SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 1860

ALGORITAM KORJENOVANJA RIJEČI TEMELJEN NA GRUPIRANJU ZNAKOVNIH NIZOVA

Zlatan Hot

Zagreb, srpanj 2010.

Sadržaj

1. Uvod

Perfomanse računalnih sustava za pretraživanje informacija i klasifikaciju teksta moguće je značajno poboljšati sažimanjem različitih morfoloških varijanti riječi na jedan jedinstveni oblik. U području obrade prirodnog jezika (engl. natural language processing – NLP) taj se postupak naziva morfološkom normalizacijom.

1.1 Problem

U morfološki složenim jezicima ista se riječ zbog fleksija, prefiksa, sufiksa, nastavaka[1] ili složenica pojavljuje u različitim oblicima. Jedan od bitnih problema u računalnoj lingvistici je razviti algoritam za nenadzirano strojno učenje koji bi prepoznavao sve varijante iste riječi. Različiti pokušaji realizacije u primjeni su u pretraživanju informacija (engl. information retrieval – IR), dubinskoj analizi teksta (engl. data mining – DM), raspoznavanju govora (engl. automated speech recognition – ASR) i strojnom prevođenju (engl. statistical machine translation – SMT) [1].

Alternativno, moguće je izgraditi alate temeljene na predefiniranim gramatičkim pravilima. Takvi alati postoje za neke jezike i oni morfološku analizu obavljaju prilično dobro, ali od velikog broja jezika u svijetu relativno malo ima ih na raspolaganju [13]. Među njima su: engleski, arapski, nizozemski, francuski, grčki, mađarski, portugalski, latinski, malezijski, ruski, ukrajinski, slovenski, poljski, španjolski i švedski. Razlog tako malom broju podržanih jezika je jako skupa cijena izrade morfološkog rječnika, odnosno rad lingivističkih stručnjaka na popisivanju pravila i prilagođavanju postupka morfološke analize riječi. Jeftiniji je pristup strojno učiti postupak analize na kolekciji teksta, primjerice nekoliko izdanja novina. Čak i za one jezike koji već imaju postojeće alate za analizu, statističke metode strojnog učenja i dalje predstavljaju zanimljivu i dobru alternativu [2].

1.2 Motivacija

Pojam pretraživanja informacija zahtijeva dodatno pojašnjenje. Ne tako davno, prije popularizacije pristupa Internetu, ljudi bi odlazili u gradsku knjižnicu potražiti kakvu knjigu o onome što im treba ili ih interesira. Često se sami ne bi uspjeli snaći čak i ako bi naišli na pravi odjel, pa je trebalo čekati dežurnog knjižnjičara da se vrati do info pulta i priupitati ga za naslov. Tu se dosta nade polagalo u njihovo poznavanje skoro svake od tisuća knjiga, a često bi bilo dvojbi oko točnog naziva djela[2]. Ako ni to ne bi pomoglo, preostajalo je ručno pretraživanje kartica sortiranih u ladicama po prezimenu autora. Dakako, ta zadnja opcija rijetko je davala traženi rezultat. Kako je rastao broj knjiga – s njima i broj knjižnica i knjižničara – postalo je nemoguće da jedna osoba ili čak više knjižičara u svojim odjelima pamte toliko puno informacija.

Razvojem računarstva problem indeksiranja velikog broja knjiga skoro se u potpunosti automatizirao i računala su postala moćniji alat u potrazi za informacijama od starije osobe s velikim naočalama koja povremeno utišava žamor u knjižnici.

[1] U nastavku će se ponekad koristiti računalni termini prefiks i sufiks u drugačijem smislu. Za razliku od gramatičkih, prefiks odnosno sufiks u računarstvu označava početni odnosno završni niz znakova u riječi.

[2] Poseban je problem predstavljala potraga za lektirom u osnovnoj školi. *Kako se ono zvala zbirka pjesama, možda "Cvjetovi zla"? Ma, ne, to nema previše smisla…*

Najranija pronađena arhiva pisane informacije datira oko 3000. pne, kada su Sumerani pohranjivali glinene pločice s klinastim natpisima na posebno predviđenim mjestima [3].

I stari Sumerani bili su svjesni da je pravilna organizacija i pristup arhivi od ključne važnosti za učinkovito korištenje informacija. Zato su razvili jednostavnu klasifikaciju za prepoznavanje pločica i njihovog sadržaja.

Računarstvo je promijenilo način na koji pohranjujemo, pretražujemo i dohvaćamo tekst. Broj pojmova za pretragu indeksa porastao je s tri (naslov, autor i kategorija) na gotovo beskonačan broj (čak i približno točno napisanih) pojmova koji se mogu dodati u upit tražilice.

Ranije spomenute discipline umjetne inteligencije iz poglavlja 1.1 potrebno je dodatno pojasniti. *Pretraživanje informacija* je disciplina koja se bavi traženjem dokumenata, informacijama i metapodacima unutar dokumenata te za pretraživanje relacijskih baza podataka i Interneta.

Ona je interdisciplinarna: temelji se na računalnim znanostima, matematici, informatici, arhitekturi informacija, kognitivnoj psihologiji, lingvistici i statistici [9].

Automatizirani sustavi za pretraživanje informacija koriste se za smanjenje onoga što se naziva “previše informacija” (engl. *information overload*). Među hrpom materijala na površinu isplivaju samo najrelevantniji dokumenti. Mnoga sveučilišta i javne knjižnice koriste IR sustave za pristup knjigama, časopisima i drugim dokumentima. Možda najpoznatija primjena su web – tražilice (Google, Bling, Yahoo).

Svake godine dodjeljuju se dvije nagrade za izvanredan doprinos na tom području: Tony Kent Strix award [16] i Gerard Salton Award [15]. Prvu izdaje britansko tijelo za razvoj elektroničkih informacijskih resursa UK eInformation Group, a drugu SIGIR (specijalna interesna grupa za područje pretraživanja informacija) kao članica ACM-a (engl. Association for Computing Machinery). Nagrade se dodjeljuje pojedincima koji su napravili “značajan, održiv i kontinuirani doprinos u vidu znanstvenog rada na području pretraživanja informacija”.

Dubinska analiza teksta je proces ekstrakcije uzoraka iz nekog skupa podataka. Ono postaje sve važniji alat za pretvaranje tih podataka u korisne informacije. Najčešća mu je primjena u ciljanom marketingu, detekciji plagijata, bioinformatici, genetici, medicini, obrazovanju i računalnom nadzoru.

1.3 Varijacije riječi

Arampatzis et al. [17] odredili su sljedeću podjelu varijacija riječi u tri kategorije:

(1) Morfološka varijacija vezana uz internu strukturu riječi.

> Primjerice, “veza” , “vezati”, “vezan”, “povezivati”, svode se na zajednički oblik “veza” koji konceptualno objedinjuje sve navedene izraze.

(2) Leksičko-semantičke varijacije vezane uz semantičku blizinu riječi, tako da različiti izrazi imaju isto značenje i obratno, jedan izraz može imati više različitih značenja. Uključuje homonime, antonime, homografe i homofone.

> “računalni problem” ili “kompjuterski problem”
>
> kartaški: koji se odnosi na Kartagu, koji se odnosi na kartanje
>
> Dúga je bila dúga.

(3) Sintaktičke varijacije vezane uz strukturu izraza složenih od više riječi, gdje se alternativne strukture svode na kanonski oblik.

> “uzimajući u obzir neke jezične varijacije” i “uzimajući u obzir ove jezične varijacije” svodi se na jedan oblik: “uzimajući u obzir jezične varijacije” [17].

Kako bi se riješio problem jezične varijacije, sustavi za pretraživanje i analizu primjenjuju neki od oblika normalizacije odnosno svođenja različitih oblika jedne riječi na jedan jedinstven oblik. Razlikujemo dva pristupa rješavanju problema: lingvistički i nelingvistički [5].

Lingvistički uključuje algoritme obrade prirodnog jezika (engl. Natural Language Processing – NLP) i pronalazi korijen riječi. Nelingvistički pristup se ne oslanja na gramatiku jezika i pronalazi morfološku normu koja ne mora odgovarati korijenu (*pseudokorijen*) [13].

U ovom radu fokusiramo se na prvu kategoriju (morfološka varijacija vezana uz internu strukturu riječi) i nelingvistički pristup (gramatika jezika nije uključena u postupak).

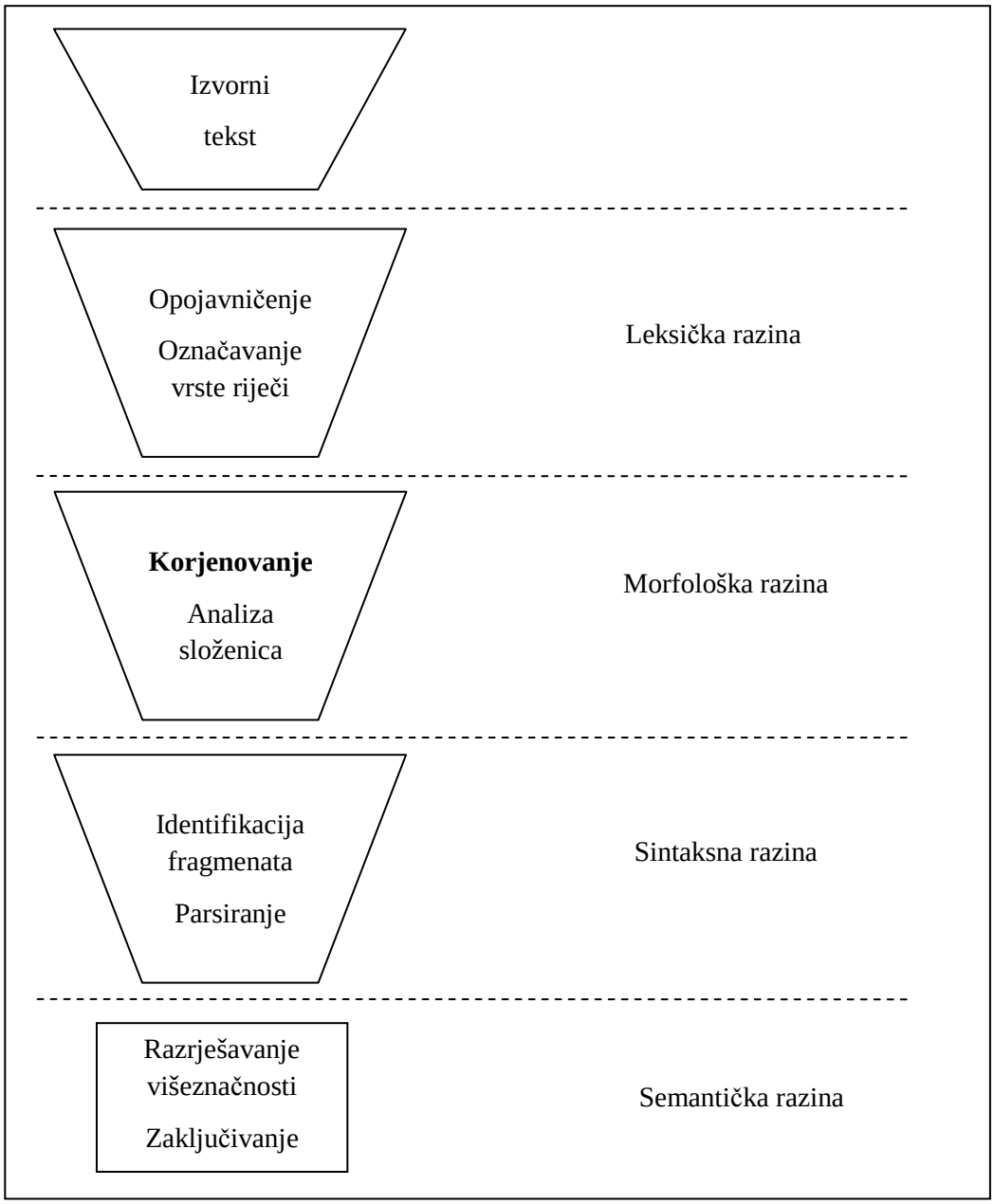
1.4 Definicija korjenovanja

Korjenovanje je postupak reduciranja flektivnih i deriviranih riječi na njihov korijenski oblik.

Fleksija se odnosi na oblike riječi koje izražavaju gramatičke značajke, primjerice: rod, broj ili padež. Kad govorimo o derivaciji, onda mislimo na tvorbu novih riječi iz postojećih, primjerice: industrija – industrijski. Pri tome se mijenja značenje i, naravno, oblik riječi. Međutim, polazna riječ i njena izvedenica su i dalje donekle semantički srodne.

Korijenski oblik koji se dobiva korjenovanjem ne mora odgovarati morfološkom obliku riječi definiranom u gramatici jezika. Za potrebe računarstva, dovoljno je da se srodne riječi reduciraju na isti zajednički oblik (*pseudokorijen*) [5].

Korjenovanje se u procesu obrade teksta nalazi na morfološkoj razini. Na slici 1.6 dan je pregled svih faza [7]:



Slika 1.1: Uloga korjenovanja u procesu obrade teksta

1.5 Pristup

Temeljna je ideja uzeti skup riječi i bez ikakvog predznanja o jeziku ekstrahirati moguća pravila pretvorbe riječi.

Postupak se provodi u tri koraka. Prvo se pronalaze ortografski slične riječi. Koriste se dva algoritma: *prefiksni* D_4 i neizraziti (engl. *fuzzy*) Ratclif-Obershelp (njem. *gestalt* pristup: gleda ukupnu sliku).

Drugi korak je pronalaženje grupa riječi. Riječi koje su u dovoljnoj mjeri slične reprezentiraju se točkama na grafu i te točke se povezuju. Provodi se algoritam iz teorija grafova koji grupira riječi u klase.

Zadnji je korak izvođenje pravila. Promatraju se transformacije riječi unutar klase i po čemu se razlikuju. Ukoliko se ustanovi da se neko izvedeno pravilo ponavlja u drugim klasama, to pravilo dobiva na jačini. Na kraju se uzimaju u obzir samo najjača pravila.

1.6 Cilj

Kako se morfološkom normalizacijom poboljšavaju perfomanse? Za početak, definirajmo dvije mjere kojima se vrednuje sustav za pretraživanje informacija: preciznost (eng. precision) P i odziv (eng. recall) R:

$$P = \frac{ukupan\ broj\ dohvaćenih\ \boldsymbol{relevantnih}\ dokumenata}{ukupan\ broj\ \boldsymbol{dohvaćenih}\ dokumenta}, \tag{1.1}$$

$$R = \frac{ukupan\ broj\ dohvaćenih\ \boldsymbol{relevantnih}\ dokumenata}{ukupan\ broj\ \boldsymbol{relevantnih}\ dokumenta}. \tag{1.2}$$

Preciznost i odziv međusobno su suprotstavljeni: preciznost pada s brojem dohvaćenih dokumenata, dok odziv raste [9].

Ako pretražujemo po upitu “prometna nesreća”, a sustav ne prepoznaje morfološke varijante riječi iz upita, dobit ćemo samo one rezultate u kojima se pojavljuju navedene riječi u tom istom obliku. Odziv bi bio slab jer bi dohvatio samo mali broj relevantnih dokumenata. Preciznost ne bi nužno bila visoka, jer se ta fraza usputno spominje i u nerelevantnim dokumentima.

Normalizacijom bi se pojavili i rezultati koji sadrže “prometne nesreće”, “u prometnoj nesreći”, “prometnim nesrećama” i time bi se značajno povećao odziv, a time i ukupne perfomanse sustava.

1.7 Struktura rada

U nastavku slijedi prezentiranje pojedinih metoda korjenovanja uz kratki opis njihovih prednosti i nedostataka. Pojašnjen je izbor nenadziranog strojnog učenja.

U poglavlju 3 je opisan postupak, a u poglavlju 4 njegova implementacija. Nakon toga u poglavlju 5 provedeno je eksperimentalno vrednovanje. Na kraju rada dan je zaključak u poglavlju 6.

2. Pregled područja

Prvi rad na temu korjenovanja u računarstvu (*Lovinsov algoritam*) objavljen je 1968. Od tada je razvijeno i u upotrebi više pristupa tom problemu (rječnička lematizacija, odbacivanje afikasa, stohastički algoritmi), a oni se primjenjuju u mnogim tražilicama za proširivanje upita ili indeksiranje, kao i za druge probleme obrade prirodnog jezika.

Metode korjenovanja možemo podijeliti u sljedeće kategorije [5,7]:

1. Pretraživanje tablice. U tablici raspršenog adresiranja su za svaki korijen spremljene mogući flektivni oblici. Problem predstavljaju nestandardne riječi i veliki zahtijevi za skladištenjem podataka.

2. *Korjenovanje odbacivanjem*. Zadržava se prvih *k* slova unutar riječi, gdje je *k* prikladan broj. Riječ s manje od *k* slova se ne dira.

3. *Korjenovanje temeljeno na pravilima* ili *eliminacija afikasa*. Uklanja se niz znakova koji se podudara se pravilom i eventualno se popravlja rezultat. Među prvim algoritmima za engleski jezik ističe se *Lovinsov algoritam* (1968.), zatim *Porterov algoritam* (1980.) i *Paice-Huskov algoritam* (1990.). Sva tri su rađena za engleski jezik, a cilj im je dobiti korijen iterativnim odsijecanjem najduljeg afiksa koji se podudara s jednim od afikasa iz pripremljenog popisa.

U zadnje vrijeme razvija se *Snowball*[2] modifikacija Porterovog algoritma (2001.) koja primjenjuje pravila specifična za pojedini jezik. Do sada su razvijene modifikacije za francuski, španjolski, talijanski, portugalski, njemački, norveški i švedski [5].

4. *Rječnička lematizacija*. Podrazumijeva korištenje ručno sastavljenih morfoloških leksikona. Njima se mogu rješavati problemi homografije i morfosintaktičkog označavanja. S druge strane, izgradnja leksikona iziskuje veliko leksičko znanje i ljudski napor, a primjenom su ograničeni samo na flektivne varijacije.

Prvi problem djelomično se rješava postupcima poluautomatske akvizicije leksikona iz korpusa.

Na temelju morfoloških *pravila produkcije* i statističkih informacija iz korpusa pokušavaju generirati tablicu korijenskih i izvedenih oblika. Primjerice, algoritam može za pronađeni glagol *trčati* iz korpusa povezati oblik za treće lice jednine *trči* i glagolsku imenicu *trčanje*.

5. *Hibridno korijenovanje*. Uz pravila odsijecanja afikasa dodatno se koristi riječnik kako bi se ublažila pogreška potkorijenovanja.

6. *Strojno učena lematizacija*. Nadzirano strojno učenje se koristi za automatsku indukciju lematizacijskih pravila iz postojećih morfoloških leksikona.

7. *Statistički pristupi korjenovanju* ili *indukcija morfologije*. Stohastički algoritmi koriste se teorijom vjerojatnosti da bi odredili korijen riječi. Prethodno ih je potrebno trenirati na poznatim parovima korijen – izvedenica kako bi "naučili" i "razvili" svoj model vjerojatnosti.

[3] Više o projektu može se saznati na http://snowball.tartarus.org

8. *Korjenovanje temeljem sličnosti nizova znakova*. Metoda nenadziranog strojonog učenja kojom se rječnici ili pravila korjenovanja induciraju automatski iz neoznačenih korpusa. Prednost ovog postupka je što je u načelu jezično neovisan i ne zahtijeva upotrebu prethodno izrađenog morfološkog leksikona. Na temelju *mjere sličnosti* nizova znakova morfološki povezane riječi grupiraju se u razrede ekvivalencije. Iz njih je zatim moguće inducirati pravila korjenovanja.

Poseban slučaj je analiza n-grama, koja se temelji na proučavanju nekoliko susjednih slova ili riječi te na temelju njihove asocijacije određuje odgovarajući korijenski oblik. Primjerice, metoda ispravno prepoznaje sintagmu "ministar vanjskih poslova", grupirajući sve 3 riječi na temelju prethodno ustanovljene asocijacije.

Nadalje, metode za nenadzirani pristup se prema očekivanom rezultatu mogu podijeliti na:

- identifikacija morfološki povezanih riječi (grupiranje),
- raščlamba riječi na morfeme (segmentacija) i
- identifikacija morfema (morfološka analiza).

Pojedina natjecanja iz područja obrade prirodnog jezika orijentirana su na specifičan problem. *Morpho Challenge Competition* [6] pokrenut je 2005. i održava se svake godine, a cilj mu je dizajnirati statistčki algoritam za strojno učenje koji otkriva od kojih se morfema (najmanjih mogućih cjelina u jeziku, a koje u sebi nose značenje) pojedine riječi sastoje.

Znanstveni ciljevi natjecanja su proučiti osnovni fenomen tvorbe riječi u prirodnim jezicima, otkriti pristup koji bi bio prikladan za primjenu na što veći skup jezika te unaprijediti metodologiju strojnog učenja.

Problem određivanja morfološki povezanih riječi najčešće se rješava kroz nenadzirani pristup ili primjenom ulaznog skupa podataka (rječnika).

Česta je ideja iza mnogih metoda ta da se riječi slažu spajanjem morfema (engl. *item–and–arrangement*), međutim takva konkatenacija ne uzima u obzir glasovne promjene na granicama morfema i time nije podobna za pojedine jezike [18].

Primjerice, prijevos i prijeglas (njem. ablaut i umlaut) u njemačkom: samostan Kloster (sg.) Klöster (pl.), odnosno promjena samoglasnika usred riječi (eng. sing, sang, sung); infiksacija (promjena afiksa koji se nalazi unutar korijena) te ekspletivna infiksacija [19] (koristi se radi pojačavanja govornikovog negativnog stava, primjerice: eng. –bloody- u riječi absobloodylutely).

U ovom radu korištena je Ratclif-Obershelp metoda [11] koja gleda cijele riječi, a ne pojedine morfeme. Nadalje, veze su ostvarene na način koji omogućava primjenu metoda za grupiranje iz teorije mreža, gdje zajednice predstavljaju grupe morfološki povezanih riječi.

3. Korjenovanje primjenom algoritma grupiranja

Problem korjenovanje riješava se u tri koraka. Prvi korak je povezivanje sličnih riječi. Idući je grupiranje riječi u kategorije. Zadnji je izvođenje pravila unutar kategorija.

1) U početnoj fazi potrebno je u ulaznom skupu pronaći dovoljno slične riječi. Postavlja se prag sličnosti, kako bi se izbjeglo povezivanje riječi koje su djelomično nalikuju jedna drugoj. Primjerice, par "igrači" i "igra" prijeći će prag, dok "izleti" i "izdržati" neće. Par "pravo" i "prvo" povezat će se ovisno o korištenoj metodi i izboru parametara.

Problemu se pristupa na dva načina. Prvi je algoritam mjere udaljenosti nizova D_4, koji je predložen u [10]. On je orijentiran na zajednički prefiks, odnosno povezuje riječi koje počinju istim nizom slova. Ako riječi ne počinju istim nizom znakova, rezultat algoritma će sugerirati da su one potpuno različite. Stoga je D_4 preporučeno koristiti samo kod jezika koji su sufiksno orijentirani (kao što je hrvatski).

U primjeru usporedbe riječi *moramo* i *moraju* na slici 1.1 zajednički prefiks je *mora-* (to je ujedno i korijen), dok je sufiks *–mo* odnosno *–ju*:

m	o	r	a	m	o
m	o	r	a	j	u

Slika 3.1: D_4 algoritam

Drugi je algoritam Ratclif-Obershelp [11]. On gleda „ukupni dojam" i nije orijentiran samo na znakovne nizove na početku ili kraju riječi. Time vjerno oponaša ljudsko opažanje jer sugerira da su riječi slične ako se razlikuju u relativno malom broju znakova. S druge strane, mnoge riječi koje se slično pišu ne dijele isti korijen. To će biti pokazano u sljedećem primjeru.

Usporedba riječi *početak* i *petak* vraća visoku ocjenu sličnosti na temelju dugačkog zajedničkog podniza *-etak* i zajedničkog slova *p* s krajnje lijeve strane, kao što je pokazano na slici 1.2:

p	o	č	e	t	a	k
p	e	t	a	k		

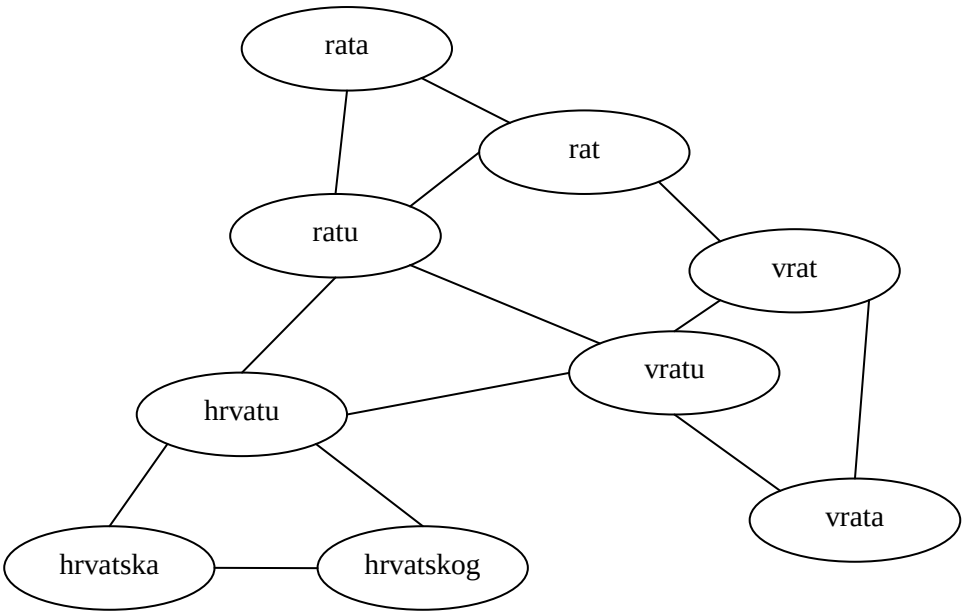
Slika 3.2: Ratclif-Obershelp algoritam

Iako primjer sugerira da je D4 bolji izbor jer zajednički početak[I] sugerira jaču vezu među riječima od ukupnog dojma, općenito to ne mora biti slučaj. Primjerice, D_4 ne bi prepoznao sličnost riječi *usporio* i *sporo*.

[I] Doista, u hrvatskom se većina riječi izvodi dodavanjem prefiksa ili promjenom sufiksa. Rijetke su promjene u sredini riječi, poput nepostojanog a ili promjene „ije" u „je".

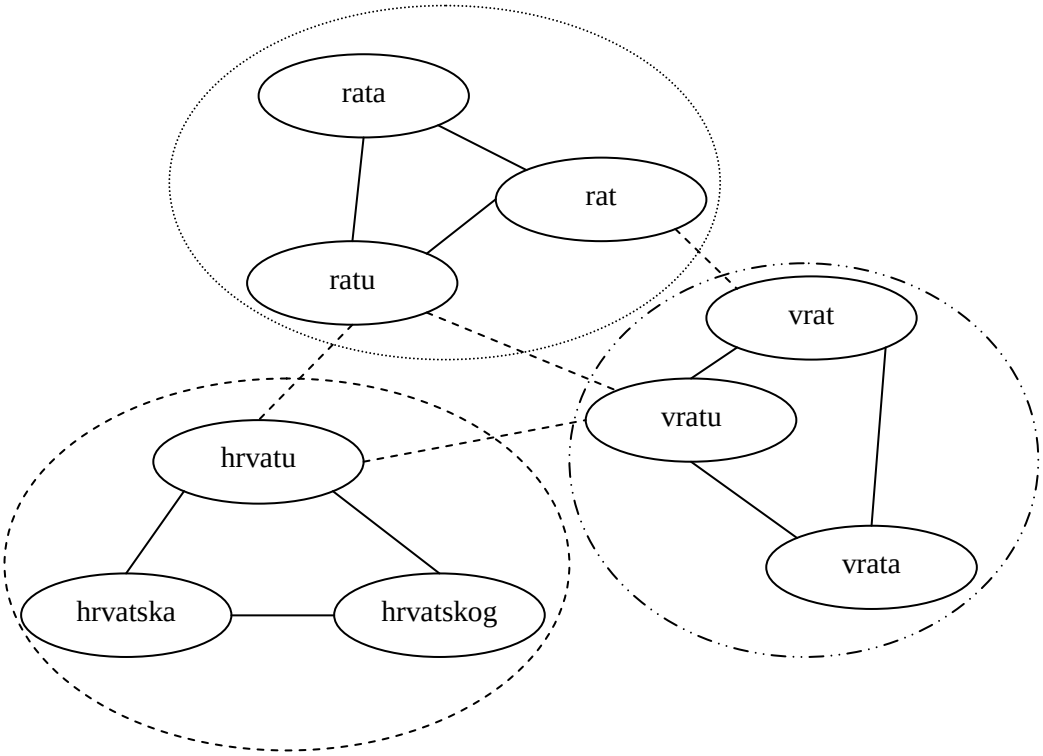
2) Sljedeći je korak grupirati riječi po sličnosti u iste kategorije ili grupe riječi. Idealno, u jednoj grupi nalazile bi se sve varijacije iste riječi iz teksta i ništa drugo. Jednostavno rečeno, svaka riječ se reprezentira točkom na grafu, a međusobno slične riječi su povezane točke. Korištenjem algoritma iz teorija grafova za detekciju zajednica, pojedine točke se grupiraju u klase.

Na slici 1.3 pokazano je početno stanje grafa i veze između riječi:



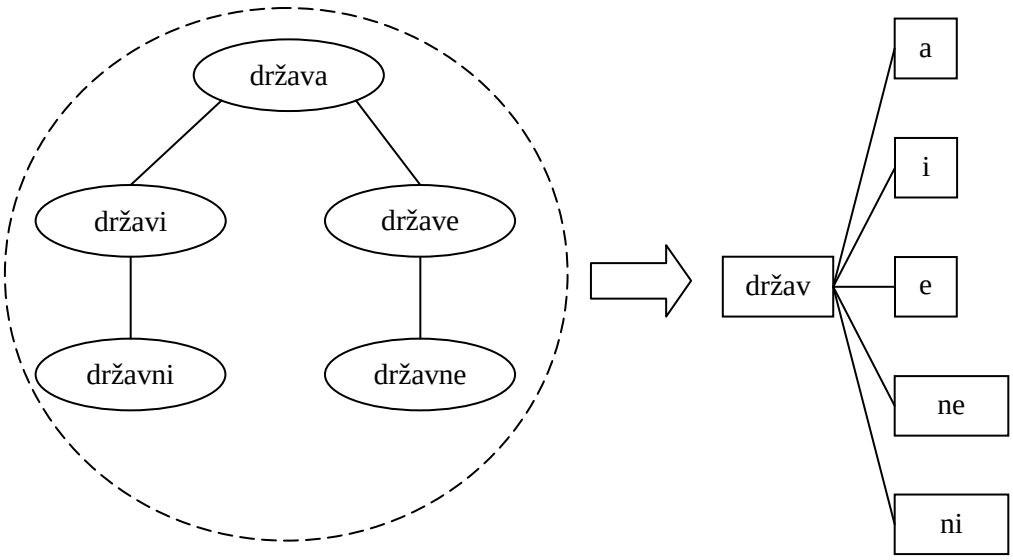
Slika 3.3: Početno stanje grafa

Algoritam prepoznaje zajednice na temelju broja susjednih veza: unutar grupe veze su “gušće”, dok su prema susjednim grupama “rjeđe”. Rezultat je otkrivanje triju grupa, kao što je pokazano na slici 1.4:



Slika 3.4: Grupiranje

3) Na kraju se na temelju veza unutar grupa riječi formiraju moguća pravila pretvorbe. Što se neko pravilo češće pojavljuje, to je vjerojatnije da se poklapa s gramatičkim pravilima jezika. Proces je ilustriran na slici 1.5:



Slika 3.5: Izvođenje pravila

Ulazni skup podataka je skup riječi ***L*** nad kojima se obavlja prvi korak. Za svaku riječ ***w*** iz zbirke riječi ***L*** traže se oblikom slične riječi koristeći prefiksni algoritam za mjeru udaljenosti D_4 [10] ili neizrazito određivanje sličnosti po Ratclif-Obershelpu [11].

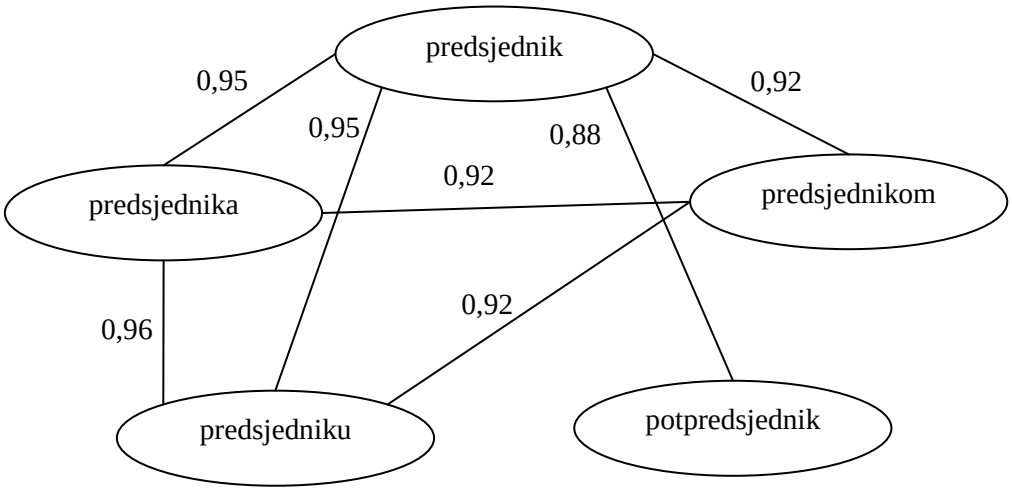
Ocjena sličnosti mora prelaziti zadani prag, koji se eksperimentalno fiksira na prikladnu vrijednost. Primjerice, za riječ *predsjednik* po algoritmu Ratclif-Obershelp uz prag 0,85 pronađene su slične riječi: *predsjednika*, *potpredsjednik*, *predsjednikom* i *predsjedniku*. Među sličnim riječima formiraju se veze s informacijom u kojoj mjeri su one slične.

Procedura za uspostavljanje veza između sličnih riječi iz zadanog skupa *L* (koji je u algoritmu koji slijedi prikazan podatkovnom strukturom liste) izgleda ovako:

```
1. links ={}
2. n = length(L)
3. for i = 1 to n {
4.  w = L[i]
5.  matches = get_string_distance(w, L, i)
6.  for each w2 in matches
7.  add_link(w, w2,links)
8. }
9. return links
```

U 3. retku izvršava se petlja koja prolazi kroz sve riječi iz ulazne liste *L***.** U 5. retku poziva se procedura `matches` koja za zadanu *i*-tu riječ pronalazi slične riječi iz liste *L*. Konačno, u 6. i 7. retku pravi se veza između pronađenih sličnih riječi.

Na temelju tih veza pravi se graf. Formalno, graf G je uređeni par (V, E) gdje je V skup vrhova, a E podskup od V×V je skup veza. Prednost grafa je da uzima u obzir višestruke veze između elemenata, tako da se jednostavno vizualiziraju rezultati dobiveni za jednu točku i njihov utjecaj na cijelu mrežu. Dobiveni graf sastoji se od točaka (riječi) povezanih vezama koje na sebi nose morfološku relaciju između te dvije riječi. Takva struktura zove se leksička mreža i prikazana je na slici 3.1:



Slika 3.6: Leksička mreža

Idući je problem određivanje grupa riječi, a to je zapravo problem otkrivanja zajednica u grafu. Taj se problem rješava korištenjem algoritma za grupiranje. Primjenjena je metoda koju je predložio Newman [2].

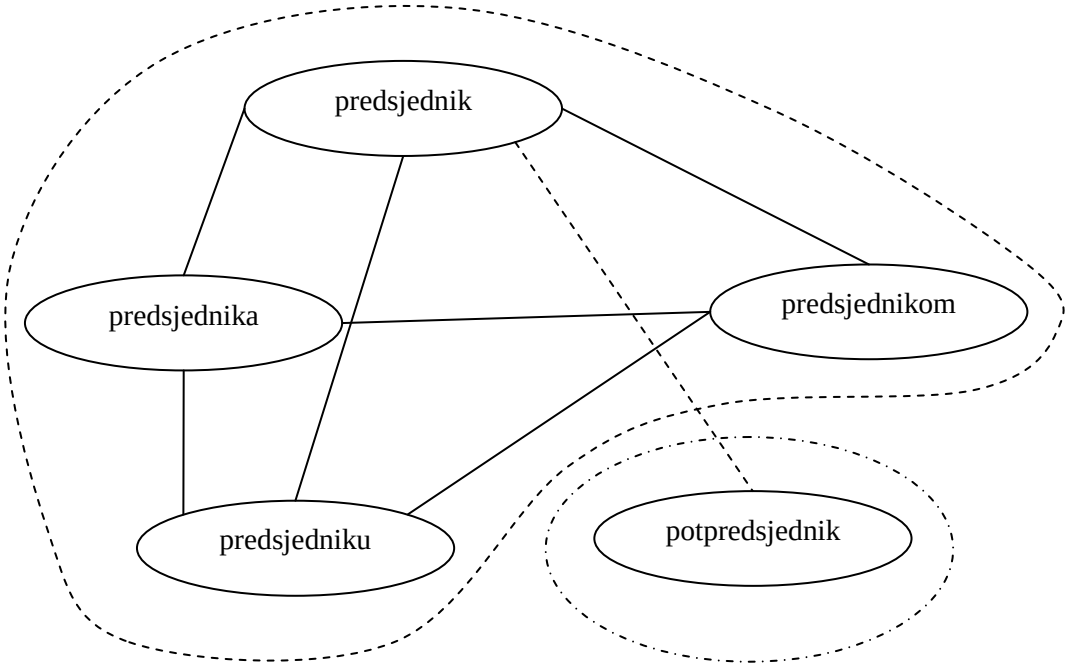
Uvodi se funkcija modularnosti *Q* koja opisuje kvalitetu povezanosti grupa u mreži. Što je *Q* veći, to su riječi unutar grupe bolje povezane. Tako se problem transformira u optimizaciju funkcije *Q*. Ona je definirana kao:

$$Q = \sum_i \left(e_{ii} - \left(\sum_j e_{ij} \right)^2 \right), \qquad (3.1)$$

gdje je e_{ii} udio rubova u mreži koji povezuju čvorove unutar zajednice *i*. e_{ij} je jedna polovica udijela rubova u mreži koji povezuju čvorove u zajednici *i* s onima u zajednici *j*. $\sum_j e_{ij}$ je udio rubova povezanih na čvorove u zajednici *i*.

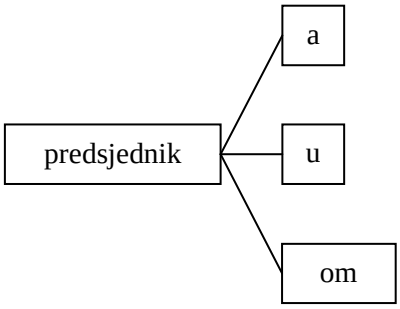
Promatraju se postojeće grupe i kako bi njihovo spajanje utjecalo na funkciju modularnosti te se odabire ono spajanje koje maksimizira *Q*. Postupak se ponavlja dok se ne dostigne maksimum.

To je prikazano na slici 3.2:



Slika 3.7: Otkrivanje zajednica

Koristi se *gestalt* pristup prilikom poređenja riječi iz iste grupe i dovoljno male razlike se preoblikuju u pravila. Primjerice, iz *predsjednika* i *predsjednik* izvodi se pravilo odbacivanja sufiksa *-a*. Radi izbjegavanja pogreški, zadržavaju se samo ona pravila koja se pojavljuju više puta. Ne nastoji se razlučiti infleksija od derivacije. Pravila su ilustrirana na slici 3.3:



Slika 3.8: Primjer pravila

3.1 Pronalažanje ortografski sličnih riječi

Za određivanje pravopisno sličnih riječi koriste se dva pristupa: D_4 i Ratclif-Obershelp algoritam.

Algoritam D_4 je odabran na temelju testiranja [8] provedenom nad korpusom na hrvatskom jeziku. Među četiri algoritma za mjerenje udaljenosti, on je postigao najbolje rezultate u smislu flektivne i derivacijske kvalitete korjenovanja.

D_4 favorizira riječi s dugačkim zajedničkim prefiksom. Neka je *m* pozicija prvog znaka s lijeve strane koji se ne poklapa u obje riječi.

U primjeru 3.4 *m* je jednak 4:

0	1	2	3	4	5
m	o	r	a	m	o
m	o	r	a	j	u

Slika 3.9: D_4 algoritam

Promatramo dva niza X = $x_0x_1 \ldots X_n$ i Y = $y_1\, y_0 \ldots y_{n'}$. Ako X i Y nisu jednake duljine, popunjavamo praznine na kraju kraće riječi kako bi oba niza bili jednake duljine. Neka je duljina riječi n + 1. Tada možemo definirati D_4 kao:

$$D_4(X,Y) = \frac{n-m+1}{n+1} \times \sum_{i=m}^{n} \frac{1}{2^{i-m}} \ . \tag{3.2}$$

Udaljenost D_4 dobiva se množenjem dvaju faktora. Prvi faktor se odnosi na relativnu duljinu sufiksa: to je omjer duljine sufiksa i ukupne duljine riječi. Drugi faktoj je broj u rasponu između 1 i 2. Što je prefiks dulji, to je vrijednost faktora bliža 1. Obratno, ako je prefiks kratak, tada se faktor približava 2. U obzir se uzima i ukupna duljini riječi. Primjer izračuna vrijednosti dan je na slici 3.5:

0	1	2	3	4	5
m	o	r	a	m	o
m	o	r	a	j	u

0	1	2	3	4	5	6	7	8
e	u	r	o	p	s	k	i	h
e	u	r	o	p	e			

$$D_4 = \frac{6-4}{6} \times \left(\frac{1}{2^{4-4}} + \frac{1}{2^{5-4}}\right) = 0{,}5 \qquad D_4 = \frac{9-5}{9} \times \left(\frac{1}{2^{5-5}} + \cdots + \frac{1}{2^{8-5}}\right) = 0{,}8\dot{3}$$

Slika 3.10: Primjer izračuna udaljenosti D_4

Drugi je algoritam Ratclif-Obershelp [11]. On je odabran kao alternativa uz očekivanje da će obuhvatiti više sličnih parova riječi od prefiksno orijentiranog algoritma. Oblikovan je tako da usporedba dva niza vraća decimalnu vrijednost sličnosti nizova. Rezultat od 0,80 znači da su dvije riječi 80% slične.

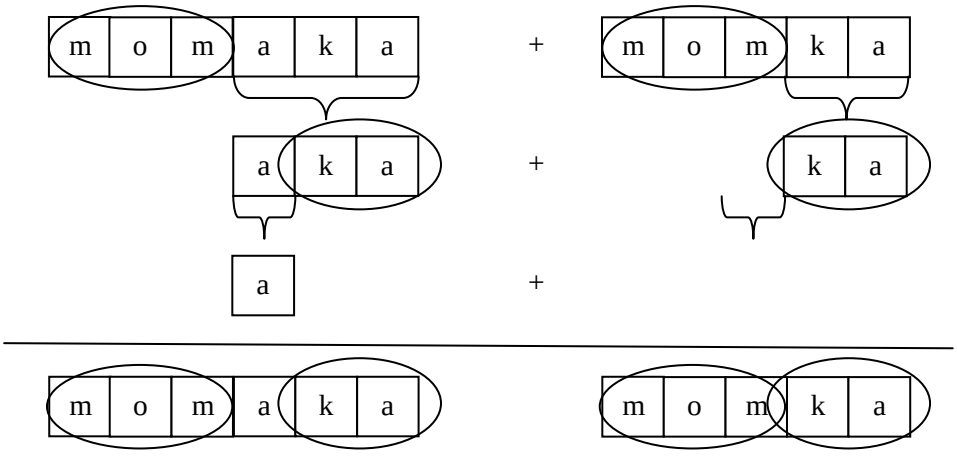
Neka je *k* ukupna duljina podudarajućih blokova, *len(X)* duljina riječi *X* te *len(Y)* duljina riječi *Y*. Tada se udaljenost računa kao:

$$RO(X,Y) = \frac{2 \times k}{len(X) + len(Y)} \ . \tag{3.2}$$

Najstarija primjena algoritma bila je ispravljanje pogrešno napisanih riječi. Na temelju poređenja s riječima iz riječnika, ona riječ s najvećim postotkom korigirala se kao pravopisno ispravna.

Osnovna je ideja pronaći najduži zajednički podniz. Ista ideja se zatim primjenjuje rekurzivno na dijelove lijevo i desno od pronađenog podniza. To rezultira povezivanjem riječi koje većini ljudi vizualno doista izgledaju slično. U nastavku su dana 3 primjera.

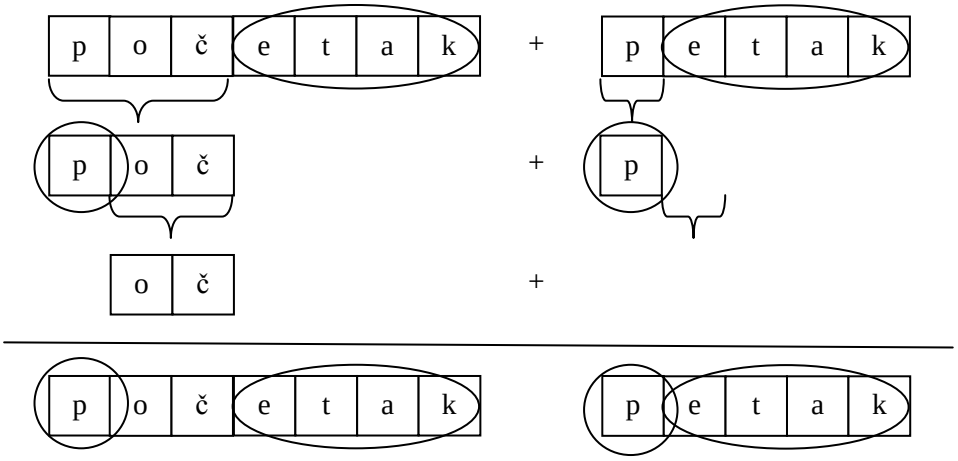
Slika 3.6 ilustrira usporedbu riječi “momaka” i “momka” po Ratclif-Obershelp algoritmu. Samo za napomenu, udaljenost D4 bila bi mala zbog relativno kratkog zajedničkog prefiksa “mom-”.



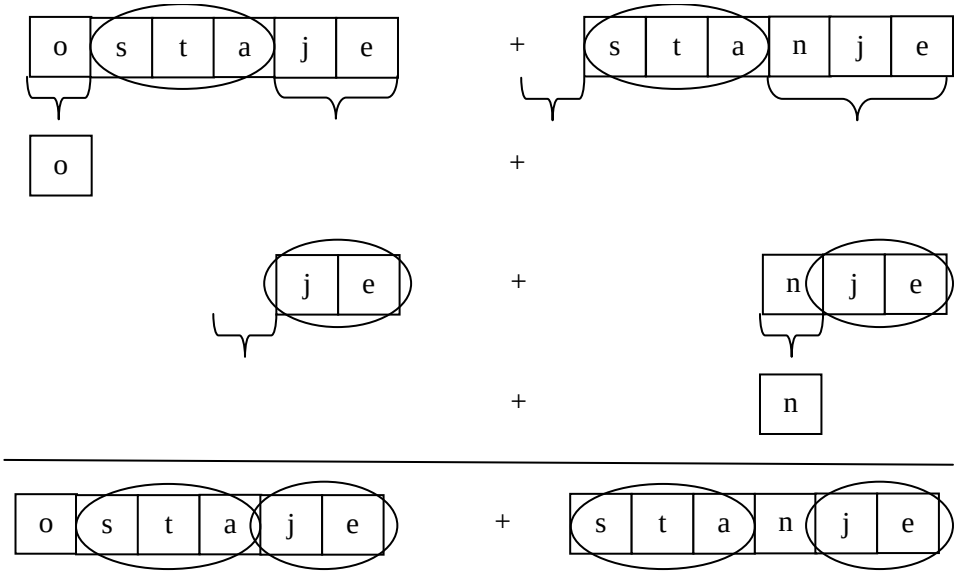
$$RO = \frac{2(3+2)}{6+5} = 0,\dot{9}\dot{0}$$

Slika 3.11: Primjer rada Ratclif-Obershelp algoritma

Na slikama 3.7 i 3.8 prikazan je izračun udaljenosti vizualno sličnih riječi “početak” i “petak”, odnosno “ostaje” i “stanje”. To je i glavni nedostatak metode, jer češće od D_4 povezuje riječi koje ne dijele isti korijen:



Slika 3.12: Primjer rada za par riječi “početak” i “petak”



Slika 3.13: Primjer rada Ratclif-Obershelp algoritma za par riječi “ostaje” i “stanje”

Procedura RO izgleda ovako:

```
1. int m = Length(a), n = Length(b), len;
2. if (m == 1) then if b[0] == a return 1 else return 0;
3. find_longest_match(a, b, i, j, len);
4. if (len == 0) return 0
5. return len + RO (a[0:i], b[0:j]) +
   RO(a[i+len:m],b[j+len,n]);
```

U pitanju je rekurzivan algoritam koji vraća broj zajedničkih slova. U 3. retku naveden je trivijalan slučaj rekurzije, kada se poredi samo jedno slovo s nizom znakova. Pronalaženje

najduljeg zajedničkog podniza obavlja funkcija `find_longest_match` u 4. retku. Njen izlaz su: *i* i *j* (pozicija zajedničkog podniza u nizu *a* odnosno *b*) te duljina podniza *len*.

Na kraju je rekurzivan poziv izračuna sume zajedničkih slova. Ona je jednaka pronađenoj duljini podniza uvećanoj za vrijednosti duljina koje će se pronaći s lijeve odnosno s desne strane podniza.

Da bi se dobila Ratclif-Obershelp udaljenost izražena kao decimalna vrijednost (odn. postotak sličnosti), rezultat procedure se dodatno dijeli sa aritmetičkom sredinom duljina ulaznih riječi.

3.2 Algoritam za grupiranje po Newmanu

Drugi korak je grupirati riječi na temelju veza iz grafa. Primjenjuje se Newmanova metoda zbog pristupa koji rezultira kraćim vremenom izvođenja u odnosu na postupke koji iterativno uklanjaju veze (primjerice, metoda po Girvanu koja vrednuje veze na „unutarnje“ i „vanjske“) [2]. Apriorno vrijeme izvođenja za optimiziranu realizaciju je $O(n^2)$.

Umjesto da se istraže sve moguće varijante grupiranja točaka u grafu, problemu se pristupa uvođenjem funkcije modularnosti *Q* koja određuje kvalitetu podjele grafa. Funkcija modularnosti poredi stvaran broj veza u zajednici s očekivanim. Dobra podjela odgovara većem broju veza unutar zajednica nego što bi se očekivalo nasumičnim izborom. Modularnost je visoka kada postoji velik broja veza unutar zajednice, a mali broj između njih.

Newmanova metoda se zasniva na optimizaciji funkcije *Q*. Prednost ove metode je da nije nužno unaprijed znati broj zajednica i nije potrebno precizno podešavanje parametara.

Postupak se izvodi u nekoliko koraka:

1. početno: svaka točka je sama u svojoj grupi,
2. za svake dvije grupe *i* i *j* određuje se promjena ΔQ_{ij} ako bi se spojile te dvije grupe,
3. odabire se najveći ΔQ_{ij} i obavlja se spajanje grupa *i* i *j*

(koraci 2 i 3 se ponavljaju sa ciljem optimizacije *Q*).

Dodatno, radi ubrzanja algoritma kreira se matrica udaljenosti ΔQ_{ij} za svaki par *i* i *j* zajednica sa barem jednom vezom između njih.

Ona se početno popunjava vrijednostima po sljedećoj formuli:

$$\Delta Q_{i,j} = 2\left(e_{ij} - a_i a_j\right), \tag{4.1}$$

gdje je e_{ij} udio svih veza između grupe *i* i *j*. a_i odnosno a_j je udio svih rubova veza koje završavaju u grupi *i* odnosno *j*.

Prilikom spajanja grupa *i* i *j*, elementi grupe *i* se dodaju grupi *j*. Zatim se matrica osvježava te se iz nje briše redak i stupac *i*.

Osvježavanje matrice radi se po sljedećem principu.

Neka je a_i udio vrhova vih veza koji završavaju u grupi *i*.

Ako je grupa *k* povezana i s *i* i s *j*, onda:

$$\Delta Q'_{\mathrm{jk}} = \Delta Q_{\mathrm{ik}} + \Delta Q_{\mathrm{jk}} . \tag{3.3}$$

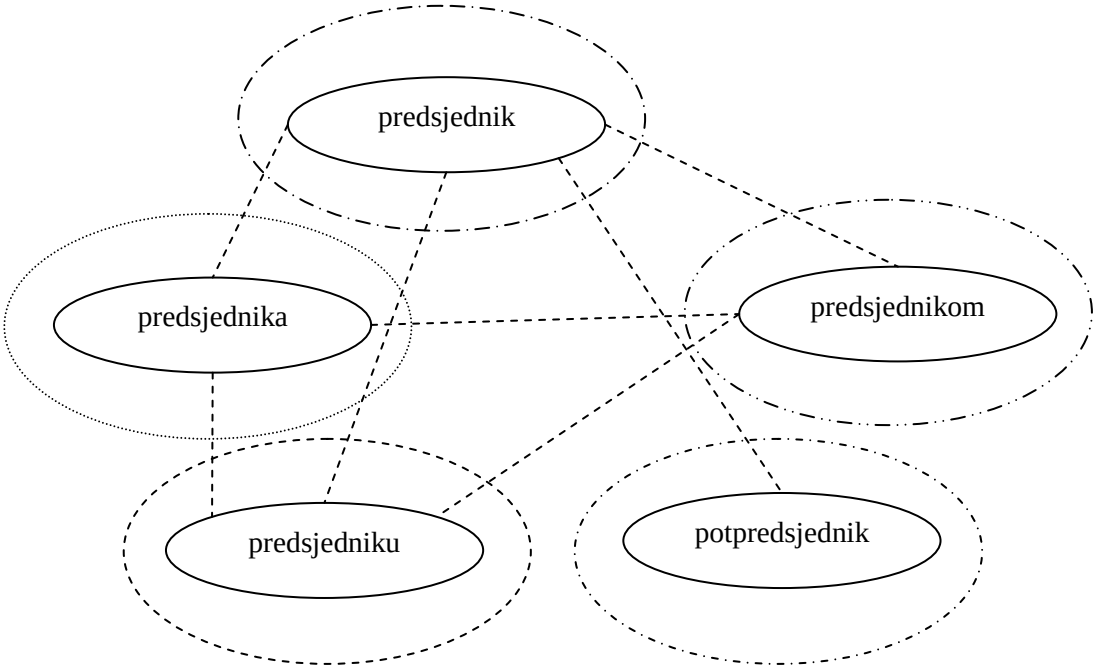
Ako je grupa *k* povezana s *i* ali ne s *j*, onda:

$$\Delta Q'_{\mathrm{jk}} = \Delta Q_{\mathrm{ik}} - 2a_{\mathrm{j}}a_{\mathrm{k}} . \tag{3.4}$$

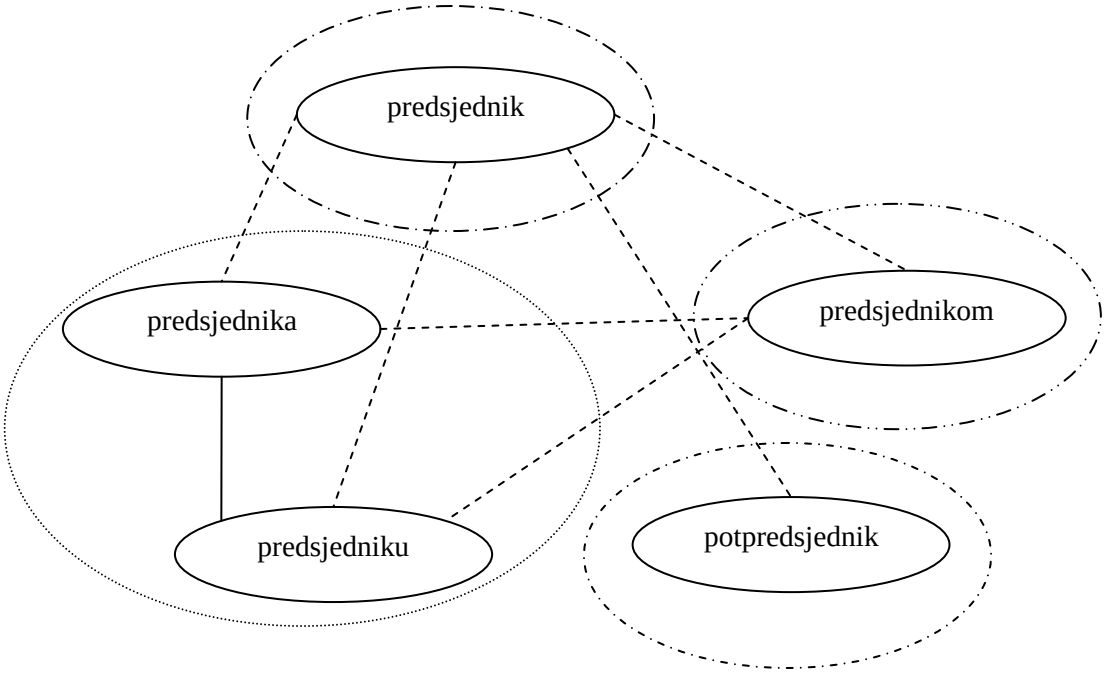
Ako je grupa *k* povezana s *j* ali ne s *i*, onda:

$$\Delta Q'_{\mathrm{jk}} = \Delta Q_{\mathrm{jk}} - 2a_{\mathrm{i}}a_{\mathrm{k}} . \tag{3.5}$$

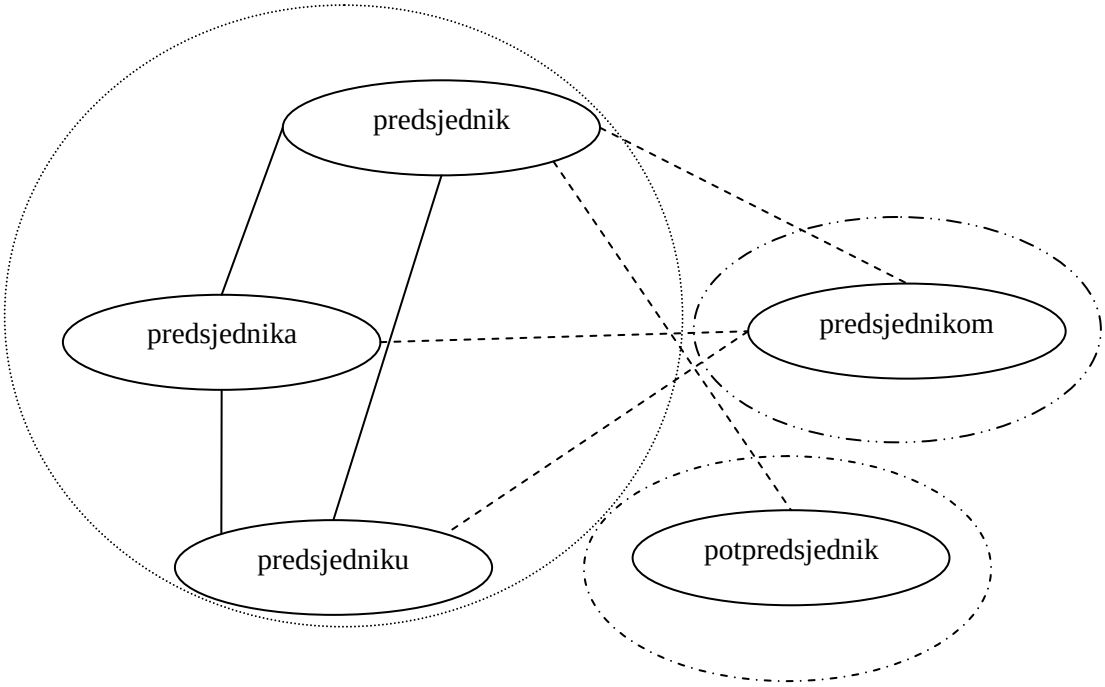
U nastavku slijedi primjer rada i detekcija dvije zajednice, koja se provodi kroz 4 koraka opisanog algoritma.



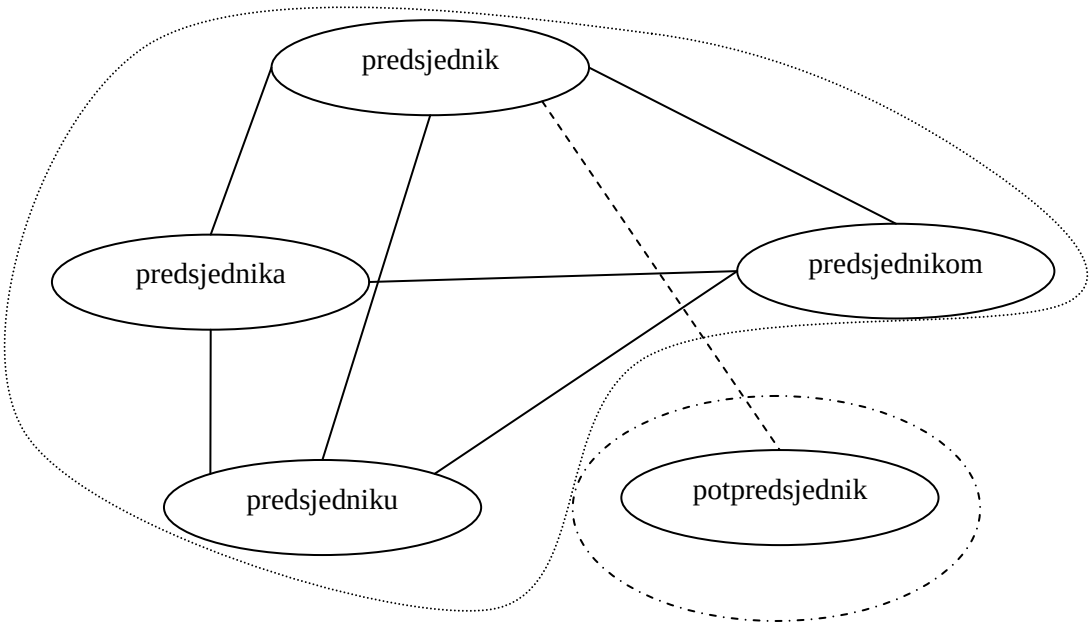
Slika 3.14: Prvi korak, svaka riječ je u svojoj grupi



Slika 3.15: Drugi korak, spojene su dvije grupe



Slika 3.16: Treći korak, nastavlja se spajanje grupa



Slika 3.17: Četvrti korak, konačni rezultat

Procedura za provođenje grupiranja riječi *words* u grupe *groups* na osnovu matrice udaljenosti *qMatrix* izgleda ovako:

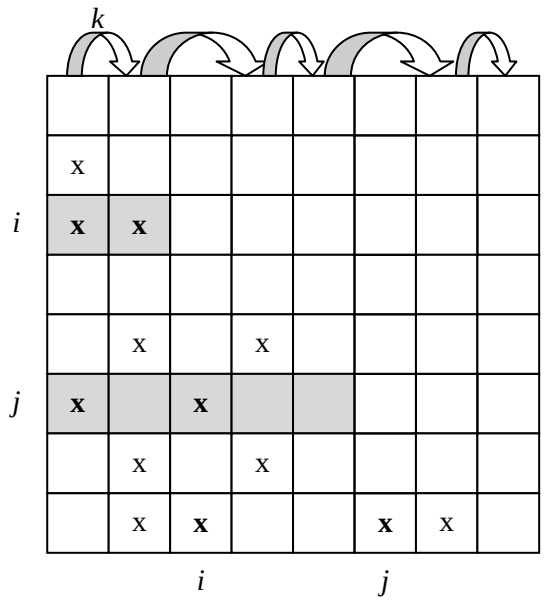
```
1. dQmax = Max(qMatrix);
2. while (dQmax > 0) {
3.  getIndex(dQmax, i, j);
4.  MergeGroups(i,j);
5.  Update_qMatrix(i,j);
6.  dQmax = Max(qMatrix);
7. }
```

U 2. retku je definirano da se procedura ponavlja dok god se ne postigne lokalni maksimum funkcije *Q*. To je ujedno i globalni maksimum, jer jednom kad u tablici ne postoji pozitivan član nemoguće je spajanjem grupa povećati *Q*. To je očito iz definiranih pravila za osvježavanje elemenata matrice $\Delta Q'_{jk}$.

Dohvaćaju se indeks grupa *i* i *j* čije spajanje rezultira maksimalnim porastom funkcije modularnosti. U 4. koraku radi se spajanje grupa *i* i *j* tako da se elementi grupe *i* dodaju grupi *j* te se zatim grupa i briše iz popisa grupa *groups*.

U 5. koraku osvježava se matrica udaljenosti po ranije defeniranim pravilima, a 6. se određuje idući najveći element matrice.

Na slici 3.13 dan je vizualna predodžba postupka:



Matrica udaljenosti je simetrična,ali radi jednostavnosti primjera prikazana je kao donja trokutasta.

Slika 3.18: Primjer postupka grupiranja

U gornjem primjeru, 0.-ti stupac je povezan i s *i* i s *j*, pa je onda:

$$\Delta Q'_{\mathrm{j0}} = \Delta Q_{\mathrm{i0}} + \Delta Q_{\mathrm{j0}} .$$

1. stupac je povezan s *i* ali ne s *j*, pa je:

$$\Delta Q'_{\mathrm{j1}} = \Delta Q_{\mathrm{i1}} - 2a_{\mathrm{j}}a_1 .$$

2. stupac je povezan s *j* ali ne s *i*, pa onda:

$$\Delta Q'_{\mathrm{j2}} = \Delta Q_{\mathrm{j2}} - 2a_{\mathrm{i}}a_2 .$$

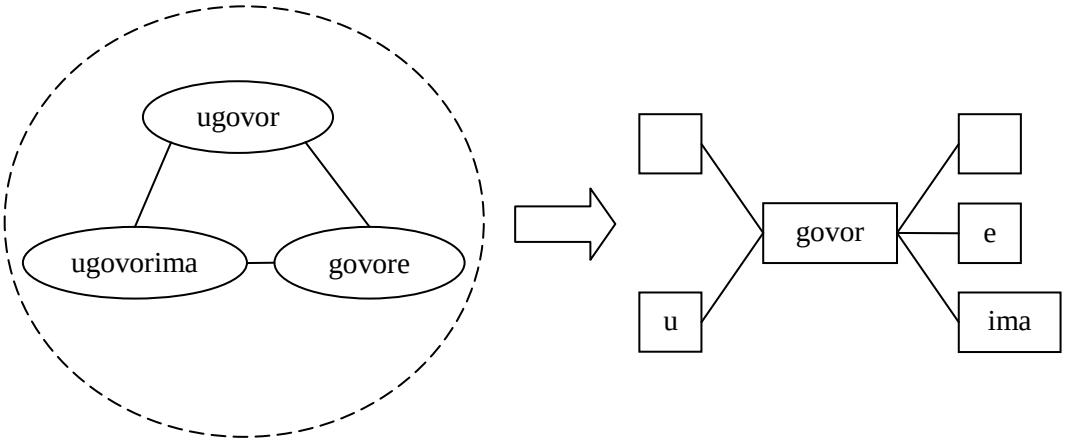
I tako dalje za sve stupce matrice.

Idealno, po završetku postupka svi flektivni i derivacijski oblici iste riječi nalaze se u istoj grupi. Različite grupe predstavljaju različite riječi.

3.3 Izvođenje pravila

Jednom kad su morfološki slične riječi sortirane u grupe, može se pristupiti posljednjem, trećem koraku. Unutar grupa proučavaju se razlike među riječima, te se na osnovu njih predlažu pravila. Pri tome je moguće izvesti i pogrešno pravilo, pa je zato bitno uočiti koliko često se novo potencijalno pravilo primjenjuje i u ostalim grupama.

Promotrimo sljedeći primjer na slici 3.14:



Slika 3.19: Primjer izvođenja pravila

Iz grupe riječi u kojima se nalaze “ugovorima”, “ugovor” i “govore” izvode se 3 potencijalna pravila. Prvo definira da se može odbaciti sufiks “-ima”. Drugo da se smije odbaciti prefiks “u-” uz dodavanje sufiksa “-e”. Treće definira da ako riječ počinje sa prefiksom “u-” a završava sufiksom “-ima”, onda se oboje može odbaciti uz dodavanje sufiksa “-e”. Može se primijetiti da je u sva tri pravila korijen riječi “govor”, međutim to je pretpostavka koja se na osnovu tri varijacije ne može potvrditi (bez poznavanja gramatike).

Potencijalno pravilo koje kaže da se može odbaciti prefiks *u-* (izvedeno is *ugovor* i *govor*) ne pojavljuje se često u ostalim grupama te se odbacuje kao nepouzdano.

Koristi se sljedeća procedura za formiranje potencijalnih pravila iz para riječi *a* i *b* unutar iste grupe:

```
1. int i = 0, string[] m = get_matching_blocks(a, b);
2. for each string s in m {
3.    replace_string(a,s,”(.+)”);
4.    i++;
5.    replace_string(b,s,”\\”+(string) i);
6. return "^" + a + "$->" + b;
```

Vektor *m* se u 1. retku procedure popunjava zajedničkim podnizovima. Zatim se u retcima 2 – 5 zamjenjuje podniz u riječi *a* sa stringom (,+), dok se u riječi *b* on zamjenjuje s oblikom \# gdje # označava redni broj zajedničkog bloka.

Primjerice, pretvorba riječi “vlada” u “vlade” definira se sljedećim pravilom: `^(.+)e$->\1a`, gdje se regularni izraz s lijeve strane zamjenjuje s desnom stranom uzimajući u obzir položaj zajedničke grupe znakova. Pretvorba “momaka” u “momka” definira se pravilom: `^(.+)a(.+)$->\1\2`.[4]

4 Ovo je pravilo dosta nezgodno za primjenu, jer opisuje ono što se u gramatici hrvatskog jezika naziva nepostojano a. Za njegovu ispravnu primjenu bilo bi nužno poznavati gramatiku jezika na koji se primjenjuje.

4. Programska izvedba

Program je u cjelosti pisan u C#-u. Iako postoje programske implementacije u pojedinih modula u drugim jezicima (primjerice difflib datoteka za Python s Ratclif-Obershelp algoritmom [14]), autoru je C puno draži i poznatiji jezik. Isto tako, programiranje od nule eliminira neke česte probleme kod čitanja tuđeg koda.

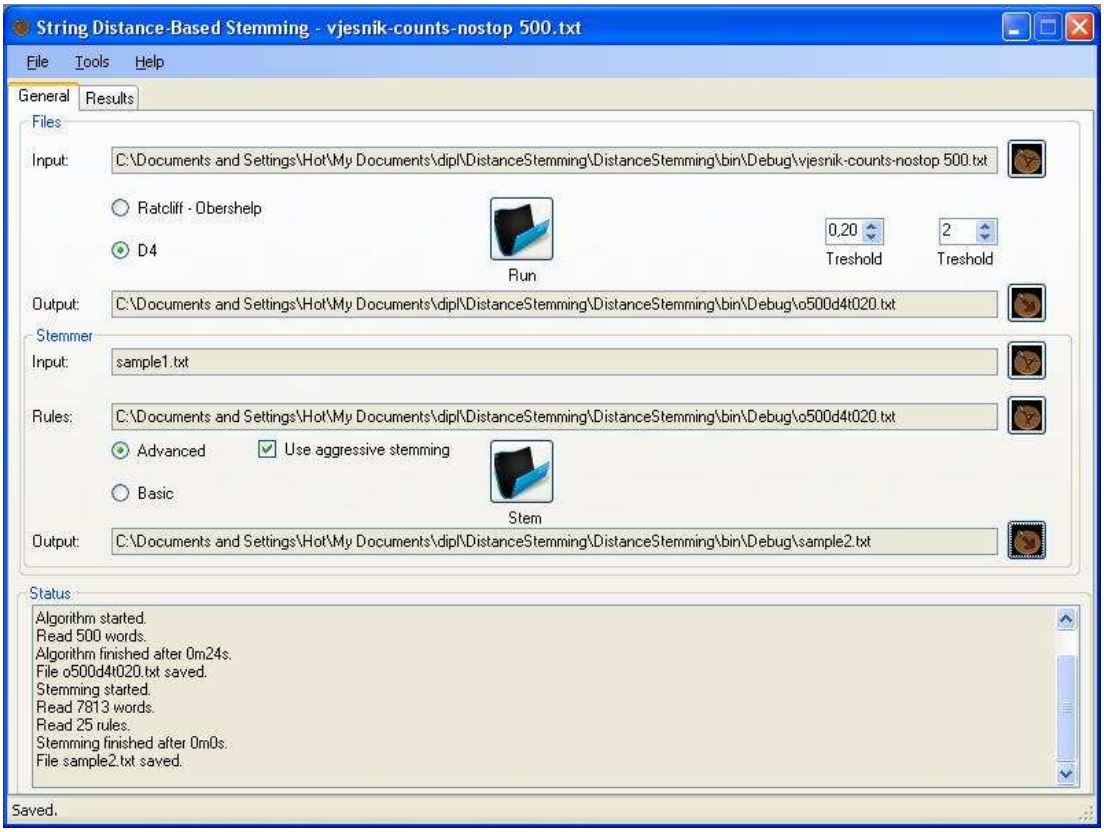
Projekt se sastoji od 3 glavne datoteke. **Mainform.cs** sadrži sve vezano uz sučelje i pozive metoda iz drugih klasa. **Words.cs** sadrži sve vezano uz tri koraka korjenovanja: određivanje udaljenosti, grupiranje sličnih riječi i izvođenje pravila. **Stemmer.cs** primjenjuje dobivena pravila i korjenjuje ulazne riječi.

4.1 Mainform.cs

U **Mainform.cs** je opisano ponašanje algoritma, upute korisniku i određene restrikcije. Primjerice, zabranjeno je pokretanje algoritma ako nije zadana ispravna ulazna datoteka. Ili nije dopuštena negativna vrijednost praga, nego se u tom slučaju automatski korigira na najmanju dopuštenu.

Samo sučelje sastoji se od 2 pogleda između kojih se prebacuje jahačem. Prvi sadrži gumbiće za odabir i učitavanje (odnosno parsiranje) ulazne datoteke, zatim odabir metode grupiranja (Ratclif-Obershelp ili algoritam D_4), odabir praga za spomenute metode, odabir praga frekvencije primjene pravila (sve ispod praga se odbacuje) te konačno gumbići za pokretanje algoritma i spremanje rezultata.

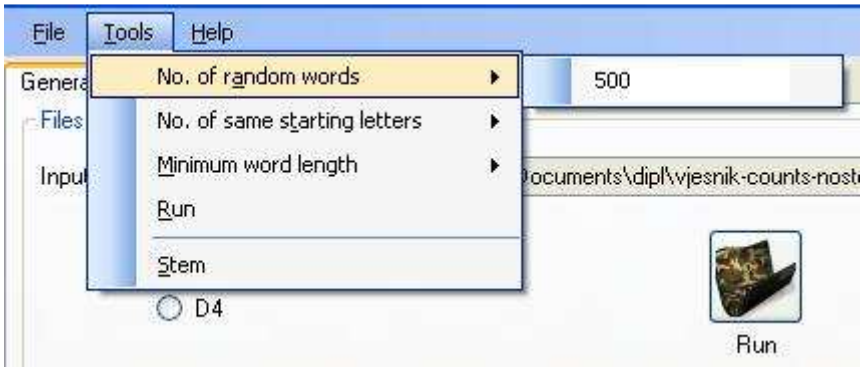
Slika 4.1 ilustrira sučelja programa:



Slika 4.1: Sučelje programa

Na alatnoj traci istiće se opcija *Tools*. Pomoću nje mogu se definirati tri parametra koja suzuju ulazni skup riječi: *No. of Random words*, *No. of same starting letters* i *Minimum word length*. Prvom opcijom se određuje ukupan broj riječi koje je potrebno nasumično učitati iz ulazne datoteke. *No. of same starting letters* je druga opcija kojom se zadaje duljina zajeničkog prefiksa. To znači da će se iz ulazne datoteke odbaciti svaka riječ koja ne dijeli zajednički prefiks sa barem još jednom drugom riječi. Treća opcija je *Minimum word length* i njom se definira minimalna duljina riječi, a sve kraće se odbacuju.

Na slici 4.2 prikazane su ponuđene opcije u izborniku:



Slika 4.2: Opcije

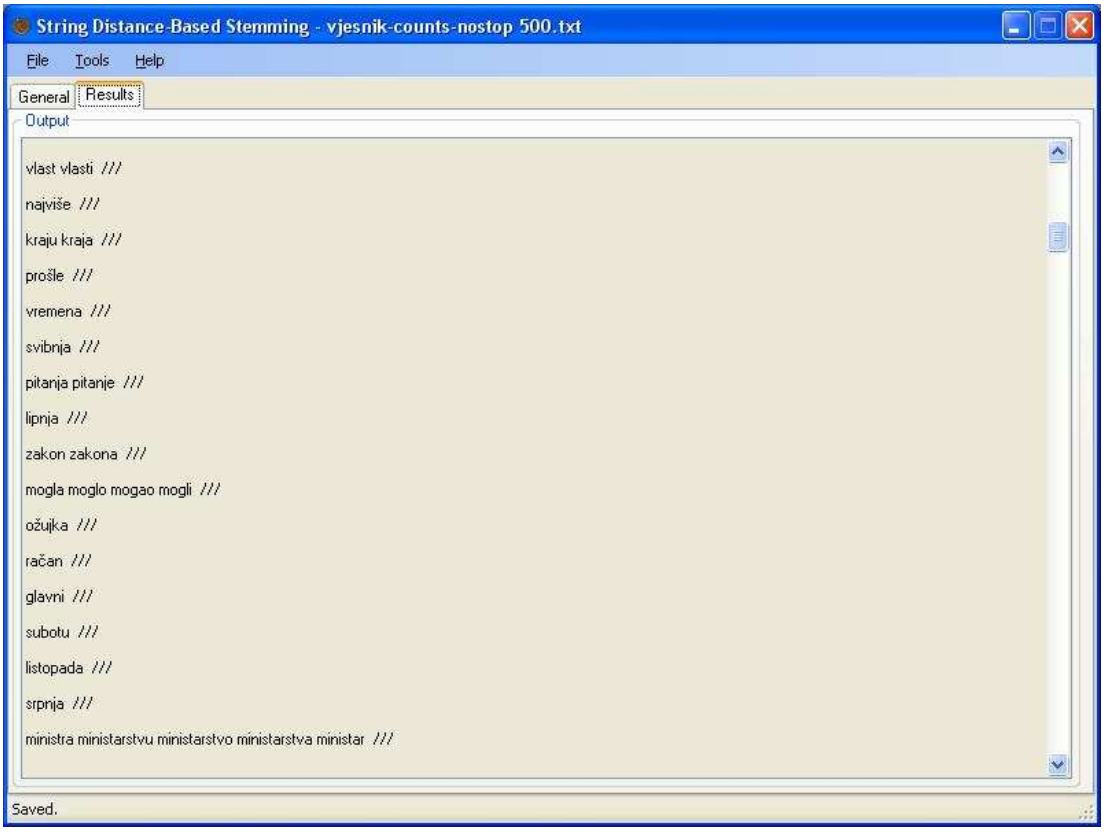
Pokretanje algoritma redom poziva metode iz klase *Words*: *Match*(), *Newman*() i *Rules*(), koje su opisane u nastavku.

Za korjenovanje su predviđena polja u sredini pogleda: gumbići za učitavanje ulazne datoteke s riječima koje treba korjenovati, gumbić za datoteku s definiranim pravilima korjenovanja (može se koristiti izlaz prethodnog koraka) te gumbići za izvršavanje korjenovanja *Stem* i spremanje rezultata. Radi jednostavnosti, ako prilikom pokretanja polja za ulazne datoteke nisu definirana, onda program postavlja predviđene vrijednosti na temelju prethodnog koraka ili pita korisnika za putanju. Dodatno, moguće je birati između osnovne i brže metode (*Basic*) ili preciznije metode (*Advanced*). One su objašnjene u poglavlju 4.3.

Podržani format datoteka je tekstovni dokument (.txt) kodiran po UTF-8 standardu (zbog podrške za hrvatske dijakritičke znakove).

Na dnu se nalazi informativni prozorčić s porukama o broju pročitanih, zapisanih riječi ili pravila, trenutnoj aktivnosti te vremenu izvođenja.

Drugi pogled predstavlja brz pregled rezultata, odnosno zapisa u izlaznoj datoteci. On je prikazan na slici 4.3:



Slika 4.3: Pregled rezultata

4.2 Words.cs

Words.cs sastoji se od tri metode, slikovito nazvane po koracima korjenovanja: *Match*() za određivanje udaljenosti, *Newman*() za grupiranje sličnih riječi i *Rules*() za izvođenje pravila.

Metoda *Match*() formira veze mežu dovoljno sličnim riječima. Same riječi pohranjene su u listi stringova *wordslist*. Metoda prolazi kroz cijelu listu i svaku riječ poredi sa svim ostalima riječima iz liste. Poređenje radi logička funkcija *CheckTreshold*(string a, string b) koja je zapravo okvir za poziv Ratclif-Obershelp metode (*StringDistanceRO*) ili algoritma D_4 (*StringDistanceD4*) ovisno o opciji odabranoj na sučelju. Njen rezultat je true ako po zadanoj metodi riječi prelazi definirani prag.

StringDistanceRO (string a, string b) vraća decimalnu vrijednost sličnosti riječi *a* i *b* po R-O algoritmu. Ona poziva pomoćnu funkciju *StringSimil* koja pronalazi broj zajedničkim znakova u zadanom paru riječi. To je funkcija sa zadanim trivijalnim slučajevima i rekurzivnim pozivima nad nizovima s lijeve odnosno desne strane najduljeg zajedničkog podniza u riječima *a* i *b*. Pri tome se ignorira razlika između velikih i malih slova. Ova metoda je implementirana na temelju članka [11] i diskusije odnosno dijela kôda pisanog u Basicu [12].

StringDistanceD4 (string a, string b) je programska implementacije jednostavne matematičke formule za prefiksno određivanje udaljenosti (3.2). Određuje se indeks *m* prvog

znaka koji je različit u obje riječi te se jednostavnom petljom računa vrijednost drugog faktora (sume).

Riječi koje prelaze prag povezuju se korištenjem liste edgeslist koja je tipa *Edge*, gdje je *Edge* struktura koja u sebi sadrži listu cjelobrojnih vrijednosti. *edgeslist*[i] sadrži sve veze koje završavaju u *i*-toj riječi iz *wordslist*. Primjerice, ako je riječ „godine“ na poziciji 0 u listi *wordslist*, struktura *edgeslist*[0] sadrži listu s vrijednostima 9 i 67. *wordslist*[9] je „godina“, a *wordslist*[67] je „godinu“. Naravno, *edgeslist*[9] sadrži 0 i 67.

Time je ostvarena funkcionalnost stvaranja veza.

Metoda *Newman*() provodi grupiranje riječi. Na početku se poziva procedura *MatrixFill*() koja inicijalizira i popunjava matricu udaljenosti *qMatrix*. Stupci i retci matrice predstavljaju grupe, a vrijednost u tablici za redak *i* i stupac *j* je promjena modularnosti $\Delta Q_{i,j}$ ako se spoje te dvije grupe. Zatim se izvršava *while* petlja s kriterijem zaustavljanja da u matrici *qMatrix* ne postoji nijedan pozitivan element. U petlji se provodi odabir grupa *i* i *j* čije spajanje rezultira najvećim porastom funkcije modularnosti *Q*. Elementi grupe *i* se dodaju grupi *j* te se zatim briše grupa *i*. Matrica udaljenosti se osvježava pozivom procedure *MatrixUpdate*() i petlja počinje iznova.

MatrixFill() popunjava matricu udaljenosti *qMatrix* početnim vrijednostima, za slučaj kad svak riječ predstavlja vlastitu grupu. *qMatrix* je zapravo lista tipa decimal, radi jednostavnosti izvršavanja kasnijih operacija eliminacije *i*-tog retka i stupca. Pamti se širina matrice, a elementima se pristupa modulo logikom. Matrica se popunjava vrijednostima koje računa pomoćna metoda *deltaQ*.

deltaQ (int i, int j) obavlja izračun po formuli (4.1). e_{ij} se računa gledajući za svaki vrh *a* iz *groupslist*[i] njegove veze u *edgeslist*[a]. Ako *groupslist*[j] sadrži iste vrhove, onda to znači da je vrh *a* povezan s vrhom iz grupe *j*. Za izračun a_i broj rubova veza koje završavaju u grupi *i* se dobiva pozivom metode Count() nad *groupslist*[i] koja vraća broj elemenata, a analogno tome dobiva se i a_j.

MatrixUpdate(int i, int j) osvježava vrijednosti matrice udaljenosti. Za svaki stupac matrice *k* u ovisnosti o tome je li grupa k povezana s grupom *i*, grupom *j* ili oboje mijenja se vrijednost elementa matrice ΔQ_{jk}. Nakon toga se prolaskom unatrag kroz listu briše *i*-ti redak i *i*-ti stupac iz matrice koristeći metode *RemoveAt*() i *RemoveRange*().

Time je dovršen drugi korak algoritma.

Metoda *Rules*() izvodi pravila nad skupom riječi unutar iste grupe.

Unutar svake grupe iz *groupslists* pokušava se iz svih kombinacija riječi formirati pravilo. Pravilo izvodi funkcija *DeduceRule*() kojoj se kao argumenti prosljeđuju redom dulja, pa kraća riječ. To je bitno za rad algoritma za korjenovanje, kako bi riječi primjenom pravila svodili na kraći oblik (a ne obratno, što bi zapravo bilo izvođenje novih riječi).

Pravilo koje se izvede poredi se s već postojećima unutar liste stringova *ruleslist*. Ako nema takvog pravila, ono se dodaje u listu. Ako identično pravilo *i* već postoji od ranije, povećava se odgovarajući brojač *rulescountlist*[i] koji pokazuje na to pravilo. Ovisno o pragu pojavljivanja pravila definiranom prije pokretanja algoritma, preživjet će samo najčešća pravila.

DeduceRule(string a, string b) pronalazi zajedničke podnizove unutar riječi *a* i *b* te ih zamijenjuje regularnim izrazima. U Pythonovom difflibu modulu pokazan dan je rekurzivan algoritam *get_matching_blocks*, ali on se oslanja na neke druge procedure iz tog modula. Iz tog razloga programska implementacija u C#-u koristi dvostruku petlju umjesto rekurzije. Prva riječ *a* rastavlja se na blokove varijabilne duljine. Početak bloka i njegova duljine su parametri petlji. Koristi se metoda Substring() klase String za utvrđivanje da li riječ *b* sadrži takav isti blok. Ako da, onda se duljina zajedničkog bloka povećava sve dok se ne nađe njen maksimum. Tada se taj blok u riječi *a* zamjenjuje nizom „(.+)“, a u riječi *b* nizom „\#“, gdje # upućuje na redni broj bloka. Primjerice, iz „momaka“ i „momka“ izvodi se pravilo da niz „(.+)a(.+)“ prelazi u niz „\1\2“.

Time je dovršen i treći korak algoritma.

4.3 Stemmer.cs

Stemmer.cs rezultate dobivene na skupu za učenje u prethodnim koracima primjenjuje na skup za testiranje. Prvo se učitava datoteka sa skupom za testiranje i datoteka sa pravilima korjenovanja. Zatim se provodi jedna od 2 metode korjenovanja: osnovna ili napredna.

Osnovna metoda nije previše izbirljiva kod izbora pravila koje može primijeniti. Od svih pravila odabiru se samo ona koja su ili prefiksna (ne počinju sa ^( ) ili sufiksna (nemaju)$ u sebi). U glavnoj petlji procedure ona se primjenjuju na sve riječi sa ulaza ako te riječi počinju sa odgovarajućim prefiksom, odnosno sufiksom.

Napredna metoda od svih primjenjivih bira ono pravilo koje za promatranu riječ rezultira najkraćim pseudokorijenom. Primjerice, ako promatra riječ “sunčanim” te može birati između korjenovanja na “sunčani” ili “sunčan”, odabrat će drugu opciju. Zbog načina na koji su definirana pravila (poglavlje 3.3) poseban problem predstavljaju promjene bez fiksiranog prefiksa ili sufiksa. Primjerice, pravilo promjena “novca” u “novac” (u hrv. gramatici to je pojava nepostojanog a) teško je pravilno primijeniti bez poznavanja odgovarajućeg konteksta ili specifičnosti gramatike. Iz tog razloga nije se nastojalo primijeniti pravila koja mijenjaju samo jedan znak u sredini riječi, bez nekog konteksta.

Za naprednu metodu se dodatno može uključiti opcija “agresivnog” korjenovanja. “Agresivno” u ovom kontekstu znači ponovnu primjenu pravila na prethodni rezultat korjenovanja. Ako imamo mali skup pravila, onda je takvo ponašanje poželjno, ali može rezultirati prekorjenovanjem. Primjerice, riječ “prepreden” svela bi se na pseudokorijen “den”. Ukoliko se ta opcija isključi, algoritam nakon primjene jednog pravila prelazi na promatranje iduće riječi.

Korjenovane riječi spremaju se u izlaznu datoteku.

5. Eksperimentalno vrednovanje

5.1 Način vrednovanja

U poglavlju 1.6 objašnjeno je kako korjenovanje poboljšava perfomanse sustava za pretraživanje informacija, te su definirani pojmovi preciznosti i odziva. Na osnovu te dvije vrijednosti moguće je napraviti ekstrinzično vrednovanje sustava. Primjerice, ako tražimo po pojmu „psi“, moguće je točno izračunati preciznost i odziv za dohvaćene dokumente o psima. Međutim, za ocjenu točnosti normalizacijskog postupka neovisno o konkretnom zadatku potrebno je provesti intrinzično vrednovanje.

Ovdje korišteni intrinzični način vrednovanja temeljen je na postupku predloženom u [21] i proširenju predloženom u [20]. On je temeljen na prebrojavanju grešaka potkorjenovanja (engl. understemming) i prekorjenovanja (engl. overstemming). Do pogreške potkorjenovanja dolazi u slučaju kad dva flektivna ili derivacijska oblika iste riječi nisu svedena na zajedničku normu. Primjerice, ako se riječi „stanovi“ i „stana“ svedu na različitu normu: „stanov“ i „stan“. Obrnuto, do pogreške potkorjenovanja dolazi u slučaju kad se na istu normu svedu dva oblika koja nisu morfološki povezana. Primjerice, „arterije“ i „baterije“ se svedu na istu normu: „aterije“.

Ova dva tipa pogreški računaju se na ručno sastavljenom uzorku. Korišten je uzorak [20] koji je načinjen iz tekstova Vjesnika, rubrike Kulture. On sadržava 5052 oblika riječi, a od toga 5014 različitih oblika (homografi su pridruženi u više grupa). Riječi su grupirane u 1985 derivacijskih grupa međusobno odvojenih praznim redom. Unutar jedne derivacijske, flektivne grupe (kojih ima ukupno 2762) odvojene su redom koji sadrži samo crticu -. Točna normalizacijska procedura na takvom bi uzorku trebala praviti što manji broj pogrešaka potkorjenovanja i prekorjenovanja.

Indeks potkorjenovanja (UI) i indeks prekorjenovanja (OI) računaju se na sljedeći način [13]:

$$UI = \frac{\#(različito\ normalizirani\ parovi\ u\ svakoj\ grupi)}{\#(\text{parovi riječi u svakoj grupi})}, \tag{5.1}$$

$$OI = \frac{\#(\text{parovi iz različitih grupa svedeni na istu normu})}{\#(\text{parovi svedeni na istu normu})}. \tag{5.2}$$

U praksi su ova dva indeksa međusobno povezana. Primjerice, konzervativno korjenovanje rezultirat će s malo pogrešaka prekorjenovanja i puno pogrešaka potkorjenovanja, dok će kod agresivnog korjenovanja situacija biti upravo obrnuta (previše će se odsijecati i svoditi na istu normu).

Uvodimo mjeru ukupne kvalitete korjenovanja (SQ), koja ukazuje na ukupne performanse na flektivnoj i derivacijskoj razini. Nju definiramo kao:

$$SQ = \frac{2(1-UI)(1-OI)}{2-UI-OI}. \tag{5.3}$$

Pri izračunu kvalitete flektivno-derivacijske normalizacije u obzir treba uzeti samo pogreške potkorjenovanja na flektivnoj razini te pogreške prekorjenovanja na derivacijskoj razini [20]. Ona ukazuje na sveukupnu kvalitetu i na flektivnoj i na derivacijskoj razini.

Mjeru kvalitete flektivno-derivacijske normalizacije (idSQ) moguće je definirati na sljedeći način:

$$idSQ = \frac{2(1 - iUI)(1 - dOI)}{2 - iUI - dOI}. \tag{5.4}$$

gdje je iUI indeks potkorjenovanja na flektivnim grupama, a dOI indeks prekorjenovanja na derivacijskim grupama. Mjera idSQ neosjetljiva je na razlike u stupnju derivacijske normalizacije te uzima u obzir samo one pogreške potkorjenovanja i prekorjenovanja za koje je neupitno da su doista pogreške.

5.2 Rezultati

Skup za učenje birao se iz tekstovne datoteke od oko 560.000 riječi preuzetih iz kolekcije Vjesnika. Riječi su sortirane po učestalosti pojavljivanja u tekstovima. Jedan od parametara testiranja bila je i veličina ulazne datoteke. Iz praktičnih razloga, težilo se trajanju izvođenja u rangu od nekoliko minuta.

U sljedećoj tablici dani su parovi vrijednosti za broj riječi u ulaznoj datoteci i trajanju izvođenja algoritma D_4. Uzimano je prvih 300 do 900 riječi iz ranije navedene datoteke. Testno računalo sastoji se od Intelovog procesora P7350 na taktu od 2GHz, posjeduje 2GB radne memorije i disk s brzinom okretaja 5400 rpm.

Tablica 5.1: Algoritam D_4: prag 0,2; broj pojavljivanja pravila >=2 :

Broj riječi u ulaznoj datoteci	Vrijeme izvođenja (s)
300	2
400	9
500	24
600	68÷76[I]
700	168
800	320
900	531÷552[II]
1000	987

[I] 76s je trajanje uz prag 0,3

[II] 552s je trajanje uz prag pravila 4

U sljedećoj tablici provedeno je isto mjerenje za algoritam Ratclif-Obershelp.

Tablica 5.2: Algoritam Ratclif-Obershelp: prag 0,8; broj pojavljivanja pravila >=2 :

Broj riječi u ulaznoj datoteci	Vrijeme izvođenja (s)
300	5
400	14
500	27/34/36[I]
600	93
700	206
800	388
900	681÷700
1000	1127

[I] 27s je trajanje uz prag 0,85; 36s je trajanje uz prag 0,75

U nastavku su dani rezultati mjerenja za različite kombinacije veličine ulazne datoteke, korištene metode i praga, broja pojavljivanja pravila *r* te dodatnih parametara: minimalna duljina riječi (*l*), broj nasumično učitanih riječi (*n*) ili broj slova u zajedničkom prefiksu (*p*).

Promatrane su vrijednosti: mjera kvalitete flektivno-derivacijske normalizacije (*idSQ*), pogreške potkorjenovanja na flektivnoj razini (*iUI*) te pogreške prekorjenovanja na derivacijskoj razini (*dOI*). Osim ako nije drugačije navedeno, korišteno je „agresivno" korjenovanje opisano u poglavlju 4.3 (rezultat korjenovanja ponovno se pokušava korjenovati po drugim pravilima).

U tablici 5.3 mjerene su vrijednosti ovisno o odabranoj mjeri (Ratclif-Obershelp – RO ili algoritam D4) uz varijaciju broja riječi u ulaznoj datoteci.

Tablica 5.3: Usporedba kvalitete normalizacije za dvije mjere udaljenosti:

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	500	R-O	0,8	2		0,771	0,345	0,059	48
2	500	D_4	0,2	2		0,416	0,733	**0,048**	25
3	600	R-O	0,8	2		0,747	0,377	0,065	62
4	600	D_4	0,2	2		0,727	0,412	0,049	34
5	700	R-O	0,8	2		0,743	0,368	0,096	74
6	700	D_4	0,2	2		0,793	0,316	0,053	39
7	800	R-O	0,8	2		0,783	0,306	0,101	92
8	800	D_4	0,2	2		0,793	0,316	0,053	40
9	900	R-O	0,8	2		0,775	0,311	0,113	100
10	900	D_4	0,2	2		0,793	0,316	0,053	43
11	1000	R-O	0,8	2		0,758	0,275	0,203	122
12	1000	D_4	0,2	2		**0,829**	**0,245**	0,080	57

U tablici 5.4 dane su vrijednosti ovisno o pragu sličnosti riječi po odabranoj mjeri. Prilikom proučavanja rezultata treba uzeti u obzir da su kriteriji dvije metode sljedeći: Ratclif-Obershelp prihvaća sve iznad praga, dok D_4 prihvaća sve slične riječi ispod zadanog praga.

Tablica 5.4: Utjecaj praga sličnosti:

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	500	R-O	0,75	2		0,703	0,418	0,110	69
2	500	R-O	0,8	2		0,771	0,345	0,059	48
3	500	R-O	0,85	2		0,726	0,412	0,049	24
4	500	D_4	0,15	2		0,411	0,737	**0,043**	16
5	500	D_4	0,2	2		0,416	0,733	0,048	25
6	500	D_4	0,30	2		0,776	0,343	0,049	42
7	600	R-O	0,75	2		0,676	0,452	0,115	87
8	600	R-O	0,80	2		0,747	0,377	0,065	62
9	600	R-O	0,85	2		0,726	0,412	0,049	30
10	600	D_4	0,15	2		0,411	0,737	**0,043**	17
11	600	D_4	0,20	2		0,727	0,412	0,049	25
12	600	D_4	0,30	2		0,773	0,347	0,049	46
13	900	D_4	0,30	2		**0,811**	**0,274**	0,079	203

U tablici 5.5 dane su vrijednosti ovisno o parametarima: minimalna duljina riječi (*l*), broj nasumično učitanih riječi (*n*) ili broj slova u zajedničkom prefiksu (*p*).

Tablica 5.5: Utjecaj parametara:

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	331 [I]	R-O	0,8	2	l=6	0,475	0,684	0,036	29
2	381 [II]	R-O	0,8	2	p=3	0,767	0,351	0,060	40
3	500 [III]	R-O	0,8	2	n=500	0,070	0,964	0,106	4
4	847 [IV]	R-O	0,8	2	p=3	0,767	0,276	0,181	113
5	900 [V]	R-O	0,8	4	n=900	-			0
6	900 [VI]	D_4	0,2	4	n=900	-			0

[I] pravila se izvode iz 331 riječi koje imaju barem 6 slova, iz datoteke od 1000 riječi

[II] pravila se izvode iz 381 riječi koje imaju barem 3 zajednička početna slova, iz datoteke od 500 riječi

[III] pravila se izvode iz 500 nasumično odabranih riječi iz datoteke od 10.000 riječi

[IV] pravila se izvode iz 847 riječi koje imaju barem 3 zajednička početna slova, iz datoteke od 1000 riječi

[V] pravila se izvode iz 900 riječi nasumično odabranih iz datoteke od oko 560.000 riječi

[VI] pravila se izvode iz 900 riječi nasumično odabranih iz datoteke od oko 560.000 riječi

Tablici 5.6 sadrži mjerenja za minimalni broj pojavljivanja pravila *r* >=4. Sva predložena pravila koja se u ulaznom skupu ne ponavljaju barem *r* puta odbacuju se kao nevažeća.

Tablica 5.6: Utjecaj broja pojavljivanja pravila *r* :

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	700	R-O	0,2	4		**0,778**	0,344	0,040	26
2	700	D_4	0,2	4		0,591	0,572	**0,036**	19
3	800	R-O	0,2	4		0,772	0,351	0,043	37
4	800	D_4	0,2	4		0,724	0,417	0,039	24
5	900	R-O	0,2	4		0,767	**0,276**	0,181	43
6	900	D_4	0,2	4		0,724	0,417	0,039	27

U tablici 5.7 dani su rezultati za $r = 1$ (primjena svih predloženih pravila) uz variranje praga sličnosti i broja ulaznih riječi:

Tablica 5.7: Usporedba kvalitete normalizacije ovisno o pragu sličnosti i broju riječi:

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	900	D_4	0,12	1		0,634	0,519	0,064	27
2	900	D_4	0,15	1		0,833	0,249	0,063	68
3	900	D_4	0,17	1		0,833	0,249	0,063	73
4	900	D_4	0,20	1		0,806	0,240	0.141	84
5	900	D_4	0,25	1		0,689	0,342	0,273	126
6	2000	D_4	0,15	1		0,758	0,215	0,266	78
7	2000	D_4	0,20	1		0,619	0,200	0,493	138

Tablica 5.8 sadrži mjerenja za slučajeve “neagresivnog” korjenovanja. Jednom korjenovana riječ nije se razmatrala za ponovno korjenovanje.

Tablica 5.8: Rezultati “neagresivnog” korjenovanja:

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	500	D_4	0,20	1		0,537	0,628	0,032	68
2	900	D_4	0,20	1		0,635	0,527	0,031	84
3	900	D_4	0,25	1		0,604	0,556	0,052	126
4	2000	D_4	0,15	1		0,500	0,662	0,034	78
5	2000	D_4	0,20	1		0,716	0,429	0,037	138

Tablica 5.9: Rezultati sortirani po **idSQ** (više je bolje):

#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila
1	500 [III]	R-O	0,8	2	n=500	0,07	0,964	0,106	4
2	500	D4	0,15	2		0,411	0,737	0,043	16
3	600	D4	0,15	2		0,411	0,737	0,043	17
4	500	D4	0,2	2		0,416	0,733	0,048	25
5	331 [I]	R-O	0,8	2	l=6	0,475	0,684	0,036	29
6	700	D4	0,2	4		0,591	0,572	0,036	19
7	2000	D4	0,2	1		0,619	**0,200**	0,493	**138**
8	900	D4	0,12	1		0,634	0,519	0,064	27
9	600	R-O	0,75	2		0,676	0,452	0,115	87
10	900	D4	0,25	1		0,689	0,342	0,273	126
11	500	R-O	0,75	2		0,703	0,418	0,11	69
12	800	D4	0,2	4		0,724	0,417	0,039	24
13	900	D4	0,2	4		0,724	0,417	0,039	27
14	500	R-O	0,85	2		0,726	0,412	0,049	24
15	600	R-O	0,85	2		0,726	0,412	0,049	30
16	600	D4	0,2	2		0,727	0,412	0,049	34
17	600	D4	0,2	2		0,727	0,412	0,049	25
18	700	R-O	0,8	2		0,743	0,368	0,096	74
19	600	R-O	0,8	2		0,747	0,377	0,065	62
20	2000	D4	0,15	1		0,758	0,215	0,266	78
21	381 [II]	R-O	0,8	2	p=3	0,767	0,351	0,06	40
22	847 [IV]	R-O	0,8	2	p=3	0,767	0,276	0,181	113
23	900	R-O	0,2	4		0,767	0,276	0,181	43
24	500	R-O	0,8	2		0,771	0,345	0,059	48
25	800	R-O	0,2	4		0,772	0,351	0,043	37
26	600	D4	0,3	2		0,773	0,347	0,049	46
27	900	R-O	0,8	2		0,775	0,311	0,113	100
28	500	D4	0,3	2		0,776	0,343	0,049	42
29	700	R-O	0,2	4		0,778	0,344	0,04	26
30	800	R-O	0,8	2		0,783	0,306	0,101	92
31	700	D4	0,2	2		0,793	0,316	0,053	39
32	800	D4	0,2	2		0,793	0,316	0,053	40
33	900	D4	0,2	2		0,793	0,316	0,053	43
34	900	D4	0,2	1		0,806	0,24	0.141	84
35	900	D_4	0,30	2		0,811	0,274	0,079	203
36	900	D4	0,15	1		0,833	0,249	0,063	68
37	900	D4	0,17	1		0,833	0,249	0,063	73
#	Broj riječi	Oznaka metode	prag	r	dodatni param.	idSQ	iUI	dOI	# pravila

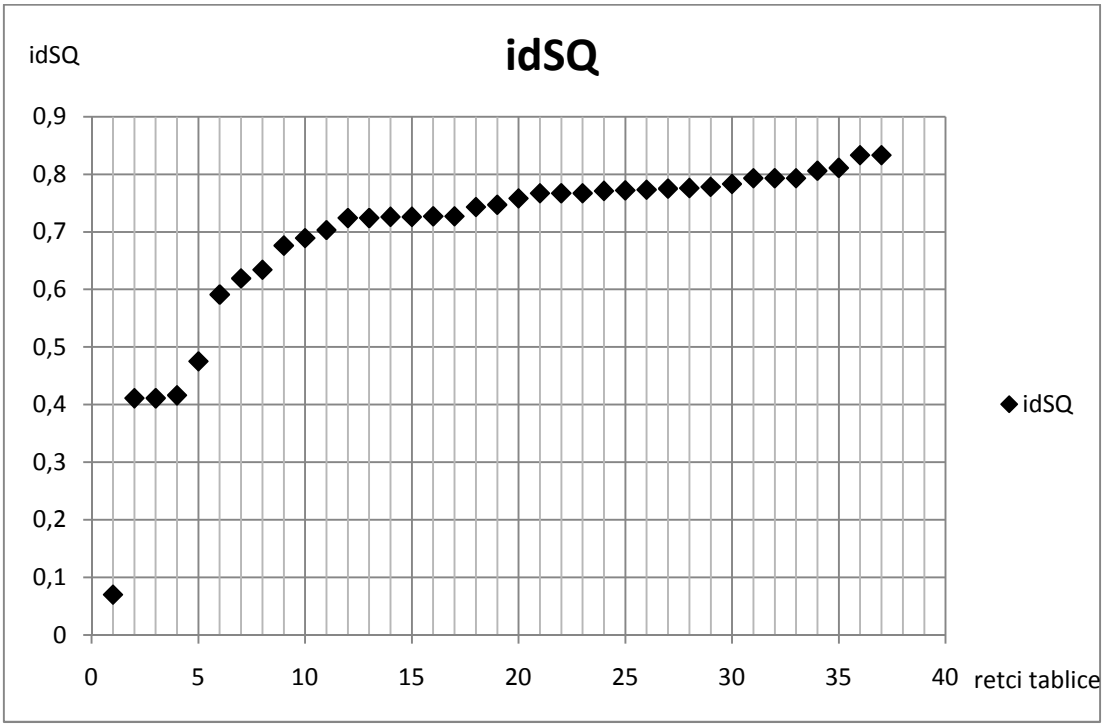
[I] pravila se izvode iz 331 riječi koje imaju barem 6 slova, iz datoteke od 1000 riječi

[II] pravila se izvode iz 381 riječi koje imaju barem 3 zajednička početna slova, iz datoteke od 500 riječi

[III] pravila se izvode iz 500 nasumično odabranih riječi iz datoteke od 10.000 riječi

[IV] pravila se izvode iz 847 riječi koje imaju barem 3 zajednička početna slova, iz datoteke od 1000 riječi

Na sljedećem grafikonu 5.10 prikazana su vrijednosti mjere kvalitete flektivno-derivacijske normalizacije u ovisnosti o pojedinom mjerenju iz tablice 5.9:



5.10: Optimiziranje ukupne kvalitete normiranja kroz prilagodbe metode

5.3 Rasprava rezultata

Iz tablice 5.3 uočljivo je da za manji ulazni skup algoritam Ratclif-Obershelp rezultira većom kvalitetom normiranja od algoritma D_4. Ta se razlika smanjuje povećanjem broja ulaznih riječi te za njih 700 prelazi u prednost algoritma D_4, koji zatim nastavlja davati bolje rezultate kako raste broj riječi.

Općenito, za veći ulazni skup bolji rezultati dobivaju se postavljanjem višeg praga broja pojavljivanja pravila *r* (dno tablice 5.3; tablice 5.6 i 5.7). S većim brojem grupiranih riječi postoji veća vjerojatnost za ponavljanjem pogrešno definiranog pravila, a prag u vrijednosti 2 ili veći filtrira takve pogreške.

Na osnovu usporedbe utjecaja praga sličnosti iz tablice 5.4 vidljivo je da se kvaliteta algoritma D_4 može poboljšati podizanjem praga, što za tu metodu rezultira prihvaćanjem riječi koje su nešto manje slične (sve ispod praga).

U tablici 5.5 razmatrao se hibridni pristup između dvije metode: uzimanjem u obzir riječi koje dijele početni niz znakova s barem još jednom drugom riječi iz ulaznog skupa njegovo ponašanje se približavalo radu algoritma D_4. Drugi pristup bio je određivanje minimalne duljine riječi, koja je za algoritam Ratclif-Obershelp usko povezana s pragom (jer on vraća postotnu sličnost). Konačno, razmatrao se utjecaj predgrupiranja riječi. One su se čitale potpuno nasumično iz cijelog ulaznog skupa. Rezultati u odnosu na tablicu 5.3 pokazuju da predsortiranje igra veliku ulogu: ako učitane riječi uopće nisu slične, ne mogu se izvoditi pravila. Ograničavanje duljine riječi ili zajedničkog prefiksa nije dalo bolje rezultate, pa se zato u ostalim mjerenjima odustalo od tog pristupa poboljšanju kvalitete.

Iz tablica 5.6 i 5.7 određene su optimalne vrijednosti praga udaljenosti i praga pojavljivanja pravila *r*. Zbog relativno malog ulaznog skupa, prag *r* je maknut (postavljen na 1), tako da su se prihvaćala sva predložena pravila radije nego da se odbacuju ona koja se ne ponavljaju.

Iz tablice 5.8 vidljivo je kako pristup korjenovanju utječe na pogrešku prekorjenovanja odnosno potkorjenovanja. "Neagresivna" metoda smanjila je prekorjenovanje, ali zato povećala potkorjenovanje. Tako je rezultirala slabijom ukupnom kvalitetom normiranja od pristupa gdje se korjenovanje nad jednom riječi ponavlja dok se ne iscrpe sva primjenjiva pravila.

Najbolji rezultati postizali su se po prefiksno orijentiranom algoritmu D_4 (idSQ iznosi 83.3%) nad ulaznim skupom od 900 riječi. Algoritam Ratclif-Obershelp rezultirao je kvalitetom normiranja koja je za 3 do 5% slabija nad istim ulaznim skupom (tablica 5.9). Takvi rezultati ukazuju da je D_4 bolji algoritam za primjenu na hrvatski jezik. Razlog tome je priroda hrvatskog jezika koji je znatno orijentiran na sufikse, dok algoritam Ratclif-Obershelp povezuje i one riječi koje se razlikuju po dovoljno malo slova na bilo kojoj poziciji. U većini slučajeva, to za rezultat ima svođenje različitih riječi na istu normu koja ne odgovara gramatičkom korijenu (a to je greška prekorjenovanja).

Rezultati su usporedivi s onima iz rada [8], gdje se uz odgovarajući izbor praga postizao idSQ od 85%. Radi usporedbe, jednostavno odsijecanje kraja riječi nakon određenog broja slova (engl. truncation) dostiže rezultat od 75%, dok rječnička normalizacija postiže i do 95%.

Pogreška flektivnog potkorjenovanja bila je u većini slučajeva 3 do 5 puta veća od greške uslijed derivacijskog prekorjenovanja. To sugerira da su izvedena pravila točna, ali nepotpuna. Odnosno, normirana forma u tim slučajevima dulja je od gramatičkog korijena riječi.

Neočekivani rezultat je slabija kvaliteta normiranja za ulaznu datoteku od 2000 riječi u odnosu na većinu mjerenja sa manjim brojem ulaznih riječi. Premda je taj slučaj dao najmanju izmjerenu vrijednost za pogrešku potkorjenovanja, prekorjenovanje je poraslo i tako smanjilo ukupnu kvalitetu.

Primjer jednog pogrešnog pravila koje je izveo algoritam Ratclif-Obershelp, a D_4 nije izveo jest: `^g(.+)$->\1`. Ono je izvedeno na temelju sličnosti parova riječi poput "grad" – "rad", a zapravo nije pravo gramatičko pravilo.

Drugi primjer je pravilo za nepostojano a: ^(.+)a(.+)$->\1\2. Ono je izvedeno na temelju sličnosti "zemlja" – "zemalja", "stranaka" – "stranka", ali isto tako i pogrešnim parom: "pravo" – "prvo". Poteškoća kod primjene takvog pravila je što ono nije inverzno. Odnosno, ne možemo iz bilo koje riječi proizvoljnom eliminacijom slova "a" dobiti smisleni oblik i obratno: proizvoljno dodavanje slova "a" u (hrvatskom) jeziku nema smisla.

U dodatku A nalaze se primjeri grupiranih riječi po oba algoritma. Dodatak B sadrži pravila koja su iz njih izvedena zajedno sa ukupnoim brojem pojavljivanja tog pravila u ulaznom skupu. U dodatku C prikazani su primjeri korjenovanja po najboljem slučaju iz tablice 3.9.

6. Zaključak

Problem koji se proučavao u radu je razvoj algoritma za nenadzirano strojno učenje koji bi prepoznavao sve morfološki srodne oblike riječi (flektivne oblike jedne riječi i derivacijsko povezane oblike više riječi). Takav algoritam imao bi primjenu u području pretraživanja informacija, dubinskoj analizi teksta, raspoznavanju govora i strojnom prevođenju. Postojeći slični algoritmi značajno pospješuju perfomanse sustava za pretraživanje informacija povećavajući odziv.

U tu svrhu napravljen je program koji korjenjuje riječi na principu ortografske udaljenosti. Program uči pravila korjenovanja kroz tri faze. To su: povezivanje sličnih riječi na temelju dva konceptualno različita algoritma, organiziranje povezanih riječi u derivacijske grupe te konačno izvođenje flektivnih pravila iz grupa.

Rezultati su usporedivi s radom [8], gdje se za metodu D_4 postizao idSQ od 85%. U navedenom radu ulazni skup bio je veličine 560.000, dok je u ovom radu za najbolji slučaj iznosio 900. Jednostavno odsijecanje kraja riječi nakon određenog broja slova dostiže rezultat od 75%, dok rječnička normalizacija postiže i do 95%. Takav rezultat ide u prilog primjeni metode opisane u ovom radu za jezično nezavisan pristup.

Jedno od predloženih budućih poboljšanja jest uvođenje predsortiranja, odnosno podjela velikog ulaznog skupa na manje cjeline riječi koje su smisleno grupirane. Time bi se program izvršao u razumno dugom periodu nad sveukupno većim ulazom nego je korišten u ovom radu. Izvelo bi se više potencijalnih pravila uz pretpostavku da bi taj pristup smanjio pogreške potkorjenovanja.

Algoritam bi se dodatno mogao poboljšati prilagodbom gramatici hrvatskog jezika i u tom smislu optimizirati za konkretan jezik. Primjerice, mogla bi se definirati najveća duljina gramatičkog prefiksa. S druge strane, tema ovog rada bila je jezično nezavisan pristup, tako da bi to eventualno bio fokus nekog drugog rada.

7. Literatura

1. Kurimo, Mikko, Virpioja, Sami and Turunen, Ville T. „Overview and Results of Morpho Challenge 2009“. Pages 1-3, 2009.
2. Newman, M. E. J., Clauset, A., and Moore, C. „Finding community structure in very large networks“. Pages 1-3, 2004.
3. Encyclopædia Britannica. cuneiform (writing system). Pristupljeno 01. lipnja 2010. (http://www.britannica.com/EBchecked/topic/146558/cuneiform).
4. John Scott: *Social Network Analysis*, 1987.
5. Galvez, Carmen, de Moya-Anegón, Félix and Solana, Vıctor H. “Term conflation methods in information retrieval: Non-linguistic and linguistic approaches“. In Journal of Documentation. Pages. 520-529, 2005.
6. Morpho Challenge 2010 - Semi-supervised and Unsupervised Analysis. Pristupljeno 20. svibnja 2010. (http://www.cis.hut.fi/morphochallenge2010/)
7. Stein, Benno and Potthast, Martina. „Putting Successor Variety Stemming to Work“. In Advances in data analysis: Proceedings of the 30th Annual Conference of The Gesellschaft für Klassifikation e.V., 2006.
8. Šnajder, Jan and Dalbelo Bašić, Bojana. „String Distance-Based Stemming of the Highly Inflected Croatian Language“. In Proceedings of Recent Advances in Natural Language Processing. 411-415, 2009.
9. Manning, Christopher D., Raghavan, Prabhakar and Schütze, Hinrich. „Introduction to Information Retrieval“. Pages 421-433, 2008.
10. Majumder, P., Mitra, M., Parui, S. K., Kole, G., Mitra, P. and Datta, K. „YASS: Yet another suffix stripper“. 2007.
11. John W. Ratclif: *Pattern Matching: the Gestalt Approach*, 1988.
12. QL Hacker's Journal, Issue #1. Pages 7-9, 2001.
13. Šnajder, Jan. „Postupci morfološke normalizacije u pretraživanju informacija i klasifikaciji teksta“. 2008.
14. Dokumentacija za Python v2.6.5. Pristupljeno 12. srpnja 2010. (http://docs.python.org/library/difflib.html)
15. ACM SIGIR – Awards. Pristupljeno 14. lipnja 2010. (http://www.sigir.org/awards/awards.html)
16. Tony Kent Strix Award | UkeiG. Pristupljeno 14. lipnja 2010. (http://www.ukeig.org.uk/awards/tony-kent-strix)
17. Arampatzis, A.T., Tsoris, T., Koster, C.H.A., van der Weide, Th.P. Phrase-based Information Retrieval. In Computing Science Institute. Pages 11-15, 1998.
18. Bernhard, Delphine. „MorphoNet: Exploring the Use of Community Structure for Unsupervised Morpheme Analysis“. Pages 1-9, 2009.
19. Marković, Ivan. “Tri nehrvatske tvorbe : infiksacija, reduplikacija, fuzija”. Rasprave Instituta za hrvatski jezik i jezikoslovlje. 217-241, 2009.
20. Šnajder, Jan. “Morfološka normalizacija tekstova na hrvatskome jeziku za dubinsku analizu i pretraživanje informacija”. Doktorska disertacija, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu. 102-110, 2010.
21. Paice, C. D. “Method for evaluation of stemming algorithms based on error counting”. In Journal of the American Society for Information Science, vol. 47, no. 8. Pages 632–649, 1996.

Dodatak A: primjeri grupiranih riječi

U nastavku su dani primjeri koji oslikavaju razliku između 2 metode grupiranja.

Ratclif-Obershelp:

- ostati sati ostali ostalim ///
- imamo imao ima ///
- zemlja zemlji zemlje zemljama zemalja ///
- djelo tijela dijela djela dijelu ///
- koalicije policije policija ///
- strani stranaka stranka strane stranke ///
- rade radu gradu grad rad rada grada ///
- ratu rast hrvata hrvatskoga hrvatskim hrvatsko hrvatskom hrvatsku hrvatskog hrvatski hrvatskoj hrvatska hrvatske hrvatskih vrata rat rata ///
- uprava uprave prvo pravo prava ///
- ostaje stanju stanje ///

D_4 za iste riječi; pojedine derivacijske grupe odijeljene su znakom "///", neke riječi su prikazane u istom redu redi lakše usporedbe za drugim algoritmom:

- ostati ///sati ///ostali ostalim ///
- imamo ///imao ///ima ///
- zemlja zemlji zemlje ///zemljama ///zemalja ///
- djelo djela ///dijela dijelu ///tijela ///
- koalicije ///policije policija ///
- strani strane ///stranka stranke ///stranaka ///
- rade ///radu ///gradu grad grada ///rada ///
- rat ///vrata ///ratu ///hrvatskim hrvatskih hrvatsko hrvatsku hrvatski hrvatska hrvatske hrvatskoga hrvatskom hrvatskog hrvatskoj ///
- uprava uprave ///prvo ///pravo prava ///
- ostaje ///stanju stanje ///

Dodatak B: primjeri izvedenih pravila

U nastavku su dani primjeri pravila izvedenih po navedenoj metodi.

Ratclif-Obershelp:

```
^po(.+)$->\1: 2 times.
^u(.+)$->\1: 2 times.
^u(.+)e$->\1a: 2 times.
^g(.+)$->\1: 3 times.
^(.+)a$->\1: 26 times.
^(.+)i$->\1: 8 times.
^(.+)a$->\1u: 20 times.
```

D_4:

```
^(.+)m$->\1: 4 times.
^(.+)a$->\1e: 3 times.
^(.+)a$->\1i: 15 times.
^(.+)o$->\1a: 8 times.
^(.+)u$->\1a: 17 times.
^(.+)a$->\1: 21 times.
```

Dodatak C: primjeri korjenovanja

U nastavku su dani primjeri korjenovanja za najbolju metodu (D_4 nad skupom od 900 riječi uz prag 0.17).