje algoritma za traženje redeskripcija) te se u posljednjem, uz primjere, prikazuje rad implementiranog alata.

# Poglavlje 1

# Simboli i pojmovi

## 1.1 Uvod

U ovom poglavlju prikazane su osnovne definicije vezane uz problem traženja redeskripcija, praćene primjerima radi lakšeg razumijevanja bitnih pojmova.

**Definicija 1.1.1** (Redeskripcija - neformalna definicija)**.** *Redeskripcija je karakterizacija podskupova objekata na dva (ili više) različitih načina, gdje karakterizacije opisuju podskupove entiteta s velikim presjekom (međusobno slične ili identične skupove objekata).*

**Definicija 1.1.2** (Zadatak traženja redeskripcija - neformalna definicija)**.** *Cilj zadataka traženja redeskripcija je pronaći sve redeskripcije koje zadovoljavaju specificirane uvjete.*

## 1.2 Definicije i primjeri

Definicije koje se navode, preuzete su iz knjige *Redescription Mining*, poglavlja *What Is Redescription Mining* [13].

Prije definicija, potrebno je uvesti simbole:

| | |
|---|---|
| $\mathcal{D}$ | skup podataka |
| $\mathcal{E}$ | skup svih entiteta u podacima |
| $\mathcal{A}$ | skup atributa u podacima |
| $\mathcal{V}$ | skup pogleda |
| $\mathbf{D}$ | tablica entiteta po atributima koja odgovara jednom pogledu |
| $\mathcal{P}$ | skup predikata nad $\mathcal{E} \times \mathcal{A}$ |
| $\mathcal{L}$ | skup literala (predikata i negacija predikata) |
| $\mathcal{Q}$ | skup pravila |

## Podaci

**Definicija 1.2.1** (Model podataka)**.** *Podaci $\mathcal{D}$ za problem traženja redeskripcije su uređena trojka $\mathcal{D} = (\mathcal{E}, \mathcal{A}, \mathcal{V})$, gdje su entiteti $e \in \mathcal{E}$ opisani atributima iz $\mathcal{A}$ i atributi su podijeljeni u poglede $V \in \mathcal{V}$.*

**Primjer 1.2.2.** *Uzmemo li za primjer nogomet, entiteti su utakmice, dok su atributi različite statistike utakmice, kao što su broj golova, udaraca, kornera, slobodnih udaraca, prekršaja, žutih/crvenih kartona itd.*
*Statistiku vezanu uz udarce i golove možemo staviti u jedan pogled (pogled napada na gol), statistiku vezanu uz kartone i prekršaje u drugi (pogled poštenog igranja) te statistiku vezanu uz eventualne sudačke pogreške u treći pogled (pogled suđenja).*

**Definicija 1.2.3** (Model podataka temeljen na tablici)**.** *U tabličnom prikazu podataka, podaci se sastoje od jedne ili više tablica $\mathbf{D}_1$, $\mathbf{D}_2$, .... Redci u tablici predstavljaju entitete, a stupci predstavljaju atribute. Svaki redak u svakoj tablici mora predstavljati iste entitete. Stupci tablica su atributi, a vrijednost $\mathbf{D}_k(i, j)$ je vrijednost koju entitet $i$ ima za atribut $j$ u tablici $k$.*

## Pravila (deskripcije)

Za definiranje pravila, potrebno je povezati entitete i atribute pomoću *predikata*. Skup $\mathcal{L}$ sadrži sve predikate i negacije predikata. Predikat za atribut $a \in \mathcal{A}$ je funkcija $p_a : \mathcal{E} \to \{istina, laž\}$. Predikati ovise o tipu atributa, a zapisuju se kao $p_a = [P_a(e)]$ gdje je $P_a(e)$ neka logička rečenica. Atributi mogu biti kategorijski, numerički ili binarni.

Za binarne atribute, predikat vraća vrijednost entiteta za taj atribut. U slučaju kategoričkih atributa, predikati mogu izgledati kao $P_a(e) = [a(e) = X]$ gdje je $X$ neka konstanta. Skraćeni način pisanja predikata je $[a = X]$. Za numeričke atribute se u većini slučajeva promatraju predikati kao $[a \leq X]$ ili $[a \geq X]$.

Pravilo $q_i$ je logička formula koja sadrži podskup atributa i-tog pogleda i logičke operatore konjunkcije, disjunkcije ili negacije.

*Deskripcija* predstavlja pravilo $q : \mathcal{E} \to \{istina, laž\}$ koje pridjeljuje vrijednost istina ili laž svakom entitetu $e \in \mathcal{E}$.

**Primjer 1.2.4.** *Primjeri deskripcija za nogometne utakmice:*

$$[1 \leq \textit{žuti karton} \leq 3], \quad [\textit{ crveni karton} = \textit{istina}], \quad [\textit{sudac} = \textit{N.Pitana}]$$

Pravilima se pridružuju dva važna skupa: podrška i atributi.

**Definicija 1.2.5** (Podrška, Atributi)**.** *Neka je $q \in Q$ pravilo. Podrška, $supp(q)$, je skup entiteta koje pravilo $q$ opisuje, odnosno, $supp(q) = \{e \in \mathcal{E} : q(e) = istina\}$. Skup atributa koji se pojavljuju u pravilu $q$ označavaju se oznakom $attr(q)$.*

## Redeskripcije

*Redeskripcija* je uređena $n$-torka pravila s disjunktnim atributima i dovoljno sličnim podrškama. Potrebno je definirati kako mjeriti razliku između skupova entiteta. Može se koristiti bilo koja funkcija udaljenosti $d : 2^{\mathcal{E}} \times 2^{\mathcal{E}} \to [0, \infty\rangle$ koja zadovoljava sljedeće uvjete:

$$\begin{aligned} &d(X, Y) = 0 && \text{akko } X = Y \\ &d(X, Y) = d(Y, X) && \text{za sve } X, Y \in 2^{\mathcal{E}}. \end{aligned}$$

Jedan mogući izbor je *Jaccardova udaljenost* koja se temelji na *Jaccardovom indeksu sličnosti*.

**Definicija 1.2.6** (Jaccardov indeks sličnosti)**.** *Jaccardov indeks sličnosti J za skupove podrški dva pravila p i q definira se kao*

$$J(p, q) = J(supp(p), supp(q)) = \left| \frac{supp(p) \bigcap supp(q)}{supp(p) \bigcup supp(q)} \right|.$$

*Jaccardova udaljenost definira se kao*

$$1 - J(p, q) = 1 - \left| \frac{supp(p) \bigcap supp(q)}{supp(p) \bigcup supp(q)} \right|.$$

Funkcijom udaljenosti mjerimo koliko su udaljene podrške dvaju pravila, dok funkcijom sličnosti, kao što je Jaccardov indeks, mjerimo koliko su slične. Ako su podrške dva pravila koja grade redeskripciju jednake, označimo li sa $d(p, q)$ udaljenost pravila $p$ i $q$, vrijedi $d(p, q) = 0$ pa je ta redeskripcija egzaktna i zapisuje se kao $p \equiv q$.

Većina redeskripcija nije egzaktna, međutim često su zadovoljavajuće redeskripcije koje su dovoljno slične. Kažemo da su redeskripcije dovoljno slične ako vrijedi $d(p, q) \leq \tau$, za neki $\tau \in [0, \infty\rangle$. Označavamo:

$$p \sim_\tau q \text{ ako i samo ako } d(p, q) \leq \tau.$$

Često se indeks $\tau$ izostavlja te se piše samo $p \sim q$.

**Definicija 1.2.7** (Redeskripcija)**.** *Redeskripcija je par pravila (p,q) takav da vrijedi*

$$p \sim q \quad i \quad attr(p) \bigcap attr(q) = \emptyset.$$

**Primjer 1.2.8.** *Sada se redeskripcija iz primjera 1.2.4 može zapisati:*

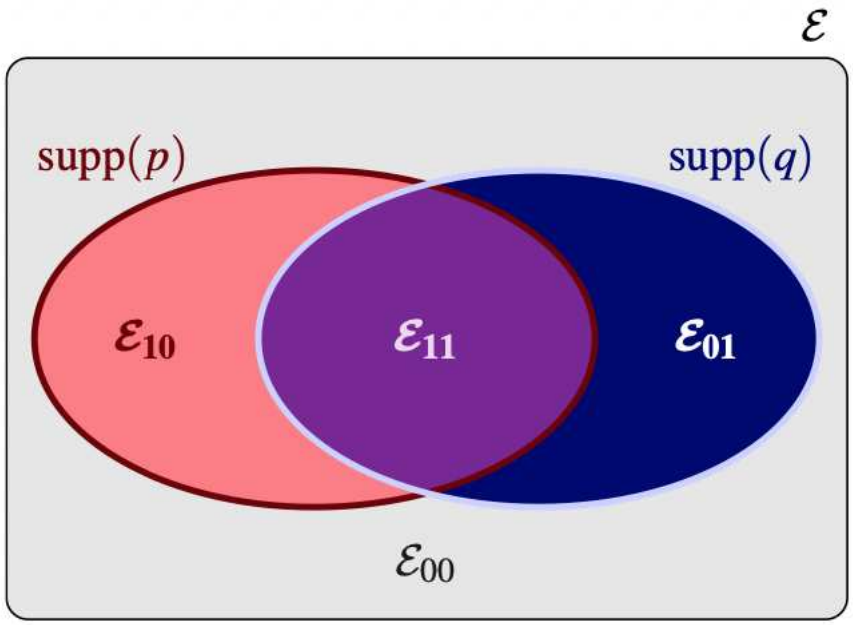
$$[1 \leq \textit{žuti karton} \leq 3] \sim [\textit{sudac} = \textit{N.Pitana}]$$

**Definicija 1.2.9** (Podrška redeskripcije)**.** *Neka je (p,q) redeskripcija dobivena na nekom skupu podataka $\mathcal{D}$. Podrška redeskripcije (p, q) je podrška pravila $p \wedge q$ u $\mathcal{D}$, odnosno,*

$$supp(p, q) = supp(p \wedge q) = supp(p) \bigcap supp(q).$$

Podrška pravila $p$ i $q$ i podrška redeskripcije $(p, q)$ definira četiri važna skupa entiteta:

$$\begin{aligned}
\mathcal{E}_{11} &= supp(p) \bigcap supp(q) \\
\mathcal{E}_{10} &= supp(p) \backslash supp(q) \\
\mathcal{E}_{01} &= supp(q) \backslash supp(p) \\
\mathcal{E}_{00} &= \mathcal{E} \backslash (supp(p) \bigcup supp(q)).
\end{aligned}$$




Slika 1.1: Vennov dijagram

Na slici 1.1 prikazan je Vennov dijagram iz kojeg je vidljiv odnos između navedenih skupova $\mathcal{E}_{11}, \mathcal{E}_{10}, \mathcal{E}_{01}$ i $\mathcal{E}_{00}$.

**Definicija 1.2.10** (Pretraživanje redeskripcija)**.** *Za skup podataka $\mathcal{D}$, skup pravila $\mathcal{Q}$, sličnost $\sim$ i skup potencijalnih ograničenja $\mathcal{C}$, cilj pretraživanja redeskripcija je pronaći sve redeskripcije $(p_i, q_i)$ koje također zadovoljavaju skup potencijalnih ograničenja.*

## Skup ograničenja

Svrha ograničenja je izbjeći neželjene redeskripcije. Koja su to ograničenja?

Uz minimalni Jaccardov indeks koriste se i sljedeća ograničenja:

*Složenost redeskripcija* je ograničenje vezano za primjerice broj atributa ili literala koje sadrži određena redeskripcija.

Jezik pravila je skup prihvatljivih pravila, ovisno o podržanim vrstama atributa, načelima za izgradnju predikata i sintaktičkim pravilima za njihovo kombiniranje u izjave. *Odabir ispravnog jezika* pravila jedan je i nedvojbeno najmoćniji - način kontrole složenosti redeskripcija. Uz to, utječe na strukturu i ekspresivnost redeskripcije. Uobičajeni jezici uključuju pravila u kojima se svaka varijabla može pojaviti najviše jednom.

Mjerenje *statističke značajnosti* pronađenih redeskripcija još je jedan način uklanjanja nezanimljivih rezultata. Postoji više različitih načina definiranja nulte hipoteze s ciljem identificiranja nezanimljive redeskripcije, ali većina postojeće literature koristi jednostavnu nultu hipotezu. Nulta hipoteza je da su podrška za $p$ i $q$ slučajni neovisni skupovi s očekivanim veličinama $|supp(p)|$ i $|supp(q)|$, respektivno, a pridružena *p-vrijednost* je vjerojatnost da vrijedi $|X \bigcap Y| \geq |supp(p \wedge q)|$. Ova vjerojatnost je rep binomne distribucije.

Neka su $X \subseteq \mathcal{E}$ i $Y \subseteq \mathcal{E}$ dva slučajna nezavisna skupa takva da je $p(e \in X) = \frac{|supp(e)|}{|\mathcal{E}|}$ i $p(e \in Y) = \frac{|supp(q)|}{|\mathcal{E}|}$ za pravila $p$ i $q$. Neka je $\alpha$ vjerojatnost da je neki nasumični entitet $e \in \mathcal{E}$ u $X \cap Y$. Označivši $|\mathcal{E}|$ s $n$ i koristeći neovisnost $X$ i $Y$, dobivamo

$$\alpha = p(e \in X, e \in Y) = P(e \in X)p(e \in Y) = \frac{|X|}{n}\frac{|Y|}{n} = \frac{|supp(p)||supp(q)|}{n^2}.$$

Vjerojatnost da vrijedi $|X \cap Y| \geq |supp(p, q)|$ je:

$$pV(p, q) = \sum_{k=|supp(p \wedge q)|}^{n} \binom{n}{k} \alpha^k (1 - \alpha)^{n-k}$$

što predstavlja željenu *p-vrijednost*.

*p-vrijednost* nam govori možemo li odbaciti nultu hipotezu koja pretpostavlja da smo dobili podskup elemenata opisan datom redeskripcijom spajanjem dva slučajna pravila s a priori vjerojatnostima jednakim udjelu obuhvaćenih elemenata. Ako je dobivena p-vrijednost niža od nekog unaprijed definiranog praga, tzv razine značajnosti, onda ovu nultu hipotezu treba odbaciti. Ova procjena je optimistična kada pretpostavka da se svi

elementi mogu uzorkovati s jednakom vjerojatnošću ne vrijedi (što je često slučaj u praksi).

Pogledamo li ponovno primjer nogometnih utakmica,

$p$=”utakmice na kojima je domaćin zabio najmanje nula golova”,

$q$=”utakmice na kojima su gosti zabili najmanje nula golova”,

odnosno, one utakmice u kojima je domaći tim postigao najmanje nula golova su upravo one utakmice u kojima je gostujući tim postigao najmanje nula golova, nije zanimljiva redeskripcija budući da niti jedna momčad ne može postići manje od nula golova. Navedena redeskripcija obuhvaća sve nogometne utakmice ikada odigrane.

## Mjere kvalitete redeskripcije

**Definicija 1.2.11** (Prosječni Jaccardov indeks entiteta opisanih redeskripcijom R)**.** *Mjera koja daje informacije o redundantnosti entiteta sadržanih u podršci redeskripcije s obzirom na ostale redeskripcije sadržane u skupu redeskripcija $\mathcal{R}$ naziva se prosječni Jaccardov indeks entiteta redeskripcije i definira se kao:*

$$AEJ(R_i) = \frac{1}{|\mathcal{R}| - 1} \cdot \sum_{j=1}^{|\mathcal{R}|} J(supp(R_i), supp(Rj)),\ i \neq j.$$

**Definicija 1.2.12** (Prosječni Jaccardov indeks atributa redeskripcije)**.** *Mjera koja pruža informacije o redundantnosti atributa sadržanih u pravilima redeskripcije, s obzirom na ostale redeskripcije sadržane u skupu redeskripcija $\mathcal{R}$, naziva se prosječni Jaccardov indeks atributa redeskripcije i definira se kao:*

$$AAJ(R_i) = \frac{1}{|\mathcal{R}| - 1} \cdot \sum_{j=1}^{|\mathcal{R}|} J(attr(R_i), attr(Rj)),\ i \neq j.$$

# Poglavlje 2

# CLUS-RM algoritam

## 2.1 Uvod

U ovom se poglavlju opisuje algoritam prije promjena napravljenih u ovom radu, odnosno prije opcije traženja redeskripcija obzirom na odabrane entitete i uvjete nad njima.

Doc. dr. sc. Matej Mihelčić osmislio je CLUS-RM algoritam za traženje redeskripcija [17]. Algoritam 1 koristi višeciljna prediktivna stabla klasteriranja.

Pseudokod i opis algoritma preuzeti su iz knjige *Redescription Mining*, poglavlja *Algorithms for Redescription Mining* [13].

Prediktivno stablo klasteriranja (PCT) [12] je vrsta stabla odlučivanja koja vrši klasteriranje entiteta na temelju deskriptivnih i/ili ciljnih atributa. Činjenica da stvara klastere, razlikuje ga od običnih stabala odlučivanja. Takvo stablo odlučivanja pruža niz testova koji grupiraju entitete u klastere, koji postaju sve homogeniji s obzirom na ciljne atribute kako se napreduje niz stablo. U tom smislu, ovu strukturu možemo vidjeti kao generiranje hijerarhijskog grupiranja entiteta.

## 2.2 Pseudokod i opis algoritma

**Algorithm 1:** Pseudokod algoritma

**Input:** Skupovi podataka $D_1$ i $D_2$, min Jaccardov indeks, max p-vrijednost, min/max podrška, broj iteracija $\kappa$
**Output:** Redeskripcija $\mathcal{R}$

1 $\mathcal{R} \leftarrow \emptyset$
2 **foreach** $s \in \{1, 2\}$ **do**
3 $\quad T_s^{(0)} \leftarrow$ induciranje stabala po $\mathbf{D}_s$ radi odvajanja izvornih entiteta od nasumičnih kopija
4 $\quad Q_s^{(0)} \leftarrow$ dobivanje pravila iz $\mathrm{T}_s^{(0)}$
5 dodati parove upita iz $Q_1^{(0)} \times Q_2^{(0)}$ u $\mathcal{R}$
6 **foreach** $k \in \{1, 2..., \kappa\}$ **do**
7 $\quad$ **foreach** $(s, t) \in \{(1, 2), (2, 1)\}$ **do**
8 $\quad\quad \tau \leftarrow$ generiranje višedimenzionalnih ciljeva iz upita u $Q_s^{(k-1)}$
9 $\quad\quad T_t^{(k)} \leftarrow$ induciranje stabala po $\mathbf{D}_t$ s ciljem $\tau$
10 $\quad\quad Q_t^{(k)} \leftarrow$ dobivanje pravila iz $T_t^{(k)}$
11 $\quad\quad$ dodati parove upita iz $Q_s^{(k-1)} \times Q_t^{(k)}$ u $\mathcal{R}$
12 $\quad$ dodati parove upita iz $Q_1^{(k)} \times Q_2^{(k)}$ u $\mathcal{R}$
13 kombinacija i redukcija kandidata redeskripcija iz $\mathcal{R}$
14 **return** $\mathcal{R}$

U inicijalizacijskom koraku algoritma se koristi postupak koji kreira dodane entitete koji su nasumične varijante izvornih. To jest, izvorni skup podataka je dupliciran, a vrijednosti svakog atributa su pomiješane među entitetima u kopiji. Početni ciljni vektor je binarni vektor s oznakom 1 za izvorne entitete i oznakom 0 za nasumične kopije. Zatim se inducira PCT nad proširenim skupom podataka koji sadrži i originalne i nasumične entitete, pokušavajući ih razlikovati (redak 3).

Upiti se zatim izdvajaju iz dobivenog stabla (redak 4). Točnije, za svaki list i svaki međučvor koji odgovara nepraznom klasteru originalnih entiteta (zanemarujući nasumične kopije), uvjeti na koje naiđemo prilikom prolaska od korijena stabla do čvora koriste se kako bi se formirao konjunktivni upit (pravilo). Ovaj se postupak primjenjuje na obje podatkovne tablice zasebno, što rezultira s dva skupa pravila $Q_1^{(0)}$ i $Q_2^{(0)}$.

Za danu zbirku upita $Q$ , višedimenzionalni cilj $\tau$ može se generirati kao binarna matrica s jednim retkom po izvornom entitetu i jednim stupcem po upitu iz $Q$, gdje svaki element matrice $(i, j)$ označava pripada li entitet $i$ podršci pravila $j$. Cilj generiran iz pravila dobivenih iz jedne tablice podataka koristi se u sljedećoj iteraciji za induciranje PCT-a

na izvornoj tablici podataka s drugog pogleda. Dobiva se nova zbirka pravila te se postupak ponavlja. Dvije takve procedure izvode se paralelno, naizmjenično između strana za određeni broj ponavljanja (redci 6-12).

U svakoj iteraciji $k$ i za obje strane, ne-redundanti upiti se prikupljaju iz induciranog stabla $T_s^{(k)}$ u $Q_s^{(k)}$ (redak 10). Ti se upiti zatim uparuju s upitima iz $Q_t^{(k-1)}$ prikupljenim u prethodnoj iteraciji generiranim koristeći atribute drugog pogleda $t$ (redak 11). Upiti prikupljeni s obje strane u istoj iteraciji također se mogu kombinirati zajedno (redak 12). Ti se kandidati redeskripcija zatim proširuju kako bi se formirali složeniji upiti koji potencijalno uključuju disjunkcije, duga pravila se svode na kraća i skup rezultata se dobiva optimizacijom skupa redeskripcija (redak 13).

# Poglavlje 3

# Alat InterSet

## 3.1 Uvod

U ovom poglavlju ukratko se opisuju neke bitnije funkcionalnosti alata InterSet prije izrade ovog diplomskog rada, odnosno prije modificiranja i spajanja s CLUS-RM algoritmom opisanim u prethodnom poglavlju.

Alat InterSet je web aplikacija koja omogućava interaktivno, sveobuhvatno istraživanje skupa redeskripcija. Alat je detaljnije opisan u radu *InterSet: Interactive redescription set exploration* [20] te *Targeted and contextual redescription set exploration* [21].

Klijent aplikacije izrađen je korištenjem standardnih tehnologija: HTML [1], CSS [2], JavaScript [3]. Glavni dio je izgrađen pomoću angular.js [4]. Poslužiteljska strana aplikacije kreirana je pomoću okruženja node.js [5].

Na web stranici mogu se testirati značajke alata istražujući redeskripcije stvorene na tri različita skupa podataka.

## 3.2 Struktura baze podataka

1. **DataTable** sadrži vrijednost elementa za sve atribute koji se koriste u procesu pretraživanja redeskripcija

2. **ElementTable** sadrži dodatne informacije o elementima kao što su opisi elemenata ili duže oznake ako su dostupne

3. **RedescriptionAttributeTable** sadrži informacije o atributima redeskripcija za svaku redeskripciju u skupu redeskripcija

4. **RedescriptionElementTable** sadrži skupove podrške za svaku redeskripciju iz skupa redeskripcija
5. **RedescriptionTable** sadrži vrijednosti raznih evaluacijskih mjera redeskripcija za svaku redeskripciju
6. **AttributeTable** omogućuje definiranje dodatnih oznaka ili opisa atributa
7. **SOMClusters** sadrži elemente sadržane u svakom SOM klasteru
8. **ElementCoverage** sadrži informacije, za svaki entitet, o broju redeskripcija koje ga sadrže u skupovima podrška redeskripcija
9. **SOMDimensions** sadrži dimenzije koje se koriste za prikaz SOM-a
10. **MeasuresNames** omogućuje unos različitih informacija o mjerama redeskripcija koje se koriste u procesu pretraživanja skupa redeskripcija
11. **AttributeCoocurenceTable** sadrži učestalosti supojavljivanja atributa u upitima za redeskripciju
12. **AttributeFrequencyTable** sadrži učestalost pojavljivanja atributa u pravilima koja tvore redeskripciju
13. **CategoryTable** sadrži informacije o kategorijama za atribut koji sadrži kategoričke vrijednosti
14. **GraphTable** sadrži Jaccardov indeks između skupova podrška dvije redeskripcije za sve parove redeskripcija sadržane u skupu redeskripcija
15. **GraphTableAttr** sadrži Jaccardov indeks između skupova atributa dvije redeskripcije za sve parove redeskripcija sadržane u skupu redeskripcija
16. **UserTable** sadrži korisnička imena i lozinke
17. Sve tablice sa sufiksom **Back** koriste se za spremanje točnih informacija o unaprijed obučenoj SOM mapi (izgled, broj klastera, članstvo entiteta itd.)
18. Postoje i tablice sa prefiksom **Selected** koje se koriste za spremanje selekcije entiteta, atributa i redeskripcija

## 3.3 Skupovi podataka

1. **Country**

   skup podataka sadrži atribute koji opisuju zemlje svijeta korištenjem socio-demografskih podataka o zemlji i obrazaca trgovanja zemlje za 2012. godinu

   - sadrži 199 zemalja svijeta
   - podaci sadrže dva prikaza:
     - Socio-demografski podaci o zemlji: sadrži 49 numeričkih atributa
     - Podaci o trgovini zemlje: sadrže 312 numeričkih atributa

2. **DBLP**

   skup podataka sadrži atribute koji opisuju graf koautorstva i bipartitni graf autor-konferencija

   - sadrži 6455 autora
   - podaci sadrže dva prikaza:
     - Bipartitni graf autor-konferencija: sadrži 304 binarna atributa
     - Graf koautorstva: sadrži 6455 binarnih atributa

3. **Phenotype**

   skup podataka sadrži informacije o prisutnosti svojstava fenotipa i klastera ortoloških gena u različitim vrstama bakterija

   - sadrži 1336 vrsta bakterija
   - podaci sadrže dva prikaza:
     - Fenotipovi: sadrži 333 binarna atributa
     - COG (Clusters of Orthologoues groups): sadrži 4602 binarna atributa

Pokretanjem alata InterSet, korisnik odabire nad kojim skupom podataka želi pretraživati skupove redeskripcija. Nakon odabira, potrebna je prijava korisnika, odnosno registracija ukoliko se korisničko ime i lozinka ne nalaze u bazi. Uspješnom prijavom, otvara se prikaz u kojem korisnik može odabrati jedan od tri pregleda opisana u sljedećoj sekciji.

## 3.4 Struktura alata InterSet

Alat se sastoji od tri različita pregleda skupa redeskripcija koji se mogu odabrati na glavnoj stranici alata:

1. Pretraživanje skupova redeskripcija na temelju **entiteta**
2. Pretraživanje skupova redeskripcija na temelju **atributa**
3. Pretraživanje skupova redeskripcija na temelju **mjera kvalitete redeskripcije** (Jaccardov indeks, p-vrijednost, podrška redeskripcije, prosječan Jaccardov indeks entiteta i atributa, odnosno redundantnost po pitanju entiteta koje redeskripcija opisuje i po pitanju atributa koje sadrži u svojim pravilima)

Redeskripcije se mogu (bez obzira na način/pogled pretraživanja) analizirati i individualno. Korisnik u svakom pogledu iz tablice redeskripcija može odabrati željenu redeskripciju i proučiti njene entitete, atribute i distribucije vrijednosti atributa za entitete iz cijelog skupa podataka, intervala sadržanog u pravilu redeskripcije i skup opisanih instanci redeskripcije.

### Pretraživanje skupova redeskripcija na temelju entiteta

U prvom se prikazu koristi samoorganizirajuća mapa (SOM map [15]) za svrstavanje elemenata u različite klastere na temelju pojavljivanja elementa u skupovima za podršku redeskripcija.
Otvaranjem ovog prikaza, na stranici su samo prikazane osnovne informacije i upute za korištenje, a odabirom određenih komponenti prikazuju se novi elementi na stranici.
Proces pretraživanja može se provesti u nekoliko koraka:

1. Odabirom bilo kojeg klastera SOM mape prikazuju se i mogu se pretražiti redeskripcije koje opisuju barem jedan element iz odabranog klastera.
2. U tablici redeskripcija mogu se odabrati redeskripcije koje su zanimljive korisniku.
3. Moguće je napraviti SOM mapu na raznim podskupovima redeskripcija.
4. Moguće je također definirati omjer entiteta iz klastera koji mora biti opisan te napraviti selekciju redeskripcija pridruženih klasteru prema tom kriteriju.
5. Moguće je spremiti selekciju entiteta ili kreirati SOM mapu baziranu na selekciji redeskripcija.

## Pretraživanje skupova redeskripcija na temelju atributa

U drugom se prikazu koristi toplinska mapa (heatmap) atributa koja prikazuje informaciju o ko-pojavljivanju parova atributa iz različitih pogleda u pravilima redeskripcija.
Otvaranjem ovog prikaza, na stranici su samo prikazane osnovne informacije i upute za korištenje, a odabirom određenih komponenti prikazuju se novi elementi na stranici.
Proces pretraživanja može se provesti u nekoliko koraka:

1. Moguće je odabrati redoslijed koji će se koristiti za raspoređivanje atributa u toplinskoj mapi.
2. Odabirom para atributa prikazuju se i mogu se pretražiti redeskripcije koje u pravilima sadrže odabrane atribute.
3. U tablici redeskripcija mogu se odabrati redeskripcije koje su zanimljive korisniku.
4. Izabrane redeskripcije se mogu spremiti, heatmap-a se može ponovo kreirati na selekciji.

## Pretraživanje skupova redeskripcija na temelju mjera kvalitete redeskripcije

Pretraživanje temeljeno na svojstvima redeskripcija omogućuje korisniku filtriranje skupa redeskripcija na temelju različitih kriterija koristeći tzv. kros-filter [6].
Proces istraživanja može se provesti u nekoliko koraka:

1. Moguće je odabrati kriterij prema kojem korisnik želi filtrirati te odabrati raspon vrijednosti odgovarajuće mjere redeskripcije.
2. U tablici redeskripcija mogu se odabrati redeskripcije koje su zanimljive korisniku.
3. Selekcija se može spremiti i koristiti u drugim pogledima.

# Poglavlje 4

# Razvojno okruženje i korišteni programski alati

## 4.1 Uvod

Glavni cilj ovog rada bilo je povezivanje alata InterSet s algoritmom CLUS-RM korištenjem Java Spring Boot [7] [16] okruženja. Uz samo povezivanje dodane su i nove funkcionalnosti kako bi alat postao interaktivan, odnosno kako bi se korisniku, osim pretraživanja, omogućilo i traženje novih redeskripcija. U ovom poglavlju opisuje se Spring Boot [7] okruženje u kojem je kreirana poslužiteljska strana aplikacije, Angular.js [4] u kojem su dodane nove funkcionalnosti aplikacije te programski alat IntelliJ IDEA [8] u kojem je programski dio rada pisan.

## 4.2 Java Spring

Java Spring Framework (Spring Framework) [9] razvojno je okruženje otvorenog koda za stvaranje samostalnih aplikacija koje se izvode na Java virtualnom stroju (JVM [10]). Kreirao ga je Rod Johnson 2013. godine kao poboljšanje dotad korištenog Java Enterprise Edition (JEE) standarda za izradu web aplikacija.

### Spring Boot

Java Spring Boot (Spring Boot) [11] je modul Java Spring-a (poboljšanje, odnosno nezavisan projekt baziran na Springu) koji čini razvoj web aplikacija i mikroservisa uz Spring Framework bržim i lakšim kroz tri osnovne mogućnosti:

1. Autokonfiguracija - aplikacije se inicijaliziraju s unaprijed postavljenim ovisnostima, omogućava brži razvoj aplikacije te smanjuje mogućnost ljudske pogreške

2. Samostalan pristup konfiguraciji - dodaje i konfigurira početne ovisnosti, na temelju potreba projekta, slijedeći vlastitu prosudbu (odabire koje će pakete instalirati i koje će zadane vrijednosti koristiti)

3. Mogućnost izrade samostalnih aplikacija - aplikacije koje se pokreću same, bez oslanjanja na vanjskog web poslužitelja, ugradnjom web poslužitelja (kao što su Tomcat ili Netty) u aplikaciju tijekom procesa inicijalizacije (moguće pokretanje aplikacije na bilo kojoj platformi)

## 4.3 Angular.js

Angular.js [4] je besplatni web okvir otvorenog koda koji se temelji na JavaScriptu za razvoj aplikacija na web stranici. Održavao ga je uglavnom Google i zajednica pojedinaca i korporacija. Razvijen je 2009. godine te je sada open-source okvir (dostupan je izvorni kod - može se redistribuirati i mijenjati).

## 4.4 IntelliJ IDEA

IntelliJ IDEA [8] je integrirano razvojno okruženje (*eng. IDE - Integrated Development Environment*) poduzeća *JetBrains* osmišljeno za povećanje produktivnosti programera. Obavlja rutinske i ponavljajuće zadatke umjesto korisnika pružajući pametno dovršavanje koda, statičku analizu koda i refaktoriranje.

# Poglavlje 5

# Praktični rad

## 5.1 CLUS-RM algoritam s uvjetima na entitete

Uvjetovano traženje redeskripcija baziranih na entitetima pronalazi i grupira slične entitete na temelju unaprijed definirane mjere sličnosti. Primarno se koristi za ubacivanje uvjeta korisnika u proces traženja redeskripcija.
Domenski stručnjak ima hipotezu da bi za neki podskup mogla vrijediti neka svojstva. Činjenicu ispituje dodavanjem uvjeta i ispitivanjem rezultantnih redeskripcija [19].
CLUS-RM algoritam do pisanja ovog rada imao je opciju dodavanja atributa koje će, ovisno o odabiru važnosti, pravila generirane redeskripcije sadržavati.

U ovom je radu dodana takva opcija za entitete. U datoteku u kojoj se nalaze postavke algoritma dodane su dvije nove mogućnosti:

1. elementImportance
    - označava važnost odabranih entiteta, odnosno koliko navedenih entiteta generirane redeskripcije moraju opisivati
    - none (niti jedan entitet) soft (barem jedan entitet od navedenih) ili hard (sve navedene entitete)
2. importantElements
    - imena entiteta odvojena ”;”
    - navedeni entiteti provjeravaju se tijekom izvršavanja algoritma ovisno o odabranom gornjem uvjetu

**Primjer 5.1.1.** *Odaberemo li entitete čiji su nazivi ”5”, ”9” i ”37” i želimo da redeskripcije koje generira algoritam opisuju sve navedene entitete, u datoteku „Settings.set“ upisujemo sljedeća dva retka:*

*elementImportance = hard*
*importantElements: ”5”;”9”;”37” .*

*Želimo li da se barem jedan od navedenih entiteta nalazi u skupu kojeg opisuju redeskripcije, zapisat ćemo ovako:*

*elementImportance = soft*
*importantElements: ”5”;”9”;”37” .*

Prilikom pokretanja algoritma, iz datoteke se čitaju postavke i spremaju u varijable. Kad se pročita linija koja počinje stringom „elementImportance“, u istoimenu se varijablu sprema vrijednost ovisno o odabranoj važnosti:

- *elementImportance* = 0, ako je postavljen none uvjet
- *elementImportance* = 1, ako je postavljen soft uvjet
- *elementImportance* = 2, ako je postavljen hard uvjet.

Isto tako, kad se pročita linija koja počinje stringom „importantElements“, u istoimeno polje stringova spremaju se entiteti navedeni u datoteci.

Za vrijeme traženja redeskripcija, ukoliko je varijabla *elementImportance* > 0, poziva se funkcija koja provjerava sadrži li redeskripcija one entitete koji su odabrani i pod uvjetom koji je određen.
Na sljedećoj je stranici prikazan pseudokod funkcije. Za svaki entitet koji je zapisan u postavkama, provjerava se opisuje li ga redeskripcija ili ne. Funkcija ovisno o provjerama vraća vrijednost 1 ako redeskripcija zadovoljava uvjete, odnosno 0 ako ih ne zadovoljava.

## Pseudokod funkcije

**Algorithm 2:** Pseudokod funkcije checkElements

**Input:** Entiteti trenutno provjeravane redeskripcije *redescriptionElements*, odabrani entiteti *importantElements*

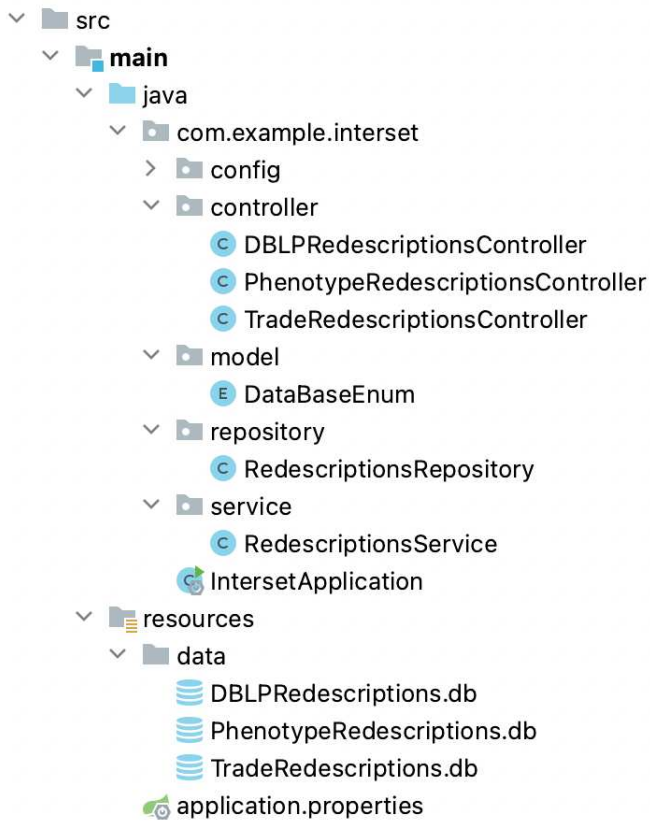
**Output:** vrijednost koja označava sadrži li redeskripcija željene entitete (*contains* = 1) ili ne (*contains* = 0)

```
1  contains ← 0
2  notContained ← 0
3  foreach element ∈ importantElements do
4      if redescriptionElements sadrži element i elementImportance == 1 then
5          contains ← 1
6          return contains
7      else if redescriptionElements ne sadrži element i elementImportance == 2
         then
8          notContained ← 1
9  if elementImportance == 2 then
10     if notContained == 1 then
11         contains ← 0
12     else
13         contains ← 1
14 return contains
```

## 5.2 Implementacija poslužiteljske aplikacije alata InterSet

Poslužiteljska aplikacija je originalno napisana pomoću node.js okruženja te je sada zamijenjena implementacijom u Java Spring Boot-u radi boljeg povezivanja i omogućavanja poziva algoritma CLUS-RM iz alata InterSet. Obzirom da je algoritam CLUS-RM pisan u Javi, tim pristupom bi se CLUS-RM i InterSet mogle snažno integrirati.

### Struktura koda




Slika 5.1: Struktura koda poslužiteljske aplikacije

Na slici 5.1 prikazana je struktura koda poslužiteljske aplikacije pisanog u Spring Boot-u. Prikaz je preuzet iz razvojnog okruženja IntelliJ IDEA Ultimate verzije.

## Model

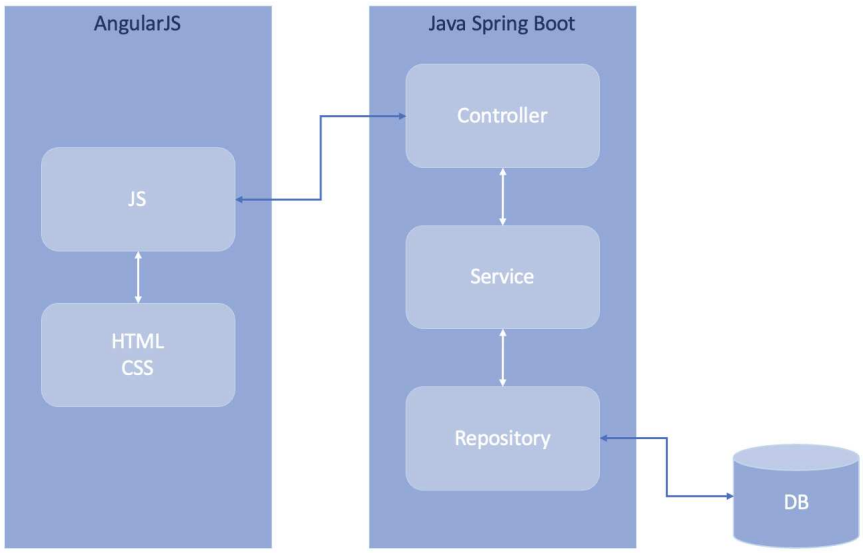
U paketu *model* nalazi se sljedeća klasa:

```
public enum DataBaseEnum {
   TRADE("data/TradeRedescriptions.db"),
   DBL("data/DBLPRedescriptions.db"),
   PHENOTYPE("data/PhenotypeRedescriptions.db");

   public final String databaseName;

   DataBaseEnum(String databaseName) {
       this.databaseName = databaseName;
   }
}
```

*DataBaseEnum* je nabrajanje, odnosno enumeracija koja će putem imena odabrane baze slati pravu vrijednost putanje do baze za upravljanje u ostalim slojevima aplikacije. Obzirom da postoje tri skupa podataka na kojima se aplikacije može pokretati, na ovaj je način odabrana baza podataka iz koje se kasnije čitaju svi podaci koji su potrebni za daljnji rad.
U svrhu poboljšanja razumijevanja sustava, konstruiran je sljedeći dijagram:




Slika 5.2: Dijagram odnosa

## Controller

Unutar paketa *controller* nalaze se tri klase za tri skupa podataka na kojima se izvršava traženje i pretraživanje redeskripcija.
Svaka klasa ima anotaciju *@RestController* koja govori Spring Boot aplikaciji da klase obrađuju HTTP zahtjeve. @RequestMapping definiran na razini klase preslikava određeni put zahtjeva ili uzorak na kontroler. U ovom slučaju, svaki zahtjev prema krajnjim točkama bit će nakon "/databaseName" (gdje databaseName može biti TRADE, DBLP ili PHENOTYPE).

Dio klase TradeRedescriptionsController:

```
@RestController
@RequestMapping("/TRADE")
public class TradeRedescriptionsController {
    RedescriptionsService redescriptionsService;
    ⋮
}
```

U svakoj klasi postoje metode koje ovisno o potrebi sadrže neku od sljedećih anotacija:

- @PostMapping - metode obrađuju HTTP POST zahtjeve koji se podudaraju s danim URI izrazom
- @GetMapping - metode obrađuju HTTP GET zahtjeve koji se podudaraju s danim URI izrazom

Funkcije odnosno metode iz sloja kontrolera pozivaju funkcije iz sloja servisa.

Primjer metode s @PostMapping anotacijom:

```
@PostMapping(value="/somData")
public JSONObject login (@RequestBody JSONObject request) {
    return redescriptionsService.login(request, DataBaseEnum.TRADE)
}
```

Primjer metode s @GetMapping anotacijom:

```
@GetMapping(value="/redescriptionElement")
public JSONObject redescriptionElement(){
    return redescriptionsService.redescriptionElement(DataBaseEnum.DBL);
}
```

## Service

Unutar *service* paketa nalazi se klasa *@RedescriptionsService* s anotacijom *@Service*. Klase označene navedenom anotacijom obično sadrže logiku aplikacije. U ovom radu, samo su implementirane funkcije u svrhu spajanja sloja kontrolera i sloja repozitorija obzirom da se sva logika odvija nakon dohvaćanja podataka iz baze. Sloj kontrolera ne bi trebao nikada izravno pozivati repozitorij pa iz tog razloga postoji ovaj sloj.

## Repository

U *repository* paketu nalazi se klasa *RedescriptionsRepository* s anotacijom *@Repository*. Implementirana klasa spaja se sa odabranom bazom podataka te ovisno o funkciji dohvaća, mijenja, upisuje ili briše podatke iz tablica u bazi.

### Spajanje na bazu

Unutar klase definiramo vezu s bazom koja je odabrana prilikom pokretanja aplikacije. *databaseName* je ime baze na koju je potrebno ostvariti spajanje radi dohvaćanja ispravnih podataka za daljnji rad.

```
private Connection connect(String databaseName) throws SQLException {
    String url =
        "jdbc:sqlite:/Users/ivakozjak/Desktop/interset/src/main/resources/"
        + databaseName;
    Connection conn = DriverManager.getConnection(url);
    return conn;
}
```

Unutar svake metode, ukoliko je potrebno spajanje na bazu, potrebno je napisati kôd na sljedeći način. Kôd u kojem se može očekivati iznimka stavimo u *try* blok iza kojeg slijedi *catch* blok s kôdom koji služi za obradu iznimke. U ovom slučaju ispisuje se greška iznimke. Unutar *if* grananja, ovisno o metodi, izvršavaju se upiti na bazu.

```
try{
    Connection conn = connect(database.databaseName);
    if(conn != null)
    {
            ⋮
    }
    conn.close();
} catch(SQLException e) {
            System.out.println(e.getMessage());
}
```

### Poziv algoritma CLUS-RM

Glavni cilj ovog rada je povezati alat InterSet s algoritmom CLUS-RM stoga je povezivanje opširnije opisano u sljedećem potpoglavlju.

## 5.3 Povezivanje alata InterSet s algoritmom CLUS-RM

Uz sve prijašnje metode koje su postojale u verziji InterSet alata koji je služio za pretraživanje redeskripcija, dodane su nove metode kako bi se korisniku omogućilo i traženje novih redeskripcija.

### Pozivanje algoritma u Angular-u (promjene u HTML-u)

Kako bi se korisniku omogućilo pokretanje algoritma, potrebno je na sučelju dodati gumbe. Obzirom da postoje dva načina pokretanja algoritma: s uvjetima na entitete i uvjetima na atribute, gumbi su dodani u dvije datoteke: *elementInfo.html* i *attributeInfo.html*.

```
<button ng-model="button" ng-click="runCLUSRMSettingsElements()">Run
    CLUS-RM</button>
<button ng-model="button" ng-click="runCLUSRMSettings()">Run
    CLUS-RM</button>
```

Ukoliko nije odabran klaster u kartici entiteta, odnosno par atributa u kartici atributa, gumbi nisu vidljivi. Također, uz gumbe za poziv algoritma, dodani su i gumb za dodatne postavke algoritma te polja odabira težine uvjeta (*radio button*) koji svojstvo vidljivosti imaju kao i prethodno navedeni gumbi.

```
<label><input type="radio" ng-model="conditionElements" value="none"> none
    </label>
<label><input type="radio" ng-model="conditionElements" value="soft"> soft
    </label>
<label><input type="radio" ng-model="conditionElements" value="hard"> hard
    </label>
```

Pritiskom na gumbe za pokretanje algoritma, pozivaju se sljedeće funkcije: `runCLUSRMSettingsElements()`, odnosno `runCLUSRMSettings()`.

## Pozivanje algoritma u Angular-u (promjene u JS-u)

Spomenute funkcije implementirane su na sličan način pa će biti prikazana funkcija *runCLUSRMSettingsElements()* te će biti objašnjene razlike u drugoj funkciji.
Obje funkcije implementirane su u datoteci *visual.js*.

Prije pokretanja poslužiteljske strane aplikacije, potrebno je provjeriti je li postavljen uvjet na entitete, odnosno atribute. Provjerava se vrijednost *conditionElements* unutar *if* petlje. Ukoliko uvjet nije odabran, odnosno vrijednost nije *none/soft/hard*, unutar funkcije se poziva

```
window.alert('Select condition for elements!');
```

čijim se pozivom korisniku otvara skočni prozor s obavijesti o obaveznom odabiru. Na isti se način provjerava i u drugoj funkciji. Jedina je razlika što umjesto entiteta pišu atributi i provjeravaju se postavljeni uvjeti za dva pogleda: *conditionw1*, *conditionw2*.

Ako se skočni prozor ne otvori, slijedi pokretanje Spring Boot dijela aplikacije.

```
var clusLink = "http://127.0.0.1:8089/databaseName/clusrmSettingsElements";
var options = {
    url: clusLink,
    method: 'POST',
    params: {},
    data: {
        conditionElements: JSON.stringify($scope.conditionElements),
        selected: JSON.stringify($scope.selSOMElements),
        numRandomRestarts: JSON.stringify($scope.numRandomRestarts),
        numIterations: JSON.stringify($scope.numIterations),
        numRetRed: JSON.stringify($scope.numRetRed),
        minSupport: JSON.stringify($scope.minSupport),
        maxSupport: JSON.stringify($scope.maxSupport),
        minJS: JSON.stringify($scope.minJS),
        maxPval: JSON.stringify($scope.maxPval),
        allowLeftNeg: JSON.stringify($scope.allowLeftNeg),
        allowRightNeg: JSON.stringify($scope.allowRightNeg),
        allowLeftDisj: JSON.stringify($scope.allowLeftDisj),
        allowRightDisj: JSON.stringify($scope.allowRightDisj)
    }
};
```

```
$http(options)
    .success(function (data) {detaljnije na stranici 32});
    .error(function (data, status) {
        if (status === 404) {
            $scope.error = 'REST server is offline!';
            }
        else {
            $scope.error = 'Error: ' + status;
            }
    });
```

Ponovno, *databaseName* označava ime baze na koju se spajamo, ovisno o odabiru skupa na kojem je aplikacija pokrenuta. Obzirom da se radi o *POST* metodi, skup parametara je prazan. Unutar *data* šalje se odabrana težina uvjeta na entitete, odabrani entiteti (entiteti odabranog klastera) te opcije dodatnih postavki algoritma.

Na sličan je način implementirana i funkcija za atribute. Link je oblika *„http://127.0.0.1:8089/databaseName/clusrmSettings“*, šalju se dvije težine uvjeta, odabrani atributi i dodatne postavke.

## Pozivanje algoritma u Spring Boot-u

Unutar *controller* paketa, u sve tri klase, napisana je funkcija:

```
@PostMapping(value="/clusrmSettingsElements")
public JSONObject clusrmSettingsElements(@RequestBody JSONObject request){
    return redescriptionsService.clusrmSettingsElements(request,
        DataBaseEnum.databaseName);
}
```

Metoda obrađuje HTTP POST zahtjev koji se podudara s danim URI izrazom. U ovom je slučaju vrijednost izraza *clusrmSettingsElements*.

Iz ove se metode poziva istoimena metoda unutar *service* paketa:

```
public JSONObject clusrmSettingsElements(JSONObject request, DataBaseEnum
    database){
    return redescriptionsRepository.clusrmSettingsElements(request,
        database);
}
```

Također slijedi poziv istoimene metode iz *repository* paketa:

```
public JSONObject clusrmSettingsElements(JSONObject request,DataBaseEnum
    database) {...}
```

Unutar funkcije `clusrmSettingsElements()` odvija se cijela logika promjene datoteke s postavkama algoritma, pozivanje algoritma te slanje novih redeskripcija korisniku.

Prije samog rada s datotekama, potrebno je iz objekta *JSONObject request* pročitati sve podatke koje je korisnik odabrao/unio putem aplikacije prije odabira gumba za pokretanje algoritma.

Čitanje podataka zapisano je unutar *try-catch* bloka. Podatke dobivamo pomoću funkcije *get*. Ovisno o tipu podatka, dohvaćamo vrijednosti na različite načine. Prikazuju se tri primjera dohvaćanja tipa: *string*, *double* i *array*.
Želimo li dohvatiti string u kojem je zapisana težina uvjeta, dohvaćamo ga na sljedeći način:

```
conditionElements = String.valueOf(request.get("conditionElements"));
```

Želimo li dohvatiti *double* vrijednost varijable u koju spremamo maksimalnu p-vrijednost redeskripcije, dohvaćamo:

```
maxPval = Double.parseDouble(String.valueOf(request.get("maxPval")));
```

Želimo li dohvatiti polje odabranih entiteta, potrebno je za početak definirati parser te pomoću njega dohvatiti željene podatke:

```
JSONParser jsonParser = new JSONParser();
selected = (JSONArray)
    jsonParser.parse(String.valueOf(request.get("selected")));
```

Sve prethodno navedeno vrijedi i za poziv funkcije `clusrmSettings`. Razlike su u nazivima funkcija i varijabli koje dohvaćamo.

Nakon uspješnog spremanja svih podataka iz varijable *request* u lokalne varijable, slijedi promjena datoteke s postavkama koju koristimo za pokretanje algoritma.

Upisivanje novih postavki ovisno o odabiru korisnika:

```
String path_file = "path_to_Settings.set";
List<String> newLines = new ArrayList<>();
```

```
try {
    for (String line : Files.readAllLines(Paths.get(path_file),
        StandardCharsets.UTF_8)) {
            ⋮
    }
    Files.write(Paths.get(path_file), newLines, StandardCharsets.UTF_8);
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
```

*path_to_Settings.set* označava putanju do datoteke u koju su upisane postavke. Ovisno o korisnikovom odabiru, prolazi se po čitavoj datoteci i dodaju se nove ili izmijenjuju već postojeće linije.
U *for* petlji, u kojoj se prolazi po svim linijama datoteke, upisuju se postavke koje je korisnik odabrao. Postavke koje korisnik nema mogućnost mijenjati ostaju iste. Kad se pročita njihov sadržaj, izvrši se naredba *newLines.add(line)*. Dakle, linija koja se pročita, dodaje se u listu stringova *newLines*.
Za postavke koje je korisnik promijenio, ovisno o sadržaju datoteke, mogu se dogoditi dva slučaja:

1. linija s tom opcijom postoji, odnosno već je upisana u datoteku s postavkama i potrebna je samo izmjena (neka je na primjer nova vrijednost linije string *„new line“*), tada se u kodu izvršava sljedeća naredba

   ```
   newLines.add(line.replace(line, "new line");
   ```

2. linija ne postoji, odnosno opcija nije upisana u datoteku s postavkama i potrebno je dodati novu liniju (neka je ponovno nova vrijednost *„new line“*), tada se izvršava sljedeća naredba

   ```
   newLines.add("new line");
   ```

Po izlasku iz *for* petlje, funkcijom *write* zapisujemo nove linije u datoteku.

CLUS-RM algoritam je algoritam napisan u Javi. Nakon dodavanja uvjeta na entitete, generirana je .jar datoteka. Datoteka s ekstenzijom .jar je datoteka Java Archive koja se koristi za pohranu Java programa.

Pokretanje algoritma također se nalazi unutar *try-catch* bloka ukoliko dođe do greške prilikom pokretanja.

```
try {
    Process proc = Runtime.getRuntime().exec("java -jar
        path_to_RMWConstrainedAdaptive.jar path_to_Settings.set");
    proc.waitFor();
} catch(Throwable e){
    System.out.println("algorithm-error");
}
```

*path_to_RMWConstrainedAdaptive.jar* označava putanju do *RMWConstrainedAdaptive.jar* datoteke koja sadrži izvršni kod algoritma, dok *path_to_Settings.set* označava putanju do datoteke u koju su upisane postavke potrebne za pokretanje algoritma.

Po završetku rada algoritma, stvara se nova datoteka s popisom pronađenih redeskripcija. Svaka redeskripcija sadrži dva pravila koja je opisuju, Jaccardov indeks sličnosti, p-vrijednost, broj i popis entiteta koje redeskripcija opisuje (*intersection*) te broj i popis entiteta koje opisuje barem jedno pravilo redeskripcije (*union*).

Primjer prikaza jedne redeskripcije (preuzeto iz datoteke nastale kao rezultat algoritma):

```
Rules:

W1R: BAL >= -6.6159 <= 14.3827 AND EMPL_INDUST_M >= 19.0 <= 35.6 AND EMPL_INDUST_F >= 3.8 <= 15.1 AND
COMP_CAPIT >= 3.9294 <= 127.4752 AND POP_64 >= 10.8274 <= 19.4025
W2R: E91 >= 1.0 <= 11.0 AND E/I66 >= 0.348 <= 4.563 AND E/I19 >= 0.944 <= 4.678 AND E/I37 >= 0.039 <=
1.128 AND E76 >= 0.0 <= 3.0 AND I38 >= 0.0 <= 1.0 AND E26 >= 4.0 <= 26.0
JS: 0.5555555555555556
p-value : 4.230371275504297E-9
Support intersection: 10
Support union: 18

Covered examples (intersection):
"214" "150" "126" "120" "114" "94" "82" "74" "20" "9"
Union elements:
"214" "195" "188" "160" "150" "126" "120" "114" "110" "100" "94" "82" "74" "73" "59" "56" "20" "9"
```

Slika 5.3: Nova redeskripcija

Ponovno unutar *try-catch* bloka, slijedi čitanje prethodno opisane datoteke s novonastalim redeskripcijama u svrhu spremanja i slanja podataka za prikaz u korisničkom sučelju.

```
File myObj = new File("path_to_outputFile.rr");
Scanner myReader = new Scanner(myObj);
while (myReader.hasNextLine()) {...}
```

*path_to_outputFile.rr* označava putanju do prethodno opisane datoteke.

Osim podataka koji se nalaze u datoteci, spremaju se i:

- Jaccardovi indeksi između skupa entiteta redeskripcije i skupova entiteta redeskripcija iz baze,
- Jaccardovi indeksi između skupa atributa redeskripcije i skupova atributa redeskripcija iz baze,
- Jaccardov indeks između skupa entiteta redeskripcije i skupa entiteta odabranog klastera.

Sva tri navedena polja dobivaju se upitima na bazu te provjerom unija i presjeka dviju redeskripcija. Također je korisno spremiti i sume svih indeksa prva dva polja koje će biti korisne ukoliko kasnije korisnik odabere spremanje jedne ili više redeskripcija u bazu podataka.

Svi navedeni podaci se vraćaju kako bi se mogli prikazati u sučelju.

## Prikaz dohvaćenih podataka (promjene u JS-u)

Unutar *success* funkcije, navedene u potpoglavlju *Pozivanje algoritma u Angular-u (promjene u JS-u)* na stranici 28, spremaju se podaci potrebni za prikaz tablica s redeskripcijama kao rezultat CLUS-RM algoritma. Ako algoritam završi s nula pronađenih redeskripcija, korisnik dobiva poruku *„No redescriptions found“* i tablice se ne prikazuju. Pronađe li algoritam barem jednu redeskripciju koja zadovoljava uvjete korisnika, tablicu novih redeskripcija popunjavamo na sljedeći način:

```
for (i=0; i<data.redescriptionsCountCLUS; i++){
    $scope.redescriptionTable[i] =
        {redescription:$scope.redescriptionsCLUS[i], EJsel:$scope.EJsel[i],
        maxAEJs:$scope.maxEJs[i], attJac:$scope.attJac[i]};
}
```

*redescriptionsCountCLUS* je broj pronađenih redeskripcija. *redescriptionsCLUS* je polje pronađenih redeskripcija koje sadrži podatke pročitane iz datoteke nastale izvršavanjem algoritma. *EJsel* označava Jaccardov indeks sličnosti skupa entiteta redeskripcije i skupa entiteta odabranog klastera. *maxEJs* označava najveći Jaccardov indeks sličnosti skupa entiteta redeskripcije i skupa entiteta redeskripcije iz baze. *attJac* označava Jaccardov indeks sličnosti skupa atributa redeskripcije i skupa atributa redeskripcije iz baze s kojom ima maksimalni Jaccardov indeks entiteta (*maxEJ*).

## Prikaz dohvaćenih podataka (promjene u HTML-u)

Sukladno prethodno navedenom spremanju podataka, nastaje nova tablica koja postaje vidljiva korisniku ukoliko je pronađena barem jedna nova redeskripcija.

```
<table id="redescriptionsCLUSRM" ng-hide="!redescriptionsCLUS">
    <tr>
        <th> #</th>
        <th> Left query</th>
        <th> Right query</th>
        <th> J</th>
        <th> Support</th>
        <th> EJ (with selected cluster)</th>
        <th> EJ (max pairwise from db)</th>
        <th> AJ</th>
   </tr>
    <tr id="id{{ x.redescription.id }}" ng-click="showElements()"
        ng-repeat="x in redescriptionTable">
        <td><input ng-model="x.selected" type="checkbox"></td>
        <td >{{ x.redescription.w1 }}</td>
        <td>{{ x.redescription.w2 }}</td>
        <td>{{ x.redescription.dfJS }}</td>
        <td>{{ x.redescription.intersection}}</td>
        <td>{{ x.EJsel }}</td>
        <td>{{ x.maxAEJs }}</td>
        <td>{{ x.attJac }}</td>
    </tr>
</table>
```

Također je dodan i gumb za spremanje odabranih novih redeskripcija u bazu čije je svojstvo vidljivosti definirano kao i za prethodno navedenu tablicu. Pritiskom na njega, poziva se funkcija `saveCLUSRMRedescriptions()`.

```
<button ng-hide="!redescriptionsCLUS" ng-model="button"
    ng-click="saveCLUSRMRedescriptions()">Save</button>
```

## Spremanje odabranih redeskripcija (promjene u JS-u)

Funkcija `saveCLUSRMRedescriptions()` implementirana je u datoteci *visual.js*.

Prije pokretanja poslužiteljske strane aplikacije, potrebno je provjeriti je li označena barem jedna redeskripcija iz tablice. Ukoliko nije označena niti jedna, unutar funkcije se poziva

```
window.alert('Select redescriptions to save');
```

čijim se pozivom korisniku otvara skočni prozor s obavijesti o obaveznom odabiru.

Ako se skočni prozor ne otvori, slijedi pokretanje Spring Boot dijela aplikacije.

```
var clusLink =
    "http://127.0.0.1:8089/databaseName/saveCLUSRMRedescriptions";
var options = {
    url: clusLink,
    method: 'POST',
    params: {},
   data: {
        userId: JSON.stringify($scope.userInfo.userInfo[0].userId),
        selectedToSave: JSON.stringify($scope.result),
        redescriptionsCLUS: JSON.stringify($scope.redescriptionsCLUS),
        sumEJ: JSON.stringify($scope.sumEJ),
        sumAJ: JSON.stringify($scope.sumAJ),
        EJs: JSON.stringify($scope.EJs),
        AJs: JSON.stringify($scope.AJs)
    }
};

$http(options)
    .success(function (data) {
        window.alert('Redescriptions: ' + $scope.result + ' saved
            successfully');
        location.reload();
    }
    .error(function (data, status) {
        if (status === 404) {
            $scope.error = 'REST server is offline!';
        } else {
            $scope.error = 'Error: ' + status;
        }
    });
```

Ponovno, *databaseName* označava ime baze na koju se spajamo, ovisno o odabiru skupa na kojem je aplikacija pokrenuta. Obzirom da se radi o *POST* metodi, skup parametara je

prazan. Unutar *data* šalje se *id* korisnika, odabrane redeskripcije te prethodno izračunati Jaccardovi indeksi sličnosti.

Ukoliko se redeskripcije uspješno spreme, otvara se skočni prozor s obavijesti o uspješnom spremanju te se osvježi aplikacija kako bi korisniku bile vidljive promjene na svim karticama aplikacije. Ukupan broj redeskripcija je povećan te su vidljive promjene u grafovima.

## Spremanje odabranih redeskripcija u Spring Boot-u

Unutar *controller* paketa, u sve tri klase, napisana je funkcija:

```
@PostMapping(value="/saveCLUSRMRedescriptions")
public JSONObject saveCLUSRMRedescriptions(@RequestBody JSONObject
    request){
    return redescriptionsService.saveCLUSRMRedescriptions(request,
        DataBaseEnum.databaseName);
}
```

Metoda obrađuje HTTP POST zahtjev koji se podudara s danim URI izrazom. U ovom je slučaju vrijednost izraza *saveCLUSRMRedescriptions*.
Iz ove se metode poziva istoimena metoda unutar *service* paketa:

```
public JSONObject saveCLUSRMRedescriptions(JSONObject request,
    DataBaseEnum database){
    return redescriptionsRepository.saveCLUSRMRedescriptions(request,
        database);
}
```

Također slijedi poziv istoimene metode iz *repository* paketa:

```
public JSONObject saveCLUSRMRedescriptions(JSONObject request,DataBaseEnum
    database) {...}
```

Unutar `saveCLUSRMRedescriptions()` funkcije odvija se cijela logika promjena tablica u bazi podataka.

Prije samog rada s bazom, potrebno je iz objekta *JSONObject request* pročitati sve podatke vezane uz redeskripcije koje je korisnik odabrao putem aplikacije prije odabira gumba za spremanje.

Čitanje podataka zapisano je unutar *try-catch* bloka. Podatke dobivamo pomoću funkcije *get*. Ovisno o tipu podatka, dohvaćamo vrijednosti na različite načine. Prikazuju se dva primjera dohvaćanja tipa: *string* i *array*.
Želimo li dohvatiti string u kojem su zapisani redni brojevi odabranih redeskripcija, dohvaćamo ga na sljedeći način:

```
selectedToSave = String.valueOf(request.get("selectedToSave"));
```

Želimo li dohvatiti polje Jaccardovih indeksa sličnosti entiteta, potrebno je za početak definirati parser te pomoću njega dohvatiti željene podatke:

```
JSONParser jsonParser = new JSONParser();
EJs = (JSONArray) jsonParser.parse(String.valueOf(request.get("EJs")));
```

Nakon uspješnog spremanja svih podataka iz varijable *request* u lokalne varijable, slijedi spajanje na bazu podataka i promjena tablica u bazi.
Spajanje na bazu opisano je na stranici 24.
Spremanjem novih redeskripcija u bazu potrebno je ažurirati više tablica.

U svakoj tablici bit će nam potreban broj redeskripcija koje postoje u bazi, a njega pronalazimo na sljedeći način:

```
ResultSet maxRedescriptionId = stmt.executeQuery("SELECT
    MAX(redescriptionID) as maxId from RedescriptionTable");
while (maxRedescriptionId.next()) {
    maxID = maxRedescriptionId.getInt("maxId");
}
```

### RedescriptionTable

U tablici *RedescriptionTable* potrebno je ažurirati sve postojeće redeskripcije i dodati nove. Promjene postojećih redaka se događaju u stupcima u kojima su upisani prosječni Jaccardovi indeksi entiteta i atributa. Za svaki se redak pomnože vrijednosti tih stupaca sa $maxID - 1$, pribroje im se prethodno izračunate vrijednosti Jaccardovih indeksa entiteta/atributa i na kraju podijele s ukupnim brojem redeskripcija $-1$. Za svaku novu redeskripciju potrebno je dodati vrijednosti svih stupaca: dva pravila koja opisuju redeskripciju, Jaccardov indeks sličnosti, broj entiteta koje redeskripcija opisuje, p-vrijednost te prosječne Jaccardove indekse sličnosti entiteta i atributa.

#### AttributeFrequencyTable

U tablici *AttributeFrequencyTable* potrebno je povećati broj pojavljivanja atributa koji se nalaze u pravilima redeskripcija.

#### AttributeCoocurenceTable

U tablici *AttributeCoocurenceTable* potrebno je povećati frekvenciju, odnosno broj pojavljivanja, parova atributa koji se nalaze u pravilima redeskripcija.

#### RedescriptionElementTable

U tablicu *RedescriptionElementTable* potrebno je dodati sve entitete koje nova redeskripcija opisuje. Tablica sadrži dva stupca: id redeskripcije i id entiteta.

#### ElementCoverage

U tablici *ElementCoverage* potrebno je svakom entitetu kojeg opisuje nova redeskripcija povećati broj redeskripcija koje ga opisuju.

#### RedescriptionAttributeTable

U tablicu *RedescriptionAttributeTable* potrebno je dodati sve atribute koji se nalaze u pravilima redeskripcije. Tablica sadrži šest stupaca: id redeskripcije, redni broj disjunkcije u kojoj se atribut nalazi (počevši od 0), id atributa, granice vrijednosti (minimum i maksimum) te binarnu vrijednost koja označava je li atribut, odnosno njegova vrijednost negirana (1) ili nije (0).

#### GraphTable

U tablicu *GraphTable* potrebno je dodati parove novih i postojećih redeskripcija i njihove Jaccardove indekse sličnosti entiteta.

#### GraphAttrTable

U tablicu *GraphAttrTable* potrebno je dodati parove novih i postojećih redeskripcija i njihove Jaccardove indekse sličnosti atributa.

Cijeli kod, kao i poveznice na prethodne verzije InterSet alata i CLUS-RM algoritma mogu se pronaći na sljedećoj poveznici: `https://github.com/ivakozjak/InterSet`.

# Poglavlje 6

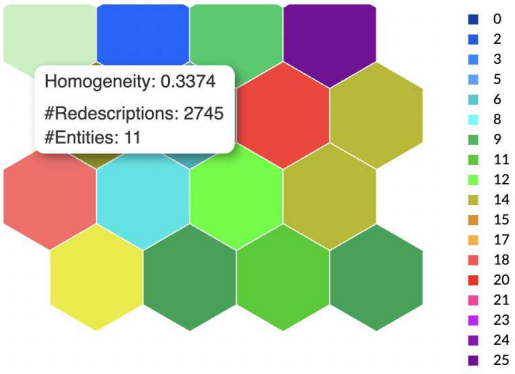
# Opis i prikaz novih funkcionalnosti

## 6.1 Pokretanje CLUS-RM algoritma

### Uvjeti na entitete

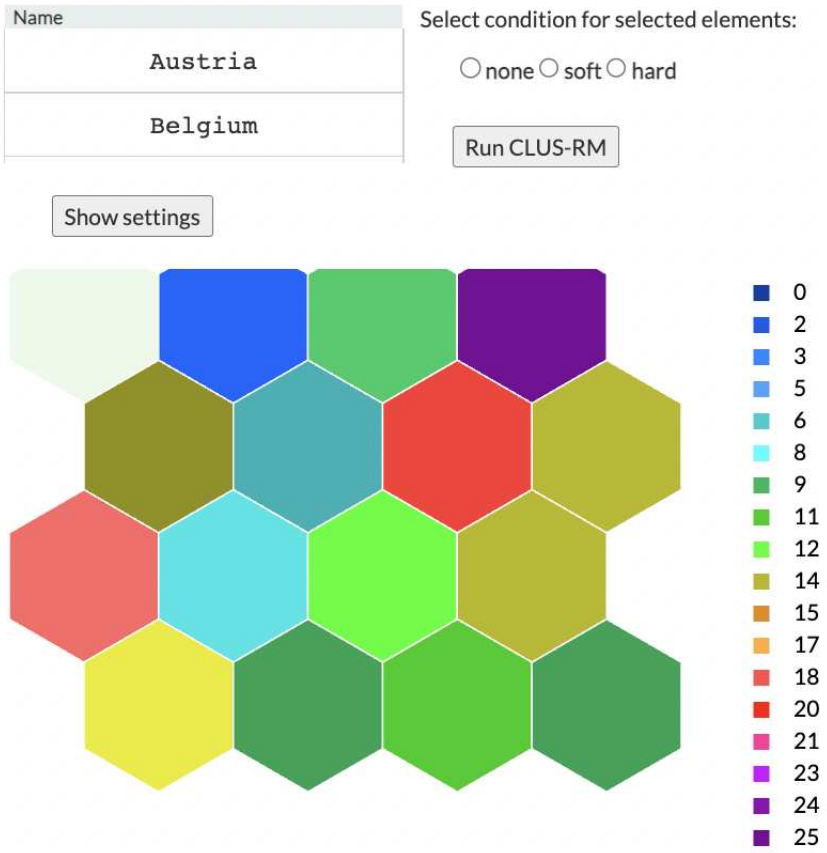
Odabirom kartice *„Entity - based exploration“* korisnik dobiva novu mogućnost nakon odabira klastera iz SOM mape dobivene treniranjem ili iz baze.

Slike u ovom poglavlju prikazuju primjer u kojem se odabire prvi klaster iz SOM mape dobivene iz podataka zapisanih u bazi. Prije pokretanja CLUS-RM algoritma, odabrani klaster opisuje 2745 redeskripcija te je homogenost klastera 0.3374.




Slika 6.1: Podaci o klasteru prije pokretanja algoritma

Nakon što se odabere željeni klaster, iznad SOM mape prikazuje se tablica entiteta (s mogučnošću pomicanja kako bi se vidjeli svi entiteti) koji se nalaze u klasteru, polja kojima se označavaju uvjeti na entitete, gumb sa dodatnim postavkama CLUS-RM algoritma, te gumb za pokretanje navedenog algoritma.




Slika 6.2: Odabir klastera omogućava pokretanje algoritma

Uvjeti na entitete koje korisnik može odabrati su:

1. *none* - nema uvjeta
2. *soft* - bilo koji entitet iz odabranog klastera (ali barem jedan) se mora nalaziti u redeskripciji
3. *hard* - svi navedeni entiteti moraju biti u redeskripciji

Ukoliko korisnik ne odabere niti jedno od ponuđenih polja, otvara se skočni prozor s obavijesti o nužnom odabiru toga uvjeta. Dodatne postavke korisnik može, ali ne mora, mijenjati. Detaljnije o dodatnim postavkama slijedi u nastavku ovog poglavlja.

## Uvjeti na atribute

Odabirom kartice *„Attribute - based exploration“* korisnik dobiva novu mogućnost nakon odabira para atributa iz Heatmap-a.

Slike u ovom poglavlju prikazuju primjer u kojem se odabire par atributa: *CRED_COVER* iz prvog pogleda te *E/I66* iz drugog pogleda.


Slika 6.3: Odabir para atributa omogućava pokretanje algoritma

Nakon što se odabere željeni par, iznad Heatmap-a prikazuju se odabrani atributi, polja kojima se označavaju uvjeti na atribute, gumb sa dodatnim postavkama CLUS-RM algoritma, te gumb za pokretanje navedenog algoritma.

Atribut *CRED_COVER* pripada w1, odnosno prvom pogledu, a atribut *E/I66* w2, odnosno drugom pogledu. Uvjeti na atribute koje korisnik može odabrati su:

1. *none* - nema uvjeta
2. *soft* - bilo koji od odabranih atributa (ali barem jedan) se mora nalaziti u pravilu redeskripcije
3. *hard* - svi navedeni atributi moraju biti u pravilu redeskripcije

Pritom treba uzeti u obzir:

- ukoliko se za jedan od uvjeta odabere opcija none, atribut/atributi iz tog pogleda se brišu i ne zapisuju u postavke algoritma
- ukoliko se na oba odabere soft uvjet, prilikom generiranja pravila redeskripcije, bit će dovoljno da se jedan od zapisanih atributa pojavi u pravilu
- ukoliko se na jednom odabere soft, a na drugom hard uvjet, oba poprimaju hard vrijednost, odnosno svi navedeni atributi moraju se nalaziti u pravilu redeskripcije.

Ukoliko korisnik ne odabere niti jedno od ponuđenih polja (vrijedi za oba pogleda) otvara se skočni prozor s obavijesti o nužnom odabiru uvjeta. Dodatne postavke korisnik može, ali ne mora, mijenjati. Detaljnije o dodatnim postavkama slijedi u nastavku ovog poglavlja.

Na sljedećoj slici prikazuje se primjer u kojem korisnik odabire prethodno navedene atribute, zatim kreira heatmap na odabranoj selekciji te odabere novi par atributa. U tom slučaju, ukupno su odabrana četiri atributa te se tada uvjeti odnose na sljedeći način:

- w1: *CRED_COVER* i *POP_64*
- w2: *E/I66* i *E/I83*.


Slika 6.4: Odabir parova atributa omogućava pokretanje algoritma

## Dodatne postavke

Pritiskom na gumb *„Show settings“*, gumb postaje *„Hide settings“* i otvaraju se postavke u kojima su prikazane inicijalne vrijednosti. Nakon što se promijene željene opcije, pritiskom gumba *„Run CLUS-RM“* pokreće se algoritam za traženje novih redeskripcija te se postavke automatski zatvaraju.


Slika 6.5: Prikaz dodatnih postavki

Dodatne postavke:

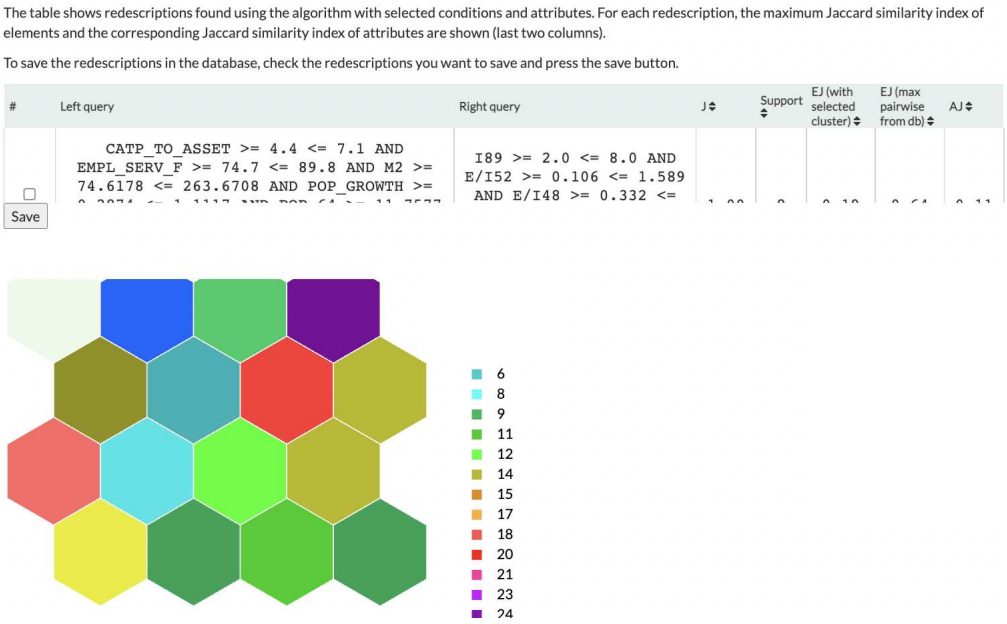
1. *numRandomRestarts* - broj pokretanja s nasumično generiranom inicijalizacijom
2. *numIterations* - broj iteracija
3. *numRetRed* - maksimalni broj vraćenih redeskripcija

4. *minSupport* - minimalni broj entiteta koje opisuje redeskripcija
5. *maxSupport* - maksimalni broj entiteta koje opisuje redeskripcija
6. *minJS* - minimalni Jaccardov indeks sličnosti
7. *maxPval* - maksimalna p-vrijednost
8. *allowLeftNeg* - dopuštanje negacije u prvom pravilu redeskripcije
9. *allowRightNeg* - dopuštanje negacije u drugom pravilu redeskripcije
10. *allowLeftDisj* - dopuštanje disjunkcije u prvom pravilu redeskripcije
11. *allowRightDisj* - dopuštanje disjunkcije u drugom pravilu redeskripcije

## 6.2 Prikaz i spremanje novih redeskripcija u bazu

### Kartica entiteti

Nakon izvršavanja algoritma, na stranici se prikazuje tablica s novim redeskripcijama.




Slika 6.6: Tablica s novim redeskripcijama

Tablicu je moguće sortirati po zadnjih pet stupaca:

1. *J* - Jaccardov indeks sličnosti redeskripcije
2. *Support* - broj entiteta koje opisuje redeskripcija
3. *EJ with selected cluster* - Jaccardov indeks skupa entiteta opisanih redeskripcijom i skupa entiteta klastera
4. *EJ max pairwise from db* - maksimalni Jaccardov indeks skupa entiteta opisanih redeskripcijom i skupa entiteta opisanih redeskripcijom (nekom) iz skupa redeskripcija
5. *AJ* - Jaccardov indeks skupa atributa redeskripcije i skupa atributa one redeskripcije s kojom ima maksimalni Jaccardov indeks entiteta

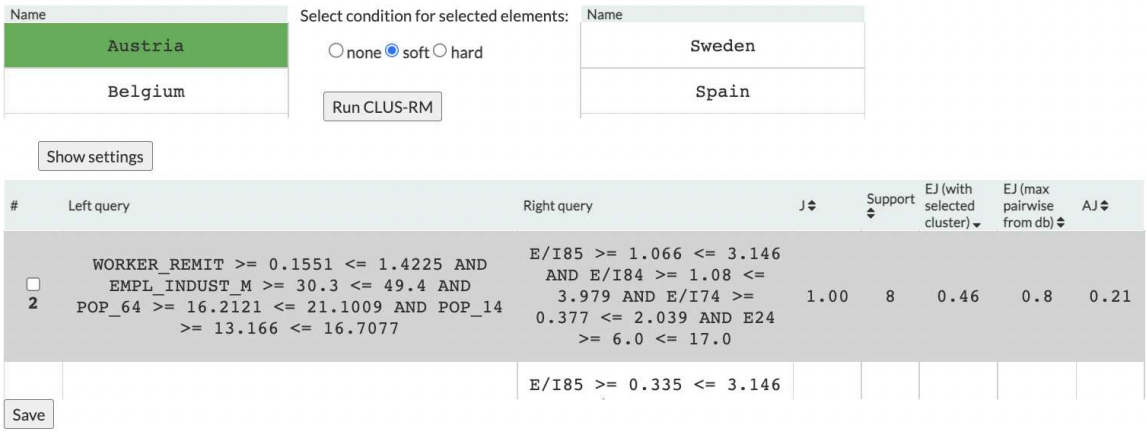
Obzirom da tražimo redeskripcije koje opisuju odabrani klaster, najbolje je sortirati tablicu po stupcu *EJ with selected cluster*.

| # | Left query | Right query | J | Support | EJ (with selected cluster) | EJ (max pairwise from db) | AJ |
|---|---|---|---|---|---|---|---|
| ☐ 2 | WORKER_REMIT >= 0.1551 <= 1.4225 AND EMPL_INDUST_M >= 30.3 <= 49.4 AND POP_64 >= 16.2121 <= 21.1009 AND POP_14 >= 13.166 <= 16.7077 | E/I85 >= 1.066 <= 3.146 AND E/I84 >= 1.08 <= 3.979 AND E/I74 >= 0.377 <= 2.039 AND E24 >= 6.0 <= 17.0 | 1.00 | 8 | 0.46 | 0.8 | 0.21 |
| | | E/I85 >= 0.335 <= 3.146 | | | | | |

Save

Slika 6.7: Sortirana tablica redeskripcija

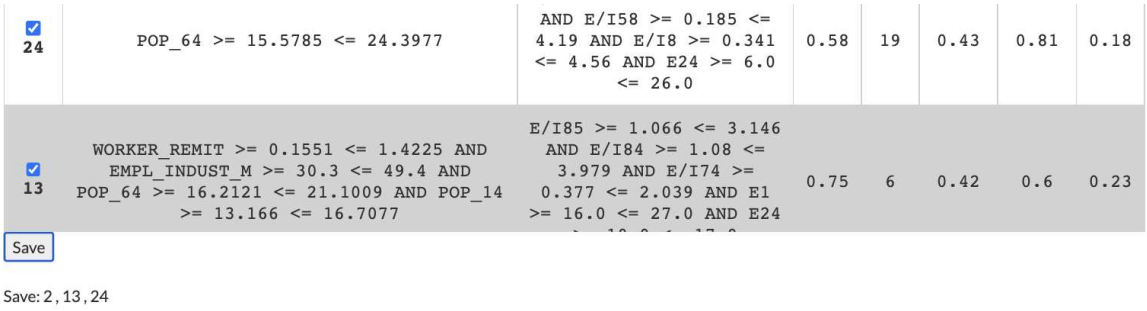
Odabirom redeskripcije iz tablice, prikazuje se nova tablica s entitetima koje ta redeskripcija opisuje. Obzirom da je korisnik pokretao algoritam s uvjetima na entitete iz klastera, također se događaju i promjene u tablici entiteta iz klastera - entiteti koji se nalaze i u redeskripciji i u klasteru označeni su zelenom bojom.




Slika 6.8: Odabrana redeskripcija iz tablice

Ukoliko korisnik želi spremiti neke od novih redeskripcija, potrebno ih je označiti kvačicom i pritisnuti tipku *„Save“*. Pritiskom na tipku, ispisuju se redni brojevi označenih redeskripcija te se željene redeskripcije spremaju u bazu.

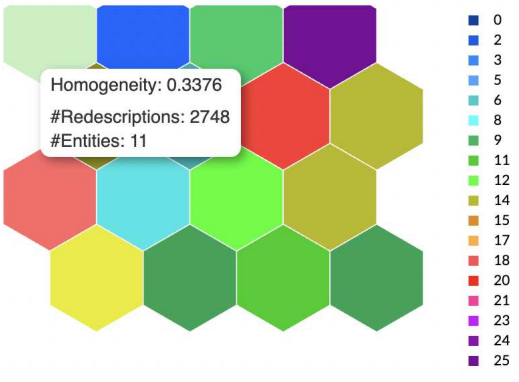
U primjeru su odabrane redeskripcije označene brojevima 2, 13 i 24.




Slika 6.9: Spremanje redeskripcija u bazu

Nakon upisa u bazu, pojavljuje se skočni prozor s obavijesti o uspješnom spremanju te se po potvrdi obavijesti ažuriraju svi podaci i prikazi u aplikaciji.

Pogledamo li sada ponovno SOM mapu koja je bila prikazana prije spremanja, možemo primijetiti da se broj redeskripcija koje opisuje opisani klaster povećao sa 2745 na 2748 te je homogenost narasla sa 0.3374 na 0.3376.

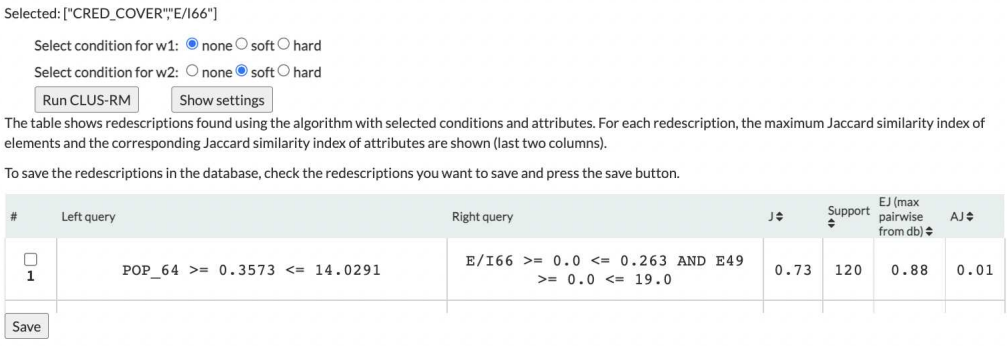



Slika 6.10: Podaci o klasteru nakon pokretanja algoritma

Također, promatra li se kartica *„Property - based exploration“* prije i nakon spremanja redeskripcija, može se primijetiti promjena u grafovima i ukupnom broju redeskripcija koje se nalaze u bazi.

## Kartica atributi

Nakon izvršavanja algoritma, na stranici se prikazuje tablica s novim redeskripcijama.

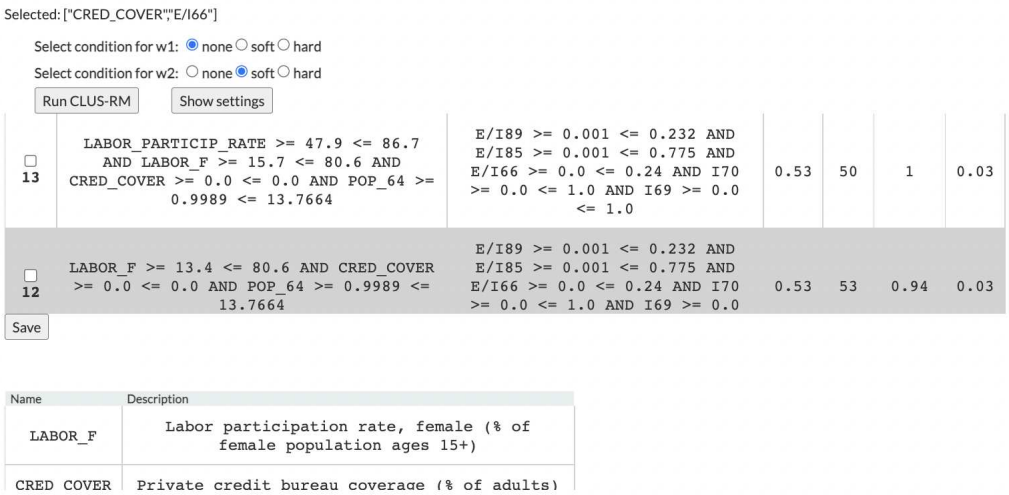



Slika 6.11: Tablica s novim redeskripcijama

Tablicu je moguće sortirati po zadnja četiri stupca:

1. *J* - Jaccardov indeks sličnosti redeskripcije
2. *Support* - broj entiteta koje opisuje redeskripcija
3. *EJ max pairwise from db* - maksimalni Jaccardov indeks skupa entiteta redeskripcije i skupa entiteta redeskripcije iz baze
4. *AJ* - Jaccardov indeks skupa atributa redeskripcije i skupa atributa redeskripcije s kojom ima maksimalni Jaccardov indeks entiteta

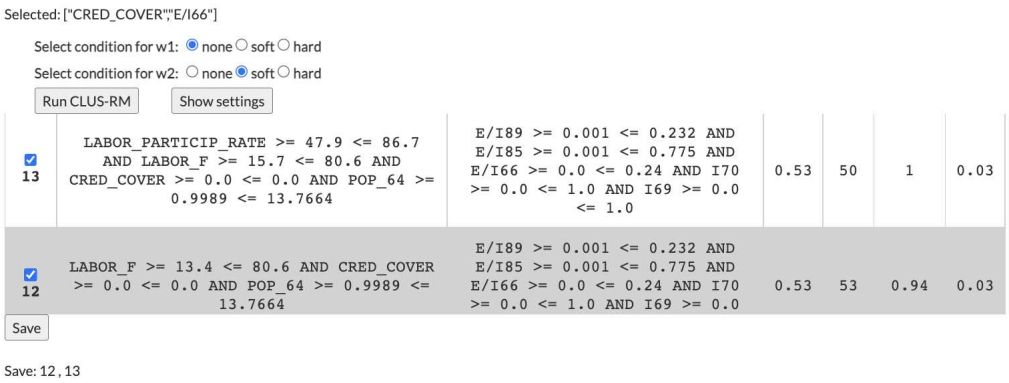
Odabirom redeskripcije iz tablice, prikazuje se nova tablica s atributima koje pravila te redeskripcije sadrže. Uz svaki atribut, prikazan je i njegov opis, odnosno što on označava.




Slika 6.12: Odabrana redeskripcija iz tablice

Ukoliko korisnik želi spremiti neke od novih redeskripcija, potrebno ih je označiti kvačicom i pritisnuti tipku *„Save“*. Pritiskom na tipku, ispisuju se redni brojevi označenih redeskripcija te se željene redeskripcije spremaju u bazu.

U primjeru su odabrane redeskripcije označene brojevima 12 i 13.




Slika 6.13: Spremanje redeskripcija u bazu

Nakon upisa u bazu, pojavljuje se skočni prozor s obavijesti o uspješnom spremanju te se po potvrdi obavijesti ažuriraju svi podaci i prikazi u aplikaciji.

Ponovno, promatra li se kartica *„Property - based exploration“* prije i nakon spremanja redeskripcija, može se primijetiti promjena u grafovima i ukupnom broju redeskripcija koje se nalaze u bazi.

# Bibliografija


[1] https://html.com, posjećena 31.12.2022.

[2] https://css.com, posjećena 31.12.2022.

[3] https://www.javascript.com, posjećena 31.12.2022.

[4] https://angularjs.org, posjećena 11.12.2022.

[5] https://nodejs.org, posjećena 11.12.2022.

[6] https://square.github.io/crossfilter/, posjećena 19.12.2022.

[7] https://spring.io/projects/spring-boot, posjećena 11.12.2022.

[8] https://www.jetbrains.com/idea/, posjećena 11.12.2022.

[9] https://spring.io, posjećena 19.12.2022.

[10] https://docs.oracle.com/javase/specs/jvms/se8/html/index.html, posjećena 19.12.2022.

[11] https://www.ibm.com/cloud/learn/java-spring-boot, posjećena 12.12.2022.

[12] Hendrik Blockeel, Luc De Raedt i Jan Ramon, *Top-Down Induction of Clustering Trees*, ICML '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998, ISBN 1558605568.

[13] Esther Galbrun i Pauli Miettinen, *Redescription Mining*, Springer International Publishing, Cham, 2017, ISBN 978-3-319-72889-6, https://doi.org/10.1007/978-3-319-72889-6_1.

[14] Dragi Kocev et al., *Tree ensembles for predicting structured outputs*, Pattern Recognition **46** (2013), br. 3, 817–833, ISSN 0031-3203, https://www.sciencedirect.com/science/article/pii/S003132031200430X.

[15] Teuvo Kohonen, *Self-Organized Formation of Topologically Correct Feature Maps*, Biological Cybernetics, 1982.

[16] Jelena Kurilić, *Razvoj web aplikacija pomoću razvojnog okvira Java Spring Boot*, (2021), https://urn.nsk.hr/urn:nbn:hr:217:408397.

[17] Matej Mihelčić, *Redescription Mining with Multi-target Predictive Clustering Trees*, Springer, Cham, 2015.

[18] Matej Mihelčić, Sašo Džeroski, Nada Lavrač i Tomislav Šmuc, *Redescription Mining with Multi-target Predictive Clustering Trees*, Springer International Publishing, Cham, 2016.

[19] Matej Mihelčić, Goran Šimić, Mirjana Babić Leko, Nada Lavrač, Sašo Džeroski i Tomislav Šmuc, *Using redescription mining to relate clinical and biological characteristics of cognitively impaired and Alzheimer's disease patients*, PLOS ONE **12** (2017), 1–35, https://doi.org/10.1371/journal.pone.0187364.

[20] Matej Mihelčić i Tomislav Šmuc, *InterSet: Interactive Redescription Set Exploration*, Springer International Publishing, Cham, 2016, ISBN 978-3-319-46307-0.

[21] _____, *Targeted and contextual redescription set exploration*, (2018), 1809–1846, https://doi.org/10.1007/s10994-018-5738-9.

# Sažetak

U ovom radu opisane su originalne verzije alata InterSet i algoritma CLUS-RM te interaktivni alat koji je nastao njihovim spajanjem. U prvom se poglavlju navode i definiraju osnovni pojmovi za lakše razumijevanje teme i načina rada korištenih alata. Nakon uvodnih definicija i objašnjenja, slijedi opis alata za pretraživanje skupova redeskripcija (InterSet) i algoritma za traženje redeskripcija (CLUS-RM). U daljnjim poglavljima opisuje se način implementacije interaktivnog alata. U zadnjem poglavlju prikazan je rad novonastalog alata popraćen primjerima i slikama. Kao rezultat rada konstruirano je efektivno interaktivno okruženje za traženje, pregledavanje i duboku analizu skupova redeskripcija.

# Summary

This thesis describes the original versions of the InterSet tool and the CLUS-RM algorithm, as well as the interactive tool that was created by combining them. In the first chapter, basic terms are listed and defined for easier understanding of the topic and working methods of the tools used. After introductory definitions and explanations, there is a description of the tool for interactive redescription set exploration (InterSet) and the algorithm for redescription creation (CLUS-RM). The following chapters describe how the interactive tool was implemented. The final chapter presents the funcionality of the newly created tool, followed by examples and illustrations. As a result of this thesis, an effective interactive environment for searching, browsing and deep analysis of sets of redescriptions was constructed.

# Životopis

Rođena sam 06. lipnja 1995. godine u Zagrebu gdje započinjem školovanje upisom u Osnovnu školu bana Josipa Jelačića. Sudjelujem u natjecanjima iz raznih predmeta te tako stječem prvi interes za matematiku. Nakon završetka osnovne škole, 2010. godine, upisujem opći smjer Gimnazije Lucijana Vranjanina. Obrazovanje nastavljam upisom preddiplomskog studija Matematike, 2014. godine, na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu. Titulu Sveučilišne prvostupnice matematike, univ. bacc. math., stječem 2020. godine i upisujem studij Računarstva i matematike na istom fakultetu.