Alati za detekciju plagijata u programskim rješenjima

Devald, Valentina

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

VALENTINA DEVALD

ALATI ZA DETEKCIJU PLAGIJATA U PROGRAMSKIM RJEŠENJIMA

Završni rad

Pula, listopad, 2020.

Sveučilište Jurja Dobrile u Puli
Fakultet informatike u Puli

VALENTINA DEVALD

ALATI ZA DETEKCIJU PLAGIJATA U PROGRAMSKIM RJEŠENJIMA

JMBAG: 0303054243, redovita studentica
Studijski smjer: Informatika

Predmet: Programiranje
Znanstveno područje: Područje društvenih znanosti
Znanstveno polje: 5.04. Informacijske i komunikacijske znanosti
Znanstvena grana: 5.04.02. Informacijski sustavi i informatologija
Mentor: Izv. prof. dr. sc. Tihomir Orehovački

Pula, listopad, 2020.

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisana Valentina Devald, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

_____

Valentina Devald

U Puli, _____, 2020. godine

IZJAVA
o korištenju autorskog djela

Ja, Valentina Devald dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Alati za detekciju plagijata u programskim rješenjima“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.
Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, _____, 2020. godine

Potpis

_____

SAŽETAK

Kroz ovaj rad dat je prikaz plagijarizma programskog koda i alata za njihovu detekciju. Plagiranje programskog koda je problem koji se pojavljuje u akademskim i industrijskim krugovima, te ima izrazite negativne učinke na njihov daljnji rast i razvoj. Detekcija plagijata je potom prikazana kroz vrste tehnika i potom se izvršila kvalitativna usporedba značajki pojedinih programa za detekciju plagiranja programskog koda s obzirom na prethodna istraživanja. Osnovna hipoteza rada se sastojala od prikaza superiornosti alata koji koriste hibridne tehnike detekcije nad alatima koji koriste individualne tehnike detekcije plagijata. Rezultati prethodnih istraživanja su interpretirani i uz detaljnu analizu njihovih javno dostupnih svojstva dato je moguće objašnjenje za navedene obrasce efikasnosti pojedinih alata i njihova ograničenja.

Ključne riječi: metode detekcije plagijata računalnog koda, programska podrška detekcije plagijata, kloniranje programske podrške

ABSTRACT

This paper deals with an overview of computer code plagiarism and the tools for their detection. Programming code plagiarism is a problem that occurs in academic as well as industrial circles which has overwhelming negative effects on their further growth and development. Plagiarism detection is shown through the different types of techniques and afterwards a qualitative comparison, of the different features between individual programs for plagiarism detection, is analysed with respect to the results of previous research. The main hypothesis of this work is the superiority of hybrid technique based detection tools over single technique tools. Results of previous research are interpreted and, with a detailed analysis of their publicly available characteristics, a plausible explanation for the resulting efficiency patterns and limitations of individual products is given

Key words: code plagiarism detection techniques, plagiarism detection software, software clones,

SADRŽAJ

1. UVOD 1
1.1. Predmet i objekt istraživanja 2
1.2. Hipoteza rada 2
1.3. Ciljevi i struktura rada 3
2. PLAGIRANJE KODA I TEHNIKE DETEKCIJE 4
2.1. Razlozi plagiranja ili kloniranja koda 5
2.2. Terminologija i svojstva alata detekcije klonova 11
2.3. Kategorizacija plagijata i programskih klonova 13
2.4. Proces detekcije plagijata 14
2.5. Tehnike detekcije 18
2.5.1. Tehnike tekstualne komparacije 19
2.5.2. Tehnike znakovne komparacije 20
2.5.3. Tehnike temeljene na stablima raščlambe 23
2.5.4. Tehnike temeljene na grafikonu programske ovisnosti(GPO) 25
2.5.5. Tehnike komparacije karakterističnih vrijednosti 27
2.5.6. Hibridne tehnike 29
3. USPOREDBA ALATA ZA DETEKCIJU PLAGIJATA 31
3.1. Prikaz popularnih alata 31
3.1.1. MOSS 31
3.1.2. GPlag 33
3.1.3. JPlag 36
3.1.4. SIM 38
3.2. Prikaz prethodnih istraživanja i rezultata 40
3.3. Komparacija dostupnih alata 44
3.3.1. Promatrane karakteristike 44
3.3.2. Rezultati istraživanja 45
3.4. Diskusija rezultata istraživanja 47
4. ZAKLJUČAK 49

POPIS LITERATURE 50
POPIS SLIKA 52
POPIS TABLICA 53

1. UVOD

Brzim razvojem tehnologija, ponajviše interneta, omogućeno je jednostavno dijeljenje podataka svih vrsta. Jedan od problema navedenog je plagiranje. Prisutan je kao problem u mnogim područjima, krenuvši od educiranja do znanstvenih istraživanja, pa čak i u razvoju programa i njihovog koda. Mnoge tvrtke s nedovoljno educiranim zaposlenicima ili manjkom vremena posegnut će za nekim već postojećim projektima otvorenog koda (često nazivani u literaturi njihovim engleskim nazivom: *„open source”*) radi izrade vlastitog proizvoda. Dodatno, postoje na tisuće studenata računarstva, informatike i srodnih studija koji su dužni rješavati niz vježbi i projekata kako bi završili svoje školovanje. U mnogim slučajevima lako je pretpostaviti da su neka od tih rješenja plagijati. Međutim, malo je akademskog osoblja koji posjeduju strpljenja i vremena temeljito i detaljno proučiti svako programsko rješenje. Takvi plagijati se najčešće pronalaze slučajno, primjerice, ako student zaboravi promijeniti svoje ime u nekom od rješenja ili se pojavljuju iste greške kod izvršavanja programa. Na svu sreću danas postoje alati pomoću kojih je moguće detektirati plagijate na znatno naprednije načine. Iako se trenutni alati bez poteškoća mogu koristiti u svrhu provjere vježbi ili projekata, sveprisutna je opasnost pogreške u radu, posebice u slučajevima gdje su preoblikovane pojedine izjave ili umetnuti novi dijelovi koda. No, iz prethodnih istraživanja, uočeno je kako, u borbi s ozbiljnim plagijatorima, navedeni alati nisu od znatne pomoći. Važno je napomenuti poveznicu plagiranja programskog koda, s piratstvom, te općenito s kršenjem zakona o autorskim pravima. Posebice, je značajna problematika u softverskoj industriji vezana uz intelektualno vlasništvo i zaštitu softverske licence, a značajno pogađa prethodno spomenute projekte otvorenog koda. Zbog navedenih razloga, kroz ovaj rad prikazana je i komparacija trenutno dostupnih softverskih rješenja za borbu s plagijatima programskih rješenja.

1.1. Predmet i objekt istraživanja

Unutar ovoga rada predmet istraživanja čine programska rješenja za detekciju plagiranja u programskom kodu drugoga programa. Alati za detekciju plagijata mogu djelovati na veoma različitom broju tehnika, te se ovim radom pokušava prikazati širinu spektra korištenih tehnologija. Prikazom trenutno najznačajnijih metoda moguće ih je kvalitativno usporediti kako bi se stvorila podloga za objekt istraživanja kojeg čini kvalitativna usporedna analiza po značajkama najznačajnijih rješenja.

1.2. Hipoteza rada

Komparacija dostupnih alata je izvršena uz glavnu hipotezu koja glasi: „grafikonom programskih ovisnosti pristup detekcije plagijata daje značajno bolje rezultate od pristupa drugim tehnikama“. Navedeno je očekivana pojava, no potrebno ju je istražiti unutar akademskog okvira koristeći dostupnu literaturu i koristeći prikladne alate za njezino dokazivanje ili opovrgavanje. Prikladan alat je u ovome slučaju znanstvena metoda istraživanja. Koristeći se znanstvenom metodom, tijekom diskusije će rezultati biti analizirani. Kako bi se to moglo napraviti, potrebno je definirati različite osnovne pojmove, detaljno objasniti plagiranje radova, s izrazitim naglaskom na tehnike koje se koriste u kodu i protumjera njima u susret.

1.3. Ciljevi i struktura rada

S obzirom na navedeni okvir, formuliraju se sljedeći ciljevi ovog rada po poglavljima:

- Kroz poglavlje „Plagiranje koda i tehnike detekcije” cilj je prikazati i definirati općenito plagiranje koda. Prikazati podjelu razina plagijata, Dodatni cilj je tehnički modelirati načine i značajne tehnike detekcije plagijata.
- U poglavlju “Usporedba alata za detekciju plagijata” cilj je izvršiti komparaciju dostupnih alata kroz prikaz njihovih kvalitativnih svojstva, te prikazati rezultate prethodnih istraživanja. Dodatni cilj je pronaći uzročno-posljedične poveznice na rezultata prethodnih istraživanja s obzirom na svojstva prikazanih alata i istražiti valjanost hipoteze.

Navedeni ciljevi se potom prikazuju ukratko u zaključku, gdje se prikazuju najbitniji zaključci rada i daje pregled izvršenih ciljeva uz kratki opis značaja istraživanja.

2. PLAGIRANJE KODA I TEHNIKE DETEKCIJE

Plagijat je prisvajanje tuđeg djela u cjelini ili u pojedinim odlomcima, bez navođenja autora. Po svojim svojstvima nije ograničen na polje književnosti, već se s njima borba vodi u širokom rasponu područja poput umjetnosti, poezije, znanosti te čak i u programskim rješenjima. Moguće je zaključiti kako njihova pojava ima značajan negativni učinak neovisno o polju u kojemu se pojavljuje. Značajno je uočiti kako je njihov negativni utjecaj najznačajniji u školstvu. Razlog toga je činjenica što plagijate u školstvu daju studenti koji bi navedenu tematiku trebali detaljno naučiti i staranjem rada ili projekta usvojiti predmetno gradivo. Plagiranjem se proces usvajanja znanja preskače. Dodatno, ako se plagijat ne otkrije prethodno ocjenjivanju rada i s obzirom na njega prođe kolegij, student je osvojio kvalifikaciju koju ne zaslužuje. Razlog tomu je što polaganjem kolegija student prikazuje određenu sposobnost unutar gradiva, no ako se to postigne na temelju plagijata, položeni kolegij ne odgovara njegovoj stvarnoj razini sposobnosti u gradivu kolegija. Plagiranje tekstualnih radova poput seminara, prezentacija i sl. radova je sveprisutno i nadaleko znana pojava. Tome se stalo na kraj kroz detaljnu usporedbu seminara. Navedena se usporedba danas vrši računalo uz pomoć programskih rješenja poput Turnitin, PlagScan i dr. Njihova upotreba je drastično povećala razinu otkrivenih plagijata. No navedeni programi su prikladni za otkrivanje sličnosti u multimedijalnim dokumentima. Programski kod u svojoj osnovi nije takav dokument osim u specifičnim slučajevima. Time se dolazi do potrebe za posebnim alatima kojima bi se navedeno moglo otkriti u programskim rješenjima. Potreba za otkrivanjem programskih plagijata je nešto složenija zbog mogućnosti obfuskacije koda i specifičnosti pisanja programskog koda, čime su pojedini dijelovi često samoslični s drugima i čime se dolazilo do značajnog broja lažno pozitivnih rezultata prilikom prvih testiranja koda na plagijate. Kako bi se pristupilo u koštac navedenoj problematici, potrebno je prikazati načine i razloga njihova nastanka. Dodatno, potrebno je imati na umu i činjenicu da nisu svi plagijati isti. Banalni primjer toga bi prikazao razliku koda koji je direktno plagiran i koda koji je plagiran, no nazivi varijabli su promijenjene kako bi izbjegao detekciju. Unutar ovog poglavlja se prikazuju odgovori na navedena pitanja, te model i tehnike detekcije kojima je moguće otkriti navedene i druge česte tehnike plagiranja programskog koda.

2.1. Razlozi plagiranja ili kloniranja koda

Kopiranje dijelova koda i njegovo ponovno korištenje, bilo to s ili bez modifikacija, vrlo je učestala pojava u razvoju programskih rješenja. Plagiranje dijelova koda je učestala pojava i znanstvenim, akademskim i industrijskim krugovima neovisno o negativnoj konotaciji koja se nalaže iza toga i ima različite negativne posljedice ovisno o kontekstu plagiranja:

- Negativnost unutar znanstvenih krugova je neosporiva i moguće ju je po svojim svojstvima izjednačiti s plagiranjem znanstvenog rada. Navedeno je teška povreda pravila i normi znanstvene metode i struke što bi trebalo detektirati i sankcionirati u svim slučajevima.
- Kopiranje koda koje se vrši prilikom edukacije ima drugačiju vrstu negativne konotacije, plagiranje prilikom edukacije u pravilu vrše studenti kako bi izbjegli proučavanje tematike i stvarni rad na zadatku. U slučaju neotkrivanja takvih kodova, studenti dobivaju kvalifikaciju za nešto što nije učinjeno i time ruše standard navedene kvalifikacije ostalim studentima koji su navedeno učinili.
- Moguće je zapitati se: „Osim u akademskim, znanstvenim i edukativnim krugovima, koja su loše strane plagiranja programerskog koda koji je otvoren za korištenje?“ Odgovor na navedeno pitanje nije jednostavan zbog heterogenosti perspektiva i funkcija koda, no neki od općih pojava koreliranih s ponovnim korištenjem koda su prema literaturi i prethodnim istraživanjima (Cordy i Roy, 2007:14, vlastiti prijevod):
    - Otežano održavanje sustava.
    - Pojava neočekivanih greški u izvođenju programa(u engleskoj literaturi često nazivani „*bug*“-ovima).
    - Otežano razumijevanje računalnog koda.
    - Negativan utjecaj na daljnji razvoj programa.

  Očito je kako se su navedene pojave nepoželjne u svim industrijama, zato što otežano održavanje sustava povećava potrošeno vrijeme programera. Jednako tako, veći broj pogrešaka programa povećava potrošnju vremena provedenog na njihovom otklanjanju. Veća potrošnja vremena programera stvara dodatne

troškove što je ekonomski nepoželjna situaciju u svakoj industriji neovisno o polju i značaju.

Potrebno je dodatno razumjeti i činjenicu kako kloniranje koda ne sadrži isključivo negativne konotacije. Ukoliko se ispune sljedeći uvjeti, kloniranje koda je čak i poželjna pojava:

- Kloniranjem koda se ne krše zakonske i dr. odredbe nadležnih tijela.
- Kloniranjem koda se pripisuje autorsko pravo originalnog programera.
- Iznimna sličnost već dostupnog, postojećeg rješenja s traženim.
- Mogućnost jednostavne prilagodbe originalnog koda za funkcije zadatka.
- Kloniranjem koda se omogućuje bolja preglednost sustava nego apstrakcijom u posebne dijelove.
- Kloniranjem koda se postižu određeni uvjeti (brzine izvršenja, veličine program i dr.) za čije ostvarenje na drugačiji način bi se stvorili znatni dodatni troškovi.

Primjer pozitivne implementacije plagiranja je ponovno korištenje postojećih modula financijskih proizvoda u integraciji novog financijskoj proizvoda u računalnom sustavu banke. Detaljnije objašnjenje navedenog je da su novi financijski proizvodi nerijetko veoma bliski postojećima te da temeljno različiti proizvodi se izrazito rijetko postavljaju na tržište. Time bi se sustav praćenja i obrade jednog postojećeg proizvoda mogao uz minorne modifikacije primijeniti na novi. Naravno u navedenom slučaju, bolje rješenje bi bilo restrukturiranje sustava, no ako ne postoji potreba za restrukturacijom, a sam sustav je značajno velik, ovakvo kloniranje koda je sasvim prihvatljivo i poželjno jer se time smanjuje trošak. Kako bio se navedeni pozitivni primjeri distancirali i semantički odvojili, nadalje u radu se za, ponovno korištenje koda, u općem i pozitivnom kontekstu koristi riječ kloniranje, dok se isključivo negativnom kontekstu pridružuje riječ plagiranje.

Kloniranjem se kroz prethodni primjer izbjeglo dugotrajno testiranje novog koda na određene sigurnosne i druge zahtjeve, te se u potpunosti izbjegao proces otklanjanja greški novog koda što je uvijek izrazito temeljit i relativno dugotrajan proces. Dodatni primjer kloniranja koda je kloniranje koje se provodi unutar jednog entiteta ali između različitih timova. Poseban slučaj kloniranje je grananje (u engleskoj literaturi poznatiji

pod nazivom „*forking“)* originalnog koda za ulogu koja mu prethodno nije bila namijenjena. Učestali primjer grananja su upravljački programi(u engleskoj literaturi „*driver program*“ ili češće, samo „*driver*“) fizičkog sklopovlja računala: Primjer je stvaranje upravljačkog programa za novi model miša. Često se za navedeni program koristi već postojeći, srodan upravljački program za sličan model kako bi se izbjeglo ponovno pisanje koda.(Cordy i Roy, 2007:3). Pojavi kloniranja koda u izrazitom različitim konfiguracijama i funkcija posebno su podložni Linux upravljački programi radi jednostavnosti dizajna interakcije jezgre Linuxa s hardverskim sklopovljem računala (Cordy i Roy, 2007:4).

Generativnim programiranjem se opisuje proces generiranja programskog koda pomoću računalnog programa. Navedena tehnika se koristi u mnogim područjima, no ona se najčešće koristi kod računalnih programa koji stvaraju drugi računalni kod pomoću „povuci i postavi“ (semantički približan prijevod s engleskog termina „drag and drop“) tehnike. Primjer navedenog je „WordPress“ – poznati uređivač web stranica uz pomoć prethodne tehnike kojom se omogućuje stvaranje i uređivanje web stranica znatno široj i tehnički ne potkovanoj populaciji. Značajni nedostatak takvog pristupa je činjenica što se generativnim programiranjem time stvara značajan udio dupliciranog, tj. kloniranog koda.

Klonirani kod također nastaje ako se njime izbjegava značajna rekonstrukcija sustava koja nije ekonomski opravdana u datom trenutku. Kako je razvoj koda relativno skup i dugotrajan proces, znatna je tendencija svih industrija koje se služe programskim rješenjima da što dulje budu u fazi eksploatacije tog razvoja, a što manje u fazi samog razvoja i implementacije optimalnih sustava. Neovisno o odgađanju rekonstrukcije sustava, sama arhitektura u fazi dizajna sustava se može tako implementirati kako bi se izbjeglo dupliciranje koda, no ako se to ne provede na samom kraju faze inicijalnog dizajna, može se ukazati potreba kloniranja. Time kod postaje teži za održavanje i može doći do nenamjernih greški jer otklanjanjem nepoželjne funkcionalnosti na jednom dijelu koda, njegova kopija nije otklonjena i u posebnim slučajevima gdje se provode upravo ti klonirani dijelovi koda mogu uzrokovati greške u radu.

Bitno je napomenuti dodatno kako optimalni sustavi se također mogu smatrati primjerom kloniranja koda. Posebno kod jednostavnijih primjera, mnogi programeri su naučili iste primjere sintakse i modele pisanja programa. Nije iznenađujuće iz tog razloga da je česta pojava ekvivalencija dvaju programa zbog čiste slučajnosti koja je proizašla iz sličnosti stilova pisanja i rada navedenih programera. Osim navedenih razloga, uzroci mogu biti i limitacije s kojima se programeri susreći u stvarnom životu poput rokova isporuke i drugih ograničenja koja su prethodno napomenuta što za rezultat primoraju programere na ponovno korištenje prethodno napisanog koda.

Prethodno opisani razlozi plagiranja zajedno s drugim razlozima kloniranja koda koji nisu ovdje opisani zbog njihove banalnosti prikazani su tablicom 1.

Tablica 1 Prikaz najčešćih razloga kopiranja prema kategorijama.

Osnovni i specifični razlozi za kopiranje programa:								
Kao dio strategije razvoja programa:								
Pojednostavljeno korištenje kroz kopiraj/zalijepi.	Grananje originalnog programa (eng. „*Forking*“)	Ponovno korištenje dizajna.	Generativno programiranje	Spajanje sličnih sustava	Odgađanje rekonstrukcije sustava.			
Zbog benefita prilikom održavanja i rada:								
Izbjegavanje nepoželjnih rizika	Izbjegavanje nepoželjnih poveznica unutar programa.	Ostvarivanje dovoljne razine robusnosti programa.	Ostvarivanje boljih performansi programa.	Zbog dizajnerskih odluka makrostrukture.				
Kako bi se izbjegle određene limitacije:								
Nedostatak mehanizma za ponovno korištenje u jeziku.	Izbjegavanjem dodatne apstrakcije koda.	Drugačija implementacija koda stvara dodatne greške.	Namjerno stvaranje kompleksnosti.	Nedostatak vremena programera.	Smanjenje broja linija koda.	Nedostatak vlasništva nad kodom.	Nedostatak znanja u relevantnom području.	Nedostatak razumijevanja sustava.
Slučajno kopiranje koda:								
Zbog protokola interakcije s API sustavima i programskim bibliotekama	Mentalni model koji posjeduje programer.	Korištenje iste implementacije logike kod dva programera.						

Cordy i Roy (2007, vlastiti prijevod)

Prethodno je spomenuto kako kloniranje koda i njegovo plagiranje utječe na kvalitetu gotovog proizvoda. Kloniranjem, programer je zadužen za modifikaciju i prilagodbu koda za njegovu temeljnu, novu funkciju. Proces modifikacije može unijeti dodatne greške u kodu i time se stvara dodatni trošak. Očito je iz navedenog da je na programeru odluka o ponovnom korištenju s obzirom na njegovo znanje i iskustvo. Ukoliko se ponovno koristi pojedini kod, važno je njega i prilagoditi za novu svrhu, a to se postiže prilagodbom identifikatora i korištenih funkcija. Isti proces ako se provodi nad plagijatom može uzrokovati većom ili manjom sličnošću završnog koda s originalom. Navedene razine sličnosti je moguće opisati kao različite razine plagijata.

2.2. Terminologija i svojstva alata detekcije klonova

Ulazeći dublje u problematiku detekcije plagijata od izrazite je važnosti prikazati osnovne pojmove i procese uz pomoć kojih se opisuje detekcija plagijata. U nastavku popis pojmova korištenih pri opisivanju detektora plagijata:

- Transformacija i normalizacija izvornog koda – Transformacija i normalizacije čine promjene koda u kojim se tekst transformira u drugi oblik, dok je normalizacija naziv za izmjenu identifikatora standardnim, normalnim kako bi se uočilo proceduralno, no ne samo tekstualno plagiranje koda.
- Prikaz transformiranog koda: Ovim svojstvom se opisuje intermedijarni prikaz koji nastaje prije komparacije a nakon transformacije koda.
- Usporedna granulacija: Različiti algoritmi rade na različitim prikazima koda pri različitim razinama granularnosti (rezolucije promatranja, tj. veličina osnovne promatrane jedinice). Pojedini algoritmi se izvršavaju na granularnost jedne linije izvornog koda, dok drugi rade na ASS ili GPO čvorištima. Dakle, ovo svojstvo služi kako bi se odredila koja je vrsta i veličina granulacije klona korištena u fazi usporedbe te je li ona ograničena u svom opsegu.
- Algoritam komparacije: Ovo svojstvo definira vrstu algoritma komparacije. Pod vrstu se prikazuje osnovni način komparacije, tj. na osnovu kojeg uvjeta se vrši određivanje sličnosti. Određivanje uvjeta je iznimno važno jer se time neposredno utječe na krajnje rezultate komparatora, a prethodni se uvjeti komparacije određuju pomoću uvjeta prisutnih u drugim, već dugo ustaljenim disciplinama. u biologiji je uobičajen algoritam usporedbe DNK uz pomoć tehnike podudaranja niza (Cordy i Roy, 2007:43). Ovim se svojstvom može zaključiti definira osnovna specifikacija programa.
- Računalna složenost: Ukupna računalna složenost tehnike detekcije klona glavni je problem kod otkrivanje klonova u velikim programskim rješenjima, Razlog je opisan činjenicom da linearnim povećanje veličine programa, nastaje eksponencijalni rast broja mjesta koje je potrebno provjeriti. Složenost pristupa ovisi o vrsti transformacija i upotrijebljenom algoritmu za usporedbu. Ovo svojstvo označava sveukupnu računalnu složenost za određenu metodu.

- Sličnost klonova: Neke tehnike detekcije mogu pronaći isključivo parove istovjetnih klonova, dok druge mogu otkriti pojedine slične dijelove, podudaranje po određenim, parametrima ili klonove koji su po drugim svojstvima bliski. Ovim svojstvom prikazuje se tehnika detekcija kojom je otkriven klonirani dio koda.
- Razina granulacije: Ona može biti konstantna ili slobodna. Ako na vraćenim klonovima postoji unaprijed definirana sintaktička granica (funkcije, zagrade početka i kraja) govorimo o klonovima konstantne granularnosti. S druge strane, ako ona ne postoji, onda se radi o klonovima slobodnih granularnosti.
- Neovisnost alata: Alati za otkrivanje kloniranog koda moraju biti neovisno o jeziku, ovim svojstvom se opisuje potrebne dodatne alate kako bi se navedeno moglo postići ili je bilo potrebno postići ovisno o kontekstu.
- Vrsta izlaza: Opisuje se način izlaza informacije o kloniranju programa, tj. jesu li klonovi vraćeni u obliku pojedinačnih parova ili kao različite klase greški po određenim svojstvima ili na sasvim drugačiji način.
- Refaktoriranje klona: Ovo svojstvo označava je li odabrana tehnika prikladna u slučajevima gdje su klonovi refaktorirani za razliku od običnog kopiranja. S obzirom na dodanu razinu kompleksnosti takve primjene, otkrivanje klonova refaktoriranjem ne podržavaju sve tehnike.
- Osjetljivost na jezičnu paradigmu: Ovim se svojstvom opisuje osjetljivost na osnovnu jezičnu paradigmu. Paradigma unutar ovog konteksta označava je li jezik programa pisan proceduralno (primjerice Pascal i C) ili je jezik objektno-orijentiran (poput Jave i C++). Navedeno je od izrazitog značaja za algoritam usporedbe jer u proceduralnim jezicima je znatno lakše uočiti klonirane dijelove koda s obzirom na strogi proceduralni red koji postoji u njima.

2.3. Kategorizacija plagijata i programskih klonova

Cordy i Roy (2007:14) objašnjavaju kako je komparaciju programskog koda na plagiranje moguće vršiti s obzirom na sličnost sirovog teksta koda i sličnost funkcionalnosti koda. Takvim usporedbama moguće je stvoriti sljedeće četiri opće kategorije plagijata prema uzlaznoj kompleksnosti postupka plagiranja:

- Plagijati tipa 1: Odlikuje ih gotovo potpuna identičnost koda osim u nefunkcionalnim dijelovima koda poput komentarima i prostoru između naredbi.
- Plagijati tipa 2: Kod ovog tipa dodatno je prisutna strukturalna i sintaktička ekvivalentnost dijelova koda, uz očitu varijaciju identifikatora, objekata i drugih programskih struktura.
- Plagijati tipa 3: Plagijati uz izmijenjenu makrostrukturu programa. Izmjenom makrostrukture izjava programa, drugim riječima njihovom promjenom, dodavanjem ili brisanjem stvara se distinktivna razlika naspram originalnog koda koju je značajno teže uočiti.
- Plagijati tipa 4: Najteži su za očiti zbog činjenica da je u njihovom stvaranju došlo do izmjene sintaktičke varijante koda. Sintaktičke varijacije koda su, unutar ovoga konteksta, dva potpuno različita koda koji vrše jednaku funkciju. Navedene je najteže uočiti i po svojim svojstvima mogu se i ne moraju pojavljivati promjene prisutne u prethodna dva tipa plagijata.

Za navedene kategorije Cordy i Roy (2007) tvrde sljedeće: „Ovakvom kategorizacijom plagijata definira se povećanje razine suptilnosti uzlazno od tipa 1 do tipa 4, no osim toga i prikazuju i povišenje razine kompleksnosti tehnike detekcije. Detekcija plagijata tipa 4 je iznimno teška čak i uz iznimno visoku razinu razumijevanja razvoja softvera. Otežana detekcija je prisutna neovisno o tome je li proces automatski rađen ili ne.“ Iz navedenih kategorija, moguće je potom stvoriti model idealnog programa detekcije plagijata s obzirom na tip plagijata koji se pokušava detektirati. Primjer navedenog bi bila brza međusobna usporedba sirovog teksta za plagijate tipa 1, dok je za plagijate tipa 2 potrebna interpretacija koda u oblik u kojemu se proučava njegova makrofunkcionalnost.

2.4. Proces detekcije plagijata

Osnovna zadaća detektora kloniranja koda je pokušati pronaći dijelove koda velike sličnosti u sustavnom izvornom tekstu. Problem koji proizlazi iz navedenog je što se ne može unaprijed znati koje dijelove koda možemo pronaći više puta je iz tog razloga potrebno usporediti sve moguće dijelove koda s ostalim cjelokupnim kodom. Ovakav način usporedbe i traženja kloniranih dijelova koda je izrazito "skup", u pogledu broja računalnih operacija. Kako bi se navedenom problemu uhvatilo u koštac, koriste se različite tehnike transformacije koda kako bi se smanjio opseg traženja. Dodatni problem čini i osjetljivost samih tehnika detekcije i transformacije čime nastaje značajan broj lažno pozitivnih rezultata koje je potrebno filtrirati prema Collberg, Myles i Stepp (2004).

Slijedom prethodne problematike, utvrđuje se sljedeći tok procesa detekcije plagijata po koracima:

1. Pretprocesiranje – Nakon ulaza izvornog koda, važno je prilagoditi ga na zapis koji je prilagođen za daljnju obrade kroz faze programa. Kroz pretprocesiranje se u pravilu izvorni kod dijeli na manje jedinice i određuje se domena komparacije. Prethodno tome je također uputno odrediti i maknuti dijelove koda koji ne ulaze u komparator.

Prethodno se može opisati kroz tri glavne predmetne faze:

a. Uklanjanje nezanimljivih dijelova
b. Određivanje izvornih jedinica
c. Određivanje jedinice za usporedbu

2. Transformacija – Opisuje proces pretvorbe osnovnih jedinica izvornog koda u prikaz uz pomoću kojeg je olakšana računalna usporedba koda.

Postoji veliki broj različitih transformacija, nadalje je prikazana lista nekih uobičajenih primjera:

- Standardizirani ispis – Kroz standardizaciju koda, izvorni program se preslaguje na jedan standardni način ispis.
- Uklanjanje komentara.
- Uklanjanje praznog prostora i proreda.
- Tokenizacija – Tokenizacija opisuje proces raščlambe programa po linijama koda koje su zapisane u obliku normaliziranih zapisa za pojedine

elemente i/ili identifikatore unutar specifičnog programskog jezika.. Time se stvaraju osnovne jedinice usporedbe koje se nazivaju “tokenima”. Nerijetko se kroz normalizaciju provode i druge transformacije. Cordy i Roy (2007:41) opisuju kako se kroz proces tokenizacije također uklanjaju komentari i praznine.

- Sintaktička raščlamba – U slučaju raščlanjivanja na bazi elementarnog izvornog koda, Cjelokupni bazni kod se raščlanjuje na manje jedinice koja tvore sintaktičko stablo programa. U ovakvom prikazu, izvorne i određene jedinice se prikazuju kao podstabla raščlanjenog ili osnovnog apstraktnog sintaktičkog stabla. Algoritam komparacije potom koristi posebna pravila za usporedbu na različitim razinama programa.
- Generiranje grafikona programskih ovisnosti (GPO; u engleskoj literaturi: “*Program Dependence Graphs”*) – Poznavanjem semantičkih pravila jezika moguće je generirati grafikone koja se opisuje ovisnost pojedinih funkcija programa. Metoda ovakve usporedbe se temelji na usporedbi oblika GPO. Osim glavnog GPO, gotovo uvijek se analiziraju i podgrafovi ulaznog programskog koda. Jedinice koje se koriste u navedenim grafikonima mogu biti:
    i. Individualni elementi izvornog koda
    ii. Individualni elementi transformacija.
    iii. Manji GPO podgrafovi.
- Normaliziranje identifikatora – Gotovo svi pristupi koriste normaliziranje identifikatora na određene vrijednosti. Time je znatno olakšana usporedba pojedinih identifikatora koje je plagijator potencijalno izmjenjivao u namjeri izbjegavanja detekcije. Važno je napomenuti kako može predstavljati zasebnu transformaciju, no većinom je slučaj da se ona provodi tokom prethodno opisanog postupka tokenizacije.
- Normalizacija elemenata programa – Osim normaliziranja identifikatora, izrazito je važno normalizirati i ostale elemente koda kako bi se mogla pronaći sličnost u elementima više različitih razina. Jednako kao i kod prethodne normalizacije identifikatora, moguće ju je izvršiti posebno, no često se provodi kroz postupak tokenizacije.

- Transformacija s obzirom na karakteristične vrijednosti – Proračunavaju se određene standardizirane mjere za izvorni kod i na temelju njih se vrši transformacija.

3. Detekcija podudaranja – Unutar ove faze direktno se kompariraju transformirane jedinice međusobno kako bi se pronašle podudarnosti s arhivom jedinica transformiranih kodova.

4. Precrtavanje – Pronađene pozicije podudarnosti transformiranog koda se precrtavaju natrag u izvorni kod radi lakšeg pregleda i kasnije ručne manipulacije.

5. Algoritamsko filtriranje – Kako algoritmi nisu savršeni, postoji određeni broj lažno pozitivnih rezultata koje je potrebno odstraniti. Navedeno se preliminarno vrši pomoću posebnih algoritama.

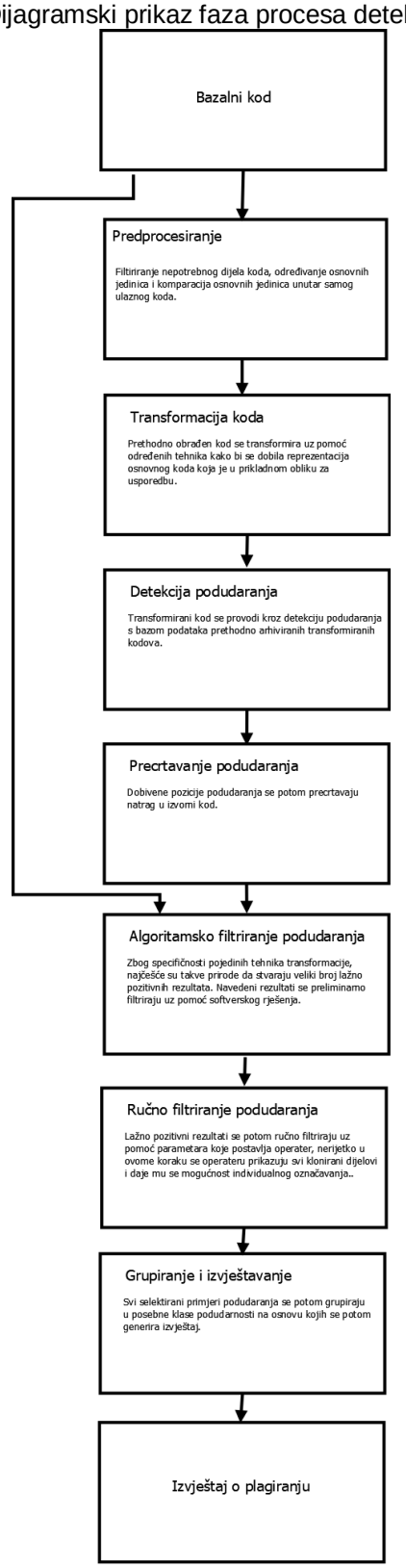
6. Ručno filtriranje – Jednako kao i prethodna faza, podaci se filtriraju ručnim pregledom od strane operatera.

U nastavku slijedi vizualan prikaz detekcije kloniranog koda.

7. Grupiranje i izvještavanje – Pronađene podudarnosti se grupiraju po određenim svojstvima u klase i potom se prema njima generira krajnji izvještaj.

Prethodno opisan proces, nastavno je prikazan na slici 1.

Slika 1 Dijagramski prikaz faza procesa detekcije.



Prema Cordy i Roy (2007:22, vlastiti prijevod).

2.5. Tehnike detekcije

Postoji veći broj tehnika pomoću kojih je moguće detektirati plagijat. Nekoliko ih je komercijalno, dok se ostala većina koristi u istraživačke svrhe s ciljem pomoći u razvoju i procesu održavanja sustava. Većina alata se primarno bazira na otkrivanje različitih vrste klonova pomoću tehnika otkrivanja u usporedivoj razini granularnosti.

Podjela tehnika detekcija unutar ovoga rada se vrši s obzirom na osnovni komparirani prikaz:

- Tehnike tekstualne komparacije – Komparacija se izvršava na razini sirovog teksta.
- Tehnike znakovne komparacije – Komparacija slijedi nakon transformacije tekstualnog koda u niz znakova.
- Tehnike temeljene na stablima raščlambe – Unutar njih se program rastavlja na svoje funkcionalne jedinice sve do najmanjih segmenata. Od navedenih elemenata se stvaraju grafička stabla s čvorištima.
- Tehnike temeljene na grafikonima programske ovisnosti(GPO) – GPO čine napredniju inačicu grafičkih stabala gdje je moguće detaljnije vršiti komparaciju pojedinih segmenata.
- Tehnike komparacije karakterističnih vrijednosti – S obzirom na izvorni tekstualni kod, stvaraju se određene karakteristične vrijednosti segmenata ili cijelog teksta koje se potom uspoređuju s drugim segmentima ili tekstovima.
- Hibridne tehnike – Sastoje se od kombinacije dviju ili više prethodno navedenih tehnika.

U nastavku, dati su prikazi pojedinih tehnika i rezultati kvalitativne komparacije koju su prethodno izvršili autori Cordy i Roy (2007) tamo gdje su dostupni.

2.5.1. Tehnike tekstualne komparacije

U ovom se pristupu ciljani izvorni kod razmatra kao redoslijed linija i/ili nizova, ovisno o individualnim svojstvima algoritma. Dva dijela koda se potom međusobno uspoređuju kako bi se pronašli nizovi istog teksta. Jednom kad se ustanovi da su dva ili više dijela koda jednaki prema svom sadržaju i opsegu, označavaju se kao klonovi. Ukoliko ih je svega dvoje, označavaju se kao par, a u slučaju višestrukog ponavljanja kao posebna klasa klonova sličnog ili istog izvora. Zbog navedenog pristupa, detektirani klonovi ne moraju odgovarati strukturalnim elementima jezika i transformacije koje se primjenjuju su male ili gotovo nikakve prije početka komparacije. No neovisno, najčešće se provodi sljedeća osnovna filtracija koda koju je moguće okarakterizirati kao transformaciju:

1. Uklanjanje komentara – Komparator zanemaruje sve komentare u izvornom kodu. Navedeno je ovisno o odabranom jeziku pisanja programa.
2. Uklanjanje praznina – Brišu se sve praznine, prijelazi u nove redove i dr.
3. Normalizacija teksta se često vrši jednom od uobičajenih shema. Primjer navedene normalizacije prikazan je u tablici 2.

Tablica 2 Primjer operacija normalizacije nad izvornim kodom.

Operacija	Element	Primjer elementa	Funkcionalna zamjena
1	Tekst	“Abort”	“...”
2	Znak	‘y’	‘.’
3	Broj	42	1
4	Decimalni broj	0.314159	1.0
5	Identifikator	Counter	p
6	Osnovni numerički elementi	Int, short, ling, double	num
7	Ime funkcije	Main	foo()

Prema Cordy i Roy (2007).

Dodatnu dimenziju usporedbe moguće je stvoriti promjenom razine usporedbe, drugim riječima promjenom veličine osnovnog komparacijskog tokena. Ovim postupkom se nastoji umanjiti varijabilnost rezultata prilikom korištenja direktne tekstualne komparacije. U nastavku se nalazi prikaz nedostataka do kojih može doći ukoliko se naivno odredi veličinu tokena da bude jedna linija koda i isključivo se po njoj vrši komparacija:

- Identifikacija prekida linije – u slučaju da je u kod ubačen umjetni prekid linije, moguće je otkrivanje parcijalnog klona za razliku od potpuno što on u naravi je.
- Promjene identifikatora: Promjenom imena identifikatora je moguće zamračiti plagirani kod alatu. Navedenom se pokušava suočiti uz pomoć prethodno opisanog postupka normalizacije koda, no to nije slučaj kod svih tehnika baziranih na direktnoj komparaciji teksta.
- Izmjena zagrada izjave: U slučaju izmjene zagrada koda, moguće je izbjeći detekciju. Time se alat u svojoj osnovi dovede u zabludu da je dio koji je dodan ili oduzet iz zagrade zapravo drugačiji blok naredbi čime je osnovni blok izmijenjen u dovoljnoj mjeri da se ne kategorizira kao plagirani dio.

2.5.2. Tehnike znakovne komparacije

Osnovna paradigma tehnika temeljenih na znakovima je transformacija osnovnog koda u znakovni niz. Tako stvoren znakovni niz se potom provodi kroz temeljnu komparaciju sa samim sobom ili s drugim nizovima u namjeri otkrivanja dupliciranih dijelova. Razlika naspram prethodnih tekstualno baziranih metoda se sastoji u činjenici njihove značajne otpornosti na prekide linija (navedeno predstavlja samo dodatni znak u kodu ako se uopće prikazuje u obliku niza), te na izmjene zagrada koda (kao i u prethodnom primjeru, dodane ili oduzete zagrade predstavljaju samo translaciju, tj. pomak niza za određeni broj znakova, dok je niz unutar zagrada gotovo nepromijenjen).

Važno je uzeti u obzir činjenicu da bi direktna komparacija cjelokupnog programa s bazom podataka bila iznimno računski skupa, što zauzvrat zahtijeva primjenu određenih transformaciju, nakon prijevoda koda u osnovni znakovni zapis, kako bi se smanjio broj potrebnih operacija komparacije. Dodatno, kod se prethodno prijelazu u

znakovni zapis normalizira čime se regulariziraju identifikatori i strukture kako bi se otkrilo pokušaje plagiranja uz izmjenu naziva identifikatora.

Sama komparacija se potom provodi na temelju sufiksnog stabla baziranog na nizovima znakova transformiranog prikaza koda. U slučaju prijelaza, empirijski određene vrijednosti sličnosti, nizovi kodova se označavaju kao parovi ili klase klonova. Prikaz pojedinih karakteristika četiriju različitih detektora baziranim na znakovnoj detekciji dat je kroz istraživanje Cordy i Roy (2007:47, vlastiti prijevod).

Tablica 3 Kvalitativna usporedba znakovnih tehnika komparacije za detekciju plagijata.

Tehnike bazirane na nizovima znakova				
Karakteristika	**Baker**	**Johnson**	**Ducasse et al.**	**Marcus Maletic**
Normalizacija ili transformacija	Uklanjanje komentara i praznina.	Uklanjanje komentara i praznina.	Uklanjanje komentara, praznina i nefunkcionalnih dijelova koda, također vrši transformaciju.	Uklanjanje komentara i sintaksne simbolike.
Reprezentacija koda	Parametrizirani tokeni teksta.	Tekstualni niz.	Efektivni niz linija.	Tekstualni prikaz.
Tehnika komparacije	Podudaranje tokena bez korištenja sufiksa.	Karp-rabin postupak identifikacije koda.	Dinamičko podudaranje uzoraka.	Grafičko teoretski pristup.
Rast kompleksnosti	*O* (n + m) n=ulazni broj linija m=broj pronađenih podudarnosti.	Nije dostupno.	$O(n^2)$ n=Broj linija koji se uz pomoć hash funkcije reducira na programske „kante“(u eng. literaturi: „*buckets*“).	Nije dostupno.
Granularnost komparacije	Linija ili tokeni linije.	Pojedini tekstualni element.	Linija.	Riječi, rečenice, paragrafi i eseji.
Sloboda granularnosti kloniranih dijelova	Slobodno,određeno granicama od 15 linija koda do maksimalne pronađene podudarnosti.	Slobodno, bazirano na minimumu od 50 linija koda.	Slobodno, ovisno o graničnoj razini.	Konstantno, funkcije, datoteke ili segmenti koda.
Osjetljivost na tip klona	Tip 1 i 2.	Tip 1 i 3	Isključivo tip 1	Tip 1,3 i 4
Neovisnost promatranog alata	Eventualno alat za raščlambu.	Nije potreban dodatni alat.	Eventualno postoji potreba za alat za raščlambu.	Uzima u obzir samo tekst izvornog koda.
Vrsta izlaza	Parovi klonova i klase klonova	Parovi klonova.	Parovi klonova.	Nije poznato.
Osjetljivost na refaktorirane klonove	Potrebna je ljudska prosudba.	Potrebna je ljudska prosudba.	Potrebna je ljudska prosudba.	Potrebna je ljudska prosudba.

Prema Cordy i Roy (2007).

2.5.3. Tehnike temeljene na stablima raščlambe

Tehnike temeljene na usporedbi stabla, imaju određene sličnosti s prethodnom tehnikom znakovne komparacije. S obzirom na programski kod stvara se stablo raščlambe, unutar ovog konteksta, često se ono naziva apstraktnim stablom sintakse (ASS). Apstraktno je s obzirom na tome da osnovna jedinica stabla vrše apstraktni elementi, dok se određivanje tih elemenata vrši prema sintaksi pojedinog odabranog jezika. ASS u svojoj osnovi sadrži sve potrebne informacije o izvornom kodu, no unutar njega ne postoje imena varijabli i doslovne vrijednosti jer se njime određuju apstraktne relacije između pojedinih naredbi, identifikatora i dr. elemenata jezika. Tim prikazom je omogućena sofisticirana komparacija stabla s podstablima. Uspoređuju li se stabla s podstablima moguće je iznimno lako pronaći samosličnost unutar koda. Dodatnu prednost čini činjenica što je, usporedba topologija dvaju ASS stabala, računski izrazito povoljna za učiniti.

Cordy i Roy (2007:51, vlastiti prijevod) navode: “Jedan od pionira ASS bazirane metode je Baxter et al.-ov CloneDR. Pomoću posebnog prevoditelja, generira se komentirano sufiksno stablo izvornog koda koje se potom uspoređuje sa svojim manjim komponentama i drugima uz pomoć posebne hash funkcije” CloneDr, program je primjer mogućnosti takvog programa, njime autori dalje navode kako je moguće komparacija neovisno o prazninama, izmjene redoslijeda u kodu koje je teško uočiti kroz prethodne metode detekcije.

U nastavku slijede rezultati komparacije algoritama stabla raščlambe iz istraživanja Cordy i Roy (2007:53, vlastiti prijevod). Rezultati su prikazani tablicom 4.

Tablica 4 Prikaz karakterističnih svojstava algoritama detekcije plagijata baziranih na strukturi stabala raščlambe.

Tehnike bazirane na stablima				
Karakteristika	**Baxter et al.**	**Yang**	**Wahler et al.**	**Evans**
Normalizacija ili transformacija	Raščlanjen u ASS.	Raščlanjen u specifično stablo.	Raščlanjen u ASS i potom pretvoren u XML.	Raščlanjen u ASS i potom pretvoren u XML.
Reprezentacija koda	ASS	Stablo raščlambe.	ASS u XML formatu.	ASS u XML formatu.
Tehnika komparacije	Komparacija stabala.	Dinamički kontrolirani proces komparacije stabala.	Komparacija podataka.	Teoretski grafički pristup.
Rast kompleksnosti	$O(N)$ N=Broj čvorišta ASS nakon optimizacije.	$O(S_1 S_2)$ S_1=Broj čvorišta prvog stabla. S_2=Broj čvorišta drugog stabla.	$O(kn^2)$ n= Broj izjava koje sadrže klonove. k=Maksimalna veličina klona	Nije dostupno.
Granularnost komparacije	ASS čvorište.	Token, u ovom slučaju čvorište stabla.	Linija koda.	ASS čvorište.
Sloboda granularnosti kloniranih dijelova	Slobodna, određena prema minimumu s obzirom na promatrane oblike.	Slobodna prema programu ili segmentu.	Slobodna, u pravilu 5 izjava, određena minimumom.	Slobodna,određena minimumom.
Osjetljivost na tip klona	Tip 1 i 3	Tip 1 i 3	Tip 1 i 2	Tip 2 i 4
Neovisnost promatranog alata	Potreban je alat raščlambe i programski lektor.	Potreban je alat raščlambe i za formatiranje prikaza.	Nije poznato.	Potreban je alat za raščlambu.
Vrsta izlaza	Parovi klonova.	Vizualni prikaz s formatiranim prikazom.	Nije poznato.	Html dokument s podacima o klonovima.
Osjetljivost na refaktorirane klonove	Moguće je pronaći mehanički refaktorirani kod.	Nije prilagođen za refaktorirani kod.	Nije preporučljivo, no moguće je.	Potrebna je ljudska prosudba.

Prema Cordy i Roy (2007).

2.5.4. Tehnike temeljene na grafikonu programske ovisnosti(GPO)

Grafikon programske ovisnosti programa nadogradnja je na prethodnu tehniku. Ako se pojedine elemente koda prikaže uz pomoć toka podataka i informacija dijagramski, dobiva se grafikon međuovisnosti pojedinih elemenata. Kontrolom toka podataka se time dobiva vjerni prikaz funkcionalnosti i način rada gotovog programa. Navedena transformacija se vrši interpretacijom koda uz pomoć semantičkih pravila za pojedini jezik. Individualni program se je potom prikaza u obliku dijagrama toka što čini njegov osnovni GPO. Osnovni GPO se ovisno o kompleksnosti programa može sastojati od manjih grafikona čime se stvara preglednost ukupne topologije programa.

GPO predstavlja grafički prikaz izvornog koda procedure. Unutar navedenog prikaza prikazuju se sljedovi podataka i kontrole nad podacima uz pomoć bridova, a pojedine naredbe, funkcije i drugi makroelementi su prikazani uz pomoć čvorišta, najčešće kuglama. Time GPO predstavlja određeni oblik s čvorištima i stranicama točno određene topologije za pojedini program.

Prednost navedenog je da komparacije topologija dvaju ili više GPO-a može točno vidjeti plagiranje programa i iznimno je velika osjetljivost na kloniranje neovisno o redoslijedu, izmjeni samog koda ili identifikatora.

Suštinska mana ovakve metode je iznimna kompleksnost GPO-a koja nastaje s obzirom na osnovni program. Navedena kompleksnost raste eksponencijalno s povećanjem kompleksnosti koda i time je računski nepovoljno tako obrađivati značajno velike programe.

Nastavno slijedi prikaz komparacije različitih algoritama za GPO usporedbu prilikom detekcije plagijata. Rezultati usporedbe su preuzeti iz Cordy i Roy(2007, vlastiti prijevod) te su prikazani tablicom 5.

Tablica 5 Prikaz komparacije detekcije plagijata pomoću GPO tehnike.

Tehnike bazirane na GPO			
Karakteristika	**Komondoor et al.**	**Krinke**	**Liu et al.**
Normalizacija ili transformacija	Stvaranje GPO pomoću CodeSurfer.	Stvaranje GPO	CodeSurfer-om se stvara GPO.
Reprezentacija koda	Skup GPO-a funkcija.	Sitnozrnati GPO.	Skup GPO s kontrolnim poveznicama.
Tehnika komparacije	Pretraga za izomorfnim GPO pomoću rezova.	Pronalazak podudaranja niza znakova koristeći djelić programa radi komparacije.	Podudaranje izomorfnih GPO
Rast kompleksnosti	Nije dostupno.	Ne ponaša se polinomski.	Polinomski.
Granularnost komparacije	GPO čvorišta.	GPO podgrafovi.	GPO čvorišta.
Sloboda granularnosti kloniranih dijelova	Slobodna, ovisno o gustoći rezova.	Slobodna, ograničena s obzirom na duljinu komparacijskog dijela.	Konstantno, u pravilu prema metodama i funkcijama.
Osjetljivost na tip klona	Izrazito visoka za tip 4.	Tip 1 i 2.	Tip 1 i 3.
Neovisnost promatranog alata	Potrebni su GPO alati.	Potrebni su GPO alati.	Potrebni su GPO alati.
Vrsta izlaza	Parovi i klase klonova.	Klase klonova.	Parovi plagiranih komponenti.
Osjetljivost na refaktorirane klonove	Pronalazi refaktorirane klonove.	Može pronaći pojedine primjere refaktoriranog koda.	Nije izvedivo, cilj je prikazati dva različita programa kroz isti format - GPO.

Prema Cordy i Roy (2007).

2.5.5. Tehnike komparacije karakterističnih vrijednosti

Analiza karakterističnih vrijednosti je markantno drugačija od prethodnih tehnika. Unutar njezina rada ne uspoređuje se sam izvorni kod, već se pojedini elementi koda prikazuju pomoću svojih vrijednosti ulaza i izlaza. Navedene vrijednosti ulaza i izlaza nazivaju se "otiskom prsta" programa (u engleskoj literaturi: *"fingerprint"*) zbog svojstva unikatnosti navedenog elementa. Komparacija za klonove provodi potom kroz usporedbu unikatnog obrasca ulaza i izlaza koje posjeduju elementi i sam program naspram baze podataka. Kako bi se postigla razdioba na elemente, kod se vrlo često provodi kroz prethodno spomenuti GPO i ASS proces raščlanjivanja kako bi se moglo prikazati ulaze i izlaze pojedinog elementa kao vektore, tj. prikazati unikatne,karakteristične vrijednosti programa i potprograma kao vektore nad ulaznim podacima. Ova tehnika komparacije nije svojstvena i primjenjiva isključivo u programskog kodu već se koristi i za otkrivanje duplikata u računalnim mrežama, ali i u dokumentima uz male preinake.

U nastavku slijedi tablični prikaz rezultata komparativne analize, algoritama detekcije uz pomoć karakterističnih mjera, koju su izvršili Cordy i Roy (2007, vlastiti prijevod). Tablični prikaz je vidljiv u tablici 6.

Tablica 6 Komparativna kvalitativna analiza algoritama detekcije plagijata na osnovi karakterističnih mjera.

Tehnike bazirane na karakterističnim mjerama				
Karakteristika	**Kontogiannis**	**Mayrand**	**Di Lucca**	**Lanubile**
Normalizacija ili transformacija	U karakteristične vektore.	U ASS pa potom u prijelazni, srednji jezik.	HTML datoteke se raščlanjuju kako bi se uklonile HTML naredbe. Kompozitne naredbe se zamjenjuju ekvivalentnima.	Korištenjem karakterističnih veličina određuju se funkcije kandidati s obzirom na njihova imena.
Reprezentacija koda	Karakteristični vektori koji prikazuju tok informacija kroz program.	Datrixom određene mjerne veličine.	HTML ili ASP tekstualne varijable.	Određene reprezentativne vrijednosti osnovne funkcije i toka programa poput broja linija, duljine linija i dr.
Tehnika komparacije	Numerička komparacija karakterističnih veličina koristeći udaljenosti u euklidskim prostorima i drugim tehnikama komparacije vektorski prikazanih programa.	21 funkcijskih karakterističnih veličina na 4 mjerne točke.	Levenshtein udaljenost.	Vizualna inspekcija izmjerenih veličina funkcije.
Rast kompleksnosti	Naivni pristup: $O(n^2)$ DP-model: $O(n \times m)$ m=veličina modela n=veličina ulaznog koda.	Polinomski.	$O(n^2)$ n=duljina dulje tekstualne varijable.	Nije dostupno.
Granularnost komparacije	Karakteristične veličine izjavnih blokova.	Karakteristične veličine pojedine funkcije.	Tekstualna reprezentacija cjelokupne stranice.	Vizualni pregled funkcije.
Sloboda granularnosti kloniranih dijelova	Konstanta koja iznosi jedinični izjavni blok.	Konstanta, na razini funkcije.	Cijela statička stranica.	Konstanta, na razini funkcije.
Osjetljivost na tip klona	Tip 1,2,3	Tip 1 i 3	Tip 2	Svi tipovi, potrebna je ljudska provjera.
Neovisnost promatranog alata	Alat za raščlambu.	Potreban je Datrix.	Potrebno je prethodno raščlaniti web stranicu.	Potreban je eMetrics alat za stvaranje karakterističnih veličina.
Vrsta izlaza	Parovi.	Klase i parovi klonova	Klase klonova.	Skup funkcija s obzirom na karakteristične veličine.
Osjetljivost na refaktorirane klonove	Potrebna je ljudska prosudba.	Može pomoći kod automatiziranog refaktoringa.	Nije primjenjiva.	Ljudska prosudba je potrebna.

Prema Cordy i Roy (2007).

2.5.6. Hibridne tehnike

Prema svojoj funkcionalnosti, možemo također odvojiti i prikazati hibridne pristupe kao odvojenu kategoriju. Njihov način rada je multidimenzionalni pristup, tj. pristup u kojemu kombiniraju različite prethodno spomenute metode. Može se kao primjer navesti Leitao – Alat za detekciju kloniranog koda za *Lisp* (obitelj računalnih programskih jezika s istaknutom sintaksom zasnovanoj na zagradama). Leitao pruža hibridni pristup koji kombinira sintaktičke(tekstualne) i semantičke(temeljene na stablima) tehnike kombinacijom specijaliziranih funkcija za usporedbu. Svaka funkcija istražuje različite značajke izvornog koda koje potom međusobno uspoređuju rezultate kroz različite aspekte. Prednost kod ovog pristupa je što je kombinacijom pristupa moguće detektirati veću i značajniju korelaciju za pojedine dijelove koda koji bi bili granični korištenjem jedne tehnike. Dodatno se prednost ostvaruje lakšim odbacivanjem lažnih pozitivnih rezultata.
Komparativna analiza je izvršena unutar rada Cordy i Roy (2007, vlastiti prijevod) i prikazana je u tablici 7.

Tablica 7 Prikaz rezultata prethodnih istraživanja u svojstva hibridno baziranih tehnika.

Hibridno bazirane tehnike					
Karakteristika	**Koschke et al.**	**Jiang et al.**	**Tairas et al.**	**Greenan**	**Balazinska et al.**
Normalizacija ili transformacija	Raščlamba u ASS i potom u serijalizirani oblik ASS.	Raščlanjuje ga se u ASS i potom se karakteristične vektorske matrice podjedinica prenose u ASS oblik.	Raščlamba u ASS i transformacija u seriju ASS čvorišta.	Raščlamba u ASS.	Diskretizirani i pribilježeni ASS tokeni.
Reprezentacija koda	Serijalizirani ASS oblik.	Vektori konstantnih dimenzija.	Niz notiranih ASS čvorišta.	Niz notiranih ASS čvorišta u obliku teksta	Niz tokena i notiranih ASS i metričkih vrijednosti metoda koje su pribavljene kroz Datrix alat.
Tehnika komparacije	Sufiksno stablo bazirano na stringovima.	Podudarnost stabala.	Podudarnost sufiksnih stabala.	Točno podudaranje najduljeg zajedničkog podniza i Smith-Waterman algoritam.	Dinamičko podudaranje uzoraka za podudarne sekvence tokena
Rast kompleksnosti	$O(N)$ N= broj ulaznih i izlaznih čvorišta	$O(n^{p+logn})$	$O(m^2)$ m=duljina serije	$O(n^2)$ n=broj metoda	$O(n \times m)$ n=broj tokena prve metode, m=broj grananja metode.
Granularnost komparacije	Serijalizirano ASS čvorište.	Podstabla s numeričkim vektorima u euklidskim prostorima.	ASS čvorište serije.	ASS čvorište serije.	Mjerne vrijednosti i serija tokena metoda.
Sloboda granularnosti kloniranih dijelova	Slobodna, vezana graničnom mjerom.	Slobodna, vezana graničnom mjerom tokena broja vektora.	Konstantna, na razini funkcija.	Konstantna, na razini metoda.	Konstantna, na razini metoda.
Osjetljivost na tip klona	Tip 1 i 2	Tip 1 i 3	Tip 1 i 2	Tip 1 i 3	18 različitih kategorija plagijata.
Neovisnost promatranog alata	Potreban je poseban raščlanjivač za novi jezik.	Uspoređuje bez poteškoća.	Potreban je Phoenix programski okvir za generiranje ASS.	Potreban je poseban raščlanjivač za novi jezik.	Potreban je poseban raščlanjivač, klasifikator i generator mjerenih vrijednosti za novi jezik.
Vrsta izlaza	Parovi	Parovi za koje je potrebna ručna obrada i selekcija.	Klonovi po cijelom kodu, imenima funkcija i klonovi klasa.	Parovi	Parovi i klase klonova.
Osjetljivost na refaktorirane klonove	Može otkriti programsko refaktoriranje.	Pronalazi programski refaktorirane klonove.	Pronalazi refaktorirane klonove.	Može otkriti programsko refaktoriranje.	Omogućava detekciju različitih vrsta refaktoriranja.

Prema Cordy i Roy (2007).

3. USPOREDBA ALATA ZA DETEKCIJU PLAGIJATA

Opća analiza dostupnih alata se vrši kroz ovo poglavlje. Navedena analiza prvo prikazuje pojedine alate s njihovim karakterističnim značajkama. Dodatno, prikazan je i jedan alat obfuskacije koda programa radi prikaza profesionalnosti pri izradi plagijata i uvjeta protiv kojih se ovi programi moraju izboriti. Program obfuskacije je dodatno prikazan zbog njegovo korištenja u prethodnim istraživanjima kojima se došlo do apsolutnih i numeričkih rezultata vrijednosti. Navedeno korištenje rezultata prethodnih istraživanja je odabrano kao izvor podataka zbog fizičke nedostupnosti i time nepotpune komparativne analize pojedinih programa čime bi se otvorila mogućnost stvaranja krivog zaključka.

3.1. Prikaz popularnih alata

Unutar ovog poglavlja, dan je prikaz trenutno popularnih alata za detekciju plagijata. Sveukupno se ovdje nalazi četiri alata koji predstavljaju najveći dio tržišta detekcije softverskih plagijata. O svakome od njih individualno više informacija slijedi u narednim poglavljima.

3.1.1. MOSS

Mjerilo softverske sličnosti (u engleskoj literaturi: „*Measure Of Software Similarity*“, MOSS) čini mogućim objektivno i automatski provjeriti sva rješenja i detektirati potencijalne plagijate. Alat je 1994. razvio Alex Aiken, izvanredni profesor računalnih znanosti na UC Berkeley.

MOSS podržava programe pisane u C, C++, Java, Pascal, Ada i drugim programskim jezicima. Programi koje se želi provjeriti šalju se direktno na MOSS server u Berkeley-u. Prije slanja potrebno je na web stranici <www.cs/berkeley.edu/~aiken/moss.html> napraviti MOSS račun te instalirati od strane MOSS-a poslanu skriptu. Link na

rezultate mjerenja točnosti korisniku se šalje mailom, a rezultati ostaju dostupni na MOSS serveru narednih dva tjedna.

MOSS uspoređuje parove programa po sličnosti i generira web stranicu s rezultatima. Za svak par programa MOSS daje izvještaj o broju linija koje se podudaraju i stupanj sličnosti. Algoritam koji se koristi u ovom alatu je puno sofisticiraniji od drugih koji se koriste u ostalim alatima za detekciju, radi se o tehnici posebnoj tehnici filtracije – engleskog naziva „*winnowing"*(doslovni prijevod na hrvatskom bio bi „odvajanje žita od ljuski puhanjem zraka") koja služi lociranju podudarajućih sekvenci između dva dokumenta. Bowyer i Hall (1999) navode kako su rezultati komparacije 75 do 100 programa pisanih u C-u, svaki dug par stotina linija, bili dostupni isti dan. Na slici 2 prikazani su rezultati njihovog istraživanja u mogućnosti MOSS-a

Slika 2 Prikaz rezultata rada u MOSS-u

Moss Results

Sun Mar 14 15:24:02 PST 1999

Options -l c -m 10

[Text Report | How to Read the Results | Tips | FAQ | Contact Moss | Submission Scripts | Credits]

File 1	File 2	Tokens Matched	Lines Matched
mike_wolf.c (79%)	mike_fox.c (80%)	463	139
bill_smyth.c (86%)	bill_smith.c (88%)	456	133
jane_white.c (59%)	jane_blanco.c (68%)	354	111
john_doe.c (100%)	john_deer.c (100%)	220	49

Any errors encountered during this query are listed below.

Iz istraživanja Bowyer i Hall(1999.)

Vidljivo je na slici kako je u rezultatima mjerenja sličnosti, dat prikaz programa koji imaju znatnu sličnost, čija je mjera izražena u postocima, moguće je vidjeti broj tokena i broj linija koji se podudaraju kod svakog para te postotak preklapanja svakog izvornog koda s drugim programom. Za svaki par podudarajućih programa moguće je vidjeti sažetak koji usporedno prikazuje izvorni kod oba programa. MOSS može pronaći i nešto sofisticiranije primjere plagijata. Brojni zasebni slični dijelovi, iako razdvojeni drugim dijelovima, također budu pronađeni i označeni. Mijenjanje imena

varijabli, promjena imena i redoslijeda funkcija, različito prorjeđivanje i komentari ne pomažu u pokušajima prekrivanja plagijata i takvi plagijati su redovito detektirani od strane MOSS-a.

Istraživači i autori članka izvršili su testiranje MOSS-a na temelju kolegija i provjere predanih radova studenata. U prvome semestru kada je korišten MOSS na uzorku od 75 studenata ukupno je njih 10 dobilo je negativnu ocjenu zbog plagijata. U idućem semestru od 140 studenata njih 9 dobilo je negativnu ocjenu zbog plagijata. Podaci pokazuju kako se postotak plagijata znatno smanjio u odnosu na prvi semestar. Jedno mišljenje kojeg navode autori rezultat je kasnijih iskustva s plagijatima i ukazuje na to da su studenti pronašli druge načinje varanja na programskim zadacima. U ovim slučajevima pokazalo se da studenti nisu međusobno kopirali programe već su ih za njih pisale treće osobe koje ne pohađaju taj kolegij.

3.1.2. GPlag

Trenutni alati za detektiranje plagijata zadovoljavajući su za akademske svrhe kada se slučajevi plagiranja nastoje prekriti promjenom imena varijabli ili promjenom imena i reda funkcija, ali su se pokazali nedovoljnim u slučajevima ozbiljnije obfuskacije plagiranja kao što je umetanje koda. Postojanje i popularnost projekata otvorenog koda stvara prilike za lako plagiranje softvera. GPlag detektira plagiranje rudarenjem grafova programskih ovisnosti (GPO). GPlag je uspješniji od modernih alata za otkrivanja plagijata zato što grafovi ovisnosti programa ostaju praktički nepromijenjeni u procesu plagiranja kako se ne bi promijenio osnovni rad programa. Kako bi se GPlag podesio za rad s velikim programima preporučuje se korištenje statističkog disipativnog filtera (u eng. literaturi: „*lossy filter*“ – filter poradi kojeg dolazi do određenog gubitaka informacija) kako bi se smanjila količina računskih operacija i time vrijeme potrebno za izvršavanje komparativne analize. Njegova rad je prikazan putem istraživanja autora programa u nastavku.

Autori programa testirali su njegov rad na primjeru plagiranja jezgrenog dijela (u eng. literaturi: „*core-part plagiarism*“). U tu svrhu izabrali su četiri programa za eksperiment.

Prvi program („join“) izabran je za procjenu efikasnosti i njega su sami autori eksperimenta plagirali oko 2 sata tako da su uspjeli prevariti MOSS i JPlag. GPlag je s druge strane uspješno otkrio plagijat u manje od 0.1 sekunde. Detektirao je šest izomorfnih parova od kojih svaki odgovara jednoj plagiranoj proceduri.

Nadanje, autori su testirali efikasnost GPlaga na tri veća programa (bc, less, tar). U tu svrhu identičnu kopiju originalnog programa tretirali su kao plagiranu verziju. GPlag ovakvu verziju programa tretira kao plagijat zato što su GPO grafovi neosjetljivi na preimenovanje identifikatora, promjenu redoslijeda naredbi i zamjenu kontroli. U istraživanju autori su dodatno testirali moć filtriranja pomoću filtara bez gubitaka (u eng.literaturi: „*lossless“)* i filtara s gubicima (u eng. literaturi: „*lossy“*), na tri načina: bez ikakvog filtara, pomoću filtara bez gubitaka te pomoću oba filtara. Rezultate efikasnosti njihovog istraživanja, moguće je vidjeti na tablici 4.

Tablica 7 Efikasnost GPlag-a s obzirom na promatrane programe

	Bez filtera			Filtracija bez gubitaka			Hibridna filtracija		
	Broj parova	Vrijeme	Pogotci	Broj parova	Vrijeme	Pogotci	Broj parova	Vrijeme	Pogotci
be	3,136	251.21	63	1724	251.78	63	293	0.056	60
less	11,449	1171.35	125	6,304	1170.3	125	1,288	7.38	114
tar	6889	853.1	110	3759	850.24	110	722	122.61	89

Studija autora Chao et al. (2006).

Cilj je bio zabilježiti koliko je GPO parova testirano, potrebno vrijeme, kao i broj pronađenih podudaranja. Za svako izomorfno testiranje postavljeno je vremensko ograničenje te ako se testiranje GPO parova ne završi u danom vremenu, GPO par biva označen kao neodgovarajući (za navedene parove, autori koriste pojam:„*time hog“.* Smisleni prijevod bi označavao par koji troši iznimno puno vremena). Zaključci istraživanja su kako filtar bez gubitaka kada korišten samostalno ne štedi mnogo vremena, ali korišten u kombinaciji s filterom s gubitcima znatno skraćuje potrebno

vrijeme, ali prvenstveno na način da zaobilazi neodgovarajuće i računski prezahtjevne parove.

U zadnjem dijelu istraživanja testirana je sposobnost i efikasnost GPlaga-a u detektiranju plagiranja jezgrenog dijela. Ponovno su uzeta tri velika programa (bc, less, tar) te dodane originalne i plagirane verzije šest procedura. Testiranje je izvršeno s filterom s gubicima i bez njega. U slučaju korištenja filtara bilo je puno manje lažno pozitivnih, a vrijeme potrebno za detektiranje plagiranja jezgrenog dijela znatno je smanjeno.

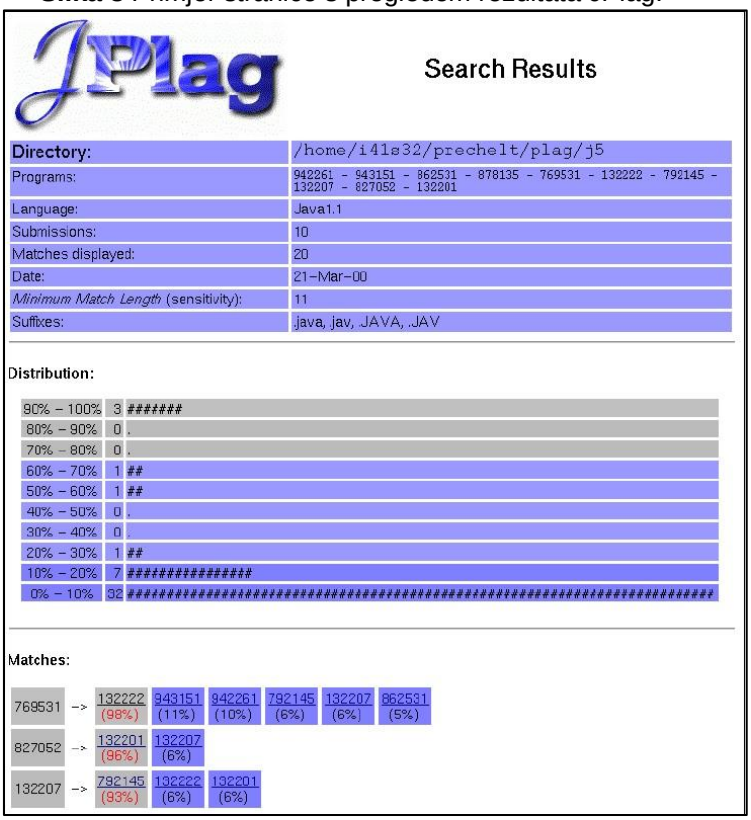
Autori istraživanja uspjeli su pokazati kako je GPlag korišten u kombinaciji s filterima s gubicima uspješniji u pronalaženju plagijata od drugih postojećih tehnika, kako konceptualno, tako i na primjeru navedenog eksperimenta. GPlag koji koristi algoritme temeljene na analizi grafova ipak je manje efikasan od onih temeljenih na analizi sekvenci. Eksperimenti su pokazali kako razlika u efikasnosti nema mnogo utjecaja na praksu s obzirom na to da GPlag obično završi u roku par sekundi čak i kada su u pitanju tisuće linija koda. Autori prepoznaju važnost problematike plagiranja softvera i učinak koji je na to imala popularnost projekata otvorenog izvora te naglašavaju važnost razvoja novih efektivnih i jasnih rješenja za detektiranja plagijata. U tu svrhu autori predlažu GPlag i naglašavaju njegovu prilagodljivost i za velike programe s preko milijun linija koda, te mogućnost jednostavnog proširenja i na druge programske jezike. GPlag je trenutno dostupan u programima pisanim u C, C++ i Java. Zanimljiva je mogućnost korištenja GPlaga u sudskim tužbama za plagiranje. Eksperimentom je pokazano kako GPlag daje jako malo lažno pozitivnih te su male šanse da nešto što nije plagijat tako bude određeno od strane GPlag-a.
Autori dodatno navode kako je plagiranje koda moguće ne detektirati usprkos korištenju GPlag alata, no autori smatraju da bi vrijeme i trud potreban za takvo plagiranje, te i samo znanje potrebno za izmjenu koda možda nadmašilo ono potrebno za pisanje vlastitog koda čime se gubi smisao plagiranja. Iz toga razloga dokle god je točnost i preciznost programa sačuvana plagiranjem, i unatoč izmjenama u kodu, GPlag bi trebao detektirati plagijarizam.

3.1.3. JPlag

JPlag je alat za detekciju sličnosti softvera koji koristi strukturalni pristup. Razvoj JPlag-a započeo je 1996., prvo kao studentski projekt, a već par mjeseci kasnije zaživio je kao online sustav. Razvio ga je Guido Malpohl na Sveučilištu Karlsruhe u Njemačkoj. Godine 2005. Emeric Kwemou i Moritz Kroll pretvorili su JPlag u web servis. JPlag analizira programe izvorno pisane u Javi, Schemi, C-u i C++-u. JPlag uzima kao ulaz skup programa, uspoređuje ih u paru (računajući za svaki par ukupnu vrijednost sličnosti i skup regija sličnosti), a kao izlaz daje skup HTML stranica koje omogućuju detaljno istraživanje i razumijevanje sličnosti.

Postoje dvije faze u radu JPlag-a. Radi tako da su prvo svi programi za usporedbu raščlanjeni i pretvoreni u niz znakova koji predstavljaju strukturu programa. Ti znakovni nizovi uspoređuju se po parovima kako bi se pronašle sličnosti između svakog para. Prilikom usporedbe JPlag uzima jedan niz znakova te ga nastoji preklopiti s podnizom drugog niza, riječ je o algoritmu polaganja niza (eng. *string tiling*). Postotak preklapanja predstavlja mjeru sličnosti između dva programa.

Slika 3 Primjer stranice s pregledom rezultata JPlag.



Preuzeto iz Prechelt, Malpohl, i Philipsen. (2000) *JPlag: Finding plagiarisms among a set of programs.*

Jedna od prednosti JPlag-a je njegovo grafičko korisničko sučelje koje prikazuje rezultate kao set HTML stranica te omogućuje detaljan pregled pronađenih sličnosti između programa. Osnivači ovog alata napravili su testiranje na 4 stvarna seta programa Jave i 8 unaprijeđenih Java programskih setova koji sadrže dodatni plagijat. Rezultate su ukratko saželi na sljedeći način:

- Za jasno plagirane programe, tj. programe preuzete u potpunosti, a zatim modificirane da sakriju podrijetlo, rezultati JPlaga gotovo su savršeni - često čak i ako su programi kraći od 100 redaka.
- Čak i za samo djelomično plagirane programe, JPlag će utvrditi sličnosti, a često ih može prilično dobro diskriminirati od slučajnih sličnosti.
- Tehnike obfuskacije plagijatora iz stvarnih programskih setova bili su beskorisni protiv JPlag-a.
- Pokušaji informiranih plagijatora da prođu su također bili uspješni u manje od 10 posto svih slučajeva. Te osobe su znale da moraju zavarati program i nisu imale drugog cilja osim navedenog.
- Uspješni napadi protiv detekcije JPlag-om zahtijevaju puno rada i time se znatno gubi čitljivost osnovnog koda.
- JPlag je pouzdan s obzirom na neoptimalan izbor svoja dva slobodna parametra, skup tokena i minimalno podudaranje duljina.
- S obzirom na vrijednosti sličnosti koje izračunava JPlag, konstantan granični prag dovoljan je kao kriterij diskriminacije koji razdvaja plagirane programske parove od originalnih s gotovo optimalnim opozivom i štoviše dobrom preciznošću.

Također su naveli kako nije poznato u kojoj se mjeri ti rezultati prenose u druge situacije. Mogu se primijeniti u manjoj mjeri za C i C ++, jer se ovi jezici trenutno ne raščlanjuju, već samo skeniraju. Mogli bi i biti slabiji za različito strukturirane (ili puno veće) programe, posebice u slučaju da napade učine učinkovitijima različiti plagijatori koriste različite vrste i sheme plagiranja (Prechelt, Malpohl i Philippsen, 2006:38)

3.1.4. SIM

SIM je jedan od alata za detekciju sličnosti koji se temelji na mjerenju strukturalne sličnosti između dva C programa što čini bazu za procjenu stila, točnosti i jedinstvenosti programa. Većina SIM-a pisana je u C++ programskom jeziku, dok je grafičko korisničko sučelje pisano u Tcl/Tk (Gitchell i Train, 1999:3)

SIM primjenjuje tehniku poravnanja niza (u engleskoj literaturi: „*string alignment technique"*) originalno razvijenu kako bi se detektirala sličnost između dva niza DNA. Koristeći standardni leksički analizator SIM dva programa najprije reducira na kompaktni oblik koji predstavlja njihovu strukturu kao niz cjelobrojnih podataka odnosno tokena. Svaki token predstavlja aritmetičku ili logičku operaciju, interpunkciju, ključnu riječ, numeričku konstantu ili konstantu niza znakova, komentar ili identifikator. Ovaj proces za cilj ima svesti dane programe na njihova stabla raščlanjivanja koja su obično mnogo kraća te ukloniti nepotrebne informacije prije usporedbe programa. SIM svako stablo raščlanjivanja tretira kao niz te ih poravnava umećući razmake dok njihova dužina ne bude ista, a s ciljem dobivanja maksimalne zajedničke sekvencije tokena. Pritom treba imati na umu da postoji više znatno različitih načina poravnavanja dva niza.

SIM prati sljedeći bodovni sustav kako bi procijenio poravnanje:

- Poklapanje dva identifikatora tokena boduju se kao 2.
- Druge vrste podudaranja boduju se kao 1
- Praznine se boduju kao -2
- Nepoklapanje 2 identifikatora ocjenjuje se kao 0
- Sve druge vrste neskladnosti i nepoklapanja se boduju kao -2

Sličnost između dva programa se potom izražava između 0.0 i 1.0. prema sljedećoj formuli:

$$s = \frac{2 \cdot bodovi(p_1, p_2)}{bodovi(p_1, p_1) + bodovi(p_2, p_2)}$$

Drugim riječima, sličnost programa se procjenjuje količnikom dvostrukog bodovanja između dva niza naspram zbroja bodovanja svakog niza sa samim sobom.

Tcl/Tk grafičko korisničko sučelje omogućava prikaz, usporedbu i ispis rezultata u obliku grafikona. Dostupno je i drugo odvojeno sučelje koje za prikaz rezultata koristi GNUPlot.

Autori članka testirali su SIM na C programu. Plagirali su ga tako da su promijenili imena svih varijabli i funkcija, uklonili komentare, zamijenili redoslijed susjednih naredbi kada god je to bilo moguće, te su promijenili i redoslijed funkcija. Sličnost između dva programa SIM je ocijenio brojem 0.4 što je prema autorima dovoljno da pobudi sumnju u plagijat. Nadalje, autori su testirali SIM na 36 programa domaćih zadaća. Rezultati navedenog istraživanja nalaze se u tablici 8.

Tablica 8 Statistika 1. grupe

	Veličina programa	Veličina tokena	Vrijeme
Prosjek	7808	714	0.20
Maksimum	15027	1361	0.80
Ukupno	296716	27157	124

Prema Gitchell i Tran (1999:3)

Svaki program korišten je kao referentni program za usporedbu sa svim ostalim programima. SIM je uspješno otkrio sličnost između dva programa (rezultat u intervalu od 0.6 do 0.85) kojeg je modificirao jedan od autora eksperimenta i referentnog programa te pet programa (sličnost između 0.7 i 1.0) koji su već ranije profesori ručno kategorizirali kao plagijatima. Zadnji eksperiment su autori proveli nad 56 programa koji su prikupljeni kao zadaće na različitim kolegijima. Učitelji niti jedan od ovih programa nisu odredili kao plagijat.

Tablica 9 Statistika 2. grupe

	Veličina programa	Veličina tokena	Vrijeme
Prosjek	3415	399	0.09
Maksimum	8050	835	0.48
Ukupno	191243	22352	139

Prema Gitchell i Tran (1999:3)

Rezultati eksperimenta pokazali su uspješnost SIM-a u detektiranju plagijata kada je riječ o promjenama kao što su ekstenzivne promjene imena, promjena redoslijeda naredbi i funkcija, dodavanje odnosno uklanjanje razmaka i komentara. Navedene promjene nisu uvijek bile uočljive čak niti čovjeku, no ovim je istraživanjem prikazana njegova korisnost i značajnost pretrage. Kada su manji programi u pitanju SIM je brz, za usporedbu svih parova 56 programa čija je prosječna duljina 3415 bitova trebalo mu je svega 105 sekundi. Time je navedeni program prikladan za usporednu analizu.

3.2. Prikaz prethodnih istraživanja i rezultata

Kako bi se stvorio uvjet za ovo istraživanje, potrebno je bilo proučiti prethodne radove na navedenu tematiku. Usporedna kvalitativna analiza se provela kroz više prethodnih istraživanja:

- Cordy i Roy (2007) – Rezultati njihovog istraživanja su prethodno prikazani i služe kao temelj ovoga istraživanja.
- Martins et al. (2014) – Rezultati vidljivi u nastavku ovog poglavlja.
- Divya et al (2014) – Autori ovog istraživanja su se također osvrnuli na značajnu brojku prethodno navedenih alata.

Unutar istraživanja provedenog od strane Martinsa et al. prikazane su tri različite korištene tehnike između promatranih alata. Krenuvši od najmanje točne, ali najbrže prema onoj s najvećom točnošću, ali najnižom efikasnošću:

1. Metodologija temeljena na atributima npr. korištenje broja varijabli, funkcija ili klasa kako bi se izdvojili izvorni kodovi s različitim brojem varijabli.
2. Metodologija temeljena na tokenima kada je izvorni kod pretvoren u token i *hash* pomoću kojih se stvara *fingerprint.*
3. Metodologija temeljena na strukturi koristi apstrakciju kako bi izvorni kod pretvorila u posrednu unutrašnju reprezentaciju (u engleskoj literaturi „*Internal Intermediate Representation“*) koji je zatim korišten za usporedbu.

Njihovo istraživanje se baziralo na sljedećih osam kriterija:

1. Podržani jezici.

2. Jednostavnost proširenja na druge programske jezike.
3. Kvaliteta rezultata (ovdje definirana kao mogućnost razlikovanja plagijata od lažno pozitivnih).
4. Grafičko korisničko sučelje (GUI).
5. Mogućnost ignoriranja bazičnog koda.
6. Mogućnost rada s grupom datoteka.
7. Off-line ili on-line alati.
8. Alat otvorenog izvornog koda.

Prikaz rezultata njihovog rada vidljiv je na slici 4 u nastavku.

Slika 4 Prikaz rezultat istraživanja Martins

	1	2	3	4	5	6	7	8
CodeMatch	36	✗	✓	✓	✓	✗	✓	✗
CPD	6	✓	✗	✓	?	?	✓	✓
JPlag	6	✗	✓	✓	✓	✓	✗	✗
Marble	5	✓	✓	✗	✓	?	✓	✗
MOSS	25	✗	✓	✓	✓	✓	✗	✗
Plaggie	1	?	?	✓	?	?	✓	✓
Sherlock	1	✗	✗	✗	✗	✗	✓	?
SIM	7	?	✓	✗	✗	✗	✓	?
YAP	5	?	✓	✗	✓	?	✓	✗

Prema Martins et al.(2014).

U pogledu primarne funkcionalnosti detekcije plagijata, autori su postavili 8 različitih vrsta plagijata koje su zatim primijenili na programima:

1. Plagijat koji je vjerna kopija originalnog programa.
2. Plagiranje tako da su samo komentari promijenjeni ili uklonjeni.
3. Promjena svakog identifikatora npr. imena varijabli i funkcija.
4. Lokalne varijable pretvorene su u globalne i obrnuto.
5. Promjena matematičkih operacija i operandi koje se uspoređuju.
6. Plagiranje tako da su tip varijable i kontrolna struktura zamijenjeni ekvivalentima.
7. Promjena redoslijeda naredbi.
8. Grupa naredbi poziva promijenjena je u pozivanje funkcije i obrnuto.

Rezultati istraživanja pokazali su kako je prvi oblik plagiranja, odnosno obična kopija programa, lako detektiran od strane svih alata. Za većinu alata uspoređivanje se pokazalo težim kada su u pitanju plagijati pod brojevima 6-8 s obzirom na to da je riječ o brojnim manjim promjenama ili premještanju više blokova izvornog koda. Različiti alati različito su reagirali na nabrojane oblike plagiranja tako je primjerice MOSS imao najviše problema s brojnim tipovima plagiranja zato što je previše fokusiran na izbjegavanje lažno pozitivnih rezultata što rezultira odbacivanjem mnogo informacija. Za alat Sherlock problematičnim su se pokazali izmijenjeni komentari zato što uspoređuje cijeli izvorni kod bez da ignorira komentare. CodeMatch je imao problema s promjenom identifikatora jer njegov algoritam mora pronaći neku poveznicu između identifikatora. Rezultati su pokazali da svaki alat ima svoju slabost, ovisno koji je oblik plagijata u pitanju, no s druge strane nijedan od rezultata nije u potpuno kriv, samo je podudarnost u nekim slučajevima jako mala.

S druge strane Divya et al.(2014) su također izvršili komparaciju na osnovi sljedećih 10 kriterija:

1. Podržani programski jezici
2. Lakoća proširenja na druge programske jezike
3. Prezentacija rezultata
4. Lakoća korištenja
5. Izuzeće predloška – primjerice, neki studentski zadaci dijele isti osnovni kod koji čini temelj zadatka.
6. Izuzeće malih datoteka koje mogu rezultirati visokim postotkom sličnosti, a ustvari ne predstavljaju plagijat – lažno pozitivni rezultati.
7. Povijesna usporedba – sposobnost alata da komparira nove programe s ranije predanim i uspoređenim programima istih programskih zadataka bez da ponovno međusobno uspoređuje stare programe
8. Mogućnost rada s grupom datoteka
9. Off-line ili on-line alati
10. Postojanje licence otvorenog koda.

Rezultate njihovog istraživanja moguće je vidjeti u tablici 10.

Tablica 10 Usporedba alata za detekciju plagijata

Značajka	GPlag	JPlag	Marble	Moss	Plaggie	SIM
Jezična podrška	Svi	6	1	23	1	5
Nadogradivost	Svi	Ne	Ne	Ne	Ne	Svi
Prezentacija rezultata (1-5)	5	5	3	4	4	2
Upotrebljivost	5	5	2	4	3	2
Izuzeće kodnih predložaka	Da	Da	Ne	Ne	Da	Ne
Izuzeće malih datoteka.	Da	Da	Das	Da	Ne	Ne
Povijesna usporedba	Ne	Ne	Da	Ne	Ne	Da
Lokalni ili web	Web	Web	Lokalni	Web	Lokalni	Lokalni
Otvorenog koda?	Da	Ne	Ne	Ne	Da	Da

Prema Divya et al. (2014).

Za numeričke kriterije korištena je skala ocjenjivanja do 5, gdje 5 predstavlja najvišu ocjenu. Od testiranih alata ističe se GPlag koji podržava sve programske jezike te je istovremeno i najbrži alat.

Na temelju ovih istraživanja prelazi se u analizu alata.

3.3. Komparacija dostupnih alata

Prethodno su prikazani alati nad kojima su se provela istraživanja njihove sposobnosti detekcije plagijata. Nad navedenim alatima provodi se istraživanje njihovih kvalitativnih osobina s obzirom na osnovnu funkcionalnost. U tu svrhu, u nastavku se nalazi popis karakteristika prema kojima se vrši komparacija. Nakon popisa karakteristika, u sljedećem potpoglavlju nalazi se tablica s rezultatima istraživanja.

3.3.1. Promatrane karakteristike

Promatrajući izrazitu raznolikost metodike i tehnika rada programa, potrebno je bilo odrediti faktore i uvjete pod kojima se određuju karakteristike programa. Prethodnim istraživanjem dostupne literature, predložile su se sljedeće karakteristične veličine usporedbe, kako bi se omogućilo držanje konzistencije s prethodnim istraživanjima:

- Normalizacija ili transformacija
- Reprezentacija koda
- Tehnika komparacije
- Rast kompleksnosti
- Granularnost komparacije
- Sloboda granularnosti kloniranih dijelova
- Osjetljivost na tip klona
- Neovisnost promatranog alata
- Vrsta izlaza
- Osjetljivost na refaktorirane klonove
- Jezična podrška
- Nadogradivost
- Prezentacija rezultata (1-5)
- Upotrebljivost
- Izuzeće kodnih predložaka
- Izuzeće malih datoteka.

- Povijesna usporedba
- Lokalni ili web
- Otvorenost koda

Odabirom ovih komparacijskih osobina moguće je dobiti približno realnu sliku rada navedenih alat zahvaljujući prethodnim istraživanjima tematike.

3.3.2. Rezultati istraživanja

Za prethodne veličine usporedbe, u nastavku se nalazi tablica s ukupnim podacima proizašlih iz testiranja, te tamo gdje to nije bilo moguće, iz prikupljenih poadataka iz prethodnih istraživanja. Rezultati su vidljivi u tablici 11.

Tablica 11 Prikaz rezultata istraživanja.

Značajka	MOSS	GPlag	JPlag	SIM
Normalizacija ili transformacija	„Winnowing“ filtracija i prethodna normalizacija	GPO	Raščlanjivanje na niz znakova.	Vrši ju pojedini leksički analizator.
Reprezentacija koda	Tokenizacija linija.	Graf PO.	Znakovni niz.	Niz stabla raščlanjivanja (ASS).
Tehnika komparacije	Poklapanje adresa tokena u ukupnom nizu	Komparacija topologije GPO.	Međusobno poklapanje nizova.	Tehnika bodovanja poravnanjem niza.
Rast kompleksnosti	Nije dostupno.	Inače eksponencijalan, no uz disipativni filter, smanjen.	$O(S_1 S_2)$ S_1=Veličina prvog niza. S_2=Veličina drugog niza.	$O(S_1 S_2)$ S_1=Veličina prvog niza. S_2=Veličina drugog niza.
Granularnost komparacije	Linija koda, segment linije.	GPO čvor.	Niz određene duljine.	Pojedino ASS čvorište.
Sloboda granularnosti kloniranih dijelova	Slobodna.	Bez ograničenja prema većim brojevima osim sve veće cijene operacija.	Slobodna.	Konstantna, određena brojem čvorišta.
Osjetljivost na tip klona	Tipovi 1, 2 ,3	Svi.	Svi.	Tipovi 1, 3.
Neovisnost alata	Da.	Zahtijeva element za raščlambu osnovnog koda.	Da.	Potreban je leksički analizator.
Vrsta izlaza	Parovi klonova.	Klase i parovi klonova.	Klase kloniranih dijelova.	Parovi kloniranih dijelova.
Osjetljivost na refaktorirane	Može pronaći, no nije savršen.	Da, iznimno osjetljiv.	Djelomična, ne preporučuje se kao trajno rješenje.	Nije prikladan.
Jezična podrška	C, C++, Java, Pascal, Ada i dr.	C, C++, Java	C, C++, Java, Python	C, C++, Java, Pascal, Modula-2, Miranda, Lisp, 8086 Assembler
Nadogradivost	Ne.	Da.	Da.	Da.
Prezentacija rezultata (1-5)	Dobar(3).	Odličan(5).	Vrlo dobra(4).	Zadovoljavajuća(2).
Upotrebljivost	Vrlo dobra (4).	Odlična(5).	Vrlo dobra(4).	Zadovoljavajuća(2).
Izuzeće kodnih predložaka	Da.	Da.	Da.	Ne.
Izuzeće malih datoteka.	Da.	Da.	Da.	Ne.
Povijesna usporedba	Da.	Da.	Ne.	Da.
Lokalni ili web	Web.	Lokalni.	Lokalni.	Lokalni.
Otvorenog koda?	Ne.	Ne.	Da.	Da.

3.4. Diskusija rezultata istraživanja

Kroz istraživanje vezano uz ovaj rad, pomno je istražena i prikazana problematika detekcije plagijata programske podrške.

Kroz prikaz procesa detekcije plagiranje objašnjen je osnovni proces i osnovne tehnike detekcije. Najprimitivniji algoritmi nisu uopće imali transformaciju izvornog koda već se on pregledavao ručno, znak po znak. Danas su algoritmi detekcije znatno napredovali i dostigli su razinu prikazivanja apstrakcije i čiste logike programa neovisno o osnovnom kodu i njegovom jeziku. Napredni prikaz kontrole i tijeka podataka se u tom pogledu naziva grafikonom programskih ovisnosti – GPO. GPO je vjerna replika procesa programa kakav je razumljiv čovjeku i omogućuje napredne procese komparacije kakve bi i čovjek mogao provoditi uz mogućnost apsolutne preciznosti koja odlikuje računala.

GPO sadrži čvorišta koja su povezana tokom informacija i kontrole. Takvim načinom prikaza, programska logika je prikazana u svojom najsirovijem obliku. Sirov oblik je time neosjetljiv na sam jezik, već je to prikaz same implementacije programa. Logično je zaključiti kako kod klonova logika implementacije programa mora biti ekvivalentna. U GPO prikazu, klonirane funkcije, programi, metode, klase i dr. očito moraju posjedovati isti oblik. Drugim riječima rečeno, njihovi GPO imaju istu topologiju. Dodatne metode komparacije osim direktne komparacije opće topologije moguće je vršiti komparacijom singularne topologije na manjim elementima. Veličinu elemenata određuje se pomoću razine granulacije, laički rečeno, hoće li razina granulacije biti pojedina izjava, sintaktički element ili pak funkcija, metoda, klasa i dr. ovisi o osnovnim postavkama koje korisnik odredi GPO algoritmu. Ovisno o razini granulacije moguće je komparirati različite razine oblika osnovnog programa tako što se pojedini zatvoreni elementi prikazuju kao čvorišta ili se ulazi u njihove detalje. Time su GPO algoritmi iznimno robusni na zamračenja pomoću odjeljivanja u različite klase i funkcije. Dodatno, ako plagijator pokuša i izmijeniti dio određenih funkcija u namjeri obfuskacije rezultata pomoću odjeljivanja, brisanja, dupliciranja i prerasporedom funkcija, ukoliko time ne želi mijenjati osnovni rad programa, ne smije mijenjati tokove podataka i kontrole. GPO će očito za znatno različite programe s istim tokom podataka i kontrole

prikazati gotovo isti oblik. Time GPO je posebno prilagođen za plagijate tipa 3 i tipa 4 koje je klasičnim metodama iznimno teško uočiti. Kombinacijom GPO algoritma s osnovnom usporedbom znakova i normalizacijom identifikatora stvara se algoritam koji je iznimno brz u uočavanju plagijata tipa 1 i tipa 2, te iznimno točan u uočavanju plagijata tipa 3 i tipa 4. Upravo se na navedenom algoritmu bazira GPlag koji je kroz sva prethodna istraživanja prikazan kao najtočniji detektor plagijata nauštrb brzine njegove izvođenja za velike programe.

Kod velikih programa je problem što njegova kompleksnost raste eksponencijalno s brojem funkcija, klasa, metoda i samih pojedinačnih manjih datoteka kojima se program koristi. Što je program veći i kompleksniji, to je kompleksnija i analiza potrebna za stvaranje simboličkog grafikona programskih ovisnosti. Nelinearnim rastom, GPO algoritmi nisu prilagođeni za izrazito velike programe. Pokušaj prilagodbe GPO algoritama na većim sustavima je prethodno vidljiv kroz primjenu filtracije osnovnog koda programa. U slučaju korištenja kombinacije filtera bez gubitaka, pa potom prolaz kroz filtera koji stvara određene gubitke, postižu se iznimne razine optimizacije kojima se povećanje kompleksnosti ne ponaša eksponencijalno s brojem osnovnih elemenata i struktura program, već polinomski. Polinomski ovisno vrijeme rješavanja naspram veličine osnovnog problema je jedan od uvjeta kako bi se omogućila primjena GPO komparacije nad najvećim računalnim programima i time učvrstila pozicija superiornosti GPO algoritama naspram ostalih načina detekcije. Iz razloga što su već stvoreni iznimni napreci u navedenome, očito je zaključiti kako su GPO algoritmi superiorni, osim u slučaju velikih sustava gdje ih je potrebno kombinirati s određenim metodama filtracije kako bi se postigla zadovoljavajuće vrijeme izvođenja. Time je osnovna hipoteza ovoga rada potvrđena kroz analizu prethodnih istraživanja.

4. ZAKLJUČAK

Kroz ovaj rad dat je prikaz cjelokupnog procesa otkrivanja softverskih plagijata. Navedeno se postiglo kroz definiranje samog pojma “plagijat”, ali i njihove kategorizacije s obzirom na razinu kopiranja. Prikazane su i pojedine algoritamske tehnike detekcije kao i prednosti i mane njihovog korištenja. Nakon prikaza osnovnih detekcijskih metoda, dan je i presjek tržišta alata. Pojedini alati poput MOSS, GPlag, JPlag, SIM su prikazani s obzirom na njihove osnovne funkcionalnosti i, gdje je to moguće, dan je prikaz njihova rada. Time je postavljen okvir naspram kojega se ušlo u istraživanje prethodnih znanstvenih istraživanja i proučavanje rezultata istraživanja koja su njime stvorena. Prikazana su dva značajna istraživanja i njihovi rezultantni podaci. Upravo su se ti rezultati potom preuzeli kao osnova za dokazivanje ili opovrgavanje osnovne hipoteze rada. Osnovna hipoteza je prethodno bila određena u smislu dokazivanja značajno boljih mogućnosti detekcije plagijata kod alata koji koriste GPO metodu naspram ostalih alata. Rezultati prethodnih istraživanja i pojedine karakteristike programa su prikazane usporedno čime se došlo do zaključka kako je navedena hipoteza točna i istinita. Istinitost superiornosti GPO tehnike je očekivan rezultat ovoga istraživanja, te se ovim radom dao odgovor uz pomoć sinteze prethodnih istraživanja nad osnovnom tematikom rada. Očito je kako su GPO alati budućnost otkrivanja plagijata programske podrške, no da će za iznimno velike programe morati postati znatno efikasniji. Samo pravovremenom detekcijom plagiranja i pravilnim sankcioniranjem se može izbjeći negativan učinak plagijata. Navedene detekcijske metode se iz tog razloga preporučuju kao alat za pomoć u redovitoj nastavi kako bi se omogućilo opominjanje plagiranja programskog koda i time omogućilo studentu da svoje obrazovanje upotpuni i vještinama programiranja koje su danas od sve veće važnosti pri zapošljavanju.

POPIS LITERATURE

A) Knjige

1. CORDY, J. i ROY, C. (2007) *A Survey on Software Clone Detection Research*, Ontario: Queen's University at Kingston

B) Članci

1. BOWYER, K. W. i HALL, L. O. (1999) Experience using "MOSS" to detect cheating on programming assignments, *Proceedings of the 29th ASEE/IEEE Frontiers in Education Conference*. vol.3.
2. CHAO, L., CHEN, C., JIAWEI, H., YU, S. P., (2006) GPLAG: Detection of Software Plagiarism by Program Dependence Graph Analysis. Na *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* Philadelphia, SAD. 20.-23. kolovoza 2006. str. 872.- str. 881. Dostupno na: https://hanj.cs.illinois.edu/pdf/kdd06_gplag.pdf [Pristupljeno 19. rujna 2020.]
3. COLLBERG, C., MYLES, G. i STEPP, M. (2004) Cheating Cheating Detectors, *arXiv*, Dostupno na: http://goto.ucsd.edu/~mstepp/publications/TR04-05.pdf [Pristupljeno 17. rujna 2020.]
4. DIVYA, L., DIVYA, P. S., JOHNSON, S. L., SREEPRABHA, S., VARGHESE, E. B., (2014) Software Plagiarism Detection Techniques: A Comparative Study. *International Journal of Computer Science and Information Technologies,* vol. 5. (4). str. 5020.-5024. Dostupno na: https://www.ijcsit.com/docs/Volume%205/vol5issue04/ijcsit2014050441.pdf [Pristupljeno: 20. rujna 2020.]
5. GITCHELL, D. i TRAN, N. (1999). Sim: A utility for detecting similarity in computer programs. *SIGCSE Bulletin*. vol. 31. str. 266.-270. Dostupno na: https://www.researchgate.net/publication/221538300_Sim_A_utility_for_detecting_similarity_in_computer_programs [Pristupljeno 19. rujna 2020.]

6. MARTINS, V. T., FONTE, D., HENRIQUES, P. R., DA CRUZ, D. (2014) Plagiarism Detection: A Tool Survey and Comparison. Na *3rd Symposium on Languages, Applications and Technologies.* Bragança, Portugal. 19.-20. lipnja 2014. Dagstuhl, Njemačka: Schloss Dagstuhl--Leibniz-Zentrum fuer Informatik. str. 143. – 158. Dostupno na: https://drops.dagstuhl.de/opus/volltexte/2014/4566/pdf/14.pdf [Pristupljeno 19. rujna 2020.]
7. PRECHELT, L., MALPOHL, G. i PHILIPPSEN, M. (2000) *JPlag: Finding plagiarisms among a set of programs.* Karlsruhe, Njemačka: Fakultat fur Informatik. Dostupno na: http://page.mi.fu-berlin.de/prechelt/Biblio/jplagTR.pdf [Pristupljeno 21. rujna 2020.]

POPIS SLIKA

Slika 1 Dijagramski prikaz faza procesa detekcije. 17
Slika 2 Prikaz rezultata rada u MOSS-u..... 32
Slika 3 Primjer stranice s pregledom rezultata JPlag. 36
Slika 4 Prikaz rezultat istraživanja Martinsa et al. 41

POPIS TABLICA

Tablica 1 Prikaz najčešćih razloga kopiranja prema kategorijama. 9
Tablica 2 Primjer operacija normalizacije nad izvornim kodom. 19
Tablica 3 Kvalitativna usporedba znakovnih tehnika komparacije za detekciju plagijata. 22
Tablica 4 Prikaz karakterističnih svojstava algoritama detekcije plagijata baziranih na strukturi stabala raščlambe. 24
Tablica 5 Prikaz komparacije detekcije plagijata pomoću GPO tehnike. 26
Tablica 6 Komparativna kvalitativna analiza algoritama detekcije plagijata na osnovi karakterističnih mjera. 28
Tablica 7 Efikasnost GPlag-a s obzirom na promatrane programe 34
Tablica 8 Statistika 1. grupe 39
Tablica 9 Statistika 2. grupe 39
Tablica 10 Usporedba alata za detekciju plagijata 43
Tablica 11 Prikaz rezultata istraživanja. 46