UDC: 519.17 Info M: str. 11-18

VIZUELIZACIJA REZULTATA DETEKCIJE PLAGIJARIZMA U IZVORNOM PROGRAMSKOM KODU
VISUALIZATION OF RESULTS OF SOURCE CODE PLAGIARISM DETECTION

Marko Mišić[1], Marko Milanović[2], Jelica Protić[3]
Univerzitet u Beogradu, Elektrotehnički fakultet, Bulevar kralja Aleksandra 73, Srbija
[1]marko.misic@etf.bg.ac.rs, [2]milanovic.etf@gmail.com, [3]jelica.protic@etf.bg.ac.rs

REZIME: Univerzitetski kursevi računarstva se u velikoj meri zasnivaju na praktičnom radu koji se realizuju kroz projektne i domaće zadatke u izabranom programskom jeziku. Kako bi se borili protiv pojave plagijarizma, nastavnici koriste alate za detekciju sličnosti u programskom kodu, poput alata JPlag i Moss. Ovaj rad se bavi analizom i vizuelizacijom rezultata pomenutih alata pomoću grafova. Za analizu i vizuelizaciju grafova je korišćen alat Gephi za koji su razvijeni dodaci koji omogućavaju učitavanje i obradu rezultata alata JPlag i Moss. U radu su izložene i smernice za podešavanje vizuelnog prikaza, kako bi se što više olakšao proces identifikacije potencijalnih plagijata.

KLJUČNE REČI: detekcija plagijarizma; grafovi; softverski alati; vizuelizacija;

ABSTRACT: University courses in the field of computing are mostly based on practical training through programming assignments and projects. To prevent plagiarism occurrences, source code similarity detection tools have been used, such as JPlag and Moss. In this paper, we discuss analysis and visualization of results of aforementioned tools using graphs. Gephi tool has been used for graph analysis and visualization, and we developed plugins for processing of JPlag and Moss results. The paper also gives directions for customization of visualization to facilitate identification of potential plagiarism cases.

KEY WORDS: graphs; plagiarism detection; software tools; visualization;

1. UVOD

Pojava plagijarizma, odnosno predstavljanja tuđeg autorskog dela kao sopstvenog predstavlja veliki problem današnjeg društva. Problem plagijarizma se često javlja i kod pisanja softvera, kako u softverskoj industriji, tako i u obrazovnom procesu na univerzitetima [1]. Programski kod je u suštini tekst, pa se mogu povući određene paralele sa detekcijom plagijarizma u tekstovima pisanim na govornom jeziku. Međutim, programski jezici imaju znatno formalniju strukturu i striktnija pravila pisanja, što donekle sužava prostor onima koji žele da počine akt plagijarizma. Govorni jezici ostavljaju mnogo više prostora za varijacije od programskih jezika, pa je stoga detekcija sličnosti dva programska koda znatno drugačija od detekcije sličnosti dva teksta napisana na govornom jeziku.

U smislu detekcije plagijarizma u programskom kodu, bitno je eksplicitno razlikovati termine sličnost i plagijat. Detektovana sličnost između dva programska koda ne mora nužno biti rezultat plagijarizma. Pored kopiranja tuđeg programskog koda, postoji niz faktora koji mogu da utiču na pojavu sličnosti: upotreba uobičajenih algoritama, konvencije imenovanja identifikatora, delovi koda nastali korišćenjem alata za automatsko generisanje koda i sl. [2, 3]. Stoga finalnu odluku da li je neki deo koda plagijat mora doneti čovek, a softverski alati razvijeni u tu svrhu predstavljaju značajno pomoćno sredstvo, koje može ukazati na potencijalne slučajeve plagijarizma.

Savremeni univerzitetski kursevi iz oblasti računarstva uključuju znatnu količinu praktičnog rada koji se ostvaruje kroz laboratorijske vežbe, domaće zadatke i projekte, koje studenti najčešće realizuju samostalno u odgovarajućem programskom jeziku [4]. U digitalnoj eri, veoma je jednostavno prepisati delove koda ili kompletan kod napisan od strane drugog lica. Jedan broj nesavesnih studenata na ovaj način pokušava da ispuni svoje predmetne obaveze i predaje profesorima plagirane radove, najčešće uz minimalne izmene. Pritom, plagijarizam u programskom zadatku predstavlja svaka namerna upotreba programskog koda koji je napisao neko drugi, a koja nije adekvatno citirana od strane onoga ko predaje programski kod kao svoj rad [5].

Navedena pojava je dovela do razvoja alata za automatizovanu detekciju sličnosti, sa ciljem da se uporedi veći broj studentskih rešenja (programa) i olakša uočavanje međusobnih preklapanja od strane nastavnog osoblja. U tom smislu, alati za detekciju sličnosti barataju sa većim brojem kraćih izvornih kodova koji predstavljaju studentska rešenja i tom vrstom plagijarizma se ovaj rad prvenstveno bavi. Sa druge strane, u softverskoj industriji povremeno postoji potreba za utvrđivanjem sličnosti manjeg broja softverskih rešenja. Tada se govori o detekciji softverskih klonova sa prevashodnim ciljem da se utvrdi narušavanje patentnih prava i neovlašćena upotreba tuđe intelektualne svojine [6].

Kao što je ranije rečeno, finalnu analizu rezultata pomenutih alata mora obaviti čovek. Alati za detekciju sličnosti najčešće poseduju tekstualni prikaz rezultata u obliku liste parova studentskih radova rangiranih po procentu sličnosti. Međutim, u realnim situacijama se može detektovati međusobna sličnost između više različitih radova, jer je plagijarizam društveni fenomen koji često obuhvata više učesnika [7]. Pritom, sami alati ne daju više mogućnosti za dublju analizu i klasterizaciju rezultata. Stoga je prirodno rezultate poređenja prikazati u obliku grafa. Postoji veliki broj algoritama za analizu grafova, a razvijen je i određeni broj alata za vizuelizaciju koji implementiraju ove algoritme. Cilj ovog rada je da se izvrši integracija alata za detekciju sličnosti sa alatima za vizuelizaciju grafova, kako bi se rezultati prikazali na prirodniji način, a nastavnicima omogućila jednostavnija i fleksibilnija analiza.

Rad je podeljen u nekoliko poglavlja. U drugom poglavlju su opisane tehnike za detekciju sličnosti u programskom kodu, kao i načini na koje studenti modifikuju programski kod da bi zataškali počinjeni akt plagijarizma. U trećem poglavlju su detaljnije opisani JPlag [8] i Moss [9], dva najpopularnija alata za detekciju sličnosti u programskom kodu, sa posebnim naglaskom na prikaz rezultata. Četvrto poglavlje sadrži kratak prikaz alata za vizuelizaciju grafova, sa posebnim osvrtom na Gephi koji je korišćen u implementaciji. U petom poglavlju izloženi su detalji integracije alata za detekciju sličnosti sa alatima za vizuelizaciju. U šestom poglavlju su izloženi detalji i smernice u vezi sa analizom dobijenih rezultata i istaknute prednosti vizualizacije rezultata u vidu grafa u odnosu na način prikaza koji se može dobiti kroz same alate. Poslednje poglavlje daje smernice za buduća istraživanja i kratak zaključak.

2. O DETEKCIJI SLIČNOSTI U PROGRAMSKOM KODU

Postoji veći broj pristupa detekciji sličnosti u izvornom programskom kodu. Algoritam za detekciju treba da ukaže da su neka dva programska koda dovoljno slična da bi bila od interesa za ručnu inspekciju kojom se može potvrditi plagijarizam. Zato je bitno identifikovati delove koda koji su slični, a ne samo utvrditi meru sličnosti.

Pored plagiranja, postoji niz drugih razloga koji mogu dovesti do sličnosti programskog koda u zadacima studenata. Studenti su najčešće programeri-početnici, a znanje usvajaju prema modelima koje im nastavnici izlažu u okviru predavanja. Takođe, što je problem koji se rešava jednostavniji, postoji manje suštinski različitih rešenja. Stoga je jasno da je problem plagijarizma nerešiv sa stopostotnom sigurnošću. Sa druge strane, skup mogućih modifikacija je takođe ograničen, pa je softverska detekcija sličnosti moguća sa visokim procentom uspeha. Zato je pre navođenja detalja o metodima za detekciju sličnosti pogodno izložiti osnovne tehnike kojima se studenti služe da prikriju akt plagijarizma.

2.1. *Metode prikrivanja plagijarizma*

Studenti se koriste najrazličitijim metodama kako bi prikrili sličnosti u prepisanom programskom kodu, a zadržali osnovnu funkcionalnost. Gotovo nikada plagirani kod nije identična kopija originalnog. Ono što ide u prilog detekciji plagijarizma jeste činjenica da se ovakvi pokušaji uglavnom svode na jednostavne i dobro poznate modifikacije. Te modifikacije se mogu podeliti na leksičke i strukturalne izmene koda, a detaljan pregled se može videti u [10, 11].

Leksičke izmene podrazumevaju jednostavne izmene programskog koda poput promene formatiranja, dodavanje, uklanjanje i izmenu komentara, dodavanje praznih linija ili drugih nevidljivih znakova. Takođe, promena imena identifikatora u kodu spada u češće korišćene leksičke izmene, kao i manipulacije deklaracijama promenljivih. Ponekad studenti menjaju i način formatiranja izlaza programa.

Strukturalne izmene zahtevaju određeno programersko znanje i imaju značajniji efekat na detekciju sličnosti. Najčešće se pribegava promeni redosleda naredbi, tako da logika samog koda ostane ista. Strukturalne izmene uključuju zamenu kontrolnih struktura ekvivalentnim, izmenu redosleda blokova u kodu i dodavanje nepotrebnog koda, kao i promenu opsega važenja promenljivih. Takođe, često se vrši izmena struktura podataka, poput klasa i interfejsa.

2.2. *Tehnike za detekciju sličnosti*

Postoji više pristupa detekciji plagijarizma u izvornom kodu [12]. Grubo se metodi za detekciju sličnosti u programskom kodu mogu podeliti na tehnike zasnovane na brojanju atributa (*attribute counting*) i tehnike zasnovane na strukturnom poređenju programskog koda, mada ima i metoda koji se zasnivaju na drugim principima, poput teorije informacija i entropije [13].

Tehnike zasnovane na brojanju atributa koriste jednostavne metrike poput broja linija koda, ključnih reči i identifikatora, veličine programa i sl. da bi formirale numerički profil zadatog koda koji se može posmatrati kao *n*-dimenzionalni vektor. Na osnovu profila se vrši međusobno poređenje kodova i određuje sličnost. Ove tehnike uglavnom eliminišu uticaj leksičkih izmena, ali nisu otporne na strukturne izmene, pa se stoga retko koriste u praksi.

Strukturno poređenje programskog koda uzima u obzir sintaksu konkretnog programskog jezika i uključuje parsiranje koda na osnovu koga se sprovodi dalje poređenje. Većina tehnika se zasniva na tehnikama uparivanja stringova (*string matching*) koji se dobijaju tokenizacijom ulaznog koda, ali se poređenje može vršiti i korišćenjem stabala parsiranja ili analizom grafa zavisnosti programa [12].

Tokenizacija je najčešće korišćen pristup. Kod njega se ulazni kod deli na tokene prilikom parsiranja i to tako da svaki njegov deo bude preveden u jednu od klasa tokena, koji mogu biti, na primer: početak metode, kraj metode, definicija promenljive, dodela vrednosti, i sl. Zatim se vrši uparivanje tokena nekom od poznatih metoda za uparivanje stringova, uz ubrzavanje postupka heširanjem sekvenci tokena i sl. Na principu tokenizacije se zasnivaju alati SIM, Plaggie, YEP, Sherlock i drugi, kao i alati JPlag i Moss koji će u ovom radu biti i detaljnije analizirani i korišćeni. Detaljniji pregled alata i tehnika za detekciju sličnosti u programskom kodu se može videti u [10, 11, 14].

3. ALATI ZA DETEKCIJU SLIČNOSTI U PROGRAMSKOM KODU

U ovom poglavlju su opisana dva popularna alata za detekciju sličnosti u programskom kodu, JPlag i Moss. Ovi alati su slobodni za korišćenje, pouzdani, otvorenog su koda, podržavaju veliki broj programskih jezika i uspešni su u praksi po oceni mnogih studija [10, 11, 14, 15].

3.1. *Moss*

Alat Moss (*Measure of Software Similarity*) je kreiran 1994. godine na univerzitetu Berkli u Kaliforniji [9]. Moss ko-

risti tehniku nazvanu *Winnowing* za selekciju i poređenje niza tokena koji predstavljaju kodove koji se porede. Moss podržava preko 20 programskih jezika, među kojima se izdvajaju: C, C++, Java, C#, 8086 asembler, i drugi. Moss je u osnovi alat za rad iz komandne linije, ali postoji nekoliko nezavisno razvijenih grafičkih okruženja za konfigurisanje i pozivanje alata. Kada je preciznost u pitanju, višegodišnji korisnici tvrde da se dva programska koda sa velikom sigurnošću mogu smatrati plagijatima ukoliko je utvrđeni procenat sličnosti bio preko 50% [16].

Winnowing tehnika [17] je osnova koju Moss koristi za detekciju sličnosti. Algoritam kreira svojevrstan digitalni potpis dokumenta (*fingerprint*) koji se koristi za poređenje sa drugim dokumentima i utvrđivanje sličnosti. Algoritam deli programski kod na *k*-gram-ove koji predstavljaju niz tokena dužine *k*. Ovi *k*-gram-ovi se heširaju, a zatim se selektuje podskup heširanih *k*-gram-ova kao digitalni potpis za upoređivanje. Sličnost dva programska koda se definiše na osnovu broja *k*-gram-ova koji se poklapaju. Konačni rezultat uključuje broj tokena i linija koji se podudaraju i procenat preklapanja programskog koda između detektovanih parova datoteka.

3.2. *JPlag*

Alat JPlag je nastao na univerzitetu u Karlsrueu u Nemačkoj 2000. godine [8]. Slično kao Moss, i ovaj alat je baziran na tokenizaciji programskog koda, a koristi Greedy-String-Tiling (GST) tehniku [18] kombinovanu sa Karp-Rabin algoritmom za uparivanje tokena i određivanje mere sličnosti između dva programska segmenta. Jezici koje podržava JPlag su: Java, C#, C, C++, Scheme, kao i običan tekst [19]. Skup tokena je preciznije definisan nego kod Moss-a, ali po analizama ovi alati daju slične rezultate [11, 14]. JPlag poseduje vrlo intuitivan grafički korisnički interfejs.

Nakon tokenizacije ulaznih datoteka, sekvenca tokena se posmatra kao običan string. Uparivanje sekvenci iz dve datoteke se vrši pomoću GST tehnike, a za uparivanje pojedinačnih tokena se koristi Karp-Rabin-ov algoritam za uparivanje stringova. Karp-Rabin-ov algoritam ubrzava uparivanje tako što vrši heširanje delova stringa, a onda vrši detaljno poređenje samo kod onih delova čiji se heševi poklapaju. GST algoritam omogućava uparivanje identičnih segmenata iz dva stringa A i B uz zadovoljavanje sledećih uslova: a) token iz stringa A odgovara tačno jednom tokenu stringa B, b) grupe tokena se uparuju nezavisno od lokacije u ulaznim datotekama i c) traže se najduži mogući podstringovi zajednički za A i B. Procenat sličnosti između A i B sada se računa kao:

*p= (2*broj linija zajedničkog dela) / (broj linija A+broj linija B)*

3.3. *Prikaz rezultata*

Alati uobičajeno određuju procenat sličnosti za svaki par upoređenih programskih kodova, ali postoje značajne varijacije u načinu prikaza rezultata i količini informacija koje su dostupne za dublju analizu. Određeni broj starijih alata poseduje čisto tekstualni izlaz, dok noviji alati poput JPlag-a i Moss-a rezultate prikazuju u vidu HTML stranice sa parovima sličnih radova. Za svaki par radova se može otvoriti posebna stranica sa uporednim prikazom kodova, a različitim bojama su obeleženi slični delovi koda, sa brojevima linija u svakom od radova. Omogućena je i jednostavna navigacija u okviru stranica.

Moss čuva rezultate u obliku HTML stranica na serverima gde se aplikacija izvršava, dok JPlag u istom formatu čuva rezultate lokalno. Na svakoj stranici su u tri kolone prikazane informacije o nazivu prvog dokumenta, nazivu drugog dokumenta i broju linija koje su pronađene u oba rada. Ove informacije postaju relativno nepregledne u slučaju velikog broja predatih radova. JPlag ima nešto unapređeni prikaz u odnosu na Moss, pa pored podataka o sličnim parovima korisniku daje raspodelu radova po procentu sličnosti, podeljenih u kategorije raspona od 10 procenata. Na kraju su sami rezultati, sortirani po sličnosti, prikazani dva puta. U prvoj polovini su rezultati raspoređeni po "prosečnoj sličnosti", a zatim po "maksimalnoj sličnosti". Naime, broj pronađenih zajedničkih linija predstavlja različit udeo kod dva rada različite veličine. Stoga je u jednoj statistici korišćen prosek, a u drugoj veći od ta dva broja. Ova podela pravi razliku kada su posmatrani radovi bitno različitih veličina, što kod studentskih radova sa istom temom, uglavnom nije slučaj, stoga je za dalju analizu korišćena "prosečna sličnost".

Međutim, nijedan od pomenutih alata ne omogućava dublju analizu veza i klasterizaciju dobijenih rezultata, već se kompletna analiza prepušta korisniku. Kako se plagijarizam kod univerzitetskih kurseva može posmatrati kao društveni fenomen, kao logičan izbor za predstavljanje rezultata detekcije sličnosti na vizuelan način izabran je graf. Prvo, graf kao apstrakcija logički odgovara problemu. Svaki student i njegov rad predstavljaju jedan čvor, dok se procenat sličnosti radova dva studenta dodeljuje grani koja ih spaja. Zatim, alati za prikazivanje grafa već postoje i dozvoljavaju modifikaciju različitih aspekata grafičkog prikaza, od veličine čvorova, debljine grana, do rasporeda čvorova na ekranu u zavisnosti od mnogih parametara. Takođe, postoji veliki broj algoritama za dublju analizu grafova, poput određivanja modularnosti, centralnosti i sl., a sve u cilju što bržeg i efektnijeg prepoznavanja sličnih, a zatim i plagiranih radova. Od analiziranih alata za detekciju sličnosti u programskom kodu, prikaz rezultata u vidu grafa je omogućavao Sherlock [1], ali se on već duže vreme ne unapređuje i nije u široj upotrebi. Od razvijenih dodataka za Moss, pomoćni alat Mossum [20] ima mogućnosti generisanja rezultata u vidu grafa. Međutim, prikaz se dobija u vidu niza statičnih slika u PNG formatu sa kojima nije moguća nikakva vrsta interakcije. JPlag nema mogućnosti za vizuelizaciju rezultata u vidu grafa.

4. ALATI ZA VIZUELIZACIJU GRAFOVA

U ovom poglavlju su opisani alati odabrani kao najpogodniji za prikazivanje rezultata detekcije sličnosti i integraciju sa alatima JPlag i Moss. Postoji veliki broj alata za analizu i vizuelizaciju grafova, a detaljniji opisi se mogu videti u

[21]. Glavni faktori prilikom izbora alata za vizuelizaciju su bili dostupnost, interaktivnost i proširivost, radi integracije sa alatima za detekciju sličnosti. Razmatrani su samo besplatni softverski paketi.

Popularni alati za analizu socijalnih i drugih mreža poput UCINET-a i Pajek-a imaju odličnu podršku za analizu grafova, ali su relativno kompleksni i akcenat je manje na grafičkom prikazu. Nekoliko alata, kao što su GraphViz, Otter i Walrus je odbačeno zbog suženih mogućnosti za manipulaciju podacima i menjanje grafičkog prikaza. U tom smislu, izabrani su NodeXL i nezavisna aplikacija Gephi. Alat NodeXL [22] je razvijen kao dodatak za Microsoft Excel, prvenstveno za vizuelizaciju i analizu društvenih mreža pomoću grafova sa kojima je omogućena odlična integracija. Ipak, zbog jednostavne proširivosti i mogućnosti za razvoj dodataka (*plugins*), izabran je alat Gephi za automatsko prikazivanje rezultata detekcije sličnosti, dok su za NodeXL razvijene skripte za uvoz podataka tokom evaluacije alata.

4.1. *Gephi*

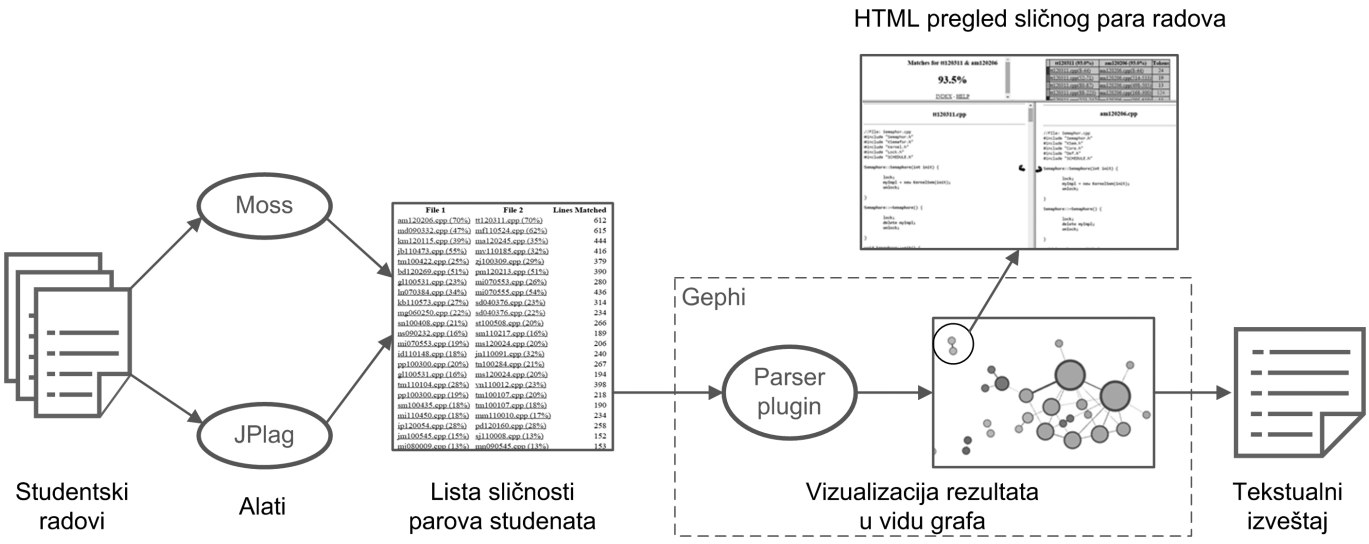
Gephi je jedan od najpoznatijih besplatnih alata za analizu i vizuelni prikaz grafova, razvijen na Univerzitetu u Kompjenju, u Francuskoj, 2009. godine [23]. Gephi nudi veliki broj opcija za vizuelizaciju grafova i manipulaciju njihovom strukturom, a podaci se mogu uvoziti iz različitih grafovskih formata. Pored osnovne manipulacije prikazom grafa, korisniku je na raspolaganju i određen broj predefinisanih načina za automatsko raspoređivanje čvorova. Pored uobičajenih, kao što su kružni, spiralni ili raspored u sinusoidu, posebno se ističe raspoređivanje čvorova na ekranu primenom Force Atlas i Force Atlas 2 algoritama, baziranih na Barnes-Hut algoritmu [24]. Kod njih se čvorovi posmatraju kao tela koja se međusobno odbijaju i privlače, a čvorovi se adekvatno prikazuju tako da prirodno reflektuju osobine grafa.

Analiza grafa se može sprovesti kroz različite algoritme, kao što su određivanje prosečnog stepena čvorova grafa, centralnosti čvorova i povezanih komponenti, vršenje klasterizacije, računanje različitih rastojanja, gustine grafa i sl. Na osnovu atributa grafa i izračunatih vrednosti se može primeniti veći broj filtera i vršiti raspoređivanje čvorova na ekranu. Jednostavno je podeliti čvorove i grane grafa u zavisnosti od odabranog parametra i menjati im veličine ili boje. Kod klasterizacije (particionisanja) elementi dobijaju boju u zavisnosti od grupe (klastera) kome pripadaju. Za problem detekcije sličnosti je izuzetno bitna klasterizacija grafa na osnovu modularnosti. Modularnost je mera kvaliteta particionisanja čvorova grafa u odgovarajuće klastere. Grafovi sa većom modularnošću su podeljeni na klastere koji su, unutar sebe, gusto povezani, a nemaju veliki broj veza sa ostatkom grafa.

Gephi je pisan u Javi, a nove mogućnosti se dodaju sa izuzetnom lakoćom, zahvaljujući opširnoj dokumentaciji dostupnoj na internetu i velikoj zajednici programera. Gotovo svaki aspekt programa može se modifikovati, i tako se mogu dodati nove metrike, filteri, načini pregleda grafa, podrška za uvoz novih grafovskih formata, kao i podrška za automatsko generisanje grafa na bilo koji način.

5. INTEGRACIJA ALATA

Integracija odabranih alata izvršena je u nekoliko faza. Kao primarni alat za prikazivanje grafova izabran je Gephi i za njega su razvijena četiri dodatka. Dva razvijena dodatka služe za parsiranje rezultata JPlag-a i Moss-a, na osnovu kojih se formira graf i vrši vizuelizacija. Treći dodatak omogućava selektovanje dva čvora i otvaranje stranice koja prikazuje sličnosti između dva rada koje oni predstavljaju, čime se korisniku omogućava ručna inspekcija. Četvrti dodatak se koristi za generisanje izveštaja. Slika 5.1 ilustruje čitav tok procesa vizuelizacije rezultata detekcije sličnosti u programskom kodu.



Slika 5.1 – *Tok procesa vizuelizacije rezultata detekcije sličnosti u programskom kodu*

5.1. *Učitavanje JPlag i Moss rezultata u Gephi*

Opcije za učitavanje rezultata rada alata JPlag i Moss su integrisane u okviru "Plugins" sekcije alata Gephi. Omogućena je selekcija direktorijuma koji sadrže HTML stranice sa rezultatima. Stranice mogu biti smeštene lokalno, ali se u slučaju alata Moss omogućava i dostavljanje odgovarajućeg linka za preuzimanje rezultata koji su smešteni na serveru. Nakon preuzimanja, stranice se smeštaju lokalno i kompletno procesiranje se slično vrši i za JPlag i za Moss uz poštovanje formata HTML stranica koje sadrže rezultate.

Sa početne HTML stranice se u oba slučaja korišćenjem regularnih izraza dohvataju sve neophodne informacije, uključujući: nazive svih čvorova (studenata), težine grana (procenti sličnosti) i linkove ka HTML stranicama koje sadrže oba predata rada, sa označenim sličnim delovima. Nakon toga, generiše se graf koji se učitava u trenutni radni prostor. Pre nego što se kontrola vrati korisniku, zarad lakšeg pregleda, primenjuju se sledeće operacije:

1. Interno rangiranje čvorova na osnovu stepena svakog čvora, radi isticanja čvorova sa velikim brojem veza. Veličina svakog čvora menja se u zavisnosti od stepena čvora odnosno broja grana koje su u vezi sa njim. Na ovaj način će biti uočljiviji oni studenti koji sa većom verovatnoćom predstavljaju izvor deljenog programskog koda.
2. Klasterizacija na osnovu osobine modularnosti. Kada se odredi klaster kome pripada svaki od čvorova, oni dobijaju različite boje. Tako postaju očiglednije potencijalne grupe studenata koje međusobno razmenjuju programski kod.
3. Labele u grafu postaju vidljive, tako da se na čvorovima prikazuje naziv studenta (njegova šifra), a grane kao labelu uzimaju težine, odnosno procenat sličnosti između radova koje povezuju. Ove labele postaju vidljive samo kada je selektovan neki čvor, tada se ističu njegove veze sa ostalim studentima. Takođe, debljina grane se povećava u zavisnosti od njene težine, što je podrazumevano za Gephi.
4. Primenjuje se Force Atlas 2 algoritam za raspored čvorova na radnoj površini, tako da postaju vizuelno grupisani studenti sa međusobnim vezama. Ovaj korak je ključan za jasno prikazivanje i lako detektovanje plagijarizma. Parametri korišćenja pomenutog algoritma odabrani su nakon testiranja različitih opcija i izbora najpreglednijeg rezultata.

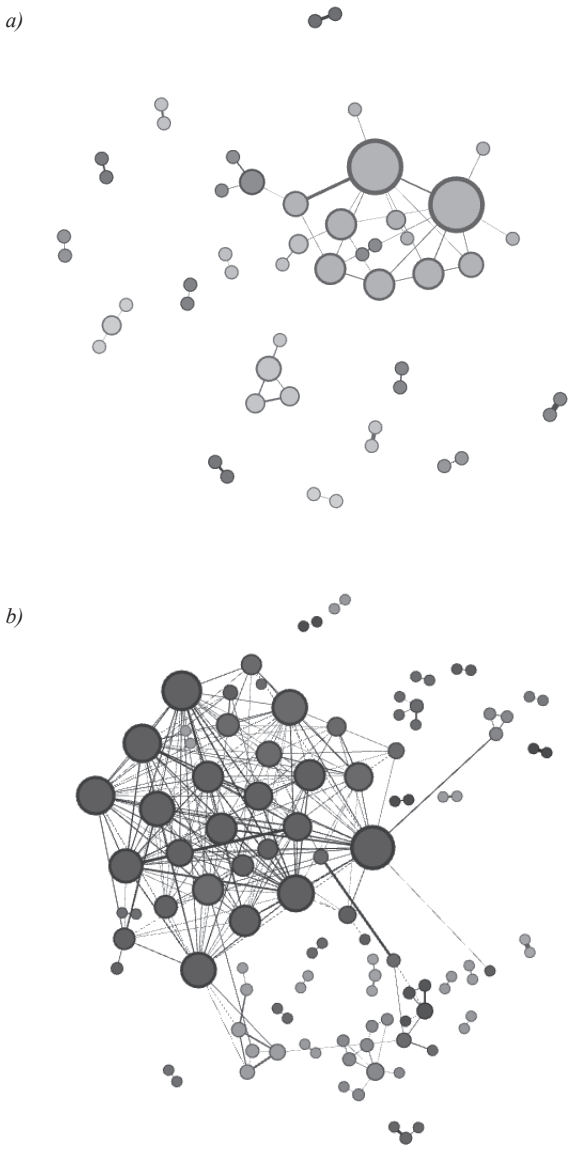
5.2. *Prikazivanje sličnosti dva rada iz sistema Gephi*

Da bi se omogućila jednostavnija ručna inspekcija sličnih kodova, omogućeno je otvaranje odgovarajućih stranica sa uporednim prikazom koje generišu JPlag i Moss iz alata Gephi. Nakon selekcije dva čvora iz generisanog grafa, korisniku je dozvoljeno da pristupi HTML stranici, koja prikazuje međusobnu sličnost radova koje oni reprezentuju.

Implementacija ove operacije je zahtevala korišćenje interfejsa za proširenje, koje nudi Gephi. Na osnovu identifikatora čvorova, traži se grana koja ih spaja i iz kolekcije njenih atributa se nalazi link ka traženoj stranici. Linkovi su prethodno prilikom parsiranja obrađeni tako da pokazuju na datoteke na lokalnom disku. Stranice se pregledaju korišćenjem podrazumevanog internet pretraživača.

6. ANALIZA GENERISANIH REZULTATA

Za testiranje integrisanih alata korišćena je kolekcija radova studenata sa kursa Operativni Sistemi 1 na Elektrotehničkom fakultetu Univerziteta u Beogradu. U kolekciji se nalazilo 175 radova napisanih na programskom jeziku C++ sa prosečno 1000 linija programskog koda.



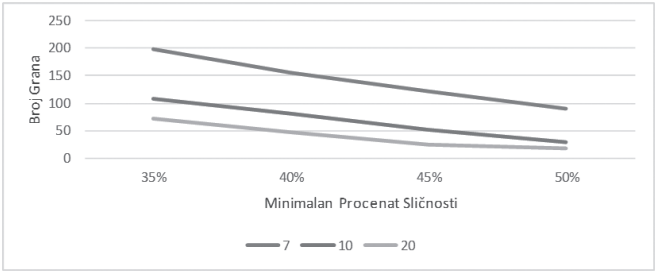
Slika 6.1 – *Prikaz generisanog grafa sa (a) 51 čvorom, (b) 107 čvorova*

Krajnji prikaz rezultata uveliko zavisi od odabranih parametara alata za detekciju sličnosti, a najveći uticaj ima broj radova relevantnih za poređenje. Pod time se podrazumevaju radovi koji sa barem jednim od preostalih imaju dovoljno veliki procenat sličnosti da postanu sumnjivi ispitivaču. Kao što je ranije pomenuto, prethodna istraživanja [11, 16] pokazuju da procenat sličnosti od preko 50% gotovo izvesno ukazuje na plagijat, mada se i niže vrednosti mogu koristiti kada se radi ručna inspekcija i odlučivanje. Empirijski je utvrđeno da ukoliko je rezultat poređenja gusto povezan graf sa preko 70 čvorova, rezultati su često nepregledni (Slika 6.1.). Slična iskustva su predstavljena u [25], gde se sugeriše da grafovi sa više od 50 čvorova brzo postaju nepregledni sa povećanjem gustine grafa.

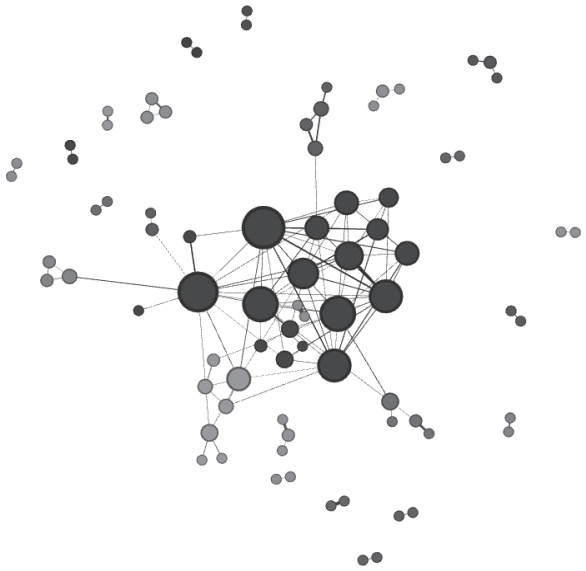
6.1. *Uticaj parametara alata za detekciju sličnosti na krajnji rezultat*

Kod JPlag alata, parametri koji mogu promeniti broj čvorova generisanog grafa su prag sličnosti i minimalna dužina niza tokena za uparivanje. Kako bi se pronašle optimalne vrednosti, posmatrane su različite vrednosti za ova dva parametra. Za prag sličnosti korišćene su vrednosti od 35% do 50%, dok je i minimalna dužina niza tokena podešavana na 20, 10 i 7. Manja vrednost ovog parametra garantuje veću preciznost. Posmatran je uticaj ovih parametara na broj čvorova i grana u generisanom grafu.

Može se zapaziti ubrzano opadanje veličine grafa sa porastom oba parametra. Kada je prag sličnosti u pitanju, takvo ponašanje je očigledno, jer se prosto odbacuju grane najmanje sličnosti i čvorovi koje oni povezuju. Što se minimalne dužine niza tokena tiče, smanjivanjem ove dužine algoritam prepoznaje mnogo više detalja, kako stvarnih plagijata, tako i slučajnih sličnosti. Stoga bi trebalo uzeti takvu vrednost da se iz razmatranja ne eliminišu delimični plagijati, ali ne premalu, jer takva vrednost produkuje veliki broj čvorova. Za dati test uzorak, empirijski je utvrđeno da je za najbolji vizuelni prikaz pogodno koristiti vrednost 10. Kako je optimalna veličina grafa oko 70 čvorova, sa Slike 6.2 se vidi da za prag sličnosti treba uzeti 40%. To garantuje da će na grafu biti prikazani samo relevantni čvorovi, sa velikim procentom sličnosti, kao na Slici 6.3. Kada je Moss u pitanju, jedini parametar koji može varirati jeste prag sličnosti.



Slika 6.2 – *Zavisnost broja čvorova od praga sličnosti i i minimalna dužina niza tokena za uparivanje*



Slika 6.3 – *Graf generisan kao rezultat alata JPlag sa pragom sličnosti 40% i minimalnom dužinom niza tokena 10*

6.2. *Smernice za podešavanje prikaza*

Način prezentovanja rezultata u vidu grafa je jako podložan promenama u zavisnosti od radova koji se porede. Na konačni rezultat utiču broj radova, nivo znanja studenata, definicija zadatka, algoritmi koji se implementiraju i sl. Rezultujući graf može biti dosta povezaniji od onih koji su do sada analizirani, a na masovnim kursevima može biti potrebe i za detaljnom analizom više od 70 čvorova. Stoga će u nastavku biti navedeni koraci koji se mogu primeniti nad generisanim grafom, kako bi se dobio što pregledniji graf.

1. Ukoliko su čvorovi previše blizu jedan drugome u rezultujućem grafu, a grane se preklapaju, može se pokušati sa ponovnom primenom Force Atlas 2 algoritma. Prilikom obrade rezultata, ovaj algoritam se poziva tačno 1,5 sekundi, što ponekad nije dovoljno, te je algoritam potrebno pozvati ponovo sa dužim vremenskim intervalom, što korisnik može da kontroliše. Ako ni takav graf ne zadovoljava, može se pokušati sa opcijom za ekspanziju grafa koja će jednostavno raširiti ceo graf tako da bude pregledniji. Rezultat ovog koraka vidi se na Slici 6.4.

Slika 6.4 – *Generisani graf (a) pre i (b) posle naknadne primene ForceAtlas 2 algoritma za raspoređivanje*

2. Ukoliko je graf nepregledan zbog velikog broja grana, mogu se koristiti filteri koji će ograničiti prikaz na samo onaj deo grafa koji je od interesa korisniku. Čvorovi i grane se mogu filtrirati po bilo kom svom atributu. Na primer, mogu se prikazati samo grane sa težinom koja pripada određenom opsegu. Takođe, filteri se u alatu Gephi mogu kombinovati. Naredni filter se primenjuje na prikaz koji je nastao primenom prethodnog. Stoga, ukoliko želimo da uz filtriranje grana po težini, dodatno eliminišemo i čvorove koji sada postaju suvišni, dovoljno je da istovremeno filtriramo po težini ivice i stepenu čvora.
3. Često će se među čvorovima izdvojiti parovi međusobno sličnih radova u grupacijama od samo nekoliko njih, dok će sa druge strane postojati veliki klaster gusto povezanih čvorova. Prilikom generisanja grafa se određuju klase modularnosti pozivanjem odgovarajućih analiza i samim tim se određuje pripadnost svakog čvora jednom od klastera. Ta informacija se može koristiti prilikom filtriranja i na taj način se može izdvojiti klaster koji se želi analizirati. Na filtrirani graf se može ponovo primeniti Force Atlas 2 algoritam kako bi se dobio optimalan prikaz. Izgled grafa pre i posle primene pomenutih filtera može se videti na Slici 6.5.

Slika 6.5 – *Uporedni prikaz generisanog grafa (a) i njegovog izgleda (b) nakon filtriranja po klasterima i težini grane*

4. Iz rezultujućeg grafa se mogu trajno eliminisati čvorovi koji nisu od naročitog interesa. To su često mali klasteri čvorova koji se mogu jednostavno analizirati. Tada je pogodno koristiti filter Partition i izabrati određeni skup čvorova sa zadatim klasama modularnosti za izdvajanje odgovarajućih čvorova. Po potrebi je moguće primeniti dodatne, već pomenute filtere.
5. Jednostavna pretraga čvorova i grana grafa, sortiranje po različitim kriterijumima i sl. se može obaviti u okviru Data Laboratory sekcije programa. Željena grana ili čvor mogu se izabrati za direktan prikaz u glavnoj sekciji za prikaz grafa.
6. Ukoliko Force Atlas 2 algoritam ne da željene rezultate prikaza, moguće je upotrebiti neki od ostalih algoritama za raspoređivanje, a izdvaja se Fruchterman Reingold, algoritam koji podrazumevano NodeXL koristi za raspoređivanje. Algoritam raspoređuje sve čvorove u koncentrične krugove, uz težnju da povezani čvorovi ostanu grupisani i sa što manjim brojem preseka između grana. Ovaj prikaz može biti koristan, uz kombinaciju sa nekim od koraka 1-4.

6.3. *Generisanje tekstualnog izveštaja*

Prilikom detekcije plagijata, često je potrebno generisanje izveštaja koji se šalju različitim instancama (disciplinske komisije i sl.) kao dokaz o počinjenom aktu plagijarizma. Stoga je u okviru alata Gephi razvijen dodatak koji omogućava generisanje jednostavnog tekstualnog izveštaja. Dodatak vrši analizu grafa i ispisuje osnovne detalje o njegovoj strukturi. Za izveštaj se definiše procenat parova sa najvećom sličnošću koji treba prikazati i željeni broj studenata sa najviše njima sličnih radova.

Generisani izveštaj sastoji se iz tri celine. U prvom delu se prikazuju klasteri studenata koji su nastali kao rezultat ispitivanja modularnosti grafa. Prikazuje se broj klastera na koje je graf podeljen. Zatim sledi ispis svake grupe zajedno sa svim studentima koji joj pripadaju. Na ovaj način može se na jednostavan način uvideti podela studenata sa visokim procentom sličnosti. Nakon toga, korisnik može videti listu parova studenata čiji radovi imaju najveći procenat sličnosti, sortiranih po procentima. Slede podaci o svim parovima studenata, čiji radovi pokazuju sličnost među sobom, ali ne sa ostalim radovima. Kako je izvor plagijarizma često jedna osoba i jedan rad, ova informacija pruža dobar uvid o parovima na koje treba obratiti pažnju. Na kraju, mogu se videti oni čvorovi koji u grafu poseduju najveći stepen. Navedeni su oni studenti čiji radovi su pokazali sličnost sa velikim brojem ostalih studenata, što može sugerisati da upravo oni predstavljaju radove koji su poslužili kao izvor plagijarizma.

7. ZAKLJUČAK

Plagijarizam kao fenomen predstavlja veliki problem u široj akademskoj zajednici, pa tako i u nastavi računarstva. Zbog kredibiliteta nastavnog procesa, nastavnici koriste različite tehnike i alate za detekciju sličnosti u programskom kodu. U praksi su se izdvojili alati JPlag i Moss koji obavljaju taj posao sa dosta visokom pouzdanošću. Međutim, zbog prirode problema, potrebna je ručna inspekcija kako bi se utvrdilo da li je neki rad plagijat ili ne. U ovom radu pokazan je jedan način omogućavanja grafičkog prikaza rezultata softverskih alata za detekciju sličnosti u programskom kodu. Graf je pritom uzet kao najprirodniji način prikaza i struktura za koju postoji veliki broj algoritama za analizu. Pritom, detektovane sličnosti između radova se mogu posmatrati kao socijalna mreža, jer je plagijarizam društveni fenomen. Za prikaz rezultata u vidu grafa iskorišćen je alat za vizuelizaciju grafova Gephi.

Alati su integrisani pisanjem odgovarajućih dodataka za obradu rezultata detekcije sličnosti u alatima za vizuelizaciju. Ovakvom integracijom alata je postignuta jednostavnost prilikom pregledanja većeg broja radova studenata. Na ovaj način od korisnika se zahteva mnogo manji napor za detekciju plagijata, jer su rezultati prikazani na prirodniji način. Dalja istraživanja bi mogli ići u nekoliko pravaca. U jednom pravcu bi se mogli razvijati dodaci koji vrše obradu rezultata drugih alata za detekciju sličnosti. Sa druge strane bi se moglo ići u realizaciju sistema koji na osnovu analize parametara rezultujućeg grafa daje sugestije o potencijalnim plagijatima. Takođe, više pažnje bi trebalo posvetiti alatu NodeXL koji se konstantno razvija i dobija sve više mogućnosti.

LITERATURA

[1] M. Joy and M. Luck, Plagiarism in programming assignments, *IEEE Transactions on Education*, **42**(2), 1999, pp. 129-133.

[2] B. Zeidman, What, Exactly, Is Software Plagiarism?, *Intelectual Property Today*, USA, Feb 2007.

[3] F. P. Yang, H. C. Jiau and K. F. Ssu, Beyond plagiarism: An active learning method to analyze causes behind code-similarity, *Computers & Education*, **70**, 2014, pp. 161-172.

[4] M. Mišić, Lj. Mitić and J. Protić, Softverska detekcija sličnosti programskog koda kao mera za otkrivanje plagijata na ispitima, *XX TREND*, Kopaonik, Serbia, 2014.

[5] G. Cosma and M. Joy, Towards a definition of source-code plagiarism, *IEEE Transactions on Education*, **51**(2), 2008, pp. 195-200.

[6] T. Kamiya, S. Kusumoto and K., CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Transactions on Software Engineering*, **28**(7), 2002, pp. 654-670.

[7] E. Luquini and N. Omar, Programming plagiarism as a social phenomenon, *IEEE Global Engineering Education Conference (EDUCON)*, Amman, Jordan, 2011, pp. 895-902.

[8] JPlag, http://jplag.ipd.kit.edu/, pristupano 08.01.2016.

[9] A. Aiken, Measure of software similarity, http://theory.stanford.edu/~aiken/moss/, pristupano 08.01.2016.

[10] Z. Đurić and D. Gašević, A source code similarity system for plagiarism detection, *The Computer Journal*, Oxford Journals, 2012.

[11] M. Mišić, Z. Šuštran and J. Protić, A Comparison of Software Tools for Plagiarism Detection in Programming Assignments**,** *International Journal of Engineering Education*, 2016, (unconditionally accepted)

[12] A. M. E. T. Ali, H. M. D. Abdulla and V. Snásel, Overview and Comparison of Plagiarism Detection Tools, *DATESO*, 2011, pp. 161-172.

[13] X. Chen, B. Francia, M. Li, B. Mckinnon and A. Seker, Shared information and program plagiarism detection, *IEEE Transactions on Information Theory*, **50**(7), 2004, pp. 1545-1551.

[14] J. Hage, P. Rademaker and N. van Vugt, Plagiarism detection for Java: a tool comparison, *Computer Science Education Research Conference (CSERC '11)*, Open Universiteit, Heerlen, The Netherlands, pp. 33-46.

[15] M. Mišić, Z. Šuštran and J. Protić, Pregled i primena sistema za otkrivanje plagijata u programskim zadacima studenata, *YUINFO 2015*, Kopaonik, 2015, pp. 473-478.

[16] K. Bowyer, L. Hall, Experience Using MOSS to Detect Cheating On Programming Assignments, *29th IEEE Frontiers in Education Conference, 1999. FIE'99,* vol. 3, Tampa, USA, 1999, pp. 13B3-18.

[17] S. Schleimer, D. S. Wilkerson and A. Aiken, Winnowing: local algorithms for document fingerprinting, *ACM SIGMOD International Conference on Management of Data*, 2003, pp. 76-85.

[18] M. J.Wise, String Similarity via Greedy String Tiling and Running Karp–Rabin Matching, Department of CS, University of Sydney, 1993

[19] L. Prechelt, G. Malpohl and M. Philippsen, Finding plagiarisms among a set of programs with JPlag, *Journal of Universal Computer Science*, **8**(11), 2002, pp. 1016.

[20] Mossum, https://github.com/hjalti/mossum, pristupano 10.02.2016.

[21] Top 30 Social Network Analysis and Visualization Tools, http://www.kdnuggets.com/2015/06/top-30-social-network-analysis-visualization-tools.html/2, pristupano 10.02.2016.

[22] D. Hansen, B. Schneidermann, and M. Smith, *Analyzing Social Media Networks with NodeXL-Insides from a Connected World*, Morgan Kaufmann, 2011.

[23] M. Bastian, S. Heymann, M. Jacomy, Gephi - An Open Source Software for Exploring and Manipulating Networks, *3rd International AAAI Conference on Weblogs and Social Media*, California, USA, May 2009.

[24] J. Barnes and P. Hut, A hierarchical O(N log N) force-calculation algorithm, *Nature*, **324**(4), 1986, pp. 446-449.

[25] M. Ghoniem, J.D. Fekete and P. Castagliola, 2005, On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis, *Information Visualization*, **4**(2), 2005, pp.114-135.

Marko Mišić, asistent, Univerzitet u Beogradu, Elektrotehnički fakultet.
Kontakt: marko.misic@etf.bg.ac.rs
Oblasti interesovanja: paralelno i distribuirano programiranje, programiranje grafičkih procesora, detekcija plagijarizma, edukacioni alati

Marko Milanović, student master studija, Univerzitet u Beogradu, Elektrotehnički fakultet.
Kontakt: milanovic.etf@gmail.com
Oblasti interesovanja: baze podataka, cloud platforme, sistemski softver

Jelica Protić, vanredni profesor, Univerzitet u Beogradu, Elektrotehnički fakultet.
Kontakt: jelica.protic@etf.bg.ac.rs
Oblasti interesovanja: inženjerska edukacija, edukacioni alati, distribuirano računarstvo, modelovanje performansi