Bojan Butolen

PLAGIATORSTVO IN PROGRAMI ZA DETEKCIJO PLAGIATOV

Diplomsko delo

Maribor, maj 2009

Diplomsko delo univerzitetnega študijskega programa

PLAGIATORSTVO IN PROGRAMI ZA DETEKCIJO PLAGIATOV

Študent:	Bojan Butolen
Študijski program:	UN ŠP Računalništvo in informatika
Smer:	Programska oprema
Mentor(ica):	izr. prof. dr. Milan Zorman
Somentor(ica):	red. prof. dr. Peter Kokol
Lektor(ica):	Nataša Ropič

Maribor, maj 2009

ZAHVALA

Zahvaljujem se mentorju in somentorju za pomoč in vodenje pri opravljanju diplomskega dela. Hvala profesorjem, ki so mi posredovali znanje, da sem lahko uspešno opravil študij, ter sošolcem, ki so s svojo tekmovalnostjo večali mojo željo po znanju.

Posebno zahvalo posvečam staršem, ki so mi omogočili študij.

PLAGIATORSTVO IN PROGRAMI ZA DETEKCIJO PLAGIATOV

Ključne besede: plagiatorstvo, primerjanje datotek, podobnost datotek, delitev besedila

UDK:

Povzetek

V diplomskem delu je predstavljen problem plagiatorstva ter možni načini reševanja le-tega. Predstavljeni so programi, s pomočjo katerih je možno primerjati velike sklope datotek in odkrivati potencialne primere plagiarizma. Ob programih, ki služijo primerjanju tekstovnih datotek, je nekaj diplomskega dela posvečenega študiji programov za primerjanje programske kode. Praktični del diplomskega dela predstavljata razvoj programa za odkrivanje plagiatov ter primerjava rezultatov, dobljenih s programom APSS, glede na rezultate, dobljene z obstoječimi programi. Predstavljeni so tudi algoritmi in pristopi, ki so uporabljeni pri razvoju programa.

PLAGIARISM AND PROGRAMS FOR PLAGIARISM DETECTION

Key words: plagiarsm, file comparison, file similarity, text segmentation

UDK:

Abstract

In this diploma thesis the problem of plagiarism and the means how to detect plagiarism in text files are introduced. It includes an introduction of programs with which we can compare large sets of text files and discover plagiarized files. Besides that, a study of some programs that compare source code files is presented. The second part contains a development of a program for file comparison and plagiarism detection and the comparison of the results gained with this program with the results of other existing programs. It also includes the explanation of the algorithms and the approaches taken in the development phase.

Kazalo vsebine

Kazalo slik

Kazalo tabel

1 UVOD

Za temo svoje diplomske naloge sem izbral plagiatorstvo iz preprostega razloga, gre namreč za temo, s katero sem se že večkrat srečal v svojem študijskem in šolskem času. Seveda ne morem trditi, da nikoli nisem plagiiral nobenega izmed svojih literarnih izdelkov, in sem tudi prepričan, da je ljudi, ki lahko to trdijo, izredno malo. Vsakdo izmed nas je v času šolanja prav gotovo kaj prepisal od katerega sošolca ali pa v svojem izdelku navedel dejstva, za katere ni podal vira, iz katerega je dejstva pridobil.

Zaradi razširjenosti ter široke uporabe interneta je dandanes zelo preprosto ustvariti pisni izdelek, ki se razteza čez več strani, prav tako se je težko izogniti skušnjavi, da ne bi dela izdelka ustvarili s plagiacijo izdelkov drugih avtorjev. Za odkrivanje takšnih izdelkov poznamo kar nekaj programov, s pomočjo katerih lahko primerjamo večje količine dokumentov ali izvorne kode. Vsak od teh programov ima unikatne lastnosti, zato je težko najti najprimernejšega, ki bi ustrezal našim potrebam.

V drugem poglavju bom poskušal predstaviti problem plagiarizma v današnjem času, navesti področja, na katerih se danes srečujemo s tem pojavom, in predstaviti definicije – kaj je plagiarizem, kaj je plagiat in kdo plagiator. V tretjem poglavju predstavljam nekaj programov za primerjanje datotek in iskanje plagiatov na področju tekstovnih dokumentov kakor tudi izvorne kode. Sledi poglavje, v katerem predstavljam program, ki sem ga sam razvil. Opisani so pristopi, ki sem jih uporabil za primerjanje datotek, ter funkcionalnosti programa, s katerim je iskanje plagiatov olajšano. V petem poglavju bom predstavil rezultate svojega programa na nekaterih dokumentih ter primerjal dobljene rezultate primerjave z rezultati, ki jih za enake dokumente dobimo s pomočjo programov, predstavljenih v tretjem poglavju. V zadnjem poglavju sledijo še moja

spoznanja ob študiji obstoječih programov in razvoju lastnega programa ter predlogi za nadaljnje delo na tem področju.

Najpomembnejši cilj, ki ga želim doseči z diplomskim delom, je razvoj aplikacije, ki bi služila možnim uporabnikom kot orodje za iskanje plagiatov in s tem zmanjšala količino plagiacije. S tem bi se študente prisililo, da obveznosti v času študija opravijo samostojno, brez nepredvidene pomoči.

2 PLAGIATORSTVO

V svoji diplomski nalogi sem se posvetil predvsem akademskemu plagiatorstvu, natančneje literarnemu akademskemu plagiatorstvu. Izdelki, ki so najbolj ogroženi za literarno plagiatorstvo, so seminarske naloge, diplomske naloge in razna poročila projektov, do plagiarizma pa lahko pride tudi v raznih esejih in spisih. Z akademskim plagiatorstvom pa se ne srečujemo le na fakultetah, temveč tudi v osnovnih in srednjih šolah. Za vedno večje možnosti plagiatorstva obsojam razširjenost interneta. Na internetu lahko ob dodatni literaturi najdemo tudi obstoječe izdelke na temo, s katero se ukvarjamo. Kot dodaten razlog pa lahko omenimo še nezainteresiranost in lenobo učencev, dijakov in študentov, a to sodi v področje psihologije, zato o tem tukaj ne bomo govorili.

S plagiatorstvom se srečujemo tudi drugje, ne le na literarnem akademskem področju. Omeniti je potrebno plagiatorstvo pri razvoju programske opreme, s katerim se kot študent fakultete za računalništvo in informatiko velikokrat srečujem. Med drugimi področji, pri katerih se srečujemo s plagiatorstvom, lahko omenimo tudi novinarstvo, internetne objave in zabavno industrijo, kot sta glasbena in filmska.

2.1 DEFINICIJE

Preden se lahko sploh lotimo obravnave takšne teme, kot je plagiatorstvo, moramo najprej razčistiti, kaj pojem plagiatorstvo sploh pomeni ter kaj pomenijo pojmi, ki so z plagiatorstvom povezani. Ob iskanju definicije samega pojma se najprej obrnemo na Slovar slovenskega knjižnega jezika [1]:

- **Plagiatorstvo**: *„dejavnost plagiatorjev“*

- **Plagiator**: „*kdor prepiše, prevzame tuje delo in ga objavi, prikaže kot lastno, navadno v književnosti*“

- **Plagiat**: „*kar je prepisano, prevzeto od drugod in objavljeno, prikazano kot lastno, navadno v književnosti“*

S plagiatorstvom pa se ne srečujemo le v Sloveniji, zato je pravično, da si pogledamo, kakšne definicije poznajo drugod po svetu.

Definicija plagiatorstva po '*Random House Compact Unabridged Dictionary (New York)'* iz leta 1995, najdena na strani Dictionary.com [2]:

„*The unauthorized use or close imitation of the language and thoughts of another author and the representation of them as one's own original work.“*

Definicija po '*Merriam-Webster Online Dictionary*' [3]:

„To use (another's production) without crediting the source, to commit literary theft, to present as new and original an idea or product derived from an existing source. to steal and pass off (the ideas or words of another) as one's own.“

Še nekaj definicij:

Plagiarism, *as defined by the Little, Brown Essential Handbook for Writers, 3rd edition, is "the presentation of someone else's ideas or words as your own. Whether deliberate or accidental, plagiarism is a serious and often punishable offense" (Aaron 133).* [4]

Deliberate plagiarism *is "copying a sentence from a source and passing it off as your own and, summarizing someone else's ideas without acknowledging your debt, or buying a term paper and handing it in as your own" (Aaron 133).* [4]

Accidental plagiarism *is "forgetting to place quotation marks around other writer's words, omitting a source citation because you're not aware of the need for it, or carelessly copying a source when you mean to paraphrase" (Aaron 133).* [4]

To je le nekaj definicij pojma plagiatorstva. Vse navedene definicije prikazujejo enak pogled na plagiatorstvo, tako jih lahko povzamemo s stavkom:

PLAGIATORSTVO JE PREDSTAVLJANJE TUJEGA DELA KOT LASTNO.

3 PROGRAMI ZA DETEKCIJO PLAGIATOV

Programe, ki jih uporabljamo za detekcijo plagiatov v šolstvu, lahko delimo v dve skupini: na programe za primerjavo programske kode in programe za primerjavo tekstovnih datotek. Poudarek v tej diplomski nalogi leži na programih za primerjanje tekstovnih datotek, saj je program, ki sem ga sam razvil, namenjen le-temu. Med programi, ki sem se jih odločil opisati in natančneje predstaviti, sem izbral tudi tri programe za primerjanje programske kode. Vsi programi za odkrivanje plagiarizma, ne glede na to, ali iščejo plagiate med tekstovnimi datotekami ali med datotekami izvorne kode, morajo imeti tri pomembne lastnosti [5]:

1.) Nedovzetnost na oblikovne znake (presledki, tabulatorji, nove vrstice) – s tem je mišljena predvsem sposobnost odkrivanja primera plagiacije ne glede na to, koliko takšnih znakov vstavimo med posamezne besede.

2.) Nedovzetnost na šum – pomeni, da morajo biti ujemajoči odseki besedil ali izvorne kode dovolj dolgi, da lahko govorimo o plagiaciji. O omenjenem pojavu ne govorimo, če gre za naključno pojavitev besede ali skupine besed v dveh različnih dokumentih.

3.) Neodvisnost od položaja – programi morajo najti plagiirane odseke ne glede na to, kje v dokumentu se nahajajo. S tem se odkrijejo tudi plagiati v primerih, kadar del besedila prestavimo iz začetka dokumenta na konec. Ta lastnost velja v programih za iskanje plagiatov za najpomembnejšo in najbolj kritično.

Programi, s katerimi primerjamo različne izdelke, naj bodo to tekstovne datoteke ali datoteke, ki vsebujejo programsko kodo, ne določijo, kateri izdelek je original in kateri je plagiat, temveč uporabniku le podajo informacijo o podobnosti dveh datotek in kaj je med njima podobno. Tako mora uporabnik sam določiti, ali je v primeru prišlo do plagiarizma ali ne. Določiti moramo mejne vrednosti o podobnosti dveh datotek. Vprašanja, s katerimi se mora uporabnik glede na rezultate primerjave ubadati, so predvsem: Ali je prišlo do naključne enakosti dela besedila? Ali je prišlo do prepisovanja enega avtorja od drugega in ali sta oba avtorja prepisovala iz istega vira? Ali gre pri enakem delu besedila za citate, dejstva, ki jih ni mogoče drugače zapisati? Ali je stopnja plagiatorstva oziroma enakosti teksta dovolj visoka za ukrepanje?

3.1 PROGRAMI ZA PRIMERJANJE TEKSTOVNIH DATOTEK

Preden sem se lotil razvoja lastnega programa za primerjanje datotek, je bilo najprej potrebno preveriti, kaj nudijo že obstoječi programi s tega področja. Programov, ki jih lahko uporabljamo za primerjanje vsebine datotek med seboj ali primerjanje vsebine s spletnimi viri, lahko v velikem številu najdemo na internetu. Nekatere izmed njih lahko uporabljamo zastonj, za druge je potrebno kupiti licenco. Ob nakupu moramo pretehtati lastnosti, da se lahko odločimo za tistega, ki najbolje služi našim potrebam. Preveril sem nekaj programov. Zanimalo me je, kako delujejo, kakšne možnosti dajejo uporabniku, katere datotečne formate podpirajo in kakšne rezultate vračajo. V naslednjih podpoglavjih se bom dotaknil nekaj teh programov, za katera sem mnenja, da sodijo med boljše programe ter jih poskušal opisati in jih primerjati z lastnim programom. Naletel sem tudi na veliko majhnih, preprostih programov, za katere razvijalci trdijo, da so sposobni služiti namenu iskanja plagiatov, vendar menim, da ti programi nudijo uporabniku premalo nastavitev glede primerjav, da bi jih lahko uporabljali za širše področje oziroma za večje potrebe iskanja plagiatov, kot jih potrebujemo na fakulteti. Izpostavil bi dva programa, ki sta mi bila v veliko pomoč pri razvoju lastnega, saj sta mi dala kar nekaj idej in smernic,

česa se naj držim. To sta programa DOC Cop in WCopyFind.

3.2 DOC COP

Program DOC Cop [6] je spletna aplikacija, namenjena primerjavi tekstovnih datotek, kar pomeni, da za uporabo te aplikacije ne potrebujemo nobene dodatne programske opreme. Program primerja datoteke, ki jih naložimo v spletno aplikacijo, in nato ustvari poročilo primerjave, tako da uporabniku prikaže enak tekst, ki ga najdemo v dveh izmed naloženih datotek. Program je uporabnikom na voljo zastonj, ni pa odprto koden, zato se nisem mogel poglobiti v sam algoritem primerjave, ki ga uporablja, in sem se zanašal le na dokumentacijo ter podatke s spletne strani programa.

Razvijalec DOC Cop-a je Mark McCrohon, ki je razvil program v času svoje zaposlitve na eni izmed avstralskih univerz, kjer je opazil potrebo po orodju za detekcijo plagiarizma zaradi dogovarjanja med tamkajšnjimi študenti. Uporaba DOC Cop programa je zelo preprosta. Obiščemo spletno stran programa, kjer se moramo pred prvo uporabo registrirati. Registracija sestoji iz dveh korakov, najprej vnesemo veljavni lastni email naslov, na katerega dobimo tako imenovani 'Guest ID' oz. identifikacijsko številko gosta, s katero lahko uporabljamo program. Nato lahko naložimo datoteke, ki jih želimo primerjati, in počakamo na rezultate primerjave. Med najpogostejše uporabnike sodijo akademiki, profesorji, študenti, uredniki, novinarji, avtorji, raziskovalci, izdajalci literature, saj je program na voljo vsakemu, ki želi primerjati tekstovne datoteke.

Kako algoritmi primerjave delujejo, zaradi nedostopnosti kode žal ni bilo mogoče ugotoviti. Kar se da razbrati iz dokumentacije in uporabe programa, je, da program ob primerjavi datotek med seboj nudi primerjavo s spletnimi viri oz. nudi iskanje teksta po internetu. Glede na objavljeno dokumentacijo je bilo možno ugotoviti, da za primerjanje

datotek program uporablja trikoračen algoritem vzorčenja in osredotočenega odkrivanja, s pomočjo katerih se izvaja primerjava le vsebinsko podobnih dokumentov. Pri primerjanju s podatki s spleta pa se uporablja algoritem koščkanja in nadzorovanega vračanja rezultatov.

Program ima tudi nekaj omejitev, ki jih moramo, preden pričnemo primerjati datoteke, poznati. Datoteke, ki so med seboj primerljive, morajo biti v formatu Microsoft Word ali Adobe PDF, in tekst, ki ga želimo primerjati, mora biti napisan in ne podan kot slika teksta. Ob primerjavi s spletnimi viri moramo upoštevati tudi omejitve glede dolžine besedila. Datoteke za spletno primerjavo ne smejo vsebovati več kot 500 ali manj kot 20 besed. Pri primerjanju s spletnimi viri je podana tudi informacija, kako se besedilo razdeli na manjše odseke. V primeru, da ima dokument dolžino 500 besed in želimo primerjati 10 besed dolge odseke, se dokument razdeli na 491 različnih odsekov, ki se nato primerjajo.

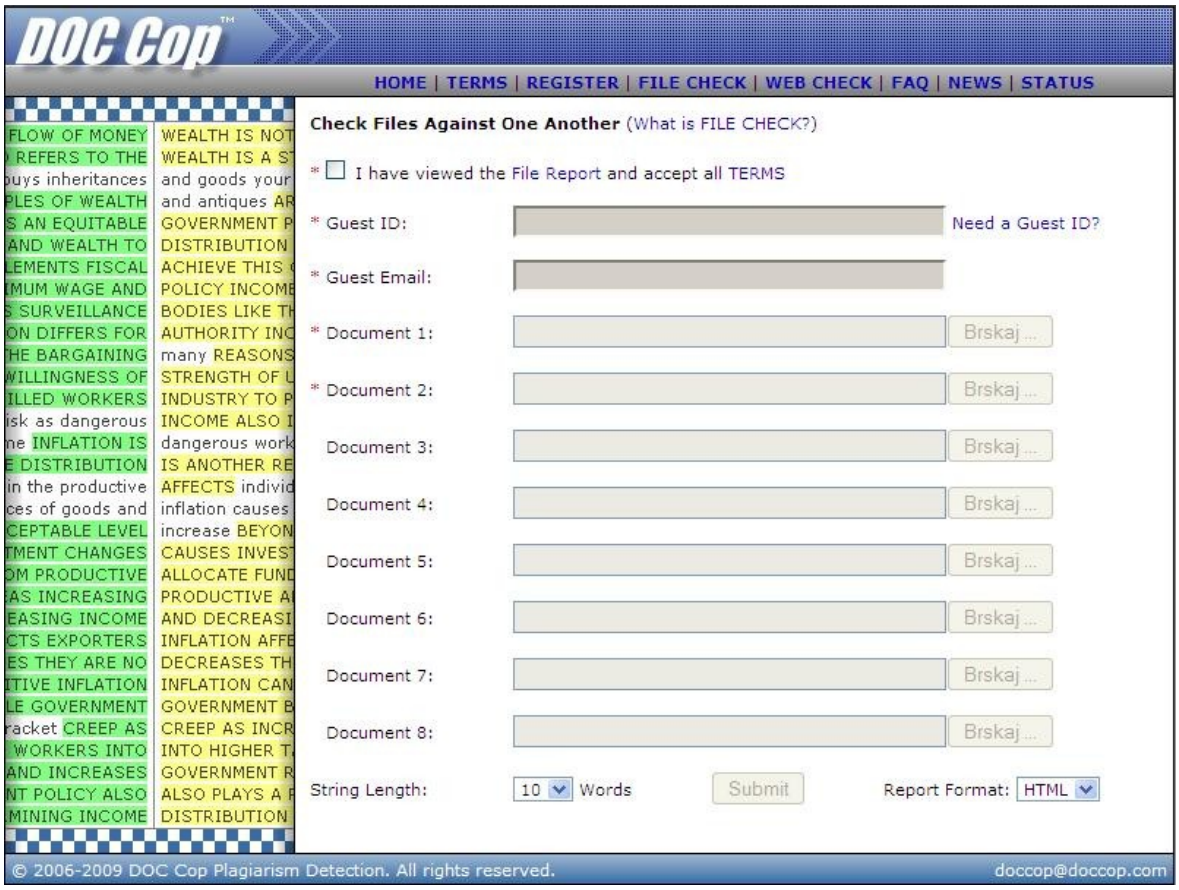
Po opravljeni primerjavi uporabnik prejme poročilo v obliki HTML- ali WORD-dokumenta. V poročilu imamo prikazano vsako primerjavo, ki je bila izvedena in je podala pozitiven rezultat podobnosti – oba besedila, označen enak tekst ter procent enakosti dokumentov. Ob primerjavi dveh dokumentov kot rezultat dobimo višji procent. To pomeni, da lahko dobimo za primerjavo dokumenta A z dokumentom B dve različni podobnosti, in sicer vsebovanost dokumenta A v B in vsebovanost dokumenta B v A. Pri taki primerjavi dobimo le rezultat višje vsebovanosti.

S tem poskušajo opozoriti na dejstvo, kateri dokument bi lahko bil plagiat in kateri original, kar pa ni nujno pravilno. S tem se kaže tudi slabost programa, saj ne poznamo obeh rezultatov. Sama primerjava traja sicer le nekaj minut, vendar zaradi obremenitve serverja, ki obravnava dokumente, je na poročilo potrebno čakati tudi do eno uro, kar lahko izpostavimo kot dodatno slabost.

Pri uporabi DOC Cop-a se pojavi vprašanje lastništva in zaupnosti dokumentov, saj gre za spletno aplikacijo in moramo poslati dokumente izven lastnega omrežja oz.

računalnika. Primerjava se vrši v Melbournu na serverju v pisarnah DOC Cop-a. DOC Cop sicer zagotavlja, da si ne jemlje kakršnih koli avtorskih pravic ali lastniških pravic nad prejetimi izdelki in ne obdrži izdelkov po končani primerjavi.

Študija DOC Cop-a je pripeljala k spoznanju, da mora biti moj program sposoben obdelovati datoteke različnih formatov ter da je potrebno uporabniku vračati poročila s podatki o opravljeni primerjavi. Uvedeno je bilo podobno deljenje besedila na manjše dele, kot je to uporabljeno pri DOC Cop-ovem primerjanju s spletnimi viri.



Slika 3.1: Program DOC Cop

3.3 WCOPYFIND

WcopyFind [7] je za razliko od programa DOC Cop namizna aplikacija, ki jo lahko snamemo z interneta in je zastonj. Avtorji ponujajo tudi izvorno kodo programa, ki je napisana v programskem jeziku C++. Program WCopyFind lahko zaganjamo na operacijskih sistemih Windows in Linux, razvili so ga na univerzi v Virginii.

Namen programa WCopyFind je primerjava zbirke dokumentov, pri katerih sumimo plagiarizem. Program iz dokumentov izvleče tekst in primerja odseke teksta med dokumenti za natančno ujemanje, nato ustvari poročilo o opravljeni primerjavi v obliki HTML dokumenta, v katerem si lahko ogledamo, kateri deli dokumentov se ujemajo ter procent podobnosti med vsakim parom dokumentov.

Lahko bi rekli, da je WCopyFind za razliko od DOC Cop-a namenjen zahtevnejšim uporabnikom, saj mu nudi veliko več opcij, s katerimi lahko opravlja primerjavo datotek. Datoteke lahko primerjamo med seboj, lahko jih primerjamo s starejšimi datotekami, ki so bile že primerjane. Na tak način opravimo primerjavo seminarskih nalog v prvem študijskem letu. Naslednje leto imamo za študente enako nalogo, ki jo primerjamo z leto starejšimi itd. Starejših nalog med seboj ne primerjamo, saj njihove rezultate že poznamo. Poljubno in po potrebi lahko nastavljamo kar nekaj parametrov, ki jih pri programu DOC Cop ne moremo spreminjati.

Nastavljamo lahko:

- minimalno dolžino odsekov, za katero želimo preveriti podobnost;
- minimalno dovoljeno število ujemajočih odsekov;

- minimalno dolžino teksta v znakih;

- ignoriranje posebnih znakov (števila, ločila, obravnavanje velikih in malih črk kot enake);

- preskakovanje skupin znakov, ki niso besede;

- preskakovanje dolgih besed;

- določanje maksimalne različnosti med odsekoma.

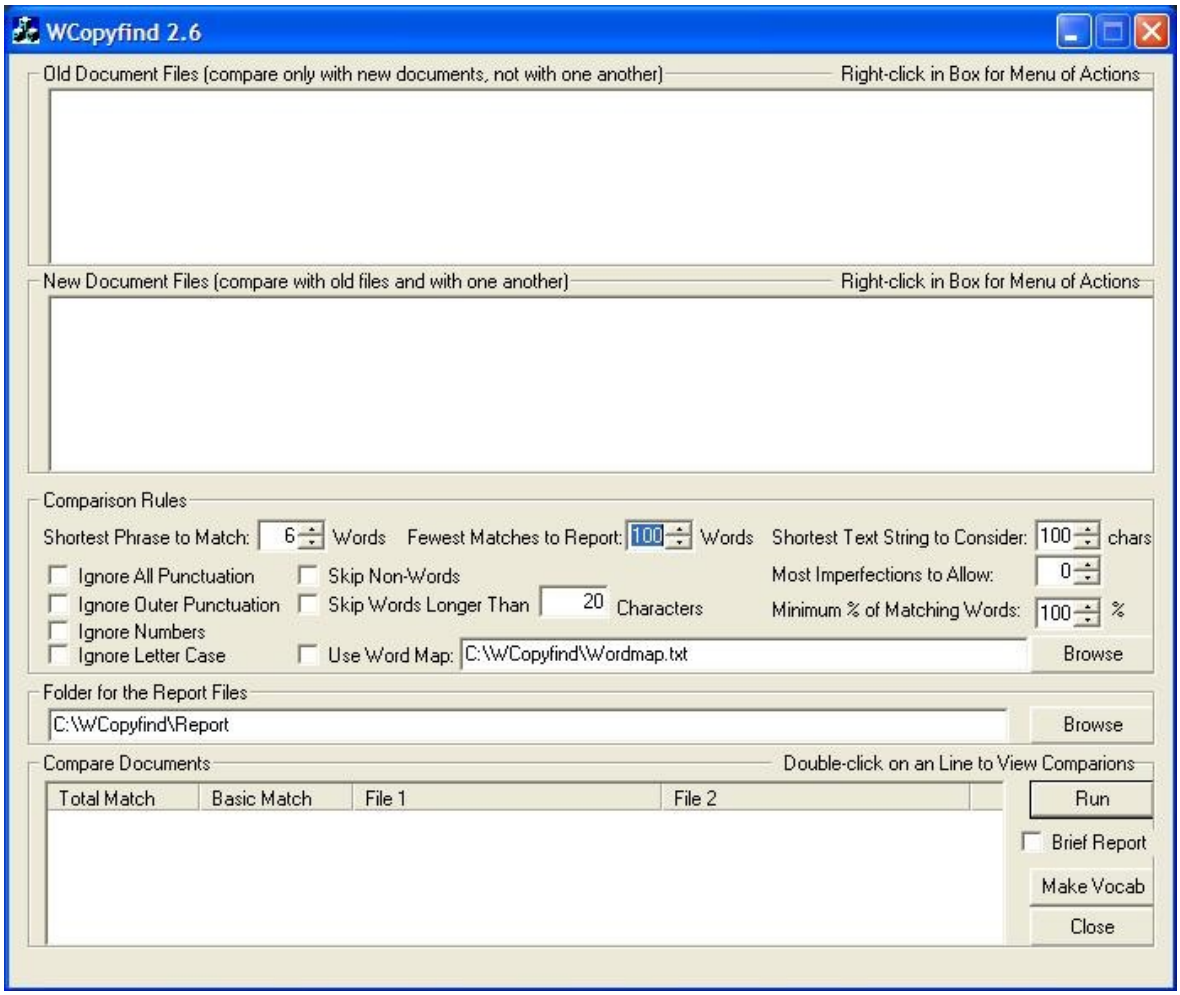
S pomočjo teh nastavitev lahko odločno vplivamo na rezultat primerjave, saj lahko odstranimo veliko motečih faktorjev in s tem dobimo natančnejšo primerjavo. Z omenjenimi nastavitvami moramo biti izredno previdni, kajti te vplivajo na čas same primerjave, prestrogi kriteriji lahko povzročijo prevelik procent podobnosti med datotekama. S tem so mišljene predvsem nastavitve za določanje različnosti med odseki besedil.

Te nastavitve predstavljajo prednost v neposredni primerjavi z DOC Cop-om glede samih primerjav, prinesejo pa tudi slabost glede uporabniške prijaznosti. Uporabnik potrebuje kar nekaj časa, da se seznani z aplikacijo, preden jo prične efektivno uporabljati. WCopyFind ima še nekaj drugih omejitev, ki mu jih lahko prištejemo k slabostim. Podpira le datotečne formate .doc in .txt ter nima podpore za iskanje in primerjavo z internetnimi viri. Na splošno lahko opravljamo primerjavo z bližnjicami do spletnih strani, kar bi sicer lahko imenovali omejena primerjava s spletnimi viri, a je težava v tem, da moramo poznati naslov strani, s katero bi radi primerjali dokument.

Program na voljo v odprto kodni obliki, zato je bilo možno tudi pogledati kako deluje sam

algoritem primerjave. Algoritem sem se odločil, s pomočjo povzetka algoritma v kodi, na kratko opisati. Sam algoritem vsebuje 900 vrstic kode, zato menim, da bi bil natančnejši opis v tem primeru predolg. Algoritem najprej prebere dokumente, ki jih želimo primerjati, in nato za vsakega od njih ustvari dinamično tabelo, v katero shranjuje vnose dokumenta. Nato prebere vsak posamezni dokument in za vsako besedo ustvari 32-bitno zgostitveno kodo, ki jih poveže v 3 sezname, s pomočjo katerih določa položaj vsake besede v besedilu. Šele nato sledi dejanska primerjava. Algoritem izbere dva dokumenta, ki jih je potrebno primerjati, in določi enega kot levega, drugega kot desnega. V levem dokumentu izbere prvo besedo, ki je daljša od treh znakov, ignorira tudi besede, ki se ne pojavijo v obeh dokumentih. Ko ima besedo, ki se nahaja v obeh dokumentih, si ogleda okolico besede, ki je izbrana v levem dokumentu, in to okolico primerja z okolico iste besede v desnem dokumentu. Ker se lahko posamezna beseda v dokumentu pojavi več kot enkrat, je potrebno preveriti vse enake besede v desnem dokumentu. Če se okolici ujemata, in ujemanje preseže maksimalno dovoljeno dolžino, potem se del besedila označi kot enak in primerjava se nadaljuje. Na koncu algoritem ustvari poročilo primerjave.

Vpliv WCopyFind-a na razvoj lastnega programa se opazi predvsem pri nastavitvah. Čeprav sem hotel obdržati čim večjo preprostost uporabniškega vmesnika in uporabe programa sem spoznal, da je marsikatera nastavitev, ki jih nudi WcopyFind, potrebna za dober program primerjanja datotek.



Slika 3.2: Program WCopyFind

3.4 PLAGIARISM DETECTOR

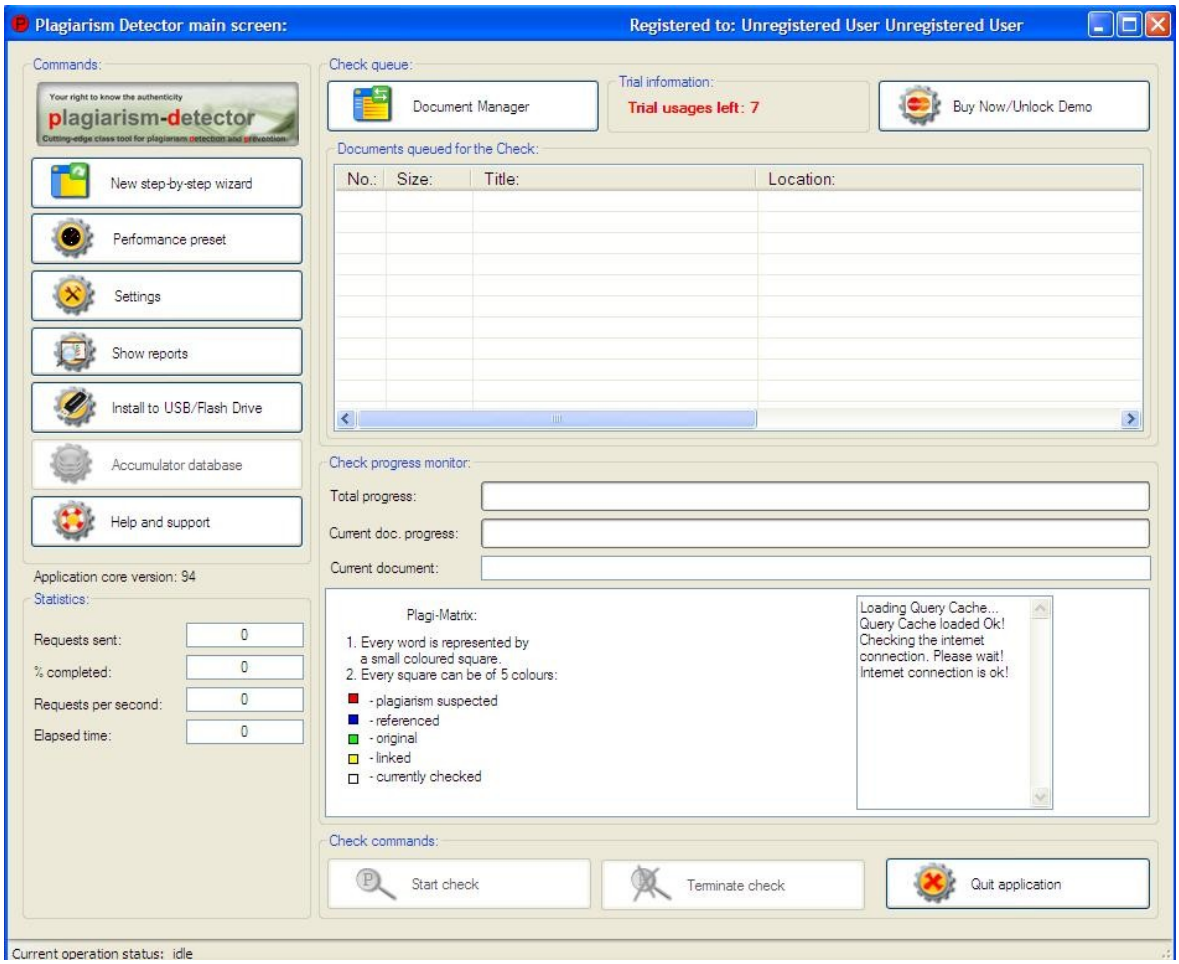
Plagiarism Detector [8] je ena izmed namiznih aplikacij za primerjanje datotek in odkrivanje plagiarizma. Program teče na operacijskem sistemu Windows in uporablja internetne iskalnike Google, Yahoo in Altavista za primerjavo datotek z viri na internetu. Aplikacija je plačljiva, a jo lahko testiramo v brezplačni demo različici.

Sama aplikacija je zelo „požrešna“, kar se tiče računalniških virov, časa primerjave, a ne porablja skoraj nobenih virov, ko primerjanje ne teče. V času primerjave lahko aplikacija zasede tudi do 100 % procesorske moči, kakor tudi celotno pasovno širino internetne povezave. Zasedenost računalniških virov je zelo odvisna od velikosti in količine dokumentov. Visoka poraba računalniških virov daje aplikaciji prednosti in slabosti. Prednost je opazna v hitrosti primerjave; če upoštevamo, da aplikacija preverja dokumente s kar tremi podatkovnimi bazami največjih internetnih iskalnikov in lahko 3 do 4 strani teksta preveri v 4 do 5 minutah. Slabost pa je tudi očitna; kadar zasedemo računalniške vire z eno aplikacijo, jih odvzamemo drugim aplikacijam, in tako oslabimo vzporedno delovanje drugih aplikacij.

Kljub temu da je za uporabo programa potrebno kupiti licenco, algoritem, ki ga uporablja, ni neznan. Program najprej prebere datoteko in iz nje izlušči tekst. Nato glede na uporabniške nastavitve iz teksta izbere odseke besedila določene dolžine na vsakih nekaj besed. Izbrane odseke besedila sestavi v seznam in za vsak odsek pošlje zahtevo za iskanje na internetne iskalnike. Rezultat, vrnjen od iskalnikov, je seznam strani, ki jih uporabi pri nadaljnji primerjavi. Vsak zaznan vir, ki bi lahko predstavljal vir plagiacije, se nato sname z interneta in primerja naprej. Algoritem uporablja tako imenovano „Plagi-Matrix“, kjer gre za matrico, v kateri se izvede primerjava besede za besedo. Za vsak potencialni vir plagiacije se ustvari lastna matrica, na koncu pa se vse združijo v končno matrico, ki je prikazana tudi v končnem poročilu. Po končani primerjavi dobimo rezultat, kolikšen del besedila predstavlja potencialno plagiatorstvo, kolikšen del je originalen, kaj je bil najdeno in kaj ne.

Podprtih je zelo veliko različnih datotečnih formatov, saj lahko ob Microsoft Word dokumentih primerjamo še Adobe PDF-dokumente, Powerpoint predstavitvene datoteke ter HTML- in PHP-internetne strani. Program nudi manj izbire nastavitev za uporabnika v primerjavi z WCopyFind. Nastavimo lahko, kako dolgi naj bodo odseki, ki jih želimo primerjati, kakšen naj bo razmik med posameznimi odseki. Poudaril bi še, da sem testiral

le demo različico programa, ki nudi le omejene funkcionalnosti in nastavitve testiranja. Program je uporabniško prijaznejši od WCopyFind in uporabniku nudi več statistike ter informacij pri poročilih.



Slika 3.3: Program Plagiarsm Detector

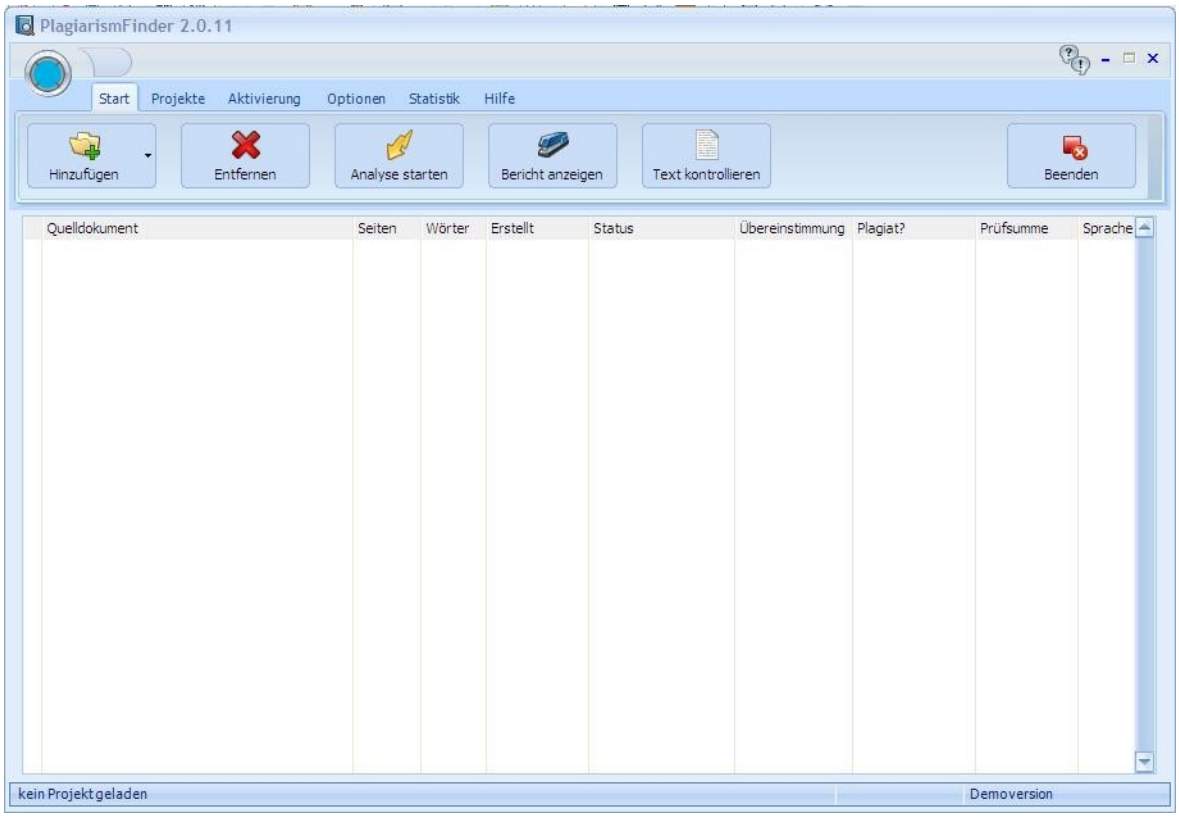
3.5 PLAGIARISM FINDER

Aplikacija Plagiarism Finder [9] je zelo podobna Plagiarism Detectorju. Je namizna aplikacija, je plačljiva in primerja datoteke s spletnimi viri. Razvijalec aplikacije je Gunter Wielage iz Mediaphor Software Entertainment AG iz Nemčije. Osnovni namen aplikacije

je preverjanje dokumentov za plagiate. Program teče na Windows operacijskih sistemih ter pošilja 9 povpraševanj na internetne iskalnike v sekundi.

Kot večina programov za primerjanje tudi Plagiarism Finder deluje na principu delitve besedila na manjše odseke. Podobnost z Plagiarism Detectorjem pa se kaže tudi v tem, da aplikacija uporablja internetne iskalnike kakor tudi podoben način delitve besedila s pomočjo določanja dolžine odsekov in razdalje med odseki. Zanimiv je tudi princip delovanja licenc. Razvijalci poudarjajo, da je uporaba internetnih iskalnikov izven spletnega brskalnika plačljiva, zato nudijo za vsako licenco 300.000 iskalnih nizov, kar je po njihovem mnenju ekvivalentno 50.000 do 100.000 preverjenim stranem dokumentov. Licenca pri povprečni uporabi naj bi tako veljala 2 do 3 leta, na kar je potrebno licenco podaljšati oziroma program ponovno kupiti. Ta lastnost daje programu veliko slabost, saj bi v primeru nabave programa za večjo organizacijo, kot na primer za fakulteto, licenca lahko potekla v manj kot enem letu.

Plagiarism Finder kakor preostale namizne aplikacije ne prenaša dokumentov na zunanje serverje in s tem zagotavlja, da z uporabo ne pride do kršenja avtorskih pravic. Sposoben je tudi obdelovati širok razpon dokumentov, od novejših Microsoft produktov, dokumentov s končnicami .docx, .pptx, xslx, starejših Microsoft produktov, dokumentov s končnicami .doc, .ppt, .xsl do PDF-dokumentov, kakor tudi HTML- in .txt dokumentov. Program avtomatsko razpozna šest različnih jezikov dokumenta, kot so angleščina, nemščina, italijanščina, francoščina, in je zmožen obdelati do 1000 dokumentov hkrati, kakor tudi dokumente z do 1000 stranmi. Nastaviti je možno želeno dolžino odsekov, primerjati in nastaviti dolžino korakov med posameznimi odseki. Nudi možnost izločanja izbranih spletnih strani z iskanja.

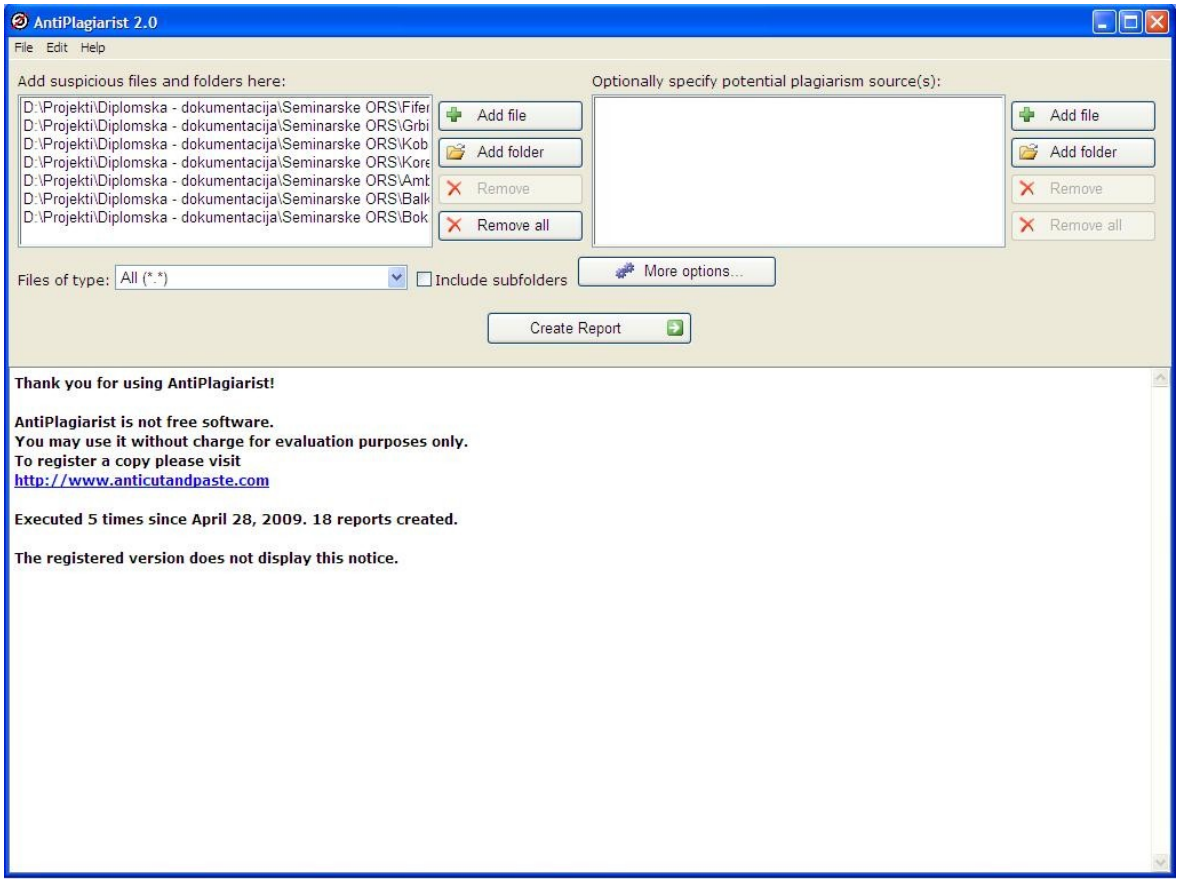


Slika 3.4: Program Plagiarism Finder

3.6 ANTI-PLAGIARIST

Aplikacija Anti-Plagiarist [10] je namizna aplikacija, zmožna primerjati sklop dokumentov med seboj. Podpira le nekaj vrst dokumentov, kot so Microsoft Word dokumenti, HTML-dokumenti in ter dokumenti s končnicami .txt in .rtf. Ob primerjavi se sproti izpisujejo podobni deli teksta, ki kažejo na možno plagiatorstvo. Nastavitve, ki jih lahko ob primerjavi uporabljamo, so zelo omejene, saj lahko nastavljamo le dolžino odseka besedila, ki se mora ujemati, lahko pa si priredimo še obliko poročila. Poročila so ne glede na obliko zelo nepregledna, saj izvemo le podatke o podobnih odsekih datotek, nimamo pa informacije, kako velike so datoteke in kolikšen procent celotne datoteke predstavlja podoben odsek. Ob samem testiranju aplikacije sem velikokrat naletel na težave z

obdelavo datotek, ki so prekinile delovanje aplikacije, zaradi česar ocenjujem, da je aplikacija neprimerna za resnejšo uporabo.



Slika 3.4: Program Anti-Plagiarist

3.7 PRIMERJALNA TABELA PROGRAMOV ZA PRIMERJANJE DATOTEK

V naslednji tabeli sem poskušal združiti najpomembnejše lastnosti opisanih programov in jih postaviti v neposredno primerjavo. V primerjavo nisem vključil lastnega programa, saj so njegove lastnosti opisane šele v poglavju 4.

Tabela 3.1: Primerjalna tabela programov za primerjanje datotek

Lastnost programa	DOC CoP	WCopyFind	Plagiarism Detector	Plagiarism Finder	Anti Plagiarist
Podprti formati dokumentov	.doc, .pdf,	.doc, .txt	.doc, .docx, .pdf , .rtf, .html, .htm, .php, .ppt	.docx, .pptx, .xslx, .doc, .ppt, .xsl, .txt, .html,	.doc, .html, .txt, .rtf
Cena	zastonj	zastonj	$35$140	149€ ali 299€	$49,95
Primerjave	Datoteke med seboj in s spletnimi viri	Datoteke med seboj, s poznanimi spletnimi viri	S spletnimi viri s pomočjo internetnih iskalnikov	S spletnimi viri s pomočjo internetnih iskalnikov	Datoteke med seboj
Vrsta aplikacije	Spletna	Namizna	Namizna	Namizna	Namizna
Zahtevan operacijski sistem	Deluje preko browserja (IE, Firefox) potrebuje JavaScript	Windows XP, Windows Vista, Linux distribucije	Windows XP, Windows Vista	Windows XP, Windows Vista	Windows XP, Windows Vista
Vrsta poročila	Word ali HTML-dokument	HTML-dokument	HTM-dokument ter grafični prikaz v aplikaciji	HTM-dokument ter grafični prikaz v aplikaciji	Prikaz v aplikaciji

3.8 DIFF-PROGRAMI

Kadar govorimo o programih za primerjanje datotek, se ne moremo izogniti tako imenovanim DIFF-programom [11]. Programi so sposobni ugotavljati razlike med dvema dokumentoma ali razlike med dvema verzijama istega dokumenta. Ti programi primarno niso namenjeni odkrivanju plagiatov, vendar jih lahko uporabljamo za ta namen. Težava je v dejstvu, da se ne da primerjati več kot dva dokumenta hkrati, kar pomeni, da bi morali

primerjati vsako kombinacijo dokumentov posebej, kar prinese veliko dodatne režije in dela, če imamo večji sklop dokumentov. Kadar imamo le nekaj dokumentov, lahko tudi s pomočjo teh programov odkrijemo možne primere plagiacije, kar se kaže tako, da program najde čim manj sprememb med dokumentoma. Težava pri teh programih je še dejstvo, da program ne vrne odstotkovne razlike med dokumentoma.

Algoritem, uporabljen v večini DIFF-programov, temelji na reševanju problema najdaljšega skupnega podzaporedja. Pri programih DIFF obravnavamo en dokument kot original in enega kot novo verzijo, pri čemer si beležimo le razlike, ki jih najdemo v novi verziji. Rezultat programa so podatki, ki povedo, kaj je bilo spremenjeno, kaj dodano in kaj odstranjeno v primerjavi z originalnim besedilom.

Testiran je bil tudi eden izmed DIFF-programov, z namenom, da bi lažje primerjali takšne programe s programi za odkrivanje plagiatov. Program, ki je bil testiran, se imenuje DOC DIFF. Program podpira veliko različnih formatov za primerjanje. Primerjamo lahko Microsoft dokumente, predstavitvene datoteke, PDF-dokumente, dokumente z končnicami .txt in .rtf, kakor tudi HTML-dokumente. Nastavitve, ki jih je program ponujal, so bile sicer zadovoljive, saj je možno odstraniti razne posebne znake kakor tudi števila, nudi pa tudi ignoriranje razlikovanja med velikimi in majhnimi črkami. Naletel sem še na eno slabost omenjenega programa, zaradi česar bi ta program lahko izključili iz skupine možnih programov za iskanje plagiatov, program je namreč izjemno počasen in ima težave z večjimi datotekami, ki vsebujejo ob samem tekstu tudi druge elemente, kot so slike in tabele. Program je za primerjavo dveh 10-stranskih datotek porabil več časa kot programi za iskanje plagiatov za obdelavo desetih datotek.

3.9 PROGRAMI ZA PRIMERJANJE IZVORNE KODE

S plagiatorstvom se srečujemo tudi na področju pisanja programov in pri razvoju aplikacij. Ob razvoju lastnega programa za detekcijo plagiatov sem razmišljal o razvoju

modula, ki bi bil zmožen preverjati izvorno kodo na plagiate, vendar sem to možnost opustil, kajti po določenem času, vloženem v razvoj takšnega modula, sem ugotovil, da je obseg algoritmov za uspešno preverjanje takšnih datotek prevelik. Podobnih algoritmov, kot jih uporabljajo programi za primerjanje datotek, ni mogoče uporabiti, takšen primerjalnik mora biti specifično vezan na določen programski jezik oziroma če želimo podpreti več programskih jezikov, je potrebno za vsakega napisati lasten primerjalnik.

Kljub temu sem se odločil predstaviti to tematiko ter nekaj programov s tega področja. Programi za primerjanje izvorne kode za uspešno ugotavljanje plagiacije uporabljajo žetone, s katerimi označujejo sestavne dele izvorne kode, razčlenitvena drevesa, grafe programske odvisnosti ter razne metrike [12]. Pri primerjanju izvorne kode se moramo zavedati, da imamo opravka z veliko višjimi podobnostmi kakor pri tekstovnem plagiatorstvu. Ponavadi namreč primerjamo manjše programe z nekaj 10 ali 100 vrsticami kode, ki morajo opravljati enako nalogo, zaradi česar so vsi programi, v kolikor so napisani v istem programskem jeziku, zelo podobni. Za primer lahko podamo podatek, da pri tekstovnem plagiarizmu lahko o plagiatu govorimo že pri 25–30 odstotni podobnosti dveh dokumentov, medtem ko pri 70–80 odstotno podobni, 200 vrstic dolgi datoteki izvorne kode, nikakor ne moremo govoriti o plagiarizmu.

3.10 MOSS

Sistem MOSS [13] je eden največjih sistemov za ugotavljanje plagiacije v izvorni kodi, kar kaže že podatek, da podpira preko 25 različnih programskih jezikov, med katere spadajo tudi C#, C, C++, Java, Matlab, Lisp in Python. Server, na katerem teče sistem MOSS, se trenutno nahaja na univerzi v Stanfordu in je v uporabi že od leta 1994. Sistem deluje kot internetni servis, ki ga lahko uporabljamo tako na Windows ali Unix/Linux operacijskih sistemih, ter je zasnovan za čim lažjo za uporabo. Sistem vrača rezultate v obliki HTML-dokumentov, v katerih so navedeni dokumenti, pri katerih se sumi plagiacija, ter označi dele dokumentov, ki so si podobni. Tako jih lahko hitro najdemo ob pregledu poročila. Iz primerjave se odstranijo tudi razne knjižnice, za katere se sklepa, da so

uporabljene v več dokumentih. Sistem je brezplačen in namenjen za nekomercialno uporabo, vendar zahteva registracijo uporabnika.

Čeprav je sistem MOSS prosto dostopen za uporabnike, algoritmi, ki jih uporablja, niso javni, vendar je na voljo članek, v katerem so opisane ideje, na katerih sistem temelji [5]. Uporablja se tako imenovani princip digitalnih prstnih odtisov in 'winnowing'. Aplikacija v digitalni prstni odtis shrani še položaj odtisa, ki je sestavljen iz številke dokumenta in številke vrstice, iz katere se ustvari odtis. V prvem koraku primerjave se zgradi seznam oziroma kazalo prstnih odtisov, v katerem se shranijo vsi odtisi iz dokumentov, ki jih med seboj primerjamo. Nato se ponovno naredijo odtisi za vsak dokument in te program išče po seznamu, ki je bil zgrajen predhodno. Rezultat drugega prehoda je seznam vseh ujemajočih odtisov za vsak posamezni dokument. Vsak izmed teh seznamov lahko vsebuje ujemajoče odtise iz različnih dokumentov, zato se odtisi uredijo glede na pripadajoči dokument. Kombinacije dveh dokumentov z več ujemanji se vrnejo kot rezultat uporabniku.

3.11 JPLAG

JPlag [14] je program za odkrivanje plagiarizma v datotekah izvorne kode in je bil razvit predvsem za uporabo na fakultetah in ustanovah, kjer se poučuje programiranje. Za razvoj programa JPlag je zaslužen Guido Malphol, ki je leta 1996 začel razvijati program v sklopu študentskega raziskovalnega projekta. Program deluje s pomočjo spletnih servisov, pri čemer si je potrebno namestiti klient aplikacijo za njihovo uporabo. Lastniki sistema ponujajo uporabnikom razvoj lastnega klienta za uporabo njihovih spletnih servisov. Ob instalaciji klient aplikacije si je potrebno ustvariti račun za uporabo. Za primerjavo je potrebno naložiti datoteke, ki jih želimo primerjati, s pomočjo klient aplikacije na server, kjer se izvrši primerjava. Po opravljeni primerjavi prejmemo poročilo, v katerem so zapisane vse podobnosti med primerjanimi datotekami.

Algoritem katerega JPlag uporablja pri primerjavi datotek temelji na žetonih. Deluje v dveh fazah, v prvi fazi najprej prebere datoteke in pretvori izvorno kodo v žetone, nato te žetone v parih primerja med seboj in določa podobnost parov. Pretvorba vrstic kode v žetone je odvisna od programskega jezika, podprti jeziki so Java, C#, C++ in C. Žetoni, ki se ustvarijo, morajo biti izbrani tako, da karakterizirajo pomen programa, kar pomeni, da presledki in komentarji niso pretvorjeni v žetone. Primerjava žetonov spominja na primerjavo tekstovnih nizov ter upošteva tri pravila:

- vsak žeton iz niza A se lahko ujema z enim žetonom iz niza B;
- podnizi se morajo najti ne glede na njihov položaj v nizu;
- daljši ujemajoči podnizi so vredni več kot krajši, saj so zanesljivejši in tako manj naključni.

Algoritem s pomočjo več iteracij skozi nize najprej poišče najdaljše nize in nato vedno krajše. Ujemajoči žetoni se označijo, da se jih pri naslednjih iteracijah ne obravnava več. Iteracije se ponavljajo tako dolgo, dokler ni več ujemajočih žetonov, kar se vedno doseže, saj se dolžina nizov manjša z vsako iteracijo. Pride lahko do naključnega ujemanja žetonov, kar je rešeno z določanjem minimalne zahtevane dolžine za ujemanje.

3.12 SID

Software Information Distance [15] je še ena izmed spletnih aplikacij za primerjanje datotek z izvorno kodo. Algoritem je bil sicer razvit za primerjavo genomov, vendar so razvijalci spoznali algoritma in ga razširili za uporabo na drugih področjih. Sistem nudi preprost spletni vmesnik za uporabo ter možnost delnega ujemanja. Podprta sta programska jezika Java in C++. Sistem je sposoben odkriti premeščanje metod, zamenjave imena spremenljivk, vstavljanje ter refaktorizacijo kode. Je brezplačen za uporabo, zahteva registracijo uporabnika.

Algoritem primerjanja temelji na metriki izpeljani iz Kolmogorove kompleksnosti. Enačba, na kateri temelji Kolmogorova metrika, ki predstavlja razdaljo med nizoma x in y [16]:

$$d(x, y) = \frac{K(x) - K(x|y)}{K(xy)} \qquad (3.1)$$

kjer je:

K(x) – je dolžina v bitih najkrajšega programa, ki ob praznem vhodu vrne izhod x [16]

K(x|y) – je dolžina najkrajšega programa, ki ob vhodu y vrne izhod [16]

K(x) – K(x|y) – količina informacij, ki jih ima y o x [16]

K(xy) – količina informacij, ki jih vsebuje, ni združen iz nizov x in y [16]

Natančneje sta definicija in izpeljava Kolmogorove metrike razložena v [16].

Sistem SID deluje v dveh korakih, najprej program pretvori datoteke, ki vsebujejo izvorno kodo, v obliko z žetoni. V drugem koraku je uporabljen algoritem, ki stisne žetone tako, da lahko hevristično ocenimo skupno količino informacij za vsak par programov, ki jih primerjamo, na podlagi Kolmogorove metrike. Na koncu so vsi pari urejeni glede na razdaljo podobnosti med dokumentoma. Program prikaže rezultate uporabniku, v katerih so za lažjo razpoznavo označeni podobni deli.

4 LASTEN PROGRAM ZA DETEKCIJO PLAGIATOV – APSS

Razvoj programa se je pričel kot študentski projekt iskanja primernega programa za primerjanje datotek in odkrivanje plagiatov. Ob študiji programov, ki omogočajo primerjanje večje količine tekstovnih dokumentov, sem prišel do ugotovitve, da noben izmed izbranih programov ne ustreza specifikacijam, ki so bile iskane, zato sem se odločil razviti svoj sistem za primerjavo datotek. V sledečem poglavju bi rad ta izdelek podrobneje predstavil, opisal uporabljene algoritme kakor tudi ponujene funkcionalnosti.

4.1 PREDSTAVITEV PROBLEMA

Razlogi, zaradi katerih obstoječi programi niso bili primerni za uporabo, so bili različni od programa do programa. Nekateri programi so zahtevali nakup licence za uporabo, drugi niso bili preprosti za uporabo, tretji niso ponujali dovolj raznolikih nastavitev, nekateri niso dajali zadovoljivih rezultatov.

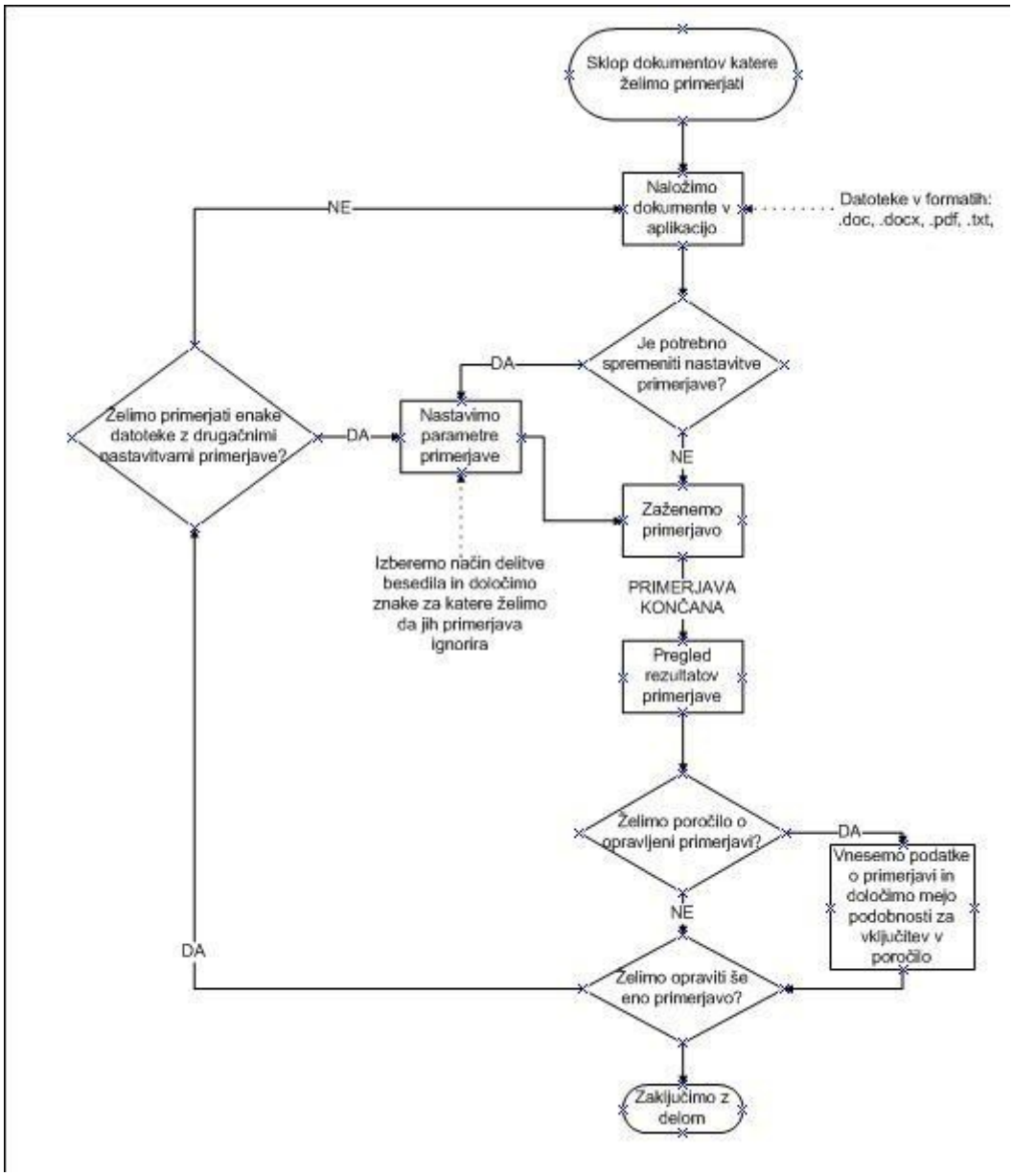
Najpomembnejše lastnosti, ki bi jih naj vseboval program, so bile: preprosta uporaba, podprtost različnih formatov za tekstovne datoteke, pregledna poročila o primerjavah ter hitro in učinkovito primerjanje velike količine dokumentov. Glede na te zahteve je bilo potrebno odkriti način, kako zagotoviti podprtost več različnih formatov tekstovnih dokumentov, ustvariti algoritem primerjave, ki bi bil tako zanesljiv kakor tudi hiter, premisliti je bilo potrebno, katere nastavitve dovoljevati uporabniku, ne da bi s temi nastavitvami nastal kompleksen uporabniški vmesnik, ter kako in kaj prikazovati

uporabniku kot rezultat primerjav. Pojavilo se je tudi vprašanje, ali naj bo aplikacija narejena kot namizna ali kot spletna.

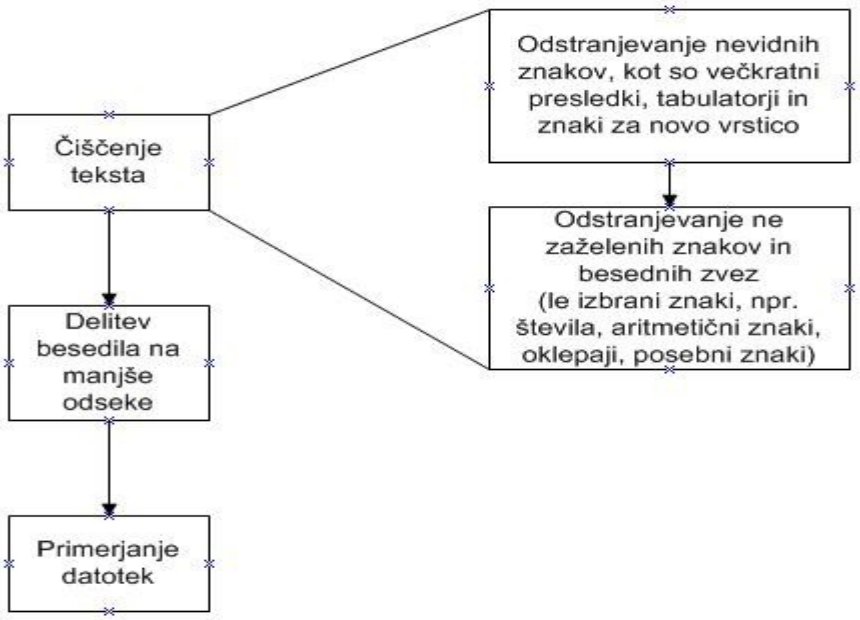
4.2 PRISTOP K REŠITVI

Na začetku razvoja je bilo potrebno definirati, v katerem programskem jeziku napisati aplikacijo in v katerem razvojnem okolju jo narediti. Pri izbiri programskega jezika sem izbiral med Javo in C#, saj jezika nudita razvoj preprostega uporabniškega vmesnika. Moja odločitev je bila programski jezik C# in s tem tudi razvoj v okolju .NET. Razlog za to odločitev so bile izkušnje z omenjenim programskim jezikom. Čeprav sem z izbranim programskim jezikom imel možnost lotiti se problema na objektno orientiran način, sem se zaradi hitrosti obdelave podatkov odločil nasprotno.

Delovanje aplikacije je predstavljeno s pomočjo diagrama poteka na sliki 4.1. V prvem koraku moramo naložiti datoteke, ki jih želimo primerjati. Aplikacija podpira več različnih formatov tekstovnih dokumentov, pri čemer delujejo vsi na principu ekstrakcije teksta, kar je podrobneje opisano v poglavju 4.3. Po nalaganju datotek izberemo nastavitve, za katere želimo, da bi bile upoštevane pri primerjavi dokumentov. Nastavitve, ki jih izberemo, se ohranijo po opravljeni primerjavi; tako lahko opravimo več primerjav z istimi nastavitvami in zamenjamo le sklop primerjanih dokumentov. Sledi operacija primerjanja, ki je sestavljena iz treh korakov, kar lahko vidimo na sliki 4.2. Najprej je potrebno počistiti besedilo, ga razdeliti na manjše odseke in opraviti dejansko primerjanje. Vsi koraki so podrobneje opisani v poglavjih 4.4, 4.5 in 4.6. Po opravljeni primerjavi dokumentov se v tabeli prikažejo rezultati za posamezne pare dokumentov, ki si jih ogledamo. Ponujena je tudi neobvezna možnost ustvarjanja poročila o primerjavi.



Slika 4.1 Diagram poteka za aplikacijo



Slika 4.2: Diagram poteka primerjanja

4.3 BRANJE DATOTEK IN PODPRTI DATOTEČNI FORMATI

Program je primarno namenjen za primerjanje tekstovnih datotek, zato mora biti sposoben brati tekstovne datoteke, da dobimo podatke za primerjavo. Tekstovne datoteke lahko ustvarjamo s pomočjo različnih programov za urejanje teksta, kar posledično privede k več datotečnim formatom. Eden izmed ciljev pri konstrukciji programa je podpora osnovnih datotečnih formatov, ki so najpogostejši v vsakdanji uporabi. Za branje teh različnih datotečnih formatov sem uporabil obstoječe odprtokodne rešitve, najdene na spletu.

Branje datotek je kljub vedno večjim hitrostim diskov še vedno zelo počasna operacija. Iz tega razloga se celotni tekst vsake prebrane datoteke shrani direktno v pomnilnik, kjer se zadržuje, dokler uporabnik ne naloži drugih datotek ali zapre programa. Pristop povzroča večjo zasedenost pomnilnika, prispeva pa k hitrosti obdelave podatkov, saj ni potrebno, da datoteko vedno preberemo ponovno, če jo želimo večkrat primerjati z

različnimi nastavitvami. Večja zasedenost pomnilnika ne predstavlja velike slabosti predvsem zaradi konfiguracij današnjih računalnikov in količine pomnilnika na voljo. To trditev lahko potrdim s podatkom, da je zasedenost pomnilnika pri 83 naloženih datotekah, ki obsegajo nekje 10 strani, približno 55 Mb.

a.) TEKSTOVNE DATOTEKE

Prvi izmed datotečnih formatov, ki jih program podpira, so navadne tekstovne datoteke, običajno napisane v beležnici ali podobnem programu, in shranjene s končnico .txt. Takšne tekstovne datoteke so najbolj idealen format za program. Branje teh datotek je preprosto, saj jih obravnavamo kot navaden tok podatkov, poleg tega format ne vsebuje veliko elementov za oblikovanje besedila, kar vpliva na hitrost branja, in s tem je branje takšnih datotek tudi najhitrejše. Zaradi hitrosti branja takšnih datotek jih dodatno uporabljamo kot varnostne kopije besedil. Ob branju datoteke podatke zapišemo še v začasne datoteke, shranjene v tem formatu, v začasni mapi. Te datoteke ohranjamo tako dolgo, kot uporabnik to želi, običajno do primerjave drugih tekstov. Datoteke ostanejo v mapi shranjene tudi po končanju programa in se ob naslednjem zagonu avtomatično naložijo v program. S tem si zagotovimo, da v primeru nepričakovanega zaključka programa, kot sta na primer izpad električne energije ali programska napaka, ni potrebno ponovno nalagati datotek v različnih formatih, kar bi povzročilo dodatno izgubo časa.

b.) DATOTEČNI FORMAT STAREJŠIH MICROSOFT WORD VERZIJ

Zaradi razširitve produktov Microsoft lahko trdimo, da datoteke, ustvarjene v različnih starejših verzijah programa Word, sodijo med najpogostejše. Sicer te datoteke zamenjujejo datoteke iz novejše verzije MS-Word, končnico .docx, ki se jim bomo posvetili v naslednjem podpoglavju. Pridobivanje teksta iz teh datotek je veliko bolj kompleksno kot v primeru navadnih tekstovnih datotek. Razlog za to je zgradba datotek, le-te vsebujejo podatke o vsebini, ki je za nas najbolj pomembna, in podatke o obliki vsebine, kakor tudi druge predmete, kot so tabele, slike, okvirji, itd. Za pridobivanje teksta

iz teh datotek si v našem programu pomagamo z odprtokodno rešitvijo za ekstrakcijo teksta z imenom Epocalipse Ifilter [17]. IFilter je vmesnik, ki ga je razvil Microsoft, in je namenjen predvsem za pridobivanje teksta. Vmesnik IFilter nam nudi funkcionalnost, s katero lahko v celoti preberemo vsebino datoteke in hkrati izloči vse informacije o obliki te datoteke. Rezultat branja je niz, ki ga potrebujemo za nadaljnjo obdelavo z našim programom.

c.) DATOTEČNI FORMAT NOVEJŠIH MICROSOFT WORD VERZIJ

Z izidom nove verzije Microsoft Word 2007 se je spremenil tudi datotečni format Word dokumentov. Čeprav uporabniku ni vidno nič drugega kot drugačna končnica dokumenta, je sprememba v samem zapisu zelo velika. Dokumente s končnico .docx je možno odpreti tudi s pomočjo programov za stiskanje in razširjanje datotek, kot je na primer program WinZip. Kadar odpremo dokument s takšnim programom, lahko vidimo njegovo sestavo, ki je zgrajena iz več XML-dokumentov. Vsak izmed teh XML-dokumentov vsebuje določene podatke; dokument za uporabljene pisave, dokument za obliko številčenja, dokument za uporabljene stile, za nas pa je najpomembnejši dokument z imenom 'document.xml', kajti le-ta vsebuje podatke o vsebini. Za ekstrakcijo te vsebine smo uporabili odprtokodno rešitev, imenovano DocxToText [18]. Potrebujemo tudi knjižnico, ki zna operirati s stisnjenimi datotekami, za kar smo uporabili SharpZipLib [19]. Datoteko, shranjeno v formatu DOCX, najprej odpremo kot stisnjeno datoteko in poiščemo datoteko z vsebino dokumenta. Zaradi XML-oblike te datoteke jo lahko zelo preprosto preberemo in iz listov XML-dokumenta pridobimo celotno vsebino dokumenta, ki jo sestavimo v en niz znakov, ki ga potrebujemo za nadaljnjo obdelavo.

d.) DATOTEČNI FORMAT ADOBE PDF

Še eden zelo popularnih formatov za shranjevanje tekstovnih datotek je format PDF. Kakor že omenjena Microsoft Word formata tudi PDF shranjuje ob vsebini datoteke tudi podatke o obliki samega teksta, zato je potrebno tudi v primeru teh dokumentov tekst

najprej ločiti od oblikovnih podatkov. Za ekstrakcijo teksta iz PDF-dokumentov se zatekamo k odprtokodnim rešitvam in uporabljamo tako imenovani PDFBox [20]. PDFBox je knjižnica metod, namenjena za programski jezik Java, obstaja pa tudi verzija za ogrodje .NET, ki deluje v kombinaciji z IKVM.NET [20]. IKVM.NET predstavlja implementacijo Jave za ogrodje .NET in vsebuje Virtualni stroj za Javo implementiran v .NET-u, .NET implementacijo Javinih razrednih knjižnic in orodja, ki omogočajo interoperabilnost Jave in .NET ogrodja. S pomočjo teh odprtokodnih rešitev se branje oziroma ekstrakcija teksta iz PDF-dokumentov prelevi v enostavno operacijo. Te knjižnice nam nudijo vse metode, ki jih potrebujemo za branje PDF-dokumenta in ekstrakcijo vsebine.

4.4 PREDOBDELAVA TEKSTA

V prvem koraku postopka smo prebrali besedila, ki jih želimo primerjati in ugotoviti njihovo medsebojno podobnost. Ta besedila imamo shranjena v pomnilniku v obliki tekstovnih nizov. Tekstovni nizi v tem koraku postopka vsebujejo veliko znakov in so v takšni obliki neuporabni za samo primerjavo. Primerjava bi vrnila rezultat podobnosti le v primeru, kadar se na vsakem položaju natančno ujemata znaka iz obeh primerjanih datotek. Da lahko niza primerjamo med seboj, jih moramo razdeliti na več krajših delov, kar imenujemo segmentacija besedila. Pred segmentacijo je potrebno opraviti še korak čiščenja teksta, kar predstavlja predobdelavo.

Omenil sem, da nizi, ki jih preberemo, vsebujejo veliko znakov. Za potrebe čiščenja teksta te znake razdelimo v tri skupine. Te so: ločila in nevidni znaki, nezaželeni znaki in zaželeni znaki. Pri čiščenju teksta gre predvsem za brisanje izbranih znakov, ki so iz skupin nevidni znakov in ločil ter nezaželenih znakov.

Z nevidnimi znaki imamo v mislih predvsem znake za novo vrstico, tabulatorje in presledke. Kadar preberemo besedilo kot tok podatkov, imamo v nizih zapisane tudi te

znake, ki jih lahko prepoznamo po oznakah /t, /r in /n. Ti znaki predstavljajo težavo pri primerjavi teksta predvsem zato, ker se jih uporablja za oblikovanje in lahko pride do primera v koraku segmentacije, da dobimo nize, ki so sestavljeni samo iz takšnih znakov ali pa se kakšen tak znak vrine med besede in tako spremeni segment. Takšni segmenti vplivajo na rezultat primerjave; kot primer lahko imamo dve vsebinsko popolnoma različni tekstovni datoteki in dobimo napačen oziroma previsok odstotek podobnosti, ker imamo v obeh datotekah niz sestavljen iz nevidnih znakov. Drug primer, kjer nevidni znaki vplivajo na rezultat podobnosti, je, kadar imamo tak znak vrinjen med besede. V tem primeru lahko imamo dva niza, ki se razlikujeta le v enem ali več nevidnih znakih in s tem povzročata nižji osdstotek podobnosti. Iz teh razlogov program iz vseh prebranih besedil pred primerjavo odstrani vse pojavitve nevidnih znakov in s tem preprečimo, da bi lahko primerjavo pretentali s preprostim vstavljanjem nove vrstice ali tabulatorjev. Z odstranjevanjem nevidnih znakov rešimo tudi problem presledkov. Govorimo o večkratnih presledkih, besedilo lahko vsebuje med besedami namesto običajnega, enega presledka, dva presledka ali celo več. Problem smo rešili na podoben način kot nevidne znake, zamenjamo vse večkratne pojavitve presledka z enojnim presledkom, kar se reši z uporabo regularnih izrazov. Onesposobimo pa še en način, ki bi lahko pretental primerjanje.

Ko obdelamo nevidne znake, imamo v nizu besedila lahko še vedno sklope nezaželenih znakov. Govorimo predvsem o številih in posebnih znakih, kot so razni oklepaji, narekovaji in aritmetični znaki. Glede na naravo in temo dokumentov, ki jih obdelujemo, se moramo odločiti, kateri znaki nas ne zanimajo. Pri dokumentih, kot so razna poročila in seminarske naloge, nas glede plagiarizma zanima predvsem vsebina dokumenta, kajti stavki oz. deli besedila, ki vsebujejo razna števila ali enačbe, so lahko definirani glede na tematiko ali standardizirani in so s tem lahko enaki v vseh primerjanih dokumentih. S pomočjo oklepajev bi lahko delno preoblikovali besedilo tako, da bi bil rezultat primerjave različen kljub enaki vsebini odseka besedila. Iz tega razloga se mi je zdelo smiselno uporabniku ponuditi izbiro znakov, ki jih naj primerjava upošteva ali ne. V nastavitvah same primerjave lahko uporabnik izbira med že vnaprej definiranimi znaki, ki

jih v primerjavi ne želi upoštevati, v seznam izbire lahko doda svoje znake ali celo besedne zveze, v kolikor želi iz primerjalnega besedila odstraniti naslove dokumentov, imena in priimke avtorjev ali mentorjev, naziv fakultete ali podjetja ali pa kakšne druge besedne zveze, za katere lahko določimo, da se bodo pojavljale pri več avtorjih. Uporabnik mora biti pazljiv, katere znake doda v seznam nezaželenih znakov, saj lahko s tem vpliva na samo vsebino dokumenta, če bi med nezaželene znake dodal pogosto pojavljajoče črke. Po odstranitvi nevidnih znakov v fazi čiščenja dokumentov se lotimo še vseh izbranih nezaželenih znakov, ki jih prav tako odstranimo iz besedila.

4.5 SEGMENTACIJA BESEDIL

Po opravljeni fazi čiščenja preidemo v fazo segmentacije besedil. V segmentaciji besedil opravljamo delitev besedila na manjše odseke, ki jih nato primerjamo z odseki iz drugih besedil. Vsa besedila, v sklopu katerega želimo primerjati, se segmentirajo na enak na način. Ne moremo namreč določiti, da bi se del besedila segmentiral drugače kot drugi. Razlog za to odločitev je preprost; če bi to dovoljevali, bi s tem lahko dobili nepravilne rezultate primerjave. Uporabnik ima na voljo izbirati med tremi različnimi vrstami segmentacije besedila, katero bo izbral, pa se mora odločiti pred samo primerjavo. Izbira načina segmentacije besedila vpliva na sam rezultat primerjave kakor tudi na hitrost primerjave.

Prvi način segmentacije je stavčna delitev. Pri stavčni delitvi razdelimo besedila na odseke različne velikosti, kjer upoštevamo, da se odsek prične za znakom, ki predstavlja stavčno ločilo, naj bo to pika, vprašaj ali klicaj, in konča s stavčnim ločilom. Ta način segmentacije besedila je najosnovnejši in velja kot priporočen za velike sklope besedil, predvsem kadar imamo opravka z daljšimi dokumenti, kot so na primer seminarske naloge. Stavčna segmentacija je tudi najhitrejša izmed treh načinov segmentacije, vendar lahko zniža procent podobnosti besedil ravno zaradi daljše dolžine odsekov besedila.

Drug način segmentacija je delitev glede na izbran znak. Segmentacija glede na izbran znak je v bistvu posplošena stavčna segmentacija, kajti če za znak izberemo piko, dobimo ravno omenjeno stavčno segmentacijo. Podobno kot pri stavčni segmentaciji so tudi pri tem načinu odseki različne dolžine in se pričnejo za izbranim znakom ter končajo z njim. Kadar izberemo segmentacijo glede na izbran znak, moramo biti pazljivi, da znaka, ki smo ga izbrali, ne dodamo med nezaželene znake, kajti s tem bi dobili za rezultat segmentacije en sam odsek, ki bi predstavljal celotno besedilo. Ob sami izbiri znaka moramo biti pazljivi na še dodatno težavo, on sicer če izberemo znak, ki se v besedilu pojavlja zelo pogosto, nastanejo zelo majhni odseki besedila, ki jih želimo primerjati, in s tem lahko dobimo napačne rezultate primerjave. Za primer si lahko predstavljamo delitev tega odstavka glede na črko 'e'. Dobili bi, ne samo zelo kratke odseke, temveč tudi delno neuporabne odseke, saj bi ponekod odseki vsebovali le del besed ali pa mogoče celo en sam znak.

Tretji način segmentacije je segmentacija glede na fiksno dolžino odsekov. Uporabnik lahko določi število besed, ki jih mora odsek vsebovati. Od izbire dolžine odseka pa vplivamo tudi na hitrost delovanja segmentacije; krajši, kot je odsek, dlje traja segmentacija, vendar pa lahko s krajšimi odseki povečamo stopnjo natančnosti primerjave. Verjetnost, da najdemo v kakšnem drugem dokumentu odsek šestih besed, je veliko večja, kot da najdemo odsek dolžine 13 besed. Pri določevanju dolžine odsekov moramo najti pravo razmerje med hitrostjo segmentacije in s tem tudi hitrostjo ter natančnostjo primerjave. Ob segmentiranju na odseke fiksne dolžine moramo upoštevati tudi smiselnost dolžine same. Dolžina vpliva na verjetnost, da najdemo v drugem dokumentu nek odsek. S krajšo dolžino je ta verjetnost večja in se zmanjšuje s podaljševanjem dolžine. S prekratko dolžino lahko odkrijemo podobnost dveh dokumentov, ki sta vsebinsko popolnoma različna. Zato je možna le nastavitev dolžine med 5 in 15 besedami.

Večkrat sem omenil težave z dolžino odsekov pri vseh načinih segmentacije. Glede na način segmentacije moramo ustrezno določiti dovoljeno mejo podobnosti dveh

dokumentov. Pri segmentiranju, glede na izbran znak, in stavčnem segmentiranju, kjer imamo opravka z odseki različnih dolžin, lahko pride tudi do situacije, kjer imamo zelo kratke odseke. Zato v tem načinu segmentiranja postavimo še dodaten pogoj, ki ga mora izpolnjevati vsak odsek. Odsek mora biti daljši od petih besed in daljši od 20 znakov. S tem dosežemo tudi izločitev internetnih naslovov iz besedila za primerjanje, kajti ob stavčni delitvi se deli tudi internetni naslov na več odsekov, od katerih pa noben ne izpolnjuje kriterija dolžine petih besed ali 20 znakov.

4.6 PRIMERJAVA BESEDIL

Po opravljeni fazi čiščenja in segmentacije besedila lahko preidemo na najpomembnejši del aplikacije, to je primerjava odsekov besedila. Aplikacija omogoča medsebojno primerjavo več dokumentov. Z višjim številom dokumentov raste število primerjav, ki morajo biti opravljene. Za zanesljiv rezultat moramo vsak dokument v sklopu besedil primerjati z vsakim. Za par dokumentov mora biti primerjava opravljena v obe smeri, kajti zaradi segmentacije lahko pride do primera, v katerem najdemo odsek le v eni smeri.

Primeri, kadar lahko naletimo na enosmerno najdbo odseka:

- daljši stavek razdelimo v drugem dokumentu na dva krajša stavka;

- pri segmentiranju na fiksno dolžino imamo odsek sestavljen iz dveh stavkov in ga zato ne najdemo v drugem dokumentu, v drugem dokumentu pa imamo podoben odsek, ki je prestavljen za eno besedo in ga zato v prvem dokumentu najdemo.

Iz te ugotovitve sledi, da lahko število primerjav izračunamo po formuli:

$$število\ primerjav = število\ dokumentov * (število\ dokumentov - 1) \quad (4.1)$$

Zato je bilo potrebno izdelati algoritem, ki opravlja te primerjave brez večje izgube časa pri obdelavi.

Preden se lotimo razlage samega algoritma primerjave, je potrebno podati, katere podatke imamo na voljo. Operiramo namreč s seznamom celotnih besedil, ki smo ga pridelali s čiščenjem besedila, ter s seznami odsekov za vsako posamezno besedilo.

PSEVDOKOD ALGORITMA:

```
function Primerjaj_besedila(besedila)
begin
int zadetki
for i = 0 to stevilo_besedil
        seznam_odsekov = besedila[i].odseki
        for j = i to stevilo_besedil
                zadetki = 0
                for k = 0 to stevilo_odsekov
                    if (Vsebuje_odsek(besedila[j].vsebina, seznam_odsekov[k])
                        zadetki++
                end
                double podobnost=IzracunajPodobnost(zadetki,
                                        seznam_odsekov.length);
                zadetki = 0
                 seznam_odsekov = besedila[j].odseki
                for k = 0 to stevilo_odsekov
                    if (Vsebuje_odsek(besedila[i].vsebina, seznam_odsekov[k])
                        zadetki++
                end
                double podobnost=IzracunajPodobnost(zadetki,
                                        seznam_odsekov.length);
        end
```

```
end



function VsebujeOdsek(besedilo, odsek)
begin
        if (besedilo.contains(odsek))
                return true
        else
                return false
end

function IzracunajPodobnost(stevilo_zadetkov, stevilo_odsekov)
        return (stevilo_zadetkov / stevilo_odsekov) * 100
```

Psevdokod je zelo posplošen za lažje razumevanje samega algoritma. V principu pa gre za zelo preprost postopek. Najprej izberemo odseke besedila, ki ga želimo primerjati, nato izberemo še besedilo, s katerim želimo opraviti primerjavo in je v seznamu besedil za tistim, katerega odseke smo izbrali. Za vsak odsek iz seznama preverimo, ali se nahaja enak del teksta, kot je v odseku, tudi v izbranem besedilu drugega dokumenta. Če se v besedilu odsek nahaja, povečamo število enakih odsekov. Po primerjavi besedil izračunamo odstotek podobnosti med njima, nato pa zamenjamo njuni vlogi, tako da izberemo odseke besedila drugega dokumenta in jih iščemo v prvem dokumentu ter še za to primerjavo izračunamo odstotek podobnosti.

S takšnim pristopom primerjanja zagotovimo, da se vsako besedilo primerja z vsakim natanko enkrat. Časovno zahtevnost algoritma primerjanja pa lahko določimo kot:

$$O(n) \approx n^2 * m \qquad (4.2)$$

kjer je:

n – število besedil v sklopu za primerjanje

m – povprečno število odsekov na besedilo

4.7 REZULTAT PRIMERJAVE IN POROČILA

Po opravljeni primerjavi besedil so ugotovitve in rezultati predstavljeni uporabniku. O vsaki posamezni primerjavi dobimo za rezultat predstavljen odstotek podobnosti in tekst, ki je bil najden v obeh dokumentih, ki smo jih med seboj primerjali. Program s tem prikazom opozori uporabnika na morebitno plagiatorstvo, če gre za dejansko plagiatorstvo, pa mora odločiti uporabnik ali ocenjevalec sam. Uporabnik je obveščen tudi o času, ki je bil potreben za samo primerjavo.

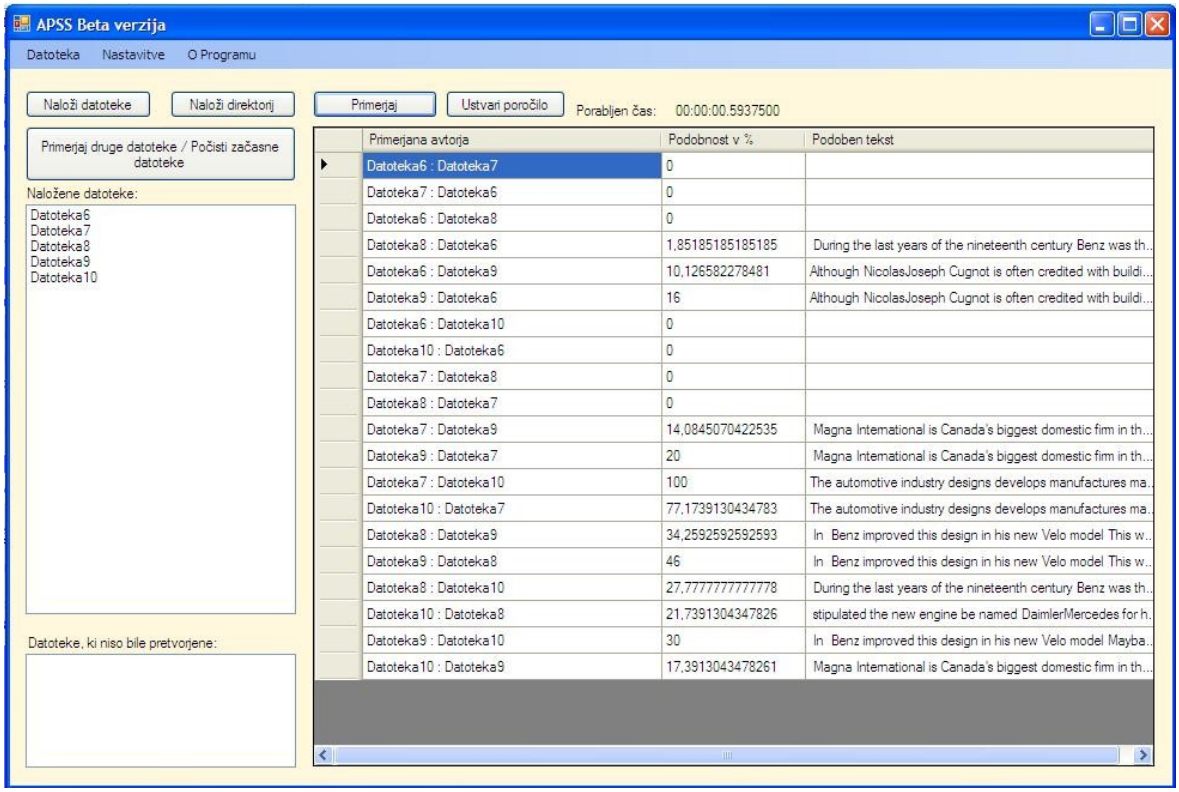
Ob novi primerjavi se obstoječi rezultati izbrišejo, zato je možno ustvariti poročilo o opravljeni primerjavi. Aplikacija je bila v prvotnem namenu zasnovana za potrebe v šolstvu, predvsem na fakultetah, zato je potrebno ob ustvarjanju poročila vnesti še podatke, kdo je izvajal primerjavo, za kateri predmet je bila izvedena primerjava, kakšni dokumenti so se primerjali in kaj je bila tema naloge. Nastaviti je potrebno, kateri rezultati naj bodo vključeni v poročilu. Priporočeno je, da so v poročilo vključene le primerjave, ki presegajo maksimalno dovoljeno stopnjo podobnosti. Kadar primerjamo 10 dokumentov, dobimo kot rezultat 90 primerjav, zato ob veliki količini dokumentov v poročilo ni smiselno vključevati vseh primerjav, saj je namen poročila ta, da imamo dokazno gradivo za primere plagiatorstva in da se v njem tudi znajdemo.

Poročilo je izdelano v obliki XML-dokumenta in ga lahko odpremo v poljubnem internetnem brskalniku. V prvotnem listu XML-dokumenta so shranjeni osnovni podatki o primerjavi, ki jih je vnesel uporabnik, in datum, kdaj je bila primerjava izvedena. Sledijo podatki za vsako posamezno primerjavo, ki izpolnjuje pogoj preseganja maksimalne dovoljene stopnje podobnosti. Za vsako primerjavo zapišemo imeni datotek, ki sta bili

primerjani, kolikšna je bila njuna podobnost ter celoten tekst, ki je v obeh datotekah enak.

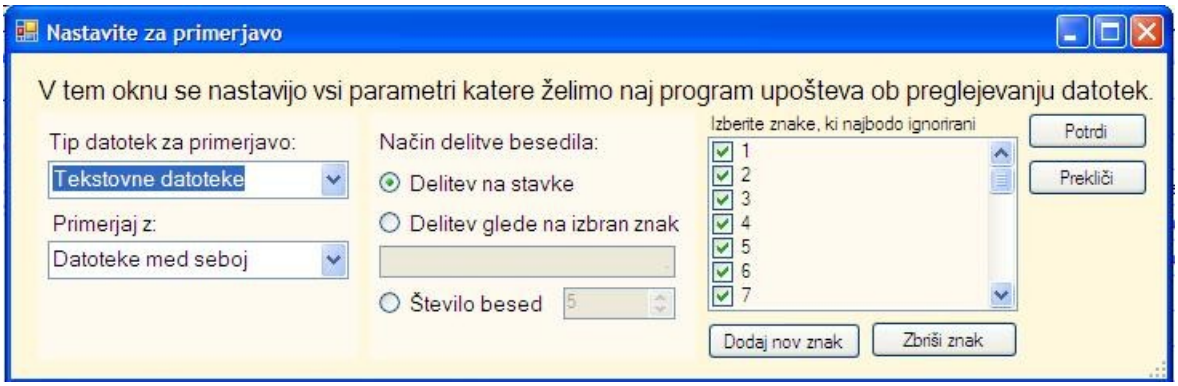
4.8 KRATEK OPIS UPORABNIŠKEGA VMESNIKA

V tem poglavju bom na kratko opisal uporabniški vmesnik svojega programa in meni za nastavitve. Uporabniški vmesnik na sliki 4.3 lahko razdelimo na 3 osnovne dele. Prvi del sestavlja meni programa, kjer dostopamo do nastavitev primerjave in informacij o programu, ter gumbi za opravljanje z datotekami. Naložiti je možno posamezne datoteke, sklope datotek ter celotne direktorije z datotekami. Drugi del vmesnika je sestavljen iz dveh seznamov, v katerih so zapisane vse naložene datoteke, ki so pripravljene za primerjavo, ter datoteke, ki jih program ni uspel naložiti zaradi težav z



Slika 4.3: Uporabniški vmesnik programa APSS za primerjanje datotek

datotečnim formatom teh datotek. Tretji del vmesnika pa predstavlja tabelo za rezultate primerjav. Tabela je sestavljena iz treh stolpcev, v katerih najdemo podatke: kateri datoteki smo primerjali, kakšno podobnost med njima smo ugotovili, kaj je med njima podobnega.



Slika 4.4: Meni za nastavitve

Slika 4.4 prikazuje meni za nastavitve, ki jih lahko uporabimo za posamezno primerjavo. Vidimo lahko, da je aplikacija pripravljena tudi za razširitve kot na primer za primerjavo izvorne kode in primerjanje datotek z viri s spleta. Izberemo lahko želeno delitev in znake, ki jih v primerjavi želimo ignorirati.

Iz obeh slik je razvidno, da gre za zelo preprost uporabniški vmesnik, ki uporabniku nudi opravljanje primerjave z nekaj preprostimi kliki. Pri nastavitvah prav tako ni potrebno vnašati nobenih podatkov, saj preprosto izbiramo med nastavitvami, ki so na voljo. Za zahtevnejšega uporabnika je omogočeno dodajanje lastnih znakov ali fraz, ki jih želimo ignorirati.

5 TESTIRANJA PROGRAMOV

Po implementaciji svojega programa sem prešel še na testiranje, da bi ugotovil, ali se lahko dobljeni rezultati primerjajo z rezultati obstoječih programov. V ta namen sem opravil dva sklopa testiranj. Prvi sklop je namenjen predvsem primerjavi rezultatov med različnimi programi, medtem ko sem v drugem sklopu testiral hitrost svojega programa ter opravil primerjavo rezultatov pri različnih nastavitvah. Za vsak posamezni test sem navedel, kaj sem testiral in na katerih programih se je test izvajal, nato sem opisal vhodne podatke ter rezultate dobljenih primerjav. Na koncu testa še podajam ugotovitve, do katerih sem prišel.

5.1 PRIMERJAVA REZULTATOV MED PROGRAMI

Programa, ki sem ju izbral za primerjavo rezultatov z mojim programom, sta DOC Cop in WCopyFind, saj oba programa nudita medsebojno primerjavo datotek. Programa Plagiarism Detector in Plagiarism Finder za primerjanje rezultatov nista uporabljena, ker nudita le iskanje po internetu, za njuno uporabo potrebujemo licenco, z demo različico pa ne moremo doseči zanesljivih rezultatov.

Za prvi test sem izbral pet popolnoma naključnih besedil iz zbirke seminarskih nalog pri predmetu ORS iz študijskega leta 2008/2009, ki mi jih je posredoval mentor. Zaradi zaščite osebnih podatkov so datoteke poimenovane kot Datoteka1 do Datoteka5. Datoteke so shranjene v formatu Microsoft Word, kajti program WCopyFind ne nudi podpore za format PDF. Povprečna dolžina datoteke je 13 strani in vsebuje okoli 1400

besed. Pri izbranih datotekah ne pričakujemo velike stopnje plagiarizma zaradi nedostopnosti internetnih virov v slovenščini glede na temo seminarske naloge, prav tako so bili študentje večkrat opozorjeni, da bodo njihovi izdelki striktno pregledani s programi za odkrivanje plagiarizma.

Rezultati programa DOC Cop pri delitvi na 10 besed dolge odseke.

Tabela 5.1: Rezultati primerjave datotek s programom DOC Cop – Test št.1

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka4	Datoteka5	1,00 %	Ni znano

Rezultati programa WCopyFind, uporabljene nastavitve so: ignoriranje ločil in števil, dolžina odsekov je 10 besed.

Tabela 5.2: Rezultati primerjave datotek s programom WCopyFind – Test št.1

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka1	Datoteka2	0,00 %	0,00 %
Datoteka1	Datoteka3	0,00 %	0,00 %
Datoteka1	Datoteka4	2,00 %	3,00 %
Datoteka1	Datoteka5	1,00 %	3,00 %
Datoteka2	Datoteka3	5,00 %	5,00 %
Datoteka2	Datoteka4	4,00 %	8,00 %
Datoteka2	Datoteka5	3,00 %	8,00 %
Datoteka3	Datoteka4	2,00 %	4,00 %
Datoteka3	Datoteka5	1,00 %	4,00 %
Datoteka4	Datoteka5	7,00 %	9,00 %

Rezultati lastnega programa, z nastavitvami na delitev na 10 besed in ignoriranje števil ter posebnih znakov.

Tabela 5.3: Rezultati primerjave datotek s programom APSS – Test št.1

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka1	Datoteka2	0,00 %	0,43 %
Datoteka1	Datoteka3	0,00 %	0,00 %
Datoteka1	Datoteka4	0,00 %	0,00 %
Datoteka1	Datoteka5	0,00 %	0,00 %
Datoteka2	Datoteka3	0,00 %	0,00 %
Datoteka2	Datoteka4	0,43 %	0,00 %
Datoteka2	Datoteka5	0,00 %	0,44 %
Datoteka3	Datoteka4	0,00 %	0,00 %
Datoteka3	Datoteka5	0,35 %	0,00 %
Datoteka4	Datoteka5	0,00 %	0,00 %

Pri rezultatih DOC Cop programa smo dobili le eno primerjavo, kar pomeni, da so ostale primerjave vrnile podobnost 0 %. Odstotki podobnosti, prikazani v tabelah, nam povedo le del rezultata primerjav, saj ne vidimo dejanskih podobnih delov. Lahko pa povem, da je bilo možno v poročilu DOC Cop programa opaziti podobnost le v naslovu dokumenta, zaradi česar je nastal odstotek podobnosti. V primerjavi z mojim programom je bilo prav tako opaziti edine podobnosti v naslovih posameznih dokumentov. Pri programu WCopyFind se podobnosti kažejo zaradi slabe obdelave vključenih slik, bilo je moč opaziti, da program slik ne prebavi dobro, saj jih pretvori v razna zaporedja znakov, kar kaže na slabost ekstrakcije teksta.

Primerjava posameznih rezultatov med programi kaže na zelo nizko stopnjo plagiarizma, kot je bilo tudi za pričakovati. Pri DOC Cop programu imamo vse primerjave razen ene na 0 %, medtem ko v mojem programu štiri primerjave vrnejo podobnost večjo od nič procentov, iz česar lahko vidimo razlikovanje do 1 % med rezultati teh dveh

programov. Rezultati programa WCopyFind dajo večje podobnosti za razliko od drugih dveh ,kar pa sem že omenil, da nastane zaradi vsebovanja slik. Odgovor na vprašanje, zakaj sploh pride do odstopanj, je ta, da imajo programi različne načine delitve besedila na odseke in različne algoritme primerjanja, tako da bi bilo kakršno koli popolno ujemanje rezultatov zgolj naključje. Opaziti je bilo mogoče težave programa WCopyFind s slovenskim jezikom, predvsem s šumniki, kar nekoliko popači rezultate.

V prvem opravljenem testu smo imeli opravka s popolnoma naključnimi dokumenti različnih avtorjev, za drugi test, ki sem ga opravil, sem osebno ustvaril 5 datotek. Datoteke sem poimenoval z imeni Datoteka6 do Datoteka10 za ohranjanje oblike testa, čeprav osebnih podatkov ni bilo potrebno varovati. Za vsebino datotek sem si izbral naključno tematiko. Najprej sem ustvaril tri datoteke s kopiranjem teksta iz člankov na angleški strani Wikipedije [21], [22], [23]. Vsi trije podani dokumenti imajo opravka z avtomobilsko industrijo, zato med njimi ni pričakovati posebno visoke podobnosti, saj bo le-ta zaradi podobne tematike kvečjemu naključna. Preostali dve datoteki sem sestavil iz vsebine že omenjenih datotek. Datoteka9 tako vsebuje dele teksta iz Datoteke6, Datoteke7 in Datoteke8 ter krajši odstavek, ki ga ne najdemo v nobeni izmed teh treh datotek. Tako lahko pričakujemo večjo podobnost z ostalimi datotekami. Datoteka10 vsebuje celotno vsebino Datoteke7 in odstavek iz Datoteke9. Pričakovana je zelo velika podobnost z Datoteko7.

Nastavitve programov za primerjanje so ostale enake kot pri prejšnjem testiranju, kar pomeni; delitev na odseke velikosti 10 besed, pri programu WCopyFind in mojem programu tudi ignoriranje vseh števil, ločil in posebnih znakov.

Tabela 5.4: Rezultati primerjave datotek s programom DOC Cop – Test št. 2

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	2,00 %	Ni znano
Datoteka6	Datoteka9	Ni znano	20,00 %
Datoteka6	Datoteka10	Ni znano	1,00 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	16,00 %	Ni znano
Datoteka7	Datoteka10	88,00 %	Ni znano
Datoteka8	Datoteka9	Ni znano	32,00 %
Datoteka8	Datoteka10	21,00 %	Ni znano
Datoteka9	Datoteka10	Ni znano	23,00 %

Tabela 5.5: Rezultati primerjave datotek s programom WCopyFind – Test št. 2

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	2,00 %	2,00 %
Datoteka6	Datoteka9	34,00 %	19,00 %
Datoteka6	Datoteka10	1,00 %	1,00 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	14,00 %	12,00 %
Datoteka7	Datoteka10	100,00 %	76,00 %
Datoteka8	Datoteka9	35,00 %	20,00 %
Datoteka8	Datoteka10	19,00 %	23,00 %
Datoteka9	Datoteka10	25,00 %	17,00 %

Tabela 5.6: Rezultati primerjave datotek s programom APSS – Test št. 2

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	1,60 %	2,17 %
Datoteka6	Datoteka9	14,00 %	21,84 %
Datoteka6	Datoteka10	0,80 %	0,80 %
Datoteka7	Datoteka8	0,00 %	0,21 %
Datoteka7	Datoteka9	12,97 %	14,28 %
Datoteka7	Datoteka10	90,84 %	68,89 %
Datoteka8	Datoteka9	34,13 %	36,13 %
Datoteka8	Datoteka10	23,48 %	20,64 %
Datoteka9	Datoteka10	22,68 %	16,27 %

Pregled rezultatov drugega izvedenega testa s prirejenimi datotekami potrjuje pričakovanja, postavljena pred testom. Opazimo lahko, da so podobnosti med Datoteko6, Datoteko7 in Datoteko8 zelo nizke, saj so datoteke ustvarjene neodvisno med seboj. Primerjave teh treh datotek z Datoteko9 in Datoteko10 kažejo veliko večje podobnosti, saj sta omenjeni datoteki sestavljeni iz vsebin preostalih treh datotek. Opazimo tudi, da so vsi trije programi odkrili zelo visoko podobnost med Datoteko7 in Datoteko10, ki vsebuje celotno vsebino Datoteke7.

Ob primerjanju rezultatov primerjav svojega programa in programa DOC Cop opazimo kar veliko podobnosti, saj največja razlika med rezultati znaša manj kot 7 %, povprečna razlika med rezultati pa je približno 1,3 %. V primerjavi z rezultati WCopyFind programa opazimo manjše razlike, kot so bile pri testu 1. Za boljše ujemanje tokratnih rezultatov lahko navedemo podatek, da datoteke, uporabljene v testu 2, niso vsebovale nikakršnih slik in so bile napisane v angleškem jeziku.

Noben izmed programov ne poda informacije, kateri dokumenti so plagiati in kateri

ne, ta odločitev pripada ocenjevalcu. S primerno postavitvijo meje dovoljene podobnosti lahko sklepamo, v katerih primerih je prišlo do plagiacije. Pri tekstovnih datotekah znaša meja nekje 15–20 % podobnosti. Glede na rezultate testa tako ugotovimo, da v prvem testu ni prišlo do plagiacije, v drugem testu pa lahko kot plagiata izločimo Datoteko9 in Datoteko10, a če ne bi poznali, kako so datoteke nastale, ne bi mogli določiti, ali je prišlo do plagiacije v primeru ustvarjanja Datoteke9 in Datoteke10 ali pa so mogoče ostale tri datoteke nastale kot plagiarizem teh dveh datotek.

5.2 PRIMERJAVA REZULTATOV APSS ZA RAZLIČNE NASTAVITVE TER TESTIRANJE ZMOGLJIVOSTI

Prva testa sta pokazala, da moj program daje enakovredne rezultate obstoječim programom za primerjanje datotek in odkrivanje plagiatov. Sledeči testi pa bodo pokazali razlike v rezultatih zaradi različnih nastavitev mojega programa ter njegovo hitrost pri obdelovanju večje količine datotek. Za primerjavo različnih nastavitev sem uporabil datoteke iz testa 2, saj nudijo najboljši pregled za to primerjavo. Za izhodišče sem uporabil primerjavo že opravljenega testa 2 z mojim programom in nadaljnje rezultate primerjal s temi rezultati.

Pri testiranju različnih nastavitev sem preveril različne nastavitve glede na delitev besedila ter različne nastavitve glede na ignoriranje znakov. Najprej sem preveril rezultate s pomočjo stavčne delitve, nato pa še delitev glede na 15 besed dolge odseke. Pri testiranju ignoriranja znakov sem uporabljal delitev na 10 besed dolge odseke, kot sem jih uporabljal že v testu 2.

Vsa testiranja mojega programa sem opravil na računalniku CPU: Intel Pentium(R) Dual T3200 @ 2.00GHz in 2,87Gb pomnilnika.

Tabela 5.7: Rezultati primerjave datotek s programom APSS – Test št. 3 stavčna delitev

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	0,00 %	1,85 %
Datoteka6	Datoteka9	10,12 %	16,00 %
Datoteka6	Datoteka10	0,00 %	0,00 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	14,08 %	24,00 %
Datoteka7	Datoteka10	100,00 %	77,17 %
Datoteka8	Datoteka9	34,26 %	46,00 %
Datoteka8	Datoteka10	27,78 %	21,74 %
Datoteka9	Datoteka10	30,00 %	17,39 %

Tabela 5.8: Rezultati primerjave datotek s programom APSS – Test št. 3 delitev na 15 besed

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	0,53 %	1,30 %
Datoteka6	Datoteka9	11,76 %	18,49 %
Datoteka6	Datoteka10	0,53 %	0,58 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	11,45 %	12,60 %
Datoteka7	Datoteka10	79,83 %	59,88 %
Datoteka8	Datoteka9	30,00 %	31,09 %
Datoteka8	Datoteka10	18,26 %	17,44 %
Datoteka9	Datoteka10	19,33 %	13,95 %

Tabela 5.9: Rezultati primerjave datotek s programom APSS – Test št. 3 upoštevanje števil

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	1,30 %	2,94 %
Datoteka6	Datoteka9	18,49 %	23,46 %
Datoteka6	Datoteka10	1,03 %	1,41 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	14,50 %	16,05 %
Datoteka7	Datoteka10	99,63 %	75,63 %
Datoteka8	Datoteka9	37,60 %	23,80 %
Datoteka8	Datoteka10	26,68 %	23,79 %
Datoteka9	Datoteka10	25,92 %	18,13 %

Tabela 5.10: Rezultati primerjave datotek s programom APSS – Test št. 3 vseh posebnih znakov

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	1,33 %	23,75 %
Datoteka6	Datoteka9	13,86 %	21,75 %
Datoteka6	Datoteka10	0,80 %	0,87 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	12,98 %	14,22 %
Datoteka7	Datoteka10	90,84 %	68,69 %
Datoteka8	Datoteka9	33,91 %	36,40 %
Datoteka8	Datoteka10	23,32 %	21,45 %
Datoteka9	Datoteka10	22,59 %	16,52 %

Tabela 5.11: Rezultati primerjave datotek s programom APSS – Test št. 3 vseh posebnih znakov in števil

Prva Datoteka	**Druga Datoteka**	**Vsebovanost prve datoteke v drugi**	**Vsebovanost druge datoteke v prvi**
Datoteka6	Datoteka7	0,00 %	0,00 %
Datoteka6	Datoteka8	2,07 %	2,94 %
Datoteka6	Datoteka9	14,73 %	23,45 %
Datoteka6	Datoteka10	1,03 %	1,41 %
Datoteka7	Datoteka8	0,00 %	0,00 %
Datoteka7	Datoteka9	14,49 %	16,05 %
Datoteka7	Datoteka10	100,00 %	75,99 %
Datoteka8	Datoteka9	37,60 %	41,56 %
Datoteka8	Datoteka10	26,68 %	23,72 %
Datoteka9	Datoteka10	25,92 %	18,36 %

Tabela 5.12: Tabela porabljenega časa za posamezno primerjavo

Primerjava	**Čas**
Stavčna delitev – ignoriranje vseh posebnih znakov in števil	0,578 s
Delitev na 10 besed – ignoriranje vseh posebnih znakov in števil	2,156 s
Delitev na 15 besed – ignoriranje vseh posebnih znakov in števil	1,156 s
Delitev na 10 besed – ignoriranje vseh posebnih znakov	2,226 s
Delitev na 10 besed – ignoriranje števil	2,187 s
Delitev na 10 besed – brez ignoriranja	2,281 s

Ob primerjavi rezultatov za različne nastavitve lahko opazimo, da ima največji vpliv na posamezne podobnosti delitev besedila na odseke. Spremembe, ki jih lahko opazimo pri različnih delitvah, so lahko kar velike, vendar je skupni rezultat vedno enak, saj opazimo, da datoteke z veliko podobnostjo le-to, ne glede na delitev, ohranijo, datoteke z majhno podobnostjo pa kvečjemu še zmanjšajo svojo podobnost pri splošnejših delitvah. V tabeli 5.12 pa opazimo prednost, ki nam jo nudi splošnejša delitev besedila. Če upoštevamo, da smo imeli samo pet datotek za primerjavo in opravili 20 primerjav, vidimo,

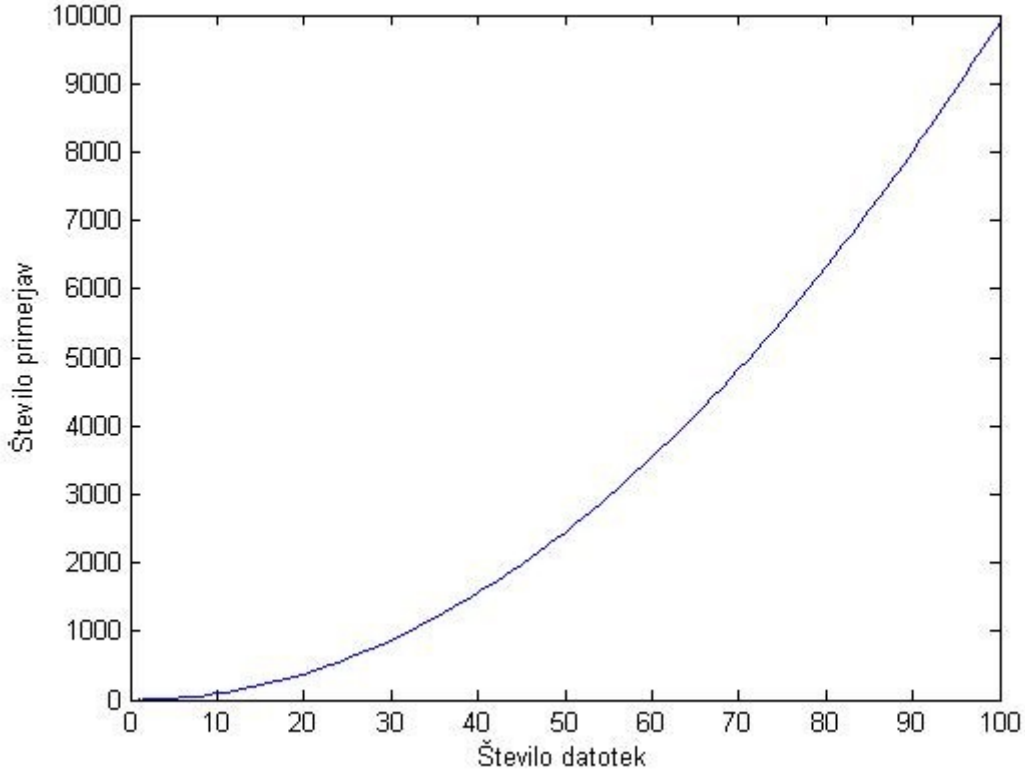
da je bil porabljen čas primerjave pri splošnejši delitvi štirikrat manjši od časa, ki smo ga porabili za delitev na 10 besed. Zato lahko s prvotno primerjavo s splošnejšo delitvijo izločimo datoteke, ki niso primer plagiarizma, kakor tudi datoteke z visoko stopnjo podobnosti, ki jih lahko osumimo kot plagiate. Nato preverimo datoteke na meji s specifično delitvijo, da dokončno določimo, katere datoteke bi lahko bili plagiati. Pri določanju različnih znakov, ki jih bomo upoštevali pri primerjavi, opazimo, da imajo števila večji vpliv na spremembe rezultatov kot posebni znaki, vendar tudi te razlike niso tako znatne. Je pa ta nastavitev zelo odvisna od samih dokumentov in količine števil ter posebnih znakov v njih.

Zadnji izveden test je namenjen predvsem za primerjavo hitrosti primerjanja datotek, zato sem opravil primerjave z večjimi količinami datotek. Opravil sem primerjavo z 20, 30, 50, 75 in 100 datotekami. Nastavitve za vsako primerjavo so bile enake, uporabljena je stavčna delitev besedila in ignorirani vsi posebni znaki kakor tudi števila. Datoteke, ki sem jih primerjal, so bile seminarske naloge s povprečno dolžino 12 strani.

Tabela 5.13: Tabela za prikaz zmogljivosti programa APSS

Opravljena primerjava	Porabljeni čas
Primerjava 20 datotek z stavčno delitvijo	22,109 s
Primerjava 30 datotek z stavčno delitvijo	44,375 s
Primerjava 50 datotek z stavčno delitvijo	2 min 08,593 s
Primerjava 75 datotek z stavčno delitvijo	4 min 32,921 s
Primerjava 100 datotek z stavčno delitvijo	9 min 07,062 s

Graf na sliki 5.1 prikazuje, kako se z večanjem števila vhodnih datotek veča število primerjav, ki jih je potrebno opraviti. S pomočjo tega grafa in tabele rezultatov zmogljivosti 5.13 pridemo do ugotovitev, s pomočjo katerih lahko ocenimo potreben čas za primerjanje specifičnega števila datotek.



Glede na graf vidimo, da je število primerjav približna kvadratna funkcija.

$$m = n*(n-1) = n*n-n \qquad (5.1)$$

kjer je:

m = število primerjav

n = število vhodnih datotek

S pomočjo tabele in formule 5.1 lahko izračunamo oceno potrebnega časa za eno primerjavo. Za dane rezultate je ta ocena 0,053 s. V kolikor poznamo število vhodnih datotek, lahko izračunamo tudi čas, potreben za primerjavo. Na primer, če imamo 143 vhodnih datotek, to pomeni, da moramo opraviti 20306 primerjav, za katere bi potrebovali približno 1076,218 sekund oziroma 17 minut in 56,218 sekund. Seveda moramo tukaj

upoštevati, da ta izračun velja le v primeru, kadar imamo opravka z datotekami povprečne dolžine 12 strani in so te datoteke seminarske naloge, kar pomeni, da vsebujejo kazalo, naslovno stran, literaturo in slike. Rezultat bi bil drugačen, če bi bile datoteke z 12 stranmi teksta ali bi bile datoteke krajše ali daljše. Upoštevati moramo tudi moč procesorja, na katerem izvajamo primerjavo, saj počasnejši ali hitrejši procesor povzroči spremembo časa potrebnega za primerjavo.

5.3 POVZETEK IN UGOTOVITVE TESTIRANJA

Na podlagi izvedenih testov lahko rečem, da se moj program brez težav primerja z obstoječimi programi za primerjanje datotek in odkrivanje plagiatov, vsaj kar se tiče dobljenih rezultatov o podobnosti datotek. Prav tako lahko trdim, da je v primerjavi z ostalimi programi zelo hiter, če ne celo hitrejši, saj opravlja dvosmerno primerjavo vsakega para datotek.

6 ZAKLJUČEK

Plagiatorstvo v šolstvu postaja vedno večji problem, zato je nujno, da imamo ustrezna orodja, s pomočjo katerih lahko odkrivamo plagiate med izdelki učencev, dijakov in študentov. Na fakultetah je problem še toliko večji, saj imamo v posameznih študijskih programih vpisanih vse več študentov in sta pregled ter primerjava njihovih izdelkov vse težja ter časovno potratna. Programi, ki za nas primerjajo izdelke med seboj, so tako velik pripomoček, saj lahko skrajšamo čas pregleda raznih nalog na le del časa, ki bi ga potrebovali, če bi te izdelke primerjali ročno.

Ob razvoju lastnega programa APSS sem spoznal, da ni težko napisati programak ki je zmožen primerjati večje sklope izdelkov, vendar se moramo ob razvoju držati pravil, ki nam pomagajo pri detekciji plagiatov. Pomembno je tudi, da predvidimo čim več načinov, kako bi lahko plagiirali nek dokument, ter da za vsako rešitev poiščemo načine, kako takšno rešitev pretentati, nato pa izboljšati to rešitev ali poiskati novo rešitev, ki ne bo imela te pomanjkljivosti. S takšnim pristopom sprotne optimizacije rešitve in učenja napak ob delu lahko v nekaj mesecih razvijemo soliden sistem za iskanje plagiatov.

Program, ki sem ga sam razvil v primeru primerjave dokumentov med seboj sicer deluje zelo dobro, vendar je še veliko možnosti za izboljšave in razširitve samega programa. Tako bi lahko razširili program s primerjavo izvorne kode kakor tudi s primerjavo s spletnimi viri. V program bi lahko dodali različne algoritme, ki bi bili za samo primerjavo še bolj temeljiti, in dovoljevali manjše razlike med samimi primerjalnimi odseki, vendar moramo biti ob razvoju takšnih algoritmov pozorni na čas obdelave. Ostaja še vprašanje, kakšen čas za primerjavo je še sprejemljiv, če primerjamo programsko obdelavo primerjave z ročnim pregledom dokumentov.

Čeprav je program za primerjavo datotek zelo koristno orodje za odkrivanje plagiatov, je še vedno potrebno pregledati vsebino datoteke na pravilnost tematike kakor tudi na to, ali so v dokumentu zajete vse zahtevane specifikacije, se pravi, da oddan dokument ne vsebuje naključnega besedila, ki nima nobene zveze s tematiko naloge. Za rešitev tega problema bi lahko razširili program s semantičnim preverjanjem, pri čemer bi program ugotovil, ali je oddani dokument vsebinsko pravilen in ali vsebuje vse zahtevane specifikacije. Takšen program bi preveril dokumente glede na njihovo vsebino in podal uporabniku povzetek vsebine, na podlagi katere bi uporabnik določil primerno oceno dokumenta, kar bi zelo olajšalo delo profesorjev in asistentov pri pregledu dokumentov.

7 LITERATURA

[1] *Definicije plagiarizma po slovarju slovenskega knjižnega jezika*. Inštitut za slovenski jezik Frana Ramovša ZRC SAZU, http://bos.zrc-sazu.si/cgi/a03.exe?name=sskj_testa&expression=plagiat&hs=1, 1. maj 2009

[2] *Definicije plagiarizma po Dictionary.com.* Dictionary.com, http://dictionary.reference.com/browse/plagiarism, 1. maj 2009

[3]*Definicije plagiarizma po Plagiarism.org.* Plagiarism.org, http://www.plagiarism.org/learning_center/what_is_plagiarism.html, 1. maj 2009

[4] *Definicije plagiarizma po Little, Brown Essential Handbook for Writers.* UNITED STATES NAVAL ACADEMY, http://www.usna.edu/Library/Plagiarism/Definitions.html, 1. maj 2009

[5] S.Schleimer, D.S.Wilkerson, A.Aiken, *Winnowing: Local Algorithms for Document Fingerprinting.* UC Berkeley, http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf, 2003

[6] *DOC Cop Plagiarism Detection,* DOC Cop ™**,** http://www.doccop.com/index.html?nc=34002901, 1. maj 2009

[7] *The Plagiarism Resource Site Charlottesville, Virginia*, WCopyFind domača stran, http://plagiarism.phys.virginia.edu/Wsoftware.html, 1. maj 2009

[8] *Plagiarism Detector – Expert Plagiarism Detection Software*, Plagiarism Detector domača stran, http://plagiarism-detector.com/, 1. maj 2009

[9] *Plagiarism Finder*, Plagiarism Finder domača stran, http://www.plagiarismfinder.de/produkte/download, 1. maj 2009

[10] *ANCP Software,* AntiPlagiarist domača stran, http://www.anticutandpaste.com/antiplagiarist/, 1. maj 2009

[11] *Diff.* Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Diff, 1. maj 2009

[12] Plagiarism detection. Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Plagiarism_detection, 1. maj 2009

[13] *MOSS, A System for Detecting Software Plagiarism,* MOSS domača stran, http://www.cs.berkeley.edu/~aiken/moss.html, 1. maj 2009

[14] *JPlag, Detecting Software Plagiarism,* JPlag domača stran, https://www.ipd.uni-karlsruhe.de/jplag/, 1. maj 2009

[15] *SID – Plagiarism Detection,* SID domača stran, http://genome.math.uwaterloo.ca/SID/, 1. maj 2009

[16] X. Chen, B.Francia, M. Li, B.McKinnon, A.Seker, *Shared Information and Program Plagiarism Detection.* University of California, http://monod.uwaterloo.ca/papers/o4sid.pdf, 28. marec 2000

[17] *Epocalipse IFilter*, CodeProject, http://www.codeproject.com/KB/cs/IFilter.aspx, 5. maj 2009

[18] *DocxToText library,* CodeProject, http://www.codeproject.com/KB/office/ExtractTextFromDOCXs.aspx, 5. maj 2009

[19] *SharpZipLib library*, ic#Code, http://www.icsharpcode.net/OpenSource/SharpZipLib/, 5. maj 2009

[20] D. Letecky, *Converting PDF to Text in C#,* CodeProject, http://www.codeproject.com/KB/string/pdf2text.aspx, 5. maj 2009

[21] *Automobile,* Wikipedia, the free encyclopedia, vir za sintetično izdelavo primerjalnih datotek, http://en.wikipedia.org/wiki/Car, 1. maj 2009

[22] *Automotive Industry,* Wikipedia, the free encyclopedia, vir za sintetično izdelavo primerjalnih datotek, http://en.wikipedia.org/wiki/Auto_industry, 1. maj 2009

[23] *Karl Benz,* Wikipedia, the free encyclopedia, vir za sintetično izdelavo primerjalnih datotek, http://en.wikipedia.org/wiki/Karl_Benz, 1. maj 2009

8 PRILOGA

Diplomskemu delu prilagam izvorno kodo najpomembnejših algoritmov v programu APSS. Ob izvorni kodi so podani so za vsak algoritem še, kratek opis samega algoritma in opis vhodnih parametrov.

8.1 KLIC PROCEDURE ZA PRIMERJAVO

```
/// <summary>
/// klic procedure za primerjavo, najprej si zapomnimo čas za določitev časa trajanja
/// nato pokličemo proceduro čiščenja, po čiščenju opravimo primerjanje na koncu pa
/// prikažemo rezultate in izračunamo porabljen čas
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void bCompare_Click(object sender, EventArgs e)
{
   resultNames = new ArrayList();
   resultSimilarty = new ArrayList();
   resultSimilartyString = new ArrayList();
   DateTime a = DateTime.Now;
   podatki.fileCleaning(initialSettings, used_ignores.listOfStrings);
   ComparingAlgorythmOne test = new ComparingAlgorythmOne();
   test.AlgorythmOne(initialSettings, fileNames, podatki.TextFileStrings, ref resultSimilarty, ref
                     resultNames, ref resultSimilartyString);
   label4.Text = (DateTime.Now - a).ToString();
   dataGridView1.Rows.Clear();
   for (int i = 0; i < resultSimilarty.Count; i++)
   {
     dataGridView1.Rows.Add();
     dataGridView1[0, i].Value = resultNames[i].ToString();
     dataGridView1[1, i].Value = (double)resultSimilarty[i];
     if (resultSimilartyString[i] != null)
       dataGridView1[2, i].Value = resultSimilartyString[i].ToString();
   }
 }
```

8.2 ALGORITEM ČIŠČENJA BESEDILA

```
/// <summary>
/// čiščenje nzaželjenih znakov iz datotečnih zapisov
/// odstranimo nezaželjene znake in spremenimo vse večkratne pojavitve presledka
/// </summary>
/// <param name="useMe">uporabniške nastavitve</param>
public void fileCleaning(Settings.Settings useMe, ArrayList ignore)
{
   ignoreChars = defaultTrimChars;
   trimChars = ignoreChars.ToCharArray();
   for (int i = 0; i < textFileStrings.Count; i++)
   {
      textFileStrings[i] = Regex.Replace(textFileStrings[i].ToString(), @"\s+", " ").Trim();
      textFileStrings[i] = Regex.Replace(textFileStrings[i].ToString(), @",", "");
      textFileStrings[i] = Regex.Replace(textFileStrings[i].ToString(), @";", "");
      textFileStrings[i] = Regex.Replace(textFileStrings[i].ToString(), @":", "");
      textFileStrings[i] = Regex.Replace(textFileStrings[i].ToString(), @"!", ".");


      cleaned = textFileStrings[i].ToString();
      for (int j = 0; j < ignore.Count; j++)
      {
         Ignore_String tren = (Ignore_String)ignore[j];
         if (tren.Yes_no)
            cleaned = cleaned.ToString().Replace(tren.IgnoreMe, "");
      }
      textFileStrings[i] = cleaned;
      textFileStrings[i] = textFileStrings[i].ToString().Replace("?", ".");
   }
}
```

8.3 ALGORITEM DELITVE BESEDILA Z VARIABILNO DOLŽINO ODSEKOV

```
/// <summary>
/// razbijanje datotek glede na delilni znak
/// hkrati pa počistimo ven prazne stringe in stringe ki
/// so krajši od 20 znakov oz vsebujejo manj od 5 besed
/// </summary>
/// <param name="filesTexts">vsebine datotek</param>
/// <param name="cmpSett">uporabniške nastavitve</param>
private void breakUpFilesOne(ArrayList filesText, Settings.Settings cmpSett)
{
   for (int i = 0; i < filesText.Count; i++)
   {
      helpArray = filesText[i].ToString().Split(cmpSett.Separator.ToCharArray());
      onebrokenFile = new ArrayList();
      for (int j = 0; j < helpArray.Length; j++)
      {
         wordCountArray = helpArray[j].Split(' ');
         if (helpArray[j] != " " && helpArray[j] != "" && helpArray[j].Length > 20 &&
```

```
                wordCountArray.Length > 5)
            {
              onebrokenFile.Add(helpArray[j]);
            }
          }
          brokenfiles.Add(onebrokenFile);
        }
      }
```

8.4 ALGORITEM DELITVE BESEDILA Z FIKSNO DOLŽINO ODSEKOV

```
/// <summary>
/// razbijanje datotek glede na nastavljeno velikost besed;
/// najprej besedilo razdelimo na besede nato pa ustvarjamo
/// ustrezno dolge stringe
/// </summary>
/// <param name="filesText">vsebine datotek</param>
/// <param name="cmpSett">uporabniške nastavitve</param>
private void breakUpFilesTwo(ArrayList filesText, Settings.Settings cmpSett)
{
  int stevec;
  string helpString;
  for (int i = 0; i < filesText.Count; i++)
  {
    helpArray = filesText[i].ToString().Split(' ');
    helpString = "";
    stevec = 0;
    onebrokenFile = new ArrayList();
    for (int j = 0; j < helpArray.Length; j++)
    {
      if (helpArray[j] != "" && helpArray[j] != " ")
      {
        helpString += helpArray[j] + " ";
        stevec++;
      }
      if (stevec == cmpSett.SeparationWordCount)
      {
        onebrokenFile.Add(helpString);
        helpString = "";
        stevec = 0;
      }
    }
    if (helpString != "" && helpString != " ")
      onebrokenFile.Add(helpString);
    brokenfiles.Add(onebrokenFile);
  }
}
```

8.5 ALGORITEM PRIMERJANJA DATOTEK

```
/// <summary>
/// primerjava datotek kot vhod dobimo ze razbite datoteke, začnemo z prvo
/// datoteko v sklopu izbranih datotek za primerjavo, to datoteko primerjamo
/// z naslednjo datoteko v sklopu in po opravljeni primerjavi naredimo še
/// obratno primerjavo nato nadaljujemo z naslednjo datoteko, s tem pristopom
/// zagotovimo da se ne rabimo nikoli vračati po seznamu nazaj
/// </summary>
/// <param name="filesTexts">vsebine datotek</param>
/// <param name="fileNames">imena vhodnih datotek</param>
/// <param name="filesTexts">vsebine datotek</param>
/// <param name="pResultSimilarity">rezultat primerjave; procent podobnosti</param>
/// <param name="pResultNames">rezultat primerjave; imena primerjanih datotek</param>
/// <param name="pSimilarityString">rezultat primerjave; enaka vsebina</param>
private void compareUS(ArrayList filesText, ArrayList fileNames, ref ArrayList pResultSimilarity, ref
                                    ArrayList pResultNames, ref ArrayList pSimilarityString)
{
   int SimilarityCounter;
   string SimilarityString;
   double SimilarityA;
   double SimilarityB;
   int i, j, k;
   int lengthA, lengthB;
   ArrayList fileOne;
   ArrayList fileTwo;

   for (i = 0; i < brokenfiles.Count - 1; i++)
   {
      fileOne = (ArrayList)brokenfiles[i];
      lengthA = fileOne.Count;
      for (k = i + 1; k < filesText.Count; k++)
      {
         SimilarityString = null;
         SimilarityCounter = 0;
         for (j = 0; j < lengthA; j++)
         {
           //primerjava datotek oz. iskanje odsekov
           if (fileOne[j].ToString() != null)
               if (filesText[k].ToString().IndexOf(fileOne[j].ToString()) != -1)
               {
                  SimilarityCounter++;
                  SimilarityString += fileOne[j].ToString();
               }
         }
         //shranjevanje rezultatov primerjave
         pResultNames.Add(fileNames[i].ToString() + " : " + fileNames[k].ToString());
         SimilarityA = ((double)SimilarityCounter / (double)lengthA) * 100;
         pSimilarityString.Add(SimilarityString);
         pResultSimilarity.Add(SimilarityA);

         fileTwo = (ArrayList)brokenfiles[k];
         lengthB = fileTwo.Count;
```

```
            SimilarityString = null;
            SimilarityCounter = 0;
            for (j = 0; j < lengthB; j++)
            {
              //primerjava datotek oz. iskanje odsekov z obrnjenima vlogama datotek
              if (fileTwo[j].ToString() != null)
                if (filesText[i].ToString().IndexOf(fileTwo[j].ToString()) != -1)
                {
                  SimilarityCounter++;
                  SimilarityString += fileTwo[j].ToString();
                }
            }
            //shranjevanje rezultatov obratne primerjave
            pResultNames.Add(fileNames[k].ToString() + " : " + fileNames[i].ToString());
            SimilarityB = ((double)SimilarityCounter / (double)lengthB) * 100;
            pSimilarityString.Add(SimilarityString);
            pResultSimilarity.Add(SimilarityB);
          }
        }
      }
```

8.6 ALGORITEM ZA KREIRANJE POROČILA

```
    /// <summary>
    /// preverimo vse primerjave ali presegajo minimalno dovoljeno
    /// mejo podobnosti nato ustvarimo poroèilo s pomoèjo serializacije podatkov
    /// </summary>
    /// <param name="fileNames">imena datotek</param>
    /// <param name="similarity">podobnosti datotek</param>
    /// <param name="similarText">podoben text</param>
    /// <param name="data">podatki o primerjavi kdo je primerjal in kaj je bilo primerjano</param>
    public void CreateReport(ArrayList fileNames, ArrayList similarity, ArrayList similarText,
Podatki_o_primerjavi data)
    {
      minLikness = data.Min_dovoljena_podobnost;
      report.Podatki_o_primerjavi = data;
      for (int i = 0; i < similarity.Count; i++)
      {
        if ((double)similarity[i] > minLikness)
        {
          matchedComparison = new Primerjava();
          string[] hlp = fileNames[i].ToString().Split(':');
          matchedComparison.Primerjana_datoteka = hlp[0];
          matchedComparison.Primerjana_z= hlp[1];
          matchedComparison.Procent_podobnosti= (double)similarity[i];
          matchedComparison.Enaka_vsebina = similarText[i].ToString();
          report.Primerjave.Add(matchedComparison);
        }
      }

      savePath += "/Poroèilo" + reportNumber.ToString() + ".xml";
      try
```

```
        {
            XmlSerializer serializer = new XmlSerializer(typeof(Porocilo));
            TextWriter writer = new StreamWriter(savePath);
            serializer.Serialize(writer, report);
            writer.Close();
        }
        catch
        {
            throw new Exception("Napaka pri kreiranju XML datoteke!");
        }

        report.Primerjave.Clear();
    }
```