Универзитет у Београду
Електротехнички факултет

Марко Ј. Мишић

Унапређења система за детекцију плагијаризма у изворном програмском коду

Докторска дисертација

Београд, 2017.

UNIVERSITY OF BELGRADE
SCHOOL OF ELECTRICAL ENGINEERING

Marko J. Mišić

IMPROVING SOURCE CODE PLAGIARISM DETECTION SYSTEMS

Doctoral dissertation

Belgrade, 2017.

Ментори:

др Јелица Протић, ванредни професор

Универзитет у Београду – Електротехнички факултет

Др Мило Томашевић, редовни професор

Универзитет у Београду – Електротехнички факултет

Чланови комисије:

др Јелица Протић, ванредни професор

Универзитет у Београду – Електротехнички факултет

Др Мило Томашевић, редовни професор

Универзитет у Београду – Електротехнички факултет

Др Бошко Николић, редовни професор

Универзитет у Београду – Електротехнички факултет

Др Иван Обрадовић, редовни професор у пензији

Универзитет у Београду – Рударско-геолошки факултет

Др Дејан Тошић, редовни професор

Универзитет у Београду – Електротехнички факултет

Датум одбране:

_____ године.

Маши и Тијани.

Захвалнице

Ова докторска дисертација је плод вишегодишњих напора аутора у освајању нових знања и ширењу научних хоризоната. На том путу немерљив допринос су дале његове колеге са факултета, студенти, пријатељи, породица и други сарадници. Аутор им овом приликом изражава изузетну захвалност.

Најпре, аутор би се захвалио својим менторима проф. др Јелици Протић и проф. др Милу Томашевићу за подршку у научноистраживачком раду и писању саме дисертације. Проф. Томашевић ме је увео у научноистраживачке воде и давао константну подршку за бављење паралелизацијом и мултипроцесорским системима. Од професора сам научио како да на темељан и систематичан начин пишем научне радове, али и како да ценим праве породичне и животне вредности. Проф. Протић ме је увела у свет истраживања у едукацији која су на крају довела до теме ове докторске дисертације. Њена изузетна енергија, константна подршка и посвећеност који су се мерили сатима заједничког рада и вера у успех су ме мотивисали да на овом путу издржим до краја. На прегледу и оцени ове дисертације сам захвалан проф. др Бошку Николићу, проф. др Ивану Обрадовићу и проф. др Дејану Тошићу.

Успеху овог истраживања су умногоме допринели колеге и сарадници. Колеге Живојин Шуштран, Дражен Драшковић и др Жарко Станисављевић су били изузетна подршка приликом писања заједничких научних радова који су изашли у часописима. Аутор дугује захвалност и некадашњем колеги Андрији Бошњаковићу на вредним саветима на почетку каријере, проф. др Вељку Милутиновићу који га је у раним фазама каријере

инспирисао и подстицао на научноистраживачки рад кроз многобројна заједничка путовања и учешће на конференцијама, као и проф. др Јовану Ђорђевићу који је путем заједничког пројекта подржавао такве одласке. Велику техничку подршку је аутору пружио и администратор Драган Миладиновић из лабораторије Катедре за рачунарску технику и информатику. Аутор би се захвалио и свом пријатељу, психологу др Витомиру Јовановићу, на помоћи у спровођењу истраживања ставова и пракси студената на тему плагијаризма у програмском коду. На крају велико хвала и свим другим наставницима и сарадницима Катедре са којима ми свакодневни заједнички рад пружа велико задовољство.

Свој допринос овој докторској дисертацији су кроз своје дипломске и мастер радове дали и ауторови студенти Владислав Губеринић, Душан Николов, Марко Живковић, Марко Милановић, Ања Јовић и Александар Дацић. Аутор им се овим путем захваљује на сарадњи.

Овај дуготрајни пут не би могао бити приведен крају без подршке пријатеља. Мирко, Витомир, Миољуб, Јелена, Васа, Марко и Милош су били део многобројних разговора на животне теме и подсећали ме на оне лепе стране живота. Исто тако дугујем захвалност и пријатељима које сам стекао у оквиру студентске организације EESTEC.

На крају, аутор изражава неизмерну захвалност својој породици, родитељима Лидији и Југославу и брату Милошу који су били велика подршка и ослонац током целокупног живота и школовања. Посебну захвалност дугујем кћерки Маши и супрузи Тијани које сваки мој дан чине радосним и испуњеним љубављу. Машино рођење 2015. године је донело један посебан подстицај целокупном мом животу и помогло да завршим истраживања и писање ове докторске дисертације на време. Надам се да ћу у будућности бити у прилици да надокнадим време које нисам провео са њима у току последњих неколико година.

У Београду, 2017. године Марко Мишић

Резиме

Наслов: Унапређења система за детекцију плагијаризма у изворном програмском коду

Образовање у области рачунарства укључује практичан рад кроз програмске задатке који су честа мета плагијаризма. У овом раду су дискутовани различити аспекти плагијаризма у програмском коду у академском окружењу, извршена је упоредна анализа софтверских система за детекцију сличности и предложена њихова унапређења. Изабрани системи су евалуирани коришћењем три различита програма над којима је коришћено више од 20 типова лексичких и структуралних измена које су примењиване на код током 1, 2, 4, и 8 сати рада. Примењено је и реално оптерећење које је укључивало задатке обима од 50 до 1000 линија програмског кода са три различита предмета које је похађало од 100 до 300 студената. Резултати су показали да 5-10% студената, сходно метрици и критеријумима ове тезе, плагира своја решења.

Анализирана су три правца за унапређење ових система: паралелизација, визуелизација резултата у виду графа и анализа резултата методама за анализу социјалних мрежа. Паралелизација над секвенцијалном верзијом базираном на алгоритму RKR-GST спроведена је на централном процесору коришћењем *Pthreads* нити и на графичком процесору коришћењем CUDA технологије. Евалуација коришћењем два скупа тест примера показује убрзања до 6 пута на централном процесору и до 4 пута на графичком процесору.

У детекцији плагијаризма се могу користити метрике и методе за анализу социјалних мрежа: централност по степену, релациона и бета централност, степен кластеризације, анализа клика и сл. Помоћу њих су карактерисане четири најчешће шеме колаборације, а потврда осмишљеног приступа је верификована на два реална скупа примера кроз које је показан и значај правилног одређивања прага сличности. Ради визуелног приказа мреже, извршена је интеграција са алатом *Gephi* и разматрани различити начини за распоређивање чворова.

Кључне речи: анализа социјалних мрежа, CUDA, детекција сличности, паралелизација, плагијаризам, програмски код, софтверски системи, визуелизација

Научна област: Електротехника и рачунарство

Ужа научна област: Рачунарска техника и информатика

УДК број: 621.3:004

ABSTRACT

Title: Improving source code plagiarism detection systems

Computing education involves practical training through programming assignments which are frequent targets for plagiarism. In this thesis, different aspects of source code plagiarism in academic environment are discussed. Comparative analysis of source code similarity detection systems was performed and several improvements were proposed. Selected systems were evaluated using simulated plagiarism based on three programming assignments produced after 1, 2, 4, and 8 hours of work on baseline version using more than 20 types of lexical and structural modifications. Real-life student codes from three different courses were also used for evaluation. The courses were attended by 100 to 300 students, and the solutions varied from 50 to 1000 lines of code. The results show that 5-10% of students plagiarized their solutions, according to the criteria used in this thesis.

Three improvements of such systems were proposed: paralellization, similarity network visualisation, and analysis of results using social network analysis. Sequentual version based on RKR-GST algorithm was parallelized on the CPU using Pthreads, while GPU implementation is based on CUDA. Observed speedups over the sequential version of the code were between 4 on the GPU and 6 on the CPU.

Social network analysis measures and methods, such as degree, betweeness, and beta centrality, clustering coefficient, clique analysis were used for plagiarism detection in this study. The most frequent four collaboration patterns were characterized using aforementioned measures. The proposed approach was verified using two real-life test sets. The importance of proper choice of similarity threshold was also discussed. Visualization of the similarity network was performed using Gephi and different node layouts were considered.

Keywords: CUDA, parallelization, plagiarism, similarity detection, social network analysis, software systems, source code, visualization

Scientific field: Electrical Engineering and Computing

Scientific subfield: Computer Engineering and Informatics

UDC number: 621.3:004

КРАТАК САДРЖАЈ

САДРЖАЈ

1. Увод

Појава плагијаризма, односно представљања туђег ауторског дела као сопственог, представља велики проблем данашњег друштва. Проблем плагијаризма се често јавља и код писања софтвера, како у софтверској индустрији, тако и у образовном процесу на универзитетима. Програмски код је у суштини текст, па се могу повући одређене паралеле са детекцијом плагијаризма у текстовима писаним на говорном језику. Међутим, програмски језици имају знатно формалнију структуру и стриктнија правила писања, што донекле сужава простор онима који желе да почине акт плагијаризма. Говорни језици остављају много више простора за варијације од програмских језика, па је стога детекција сличности два програмска кода знатно другачија од детекције сличности два текста написана на говорном језику.

Плагијаризам у програмском коду се може посматрати као свака намерна употреба програмског кода који је написао неко други, а која није адекватно цитирана од стране онога ко предаје програмски код као своје дело. У смислу детекције ове појаве, битно је експлицитно разликовати термине сличност и плагијат. Детектована сличност између два програмска кода не мора нужно бити резултат плагијаризма. Поред копирања туђег програмског кода, постоји низ фактора који могу да утичу на појаву сличности: употреба уобичајених алгоритама, конвенције именовања идентификатора, делови кода настали коришћењем алата за аутоматско генерисање кода и сл. Стога финалну одлуку да ли је неки део кода плагијат мора донети човек, а софтверски алати развијени у ту сврху представљају значајно помоћно средство, које може указати на потенцијалне случајеве плагијаризма.

Савремени универзитетски курсеви из области рачунарства захтевају знатну количину практичног рада који се остварује кроз лабораторијске вежбе, домаће задатке и пројекте. Студенти ове активности најчешће реализују самостално у одговарајућем програмском језику. У дигиталној ери, веома је једноставно преписати делове кода или комплетан код написан од стране другог лица. Један број несавесних студената на овај начин покушава да испуни своје предметне обавезе и предаје наставницима плагиране радове, најчешће уз минималне измене. Модификације које при томе чине са намером да прикрију плагијаризам варирају од једноставних промена текста програма до сложених структурних трансформација. Са повећањем броја студената уводе се системи за управљање учењем (енг. *Learning Management Systems*) као што је *Moodle*, а све чешће се користе и системи за аутоматизовано састављање тестова, као и за предају домаћих задатака путем веб сервиса. У таквим околностима комбинованог (енг. *blended*) приступа учењу, програмерски домаћи задаци и пројекти добијају на значају као јединствена прилика да студенти искажу своју креативност и вештине. У том смислу, провера њихове оригиналности постаје врло важна.

Појава плагијаризма је довела до развоја алата за аутоматизовану детекцију сличности, са циљем да се упореди већи број студентских решења (програма) и олакша уочавање међусобних преклапања од стране наставног особља. У том смислу, алати за детекцију сличности баратају са већим бројем краћих изворних кодова који представљају студентска решења и се овај рад првенствено бави том врстом плагијаризма. Са друге стране, у софтверској индустрији повремено постоји потреба за утврђивањем сличности мањег броја софтверских решења. Тада се говори о детекцији софтверских клонова са превасходним циљем да се утврди нарушавање патентних права и неовлашћена употреба туђе интелектуалне својине.

Као што је раније речено, финалну анализу резултата поменутих алата мора обавити човек. Алати за детекцију сличности најчешће поседују текстуални приказ резултата у облику листе парова студентских радова рангираних по проценту сличности. Међутим, у реалним ситуацијама се може детектовати међусобна сличност између више различитих радова, јер је плагијаризам друштвени феномен који често обухвата више учесника. Притом, сами алати не дају више могућности за дубљу анализу и кластеризацију резултата. Стога је природно резултате поређења приказати у облику графа. Постоји велики број алгоритама за анализу графова, а развијен је и одређени број алата за визуелизацију графова.

Са друге стране, у академском окружењу је потребно поредити велики број предатих радова студената. У зависности од обима појединачних радова и броја парова за поређење који расте квадратно, процес поређења може да траје одређено време. У данашње време, рачунарски хардвер је знатно узнапредовао, а за израчунавања су доступни модерни хетерогени системи са вишејезгарним централним процесорима и многојезгарним графичким процесорима. С обзиром да је проблем детекције сличности у програмском коду погодан за паралелизацију и на нивоу алгоритама и на нивоу организације послова, могућности за искоришћавање централних и графичких процесора ради убрзавања поступка детекције ће бити испитани у овом раду.

Циљ овог рада је да се на основу претходних анализа предложе и испитају нови приступи за унапређења система за детекцију сличности у програмском коду и анализу резултата. Фокус рада је усмерен на побољшање укупног корисничког искуства, с обзиром да јавно доступни системи на рудиментарном нивоу приказују сличности парова задатака, при чему не постоји подршка за дубљу анализу и визуелизацију резултата. Циљ саме анализе резултата је да се квантитативним методама утврде шаблони за препознавање потенцијалних колаборација различитих група студената, по угледу на сличне методе за анализу колаборативних и коауторских мрежа. У погледу побољшања перформанси система, разматране су различите могућности паралелизације, како на нивоу самих алгоритама за детекцију сличности, тако и на нивоу организације послова унутар система. Ова унапређења су имплементирана у оквиру експерименталног система који је коришћен за евалуацију добијених резултата. Резултати овог рада треба да допринесу бољем разумевању проблема плагијаризма у програмском коду у академском окружењу, а наставницима омогуће једноставнију и флексибилнију анализу резултата које дају софтверски системи за детекцију сличности.

Истраживање које је спроведено у оквиру ове докторске дисертације има превасходно значај у едукацији, али се одређени елементи могу применити и у индустрији. Појава плагијаризма представља изузетно актуелан проблем, који значајно утиче на кредибилитет наставног процеса, па је у том смислу неопходно деловати превентивно и предупредити акте недозвољене колаборације. Уочавање и кажњавање плагијаризма је од великог значаја, јер се тиме успостављају и етички стандарди које студент треба касније да поштује и у свом професионалном раду. Истраживање у оквиру ове тезе у том смислу доприноси превазилажењу ограничења постојећих јавно доступних алата у сврху добијања директно

применљивих система за детекцију плагијаризма. Поред тога, одређени резултати овог рада могу имати ширу примену и ван академског окружења, нарочито у области заштите ауторских и патентних права.

За евалуацију предложених унапређења је коришћена значајна база домаћих и пројектних задатака студената који садрже програмски код са више различитих предмета на Електротехничком факултету Универзитета у Београду. Радови су прикупљани током седам година рада аутора на Катедри за рачунарску технику и информатику Електротехничког факултета Универзитета у Београду.

Рад је подељен у неколико поглавља. У другом поглављу је разматран проблем плагијаризма у програмском коду. Посебна пажња је посвећена дефиницији плагијаризма, узроцима његовог настанка, као и методама којима се студенти користе за његово прикривање. Као посебна тема, обрађени су и ставови студената и наставника на ову тему и изложени резултати истраживања ставова и пракси студената на тему плагијаризма у програмском коду које је спроведено у оквиру овог истраживања. Треће поглавље се бави алгоритмима за детекцију сличности у програмском коду. Описане су основне разлике између примена у индустрији и академском окружењу, а затим детаљно разматране технике поређења и коришћене метрике. Преглед и класификација система за детекцију сличности у програмском коду су дати у четвртом поглављу. Релевантни системи су поређени и класификовани на основу дефинисаних карактеристика. У оквиру поглавља су изложени резултати експерименталне евалуације одабраних система. Поглавље се завршава дискусијом о недостацима постојећих решења.

У петом поглављу је дато више детаља о паралелизацији алгоритама за детекцију сличности. Описане су референтна секвенцијална и реализована паралелна имплементација и изложени резултати експерименталне евалуације. Шесто поглавље приказује примену метода за анализу социјалних мрежа у анализи резултата алата за детекцију сличности. Визуелизација резултата са упутствима за параметризацију је описана у седмом поглављу.

Закључак се налази у осмој глави и он сумира главне резултате рада и праваце за даља истраживања. Списак коришћене литературе се налази након закључка. У оквиру прилога је дата посебна глава која детаљније обрађује истраживање о ставовима и праксама студената на тему плагијаризма у програмском коду.

2. О ПРОБЛЕМУ ПЛАГИЈАРИЗМА У ПРОГРАМСКОМ КОДУ

Савладавање релевантних вештина програмирања је један од најважнијих исхода у образовању инжењера рачунарства и софтверских инжењера. Развијање добрих програмерских вештина је од изузетног значаја за студенте, јер на тај начин усвајају начин размишљања који је окренут ка решавању проблема помоћу рачунара (енг. *computational thinking*) и сусрећу се са решавањем практичних проблема помоћу софтвера. Стицање ових вештина је дугорочан процес који се најбоље остварује кроз самосталан, практичан рад у оквиру различитих академских курсева. Важност ових вештина је препозната и од стране стручних организација, попут IEEE и ACM и, у складу са тим, оне су инкорпориране у препоруке за наставне планове и програме из рачунарства и сродних области [1].

Практичан рад се најчешће спроводи кроз лабораторијске вежбе, пројектне радове и домаће задатке који носе одређен број бодова за коначну оцену студента. Ове активности се разликују по комплексности и обиму програмског кода који варира од неколико десетина до неколико хиљада линија. Велика већина студената се веома брзо навикне на овакав начин учења и испитивања. Међутим, одређени, мањи број студената се не придржава правила академског понашања и плагијаризују туђа решења. Плагијаризована решења долазе из различитих извора: колега са факултета, доступне литературе, Интернета, а постоје и ситуације у којима старији студенти или професионални програмери решавају млађим колегама задатке са или без неког облика накнаде [2]. Ови задаци се често бране у рачунарским лабораторијама, пред асистентима и демонстраторима, са циљем да се утврди да ли су студенти самостално решавали задатке и да ли су у стању да изврше задате

модификације кода. На основу испитивања и инспекције кода, догађа се да испитивачи уоче велику сличност у задацима различитих студената. Овакви поступци откривања плагијаризма су могући на мањим курсевима, али у условима масовних рачунарских курсева, као што су курсеви из програмирања [3], овакав приступ постаје практично немогућ.

Плагијаризам је у многим докуменатима означен као непоштена пракса, како од стране струковних организација као што је ACM [4], тако и од стране већег броја страних универзитета кроз кодове части [5] и документе о академском интегритету [6]. Међутим, то је и озбиљна претња по регуларност процеса испитивања, јер пројектни и домаћи задаци у значајној мери утичу на коначну оцену студента. Стога је потребан систематичнији, софтверски подржан приступ откривању сличних програмских задатака који би олакшао процес идентификације програмских плагијата.

2.1. Дефиниција плагијаризма

Да би се успешно извршила идентификација плагијата, потребно је најпре утврдити јасну дефиницију плагијаризма. Као што је напоменуто у уводу, сличност два текста или програмска кода не мора нужно представљати плагијат. Постоје многи разлози који могу довести до сличности, па се стога сам појам плагијаризма мора пажљиво дефинисати.

Реч плагијаризам потиче од латинске речи plagiarius, што у преводу значи отет, киднапован [7]. Сматра се да је реч први пут употребио римски песник Марко Валерије Марцијал да би описао ситуацију у којој је други песник употребио његове стихове. Реч је у енглески језик ушла почетком седамнаестог века да би означила врсту литерарне крађе. У Меријам-Вебстер речнику енглеског језика [8], глагол плагирати (енг. *to plagiarize*) има следећа значења:

- *to steal and pass off (the ideas or words of another) as one's own* – украсти или користити идеје или речи другог човека као сопствене идеје и речи
- *to use (another's production) without crediting the source* – користити туђе дело без навођења извора
- *to commit literary theft* – починити литерарну крађу
- *to present as new and original an idea or product derived from an existing source* – приказати као нову и оригиналну идеју или производ који су изведени на основу постојећег извора

У сваком случају, плагијаризам се може сматрати актом преваре. Ова појава укључује крађу туђег дела и лажно представљање након тога [9]. Исти извор класификује неколико врста понашања у литерарним делима који се могу сматрати плагијаризмом:

- представљање и предавање туђег рада као сопственог
- копирање речи или идеја из туђег дела без навођења извора
- изостанак интерпункцијских знакова навода приликом дословног преношења делова текста (цитата)
- давање некоректних информација о извору цитата
- копирање већег дела речи или идеја из неког извора, тако да он постане већи део нечијег ауторског дела, без обзира да ли је извор наведен или не

У контексту академског окружења, плагијаризам представља крађу и експлоатисање туђих идеја, језика и креативности без јасног навођења извора. Ова појава представља форму варања и генерално се може сматрати морално и етички неприхватљивом. Иако није кажњива законом, ова пракса подлеже дисциплинској одговорности на многим универзитетима и обично се регулише интерним актима, као што су правилник о дисциплинској одговорности студената на Универзитету у Београду [10], код части са Принстон универзитета [5] или документ о академском интегритету са MIT-а [6].

Појава плагијаризма је присутна и у новинарству и уметности. Новинарство је у основи засновано на поверењу јавности, па је очекивано да новинар адекватно наведе своје изворе. У уметности је ова појава присутна јако дуго, па се чак може сматрати њеним саставним делом, јер су праксе имитације, подражавање стила, ревизије, прилагођавања, препричавања, колажа и сл. доводиле до нових уметничких праваца и тенденција [11].

Изражавање мисли и идеја подлеже заштити интелектуалне својине по законским прописима у већини земаља, док год постоји нека врста њиховог записа (аудио, видео, текстуални запис и сл.). Међутим, додатна ауторска или уметничка интервенција може променити природу самог дела, па је самим тим тешко одредити да ли је нешто једноставно плагијаризам или нови ауторски израз са новом вредношћу.

Такође, у пракси се често јављају случајеви аутоплагијаризма, где исти аутор користи значајне делове претходно објављеног рада. Ова појава је сама по себи контроверзна, јер плагијаризам подразумева коришћење туђег ауторског дела. Проблеми се могу појавити

уколико су ауторска права пренета на неко друго лице у процесу публиковања. Међутим, од аутора се свакако очекује да своје претходно дело адекватно цитира и да се ново дело у одговарајућој мери разликује од претходног како би се избегло вишеструко публиковање истог материјала.

Поновно коришћење сопственог материјала се у многим контекстима може сматрати оправданим, а [12] наводи четири битна разлога:

- Резултати претходног рада морају поново да се изложе да би се поставиле основе за нови допринос
- Делови претходног рада морају да се понове како би се извршила дискусија са новим доказима или аргументима
- Публиковање истог рада је потребно на више различитих места како би порука допрла до различите публике, нарочито у интердисциплинарним истраживањима
- Аутор сматра да је неку тему први пут обрадио тако добро да нема смисла препричавати је на други начин

Стога се аутоплагијаризам више сматра врстом непоштења, него крађе и у том смислу много блаже посматра од стране академске заједнице. Такође, ограничено коришћење и репродуковање ауторског материјала је и законски и етички дозвољено, па је знатно теже проценити да ли је неко починио аутоплагијаризам или не. У контексту програмског кода, чак су и пожељне праксе поновног коришћења програмског кода, нарочито код објектно-оријентисаног програмирања.

Из свега до сада је јасно да дефиниција плагијаризма мора да укључи више битних аспеката. У контексту овог рада, плагијаризам је дефинисан по узору на истраживање изложено у [13]. У оквиру тог истраживања је спроведено испитивање на тему плагијаризма прогамског кода у академској средини на универзитетима у Великој Британији. Кроз скуп питања, аутори су покушали да дођу до опште прихваћене дефиниције шта све обухвата плагијаризам, а које праксе се могу сматрати дозвољеним.

Плагијаризам у програмском коду представља свака намерна или ненамерна употреба програмског кода који је написао неко други, а која није адекватно цитирана од стране онога ко предаје програмски код као свој рад. Ова дефиниција укључује и праксе прибављања програмског кода са или без дозволе оригиналног аутора, као и поновног коришћења

програмског кода предатог у оквиру неког другог програмског задатка који је већ бодован и оцењен. С обзиром да постоје различита гледишта на праксе прибављања, цитирања и поновне употребе кода, аутори [13] су детаљније побројали ситуације које потпадају под плагијаризам и оне ће бити наведене у наредном тексту.

Туђ програмски код студент може да набави или добије на више начина. У данашње време, студенти често размењују програмски код путем електронске поште, дискусионих група и форума. Веома често студенти решавају задатак групно, иако је предвиђен за самосталан рад, што резултује сличним решењима. Поједини студенти из добре намере прослеђују решења својим колегама, како би им она користила као узор. Међутим, таква решења често бивају злоупотребљена и предата у скоро истоветном облику. Студент може код набавити и узимањем без дозволе (крађом). Одређени студенти користе услуге трећих лица да им са надокнадом или без ње реше задатак.

Студенти често пропусте да адекватно цитирају извор и аутора програмског кода који користе, било коришћењем коментара у самом коду, било у пратећој документацији. Они понекад користе непостојеће референце или референце које не одговарају стварно коришћеном програмском коду. Често студенти нису довољно информисани о томе на који начин треба да цитирају изворе које користе.

Поновна употреба програмског кода је пракса око које постоји највише неслагања. У смислу наведене дефиниције, свако репродуковање програмског кода без измена или са минималним или умереним изменама представља плагијаризам. Међутим, значајне измене програмског кода, коришћење одређеног кода као основе за ново решење или конверзија одређеног програмског кода из једног у други програмски језик се не сматрају плагијаризмом од једног броја наставника. Такође, генерисање кода помоћу одговарајућих софтверских алата се у одређеним случајевима толерише. Међутим, иако се пракса поновне употребе кода охрабрује у објектно-оријентисаним окружењима, постоји велика разлика између плагијаризма и поновне употребе програмског кода. Плагијаризам је процес креирања сличног софтвера од постојећег из разлога лењости или немогућности да се уради задатак, а поновна употреба кода је процес креирања новог софтвера на основу постојећег решења које је пажљиво изабрано од више кандидата, строго водећи рачуна о његовом квалитету [14, 15].

2.2. Узроци који доводе до плагијаризма

Постоје различити разлози због којих се студенти одлучују да плагирају туђ рад. Они укључују кратке временске рокове, неинтересантне курсеве [16, 17], лоше предзнање студената, жељу за бољом оценом [18] и генерално бољим постигнућем на курсу или студијама. Недовољна доступност материјала за израду задатака и недовољно јасна и прецизна спецификација задатка често умањују мотивацију за његово решавање и појачавају проблем. Често студенти сматрају да су преоптерећени великим бројем домаћих задатка и пројеката које треба да реше у кратком року на више курсева, док за саме задатке мисле да су преобимни и претешки. У том смислу, незаинтересован став наставника за сам курс и материју коју предаје, благ став наставника према случајевима плагијаризма, али и некоординисаност наставника са различитих курсева могу да допринесу ширењу ове појаве. Такође, непостојање јасних правила о одговорности студената, као и непознавање тих правила доприноси проблему.

Студенти користе најразличитије стратегије да се изборе са тешким и обимним задацима [19]. Притом, они се у таквим ситуацијама сусрећу и са одређеним етичким и моралним дилемама. Аутори су проблем сагледали из три различита угла: стратегија које студенти користе да превазиђу проблем, начина на који студенти реагују када неко од њих затражи помоћ и страна које подржавају у могућој конфликтној ситуацији.

Идентификовано је седам стратегија које студенти користе да реше тешке задатке: питају наставника за помоћ, решавају проблем у сарадњи са колегом, постављају питања на интернету (форумима, дискусионим групама, друштвеним мрежама), траже решења сличних задатка на интернету, копирају и модификују задатак од колеге, деле посао међусобно и траже одлагања рока под различитим изговорима који су често медицинске природе. Интернет извори су изузетно доступни у данашње време, јер се на сајтовима попут *Stack Overflow* [20] и *GeeksforGeeks* [21] могу наћи готова решења различитих проблема. Већина ових стратегија је у колизији са оним што наставници сматрају етичким праксама при решавању задатка [13].

Са друге стране, студенти на различите начине одговорају на захтеве за помоћ који стижу од њихових колега. Одређени студенти једноставно игноришу такве захтеве. Међутим, већи је број оних који дају савете за решавање задатка, допуштају колеги да погледа њихово решење или копира њихово сопствено решење да би га користили као узор, што лако доводи

до појаве плагијаризма. У високо компетитивним академским срединама, где од успеха студирања зависе одређене бенефиције које студенти добијају, неки студенти пријављују колеге наставницима уколико им се колеге обрате за помоћ, што одређени наставници сматрају позитивним. Други студенти дају погрешне савете како би побољшали своје шансе да добију добру оцену, што се може сматрати неморалном и неетичном праксом.

Истраживање је показало да наставници нису високо на лествици приоритета студената када траже помоћ, што оставља могућност за појаву плагијаризма уколико студенти нису довољно информисани шта је дозвољено понашање, а шта не. Студенти чешће траже помоћ пријатеља или претражују доступне базе знања на интернету, него што траже помоћ од наставника. У односу са својим колегама, студенти су чешће спремни да дискутују начин решавања проблема, него да дозволе копирање сопственог задатка. Уколико дође до конфликта, студенти су лојалнији својим колегама, него наставницима и универзитетским правилима.

Недовољна информисаност студената на тему плагијаризма се показује као један од значајних фактора за његов настанак и спречавање [13, 16]. Студенти су генерално лоше информисани о правилима академске честитости, а многи нису ни упознати са постојањем било каквих правилника који регулишу овакво понашање на универзитету. У том смислу, плагијаризам у програмском коду се не разликује много од плагијаризма у текстуалним документима, мада постоје разлике у погледима наставног кадра, као што је објашњено у Поглављу 2.1.

2.3. Специфичности плагијаризма у програмском коду

У претходним поглављима је дефинисан појам плагијаризма у програмском коду и узроци који доводе до његовог настанка. Ипак, с обзиром на структурираност и већу формализацију програмских језика у односу на говорне језике, постоје одређене специфичности у вези са овом појавом које се односе искључиво на програмски код. Те специфичности се, пре свега, односе на начине које студенти користе да би прикрили свесно почињени акт плагијаризма. За разлику од говорних језика, где постоји много шири дијапазон промена и где се могу учинити значајне реформулације реченица уз задржавање смисла, програмски језици су много мање флексибилни. У овом поглављу ће бити дат преглед уобичајених измена које студенти врше, али и још неких специфичних разлога који могу да доведу до сличности у програмском коду, а не потпадају нужно под плагијаризам.

2.3.1. Могући узроци сличности

Већи број аутора је у својим радовима сугерисао да сличност два програмска кода није нужно резултат плагијаризма. Сличност може настати као последица сличног нивоа знања студената, коришћења пројектних образаца, као и праксе поновне употребе кода која је инхерентна објектно-оријентисаном програмирању. У раду [14] је наведено 12 потенцијалних разлога сличности програмског кода у задацима студената, подељних у три групе:

1) Разлози у вези са социјалним догађајима – размена кода, дискутовање фрагмената кода и резултата извршавања, дискусија поставке проблема, дискусија о додатним функционалностима
2) Разлози у вези са усвојеним програмерским вештинама – начин пројектовања и архитектура решења, употребљени алгоритми, логички ток програма, стил кодирања, поновна употреба кода, коришћење пројектних узорака
3) Разлози у вези са спецификацијом задатка – дизајн корисничког интерфејса, захтеви и ограничења постављени у задатку, додатне функционалности

Аутори [14] сматрају да је већина ових разлога позитивна и да се само размена кода и у некој мери поновна употреба кода могу сматрати плагијаризмом. Не треба заборавити да су студенти најчешће програмери-почетници, а знање често усвајају према моделима које им наставници излажу у оквиру својих предавања, што утиче на појаву сличности у задацима. Такође, понекад се од студената тражи имплементација једноставних концепата код којих не постоји пуно могућих решења, што такође резултује сличношћу. Најзад, студенти рачунарства се охрабрују да пишу квалитетна решења која могу да се поново употребе, тако да је значајна улога наставника да истакне разлике поновне употребе квалитетног решења, од пуког репродуковања које се може сматрати плагијаризмом.

Почетници често чине и сличне логичке грешке, па продукују код са сличним недостацима, као што је идентификовано у [22]. Аутор овог рада је и сам запазио један такав пример у својој наставној пракси. Приликом имплементације алгоритма за уклањање свих елемената из низа који задовољавају неки критеријум, студенти често користе алгоритам који врши померања у низу након уклањања сваког појединачног елемента, као када уклањају само један елемент. Такав алгоритам онда има квадратну временску сложеност. Иако се студентима предаје једнопролазни алгоритам који уклањање врши у линеарној

временској сложености, одређени број почетника посеже за очигледним, а мање квалитетним решењем, што онда доводи до сличних кодова.

2.3.2. Импликације на усвајање програмерских вештина

Из перспеткиве жељених исхода учења и стечених програмерских вештина, плагијаризам је свакако непожељна појава, јер студенти не стичу неопходне вештине. У неколико студија као што је [23], праћено је понашање и постигнуће студената на програмским задацима у ситуацијама када имају доступан костур решења и када га немају. Аутори су кроз наменски додатак за развојно окружење Eclipse пратили рад студената у лабораторији кроз параметре као што су број превођења, дебаговања и покретања кода, време између два таква догађаја и сл. Притом су студенте поделили у две групе за лабораторијске вежбе, како би спровели експеримент. Првој групи је дат на располагање једноставан костур решења задатка који су могли да користе у даљем развоју („плагијаризована група“). Друга група студената је морала да развија решење од почетка.

На основу сакупљених података, аутори студије су закључили да студенти из групе која је имала доступно решење постижу боље резултате и добијају више бодова, али притом много мање времена посвећују развоју програма. Плагијаризована група је трошила време углавном на модификацију постојећег костура решења да би решила задатак. Са друге стране, они су трошили много мање времена на развој, превођење и дебаговање кода који чине језгро програмерских вештина и које се уче кроз практичан рад.

2.3.3. Методе за прикривање плагијаризма

Студенти се користе најразличитијим методама како би прикрили сличности у преписаном програмском коду, а измене врше тако да задрже потпуну функционалност предатог решења. Те методе се грубо могу поделити на лексичке и структуралне измене [18, 24]. Поједине измене су општег типа и независне су од коришћеног програмског језика, док су друге директно зависне, како од коришћеног језика, тако и од парадигме програмирања. Измене које се могу начинити у објектно-оријентисаним језицима често нису применљиве у процедуралним језицима.

Лексичке измене су измене које не захтевају скоро никакво програмерско знање. Генерално, ове измене су једноставне и у већини случајева овај тип модификација се лако детектује и игнорише [24]. Оне укључују промену имена идентификатора као што су променљиве, методе, константе, типови, класе, затим додавање, уклањање и измену

коментара, празних линија или других невидљивих знакова и промену форматирања кода. Такође, у ове измене се могу укључити и додавање, уклањање и измена модификатора променљивих и класа, подела или спајање декларација променљивих и проста промена излаза програма. Могуће је модификовати константне вредности, тако да то нема утицаја на семантику програма и сл.

Структуралне модификације, за разлику од лексичких модификација, захтевају одређено познавање програмирања и коришћеног програмског језика и знатно отежавају детекцију сличности. Промене редоследа променљивих у исказима, промене редоследа програмских блокова или појединачних наредби у оквиру блокова кода се раде на начин који нема утицаја на семантику програма. Контролне структуре као што су условна гранања и петље се мењају еквивалентним контролним структурама.

Структурални редизајн изворног програма је вероватно најефикаснија техника маскирања плагијаризма. Примери су замена позива функције телом функције или обрнуто, премештање секвенце исказа у засебну функцију, премештање функције у засебну класу и сл. Редизајн може укључити додавање непотребног кода у програм. Могу се додавати функције које се никад не позивају, декларисати променљиве које се никада не користе, или додавати искази који не мењају излаз програма. Вредност подизраза се може сместити у засебну променљиву или обрнуто, а вредност променљиве се може директно користити у изразу. За променљиве и функције се може извршити модификација досега важења. Код објектно-оријентисаних језика ове измене могу бити још значајније и укључивати додавање непотребних апстрактних класа и интерфејса, измену структура података (класа) и коришћење алтернативних имплементација библиотечких функција.

У отвореној литератури се може наћи већи број класификација лексичких и структуралних измена. Већина ових подела је базирана на класификацији која је дата у [18] и која излистава 10 карактеристичних измена. Сличне описе модификација су дали и [25, 26]. Најдетаљнији и најисцрпнији списак техника плагирања је дат у [27]. Он укључује 51 различиту измену које су подељене на измене у изворном програмском коду, документацији, корисничком интерфејсу, начину имплементације програма и дизајну програма.

У контексту овог рада, усвојена је модификована класификација дата у [24] која садржи 22 различите измене. Ова класификација боље описује могуће измене у програмском коду уз задржавање одговарајућег нивоа детаља у односу на [18], а са друге стране избегава

сувише висок ниво детаља који намеће класификација [27]. Усвојена класификација је дата у наставку текста, а коришћена је у ауторовом раду [28].

Идентификоване су следеће лексичке измене:

1) промена форматирања у програмском коду (L1)
2) додавање, измена или брисање коментара (L2)
3) измена или другачије форматирање излаза програма (L3)
4) преименовање идентификатора (L4)
5) подела или спајање декларација променљивих (L5)
6) додавање, измена или брисање модификатора (L6)
7) промена константних вредности и знаковних низова (L7)

Структуралне измене су подељене на следећи начин:

8) промена редоследа и измена оператора и операнада (S1)
9) промена редоследа наредби у оквиру блока кода (S2)
10) промена редоследа блокова кода (S3)
11) додавање редундантних наредби или променљивих (S4)
12) измена или замена контролних структура еквивалентним (S5)
13) измена типова и структура података (S6)
14) рефакторисање функције и замена позива функције телом (енг. *inlining*) (S7)
15) додавање редундантног кода (интрефејса, апстрактних класа, функција) (S8)
16) додавање привремених променљивих и подизраза (S9)
17) структурални редизајн програмског кода (S10)
18) промена досега идентификатора (S11)
19) коришћење алтернативних библиотечких функција (S12)
20) превођење макроа у функцију и обратно (S13)
21) коришћење еквивалентних наредби (S14)
22) промена декларације (потписа) функције (S15)

Са L су означене лексичке измене, а са S су означене структуралне измене и помоћу ових ознака ће измене бити реферисане у даљем тексту.

Из оригиналне класификације је уклоњена лексичка измена *language translations*, јер је аутор овог рада сматрао да су текстуалне замене добро покривене изменом L7. Са друге стране, додате су измене S12 - S15, јер је у пракси запажено да их студенти повремено користе да би прикрили плагијаризам у својим кодовима.

У наставку су дата два примера програмских кодова који су настали вршењем лескичких и структуралних измена и коришћени су за потребе анкете на тему плагијаризма у програмском коду која је детаљније обрађена у Прилогу А. За сваки приказани код је наведено које измене су извршене и дат је одговарајући коментар. У примерима се сматра да кодови задржавају основну функционалност на нивоу алгоритма, али да не морају задржати поптуно једнак излаз, нити да морају бити потпуно функционално идентични. То одговара реалним случајевима, где плагијатори услед недовољног програмерског знања могу лако чињењем измена у програмском коду да би прикрили плагијаризам да ненамерно промене и његову функционалност.

У Табели 2.1 је дат пример два слична програмска кода, где је код са десне стране настао од левог углавном лексичким изменама. Извршене су промене форматирања (L1), измене над коментарима (L2) и у испису програма (L3) и извршено преименовање променљивих (L4). Учињене су и две једноставне структуралне измене. Извршено је спајање две доделе вредности у једну вишеструку доделу вредности (S2) и позив библиотечке функције **`putchar`** је замењен еквивалентним позивом функције **`printf`** (S12). Са приличном сигурношћу може се рећи да ова два кода представљају случај плагијаризма, јер је очигледно један код настао из другог.

Табела 2.1. Два слична програмска кода
од којих је један настао из другог трансформацијама L1, L2, L3, L4, S2 и S12

```
putchar('\n');
//***************************
//stampanje koordinata matrice
for (i = 0; i < dim; i++) {
  for (j = 0; j < dim; j++) {
    if (i != j)
      printf("[%d,%d] ", i, j);
    else printf("****");
  }
  putchar('\n');
}
//***************************
//da li je trougaona matrica
bot = 0;
top = 0;
for (i = 0; i < dim; i++)
  for (j = 0; j < dim; j++) {
    if (i<dim && mat[i][j] == 0)
      bot++;
    if (i>dim && mat[i][j] == 0)
      top++;
  }
```

```
printf("\n");
for(i=0;i<m;i++){ //*ISPIS INFO MATRICE*//
  for(j=0;j<m;j++)
  {
    if(i!=j)printf("[%d,%d]   ",i,j);   else
printf("XXXXX ");

  }
  printf("\n");
}

//***ISPITIVANJE DA LI JE TROUGAONA ***///
dole=gore=0;

for(i=0;i<m;i++)
  for(j=0;j<m;j++)
  {
    if(m>i && wut[i][j]==0)dole++;
    if(m<i && wut[i][j]==0)gore++;

  }
```

Табела 2.2 даје пример два слична програмска кода, где је код са десне стране настао од левог, али са доминантно коришћеним структуралним изменама. На први поглед је тешко рећи да ли је у питању случај плагијаризма, јер су измене значајне. Извршена је промена форматирања померањем ознака за почетак блока кода и додавањем празних редова (L1) и измена имена функције и променљивих (L4). Међутим, много већи утицај на сличност имају структурне измене: промена редоследа наредби у петљи (S2), замена **`while`** петље еквивалентном **`for`** петљом (S5), рефакторисање функције у смислу избацивања позива **`atol`** функције (S7) и, у складу са тим, промена потписа функције која укључује и промену типа једног од аргумента и замену њиховог редоследа (S15).

Табела 2.2. Два слична програмска кода
од којих је један настао из другог трансформацијама L1, L4, S2, S5, S7 и S15

```
void obrada (char *ul_pomeraj,
Elem *prvi){
  Elem *trenutni=prvi;
  int pomeraj=atol(ul_pomeraj);
  while(trenutni) {
    trenutni->sadrzaj->vreme_poj
    +=pomeraj;
    trenutni->sadrzaj->vreme_ukl
    +=pomeraj;
    trenutni=trenutni->sledeci;
  }
}
```

```
void titles ( Subtitle_list *list,
int disp)
{
  Subtitle_list *current;

  for(current = list; current != NULL;
      current=current->next)
  {
    current->info->t_remove+=disp;
    current->info->t_appear+=disp;
  }
}
```

2.4. Ставови студената и наставника

Као што је речено у претходним поглављима, плагијаризам је тема осетљиве природе која је често извор различитих гледишта студената и наставника, али и међу самим наставницима. Иако почињени акт плагијаризма најчешће не носи никакве законске последице по учиниоца, он носи одређене импликације и санкције у оквиру академског окружења. Стога је битно разумети ставове студената и наставника на ову тему, како би се кроз боље информисање и превенцију предупредили случајеви плагијаризма и неспоразуми који из тога произилазе. Ставови студената и наставника су били предмет више студија на које ће бити извршен кратак осврт у овом поглављу.

2.4.1. Преглед досадашњих истраживања

У раније помињаној студији [13, 27], аутори су се бавили дефиницијом плагијаризма у програмском коду на узорку од 59 наставника програмирања са универзитета у Великој Британији. Иако не постоји потпуно слагање о самој дефиницији плагијаризма, наставници се генерално слажу о политици нулте толеранције према овој појави. Слажу се да коришћење кода који је написао неко други мора бити цитирано. Разлике постоје у погледима на аутоплагијаризам, јер је један број наставника склон нешто мекшем ставу који охрабрује поновну употребу кода који је раније написан и предат у оквиру неког другог домаћег задатка сматрајући да се тиме подстичу добре праксе из објектно-оријентисане парадигме програмирања. Студија је такође показала да потенцијални случајеви плагијаризма треба да буду истражени без обзира на учешће задатка у коначној оцени. Међутим, наставници показују мање агилности да пријаве такве случајеве одговарајућим комисијама, уколико се такви задаци не бодују за оцену. Такође, наставници су понекад склони да спроводе одређене арбитрарне мере према починиоцима плагијаризма које нису део одговарајућих академских правилника.

Исти аутори су истраживали ставове студената британских универзитета на тему плагијаризма у рачунарству [29, 30]. Аутори су покушали да идентификују „сиве зоне“ у активностима које студенти чине приликом решавања програмских задатака, где границе између дозвољеног и недозвољеног понашања нису довољно добро дефинисане или схваћене од стране студената. Идентификована су четири могућа проблема међу студентском популацијом: поновно коришћење кода из претходних задатака студенти генерално не схватају као форму плагијаризма, неадекватном цитирању се не поклања довољно велика

пажња, дељење кода се сматра прихватљивим и код задатака који треба да се решавају самостално и превођење кода из једног програмског језика у други се не сматра плагијаризмом.

У Словачкој је спроведена студија која је испитивала погледе наставног кадра и студената на проблеме софтверског плагијаризма [16]. У анкети је учествовало 313 студената и 25 наставника, а обухваћене су теме као што су циљеви, одговорност, начини детекције, дисциплинска политика за прекршиоце, морални аспекти и сл. Резултати студије показују да се погледи студената и наставника у вези плагијаризма значајно разликују, а да студенти генерално имају "мекше" погледе на ову негативну праксу. Резултати анкете показују да је више од 30% студената признало да је бар једном током студија починило акт плагијаризма и предало туђ рад као свој, а више од 60% студената је навело да је једном током студија послало свој рад другима. Два коаутора истраживања су у тренутку писања студије били студенти, а један од њих је изнео издвојено мишљење у коме објашњава основне мотиве који подстичу студенте да плагирају туђа решења. У том мишљењу се на неколико примера истиче како погрешан приступ наставног особља може допринети овој појави. Са друге стране, у овом мишљењу се истиче да већина студената ипак не подржава акте плагијаризма и да студенти-варалице који то чине и покушавају да стигну до дипломе на лакши начин нису популарни међу својим колегама.

Иако се наставници генерално слажу да копирање комплетног кода представља акт плагијаризма, не постоји слагање око тога да ли копирање блока кода (једне методе) или једне једине линије представља плагијаризам или не [31]. Такође, свест студената о примени одређених алата за детекцију сличности програмског кода [31, 32], значајно утиче на њихове активности. Превентивни ефекат постоји, јер одређени студенти избегавају активности које се могу оценити као плагијаризам уколико знају да наставници примењују одређена средства детекције. Са друге стране, свест о могућој детекцији усмерава друге студенте ка софистичнијим методама плагирања о којима ће бити више речи у наредним поглављима. Ипак, поједина истраживања [32] показују да је превентивни ефекат привремен. Иако студенти генерално уче из искуства других, после неког времена та искуства се заборављају и нови студенти почињу поново да плагирају, мислећи да ће проћи неопажено. Са друге стране, исто истраживање тврди да су старији студенти значајно мање укључени у случајеве плагијаризма.

На Универзитету у Београду је појава плагијаризма обрађена у оквиру ширег истраживања на тему различитих облика академског непоштења у основном, средњем и високом образовању [33]. Укупно 391 студент са 5 различитих факултета је учествовао у испитивању о високом образовању. Иако није експлицитно фокусирана на тему плагијаризма, студија показује да преко 30% студената није довољно упознато са правилима у вези цитирања туђег рада, а преко 75% студената је у току студија било укључено у неко форму дељења садржаја у ситуацијама када то није било дозвољено.

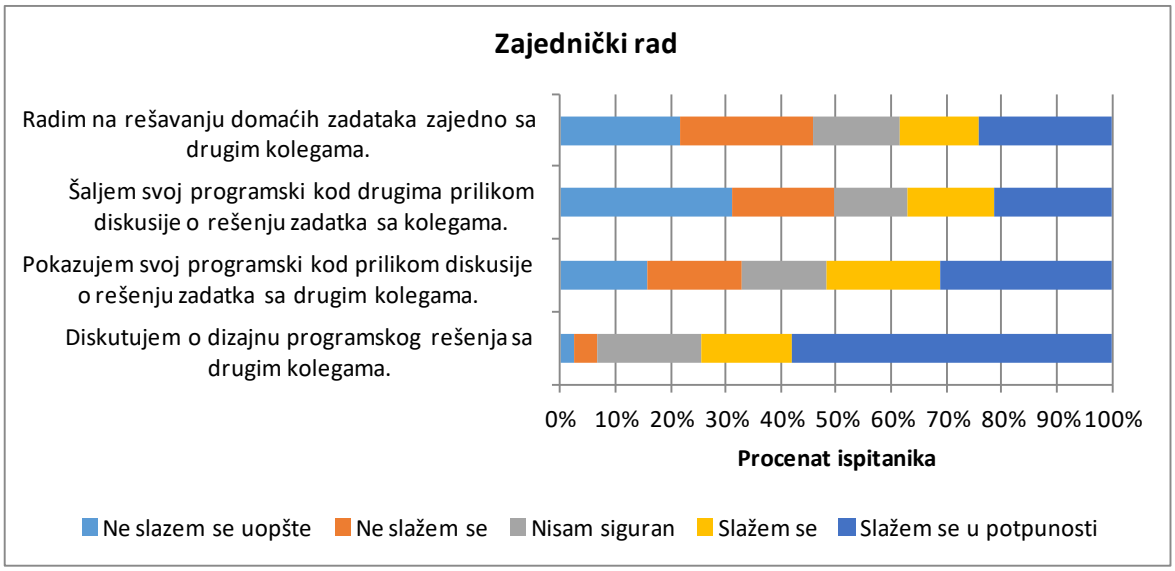
2.4.2. Резултати истраживања ставова и пракси студената на Електротехничком факултету

Као допринос истраживању проблема плагијаризма у програмском коду у оквиру ове докторске дисертације је спроведено истраживање ставова и пракси студената о овој теми на Електротехничком факултету Универзитета у Београду. С обзиром да је плагијаризам у образовању и науци сложен феномен који зависи од норми које га идентификују и одређују, било је од значаја да се испита како студенти Електротехничког факултета разумеју плагијаризам, какве ставове имају према њему и колико су склони да подлежу различитим видовима плагијаризма. Из до сада наведених чињеница је јасно да је питање плагијаризма у програмском коду нарочито осетљиво због великог броја јавно доступних извора који се могу искористити за формирање плагијата и тенденције колективне израде програмског кода као последице колаборативног учења. Истраживање је искоришћено и у циљу испитивања ширих ставова студената Електротехничког факултета на тему академске честитости и академског интегритета. Цело истраживање је приказано у Прилогу А овог документа, а у овом поглављу су приказани само најзначајнији резултати битни у контексту овог рада.

Да би се одговорило на ова питања, креиран је упитник експлоративног типа који се, поред општих питања, састојао од три креирана инструмента који користе Ликертове скале процене. Ти инструменти су:

1) скала ставова студената према плагијаризму и академској честитости,
2) скала учесталости пракси које се могу подвести под плагијаризам уопштено и плагијаризам у програмском коду,
3) скала процене за идентификацију плагијаризма у програмском коду.

Спроведено истраживање је показало неколико важних чињеница. Најпре, студенти нису добро упознати са дефиницијом плагијаризма и дозвољеним и недозвољеним праксама. То важи како за плагијаризам генерално, тако и за плагијаризам у програмском коду. Иако се слажу да ова појава представља нарушавање принципа академске честитости, већина студената није довољно упозната са тренутним академским прописима на ову тему, па велики број њих сматра да су различити облици сарадње, као што су заједнички рад, размена кодова, коришћење кодова са Интернета и сл., дозвољене праксе (Слика 2.1.). Са друге стране, ни правилници о дисциплинској одговорности не регулишу ово питање на довољном нивоу детаља, тако да свакако постоји потреба да се академска заједница позабави и информисањем студената и регулисањем различитих аспеката ове појаве.



Слика 2.1. Праксе студената у вези заједничког рада

Испитани студенти генерално имају мекше ставове према плагијаризму у образовању, него у науци, а сваки десети испитаник сматра да је плагијаризам допустив. Око 40% студената је бар једном слало свој домаћи задатака другоме. С обзиром да 80% дипломаца сматра да је давање програмског кода другом колеги у реду и да за плагијаризам није одговоран онај који шаље код, може се сматрати да постоји погодна клима за ширење ове непожељне праксе. Иако преко 90% студената тврди да никада није предавало туђ домаћи задатак, око 7% испитаних признаје да је то учинило. Са друге стране, око четвртина испитаника је навела да је користила туђе резултате са интернета или из литературе без адекватног цитирања извора.

Истраживање је обухватило и праксе тешког нарушавања правила о полагању испита, као што је полагање путем бубица или уместо колеге, преписивање на испиту и сл. Иако занемарљиво мало људи признаје ову праксу, значајно већи проценат људи наводи да су чули или видели да се оваква пракса спроводи. Слично је и са праксом плаћања трећем лицу да уради домаћи или пројектни задатак, где чак 50% испитаника тврди да зна бар за један такав случај.

Студенти доста добро процењују да ли неки конкретни сегменти програмског кода представљају плагијаризам и у којој мери га одређене модификације могу прикрити. Ипак, у складу са ставовима израженим у оквиру првог дела истраживања, њихова процена конкретних понашања значајно варира. Студенти исправно препознају да копирање и предавање комплетног кода представља плагијаризам, али се у много мањој мери слажу да плагијаризам представља копирање једне линије, дела кода или комплетног кода уз измену.

Слично важи и за неке друге аспекте програмског решења, као што су коментари, скупови података за тестирање, кориснички интерфејс, структура решења и сл. Копирање коментара и скупова података за тестирање за већину студената не представља вид плагијаризма, али је проблематичније што слично важи, додуше у мањој мери, и за кориснички интерфејс и структуру решења, што су битни аспекти које они треба самостално да науче да реализују.

У оквиру истраживања је обављена и факторска анализа. Најважнији налаз представљају фактори који описују праксе лаког и тешког плагијаризма. Под лаким плагијаризмом се подразумевају све оне праксе које могу бити схваћене и као помагање другом студенту у виду давања одређених одговора током самих испита или приликом израде домаћих задатака, заједнички рад и сл. Тешки плагијаризам описује све оне непоштене праксе које представљају теже повреде дисциплие, као што су полагање испита уз коришћење „бубица“, коришћење туђег идентитета приликом полагања испита и сл.

3. АЛГОРИТМИ ЗА ДЕТЕКЦИЈУ СЛИЧНОСТИ У ПРОГРАМСКОМ КОДУ

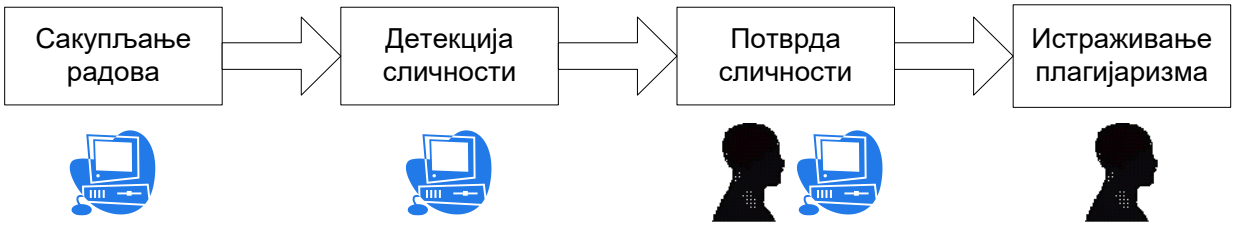
Детекција плагијаризма у програмском коду је процес који се састоји од више корака. Један од главних корака је детекција сличних делова кода. У том смислу, велики значај имају алгоритми за детекцију сличности који треба да укажу да су нека два програмска кода довољно слична да би била од интереса за ручну инспекцију којом се може потврдити плагијаризам. Стога ће у овом поглављу бити дат осврт на основне технике које се користе у детекцији сличности у програмском коду. Велики број алгоритама за детекцију сличности је заснован на техникама за упаривање стрингова (знаковних низова), па ће њима бити поклоњена нарочита пажња.

3.1. О детекцији плагијаризма у програмском коду

Програмски код није тешко плагијаризовати, а често за то и нису потребне нарочите програмерске вештине. Са друге стране, тешко је са великом поузданошћу аутоматски утврдити да ли је неки део кода плагијат или не, нарочито уколико је код измењен да би се плагијаризам сакрио. У том смислу, коначну одлуку о постојању плагијаризма у неком коду доноси човек након мануелне инспекције кода.

Процес детекције плагијаризма у програмском коду је у [34] дефинисан као процес у четири фазе: сакупљање програмских задатака, детекција сличности (анализа), потврда сличности и истраживање плагијаризма. У почетку су све четири фазе морале бити рађене ручно, али су са развојем софтверских алата и напретком на пољу алгоритама прве две фазе у

потпуности аутоматизоване, док се последње две фазе могу само делимично аутоматизовати. У фази сакупљања се програмски задаци прикупљају од студената и врши одређена предобрада у смислу униформисања, како би се задаци припремили за поређење од стране алата за детекцију сличности. Затим се врши детекција сличности, што је у данашње време типично аутоматизован поступак кроз доступне софтверске алате. У фази потврде плагијаризма се врши инспекција листе која садржи проценте сличности за парове студената. Та листа је уобичајено сортирана опадајуће по проценсту сличности. Фаза потврде има за циљ елиминацију лажно позитивних резултата (тзв. *false hit*) који се могу појавити због несавршености процеса детекције сличности. У четвртој фази, фази истраживања, наставник врши инспекцију сумњивих парова, како би утврдио да ли сличност долази из дозвољених разлога или су у питању случајеви плагијаризма. Графички приказ описаних фаза је дат на Слици 3.1.



Слика 3.1. Четири фазе процеса детекције плагијаризма у програмском коду

Већина данашњих приступа процесу детекције плагијаризма се фокусира на прве две фазе описаног поступка [35], па ће стога у овом поглављу пажња бити посвећена алгоритмима за детекцију сличности. О трећој и четвртој фази ће бити више речи у шестом и седмом поглављу рада.

У суштини, програмски код је једна специфична врста текста, јер програмски код најчешће има јасно дефинисану структуру. Интелигентно поређење текстова подразумева семантичку анализу која је кључни корак да би се уочиле сличности и поред спроведених модификација извршених да би се оне прикриле. Детекција сличности у програмском коду укључује поређење већег броја изворних фајлова који могу садржати више хиљада линија кода. Стога је детекција сличности у програмском коду компликована и временски захтевна радња. Алгоритми за детекцију сличности морају бити отпорни на различите модификације кода, како лексичке, тако и структуралне, које се врше да би плагијаризован код изгледао другачије од плагираног решења.

3.1.1. Академско окружење – детекција плагијаризма

Треба јасно направити разлику између процеса детекције плагијаризма у академском окружењу и у индустрији. Масовни академски курсеви из области рачунарства обично укључују велики број студената. Ти бројеви се најчешће крећу између 100 и 300 студената, али нису ретки ни масивни курсеви програмирања [3] са близу 1000 студената, као што је то случај на Електротехничком факултету Универзитета у Београду, што сваки покушај ручне детекције плагијаризма чини готово немогућим.

Детекција плагијаризма на оваквим курсевима уобичајено укључује поређење већег броја софтверских решења мањег обима. Типични студентски домаћи задаци се састоје од 100 до 300 линија кода, а пројектни задаци 500 до 2000 линија кода. Коришћени софтверски алати стога треба да показују добру скалабилност са бројем радова који се пореде, буду отпорни на описане модификације и дају што мање лажно позитивних резултата. Овом врстом система и њиховим унапређењем се овај рад првенствено бави.

3.1.2. Индустрија – детекција софтверских клонова

У индустријским применама се процес поређења програмских кодова своди на детекцију софтверских клонова [36-39]. Уобичајено се пореде два или неколико решења већег обима [37], а често је циљ утврђивање нарушавања патентних права или крађе интелектуалне својине [40], потреба за рефакторисањем кода, анализа рањивости софтвера, откривање злонамерног кода и сл. [41] У том смислу, плагијаризам је слична појава, али су узроци настанка најчешће различити. Код плагираног кода се тежи задржавању исте функционалности, уз модификацију решења тако да се што више разликује од оригинала. У домену производње софтвера, клонови се могу јавити као идентични или слични делови кода у оквиру једног софтверског производа који су настали из потребе за бржим развојем или одређеним оптимизацијама у оквиру кода [41].

Због нешто другачије природе проблема код детекције софтверских клонова и приступи решавању проблема су другачији. У академској средини су изворни кодови програма доступни, док код детекције софтверских клонова то не мора бити случај. Када изворни код решења није доступан за поређење, користе се процедуре за детекцију сличности у бинарним кодовима [42] које укључују свођење кодова на једнак, нормализован облик, било процесом превођења, било процесом декомпајлирања, што уноси додатне елементе комплексности у цео процес поређења.

Интересантно је приметити да је уобичајена класификација софтверских клонова према учињеним изменама у односу на оригинални код [38] знатно грубља у односу на класификацију лексичких и структуралних измена које студенти користе приликом плагирања у академској средини, а која је описана у Поглављу 2.3.3. Софтверски клонови се према [38] деле на четири типа. Софтверски клон типа 1 представља код који је идентичан оригиналном, осим евентуалних измена у форматирању и коментарима. Тип 2 се добија преименовањем идентификатора у односу на полазни код уз задржавање синтаксне идентичности. У клону типа 3 се могу појавити додавање, измена или уклањање наредби. Све остале модификације које укључују задржавање семантичке сличности, али уз измене које не дозвољавају синтаксну сличност представљају софтверске клонове типа 4.

Аутор овог рада сматра да је разлика у класификацији директна последица разлике у применама система за детекцију софтверских клонова и система за детекцију плагијата. У академској средини је, како због дисциплинских разлога (због одређивања казне), тако и због едукативних разлога (због корекције наставног процеса) потребно боље разумети на који начин је плагирани рад настао из оригиналног. У софтверској индустрији је често довољно утврдити само постојање клона. У овом раду ће стога само повремено бити направљен осврт и на процес детекције клонова, у мери у којој се он поклапа са детекцијом плагијаризма у програмском коду.

3.2. Технике за детекцију сличности у програмском коду

Постоји више техника за детекцију сличности у програмском коду. Две најчешће коришћене технике су бројање атрибута (енг. *attribute counting*) и структурно оријентисано поређење, али се такође користе и поједини методи за извлачење информација (енг. *information retrieval*) као што су латентна семантичка анализа (eng. *latent semantic analysis*) [43]. Ипак, њихов је учинак значајно слабији од метода који узимају у обзир и синтаксу програмског кода. Поред наведених, поједини експериментални методи се базирају на теорији информација и користе ентропију програма и комплексност Колмогорова за одређивање сличности [44]. У академском окружењу су технике структурно оријентисаног поређења које се заснивају на упаривању стрингова (енг. *string matching*) нарочито заступљене [18, 24], па ће овим алгоритмима бити посвећена посебна пажња.

3.2.1. Алгоритми за проналажење шаблона у стринговима

Проналажење или упаривање шаблона (енг. *pattern*) у стринговима је честа операција са применама у претраживању и истраживању података, филтрирању података, откривању злонамерног кода, анализи саобраћаја у рачунарским мрежама, и сл. Сам поступак претраживања и проналажења шаблона у неком тексту подразумева идентификацију једног или више задатих шаблона који се могу састојати од знакова из унапред дефинисаног скупа. Поступак најчешће подразумева проналажење идентичне копије шаблона у неком тексту и одређивање позиције једног или свих његових појављивања [45]. Додатне акције обично укључују и замену текста другим садржајем, попут познате опције *Find & Replace* из различитих текст процесора. Један број алгоритама дозвољава апроксимативно поређење, где се метрике, попут Левенштајнове дистанце, користе да би одредили меру сличности.

Алгоритми за проналажење шаблона у стринговима се обично деле према броју шаблона задатих у претрази на алгоритме који користе један шаблон (енг. *single pattern matching*), више шаблона (енг. *multiple pattern matching*) или неодређено (бесконачно) много шаблона. У последњу групу спадају алгоритми који као шаблон користе регуларне изразе и они нису од посебног интереса у контексту детекције сличности у програмском коду.

У групу алгоритама који користе један шаблон спадају претраживање грубом силом (тзв. "наивни метод") [45], *Knuth-Morris-Pratt* (KMP) алгоритам [45, 46], *Boyer-Moore* алгоритам [47], основни *Karp-Rabin* алгоритам [45, 48] и *bitap* алгоритам [49]. Ови алгоритми користе различите технике претпроцесирања стринга (префиксне и суфиксне табеле, битске маске, хеширање и сл.), како би прескочили део текста и смањили број поређења. Одређени алгоритами, попут KMP алгоритма, базирани су на коначним аутоматима. *Boyer-Moore* алгоритам је познат по својој брзини и често се користи као репер за поређење приликом евалуације нових приступа у откривању шаблона [50].

Алгоритам који претражују на више шаблона најчешће чине екстензије алгоритама из прве групе, попут *Rabin-Karp* алгоритма, *Aho-Corasick* [51] (екстензија KMP алгоритма), *Commentz-Walter* [52] (екстензија *Boyer-Moore*), *Wu-Manber* [49] (екстензија *bitap* алгоритма коришћена у *UNIX* алату *agrep*), али и алгоритми попут *Greedy-String-Tiling* (GST) [53, 54] који ће детаљно бити описан у једном од наредних поглавља. Ниједан од наведених алгоритама не испољава добре карактеристике и код претраге појединачних шаблона и код претраге на више шаблона, па им је стога и примена другачија. С обзиром да се детекција

сличности између два програмска кода може посматрати као претраживање скупа шаблона из једне датотеке у другој датотеци, јасно је да се за овај поступак мора одабрати један од алгоритама из друге групе.

Да би се боље разумела временска сложеност проналажења шаблона у стрингу, у Табели 3.1 су дате временске сложености карактеристичних алгоритама из прве групе. Табела је заснована на подацима из [45], а нотације асимптотских сложености (велико O, велико Ω и велико Θ) је дефинисао Кнут у раду [48]. Текст који се претражује се посматра као стринг *T[1..n]* дужине *n*, а шаблон *P[1..m]* као стринг дужине *m*, тако да важи $m \leq n$. Број знакова од којих се састоји алфабет није већи од *k*. Из табеле се јасно види да већина алгоритама у најгорем случају тежи квадратној временској сложености, али да се код неких алгоритама у одређеним ситуацијама може тежити и линеарној временској сложености.

Табела 3.1. Временска сложеност алгоритама за проналажење појединачних шаблона у стринговима

Алгоритам	Време претпроцесирања	Време упаривања
Груба сила [45]	0 (нема претпроцесирања)	Θ(n·m)
Karp-Rabin алгоритам [45, 48]	Θ(m)	просечно: Θ(n + m) најгоре: Θ((n - m)·m)
KMP алгоритам [45, 46]	Θ(m)	Θ(n)
Boyer-Moore [47]	Θ(m + k)	најбоље: Ω(n/m) најгоре: O(n·m)
Bitap алгоритам [49]	Θ(m + k)	O(n·m)

3.3. Техника бројања атрибута

Технике засноване на бројању атрибута анализирају одређене кључне показатеље (метрике) унутар програмског кода на основу кога формирају својеврсан дигитални потпис (енг. *fingerprint*) који се онда користе за одређивање сличности парова програма. Ти кључни показатељи варирају од система до система, а најранији системи, попут [55], су били засновани базирани на Халштедовој метрици [56] за одређивање сличности међу паровима програма на програмском језику FORTRAN која се састојала од четири једноставна показатеља: броју јединствених оператора, броју јединствених операнада, броју појављивања оператора и броју појављивања операнада.

Каснији системи су укључивали и друге метрике, као што су подаци о броју променљивих, броју оператора, броју метода, броју петљи, броју кључних речи, средњем броју речи по једној линији, броју јединствених речи, величини програма [18, 24], али и сложеније метрике, као што су број позива функције, број грана у графу контроле тока, цикломатска комплексност, *Henry-Kafura* метрика [41] и сл. Одређени алати су користили и различите метрике засноване на графу тока контроле и различите инваријантне показатеље који се тешко мењају променом кода [57]. Пример таквог алата је дат у [58], где су аутори користили 24 различита показатеља да опишу програм. Показатељи, притом, са различитим значајем улазе у коначан дигитални потпис програмског кода.

Ипак, пракса и различите студије [24, 28, 57] су показале да системи засновани на бројању атрибута значајно заостају по ефикасности за системима заснованим на структуралном поређењу. Системи засновани на бројању атрибута су погодни за детекцију изузетно сличних, готово идентичних копија и углавном елиминишу утицај лексичких измена у програмском коду. Међутим, они нису превише отпорни чак и на мање структурне измене и дају добре резултате само код програма мањег обима, чак и у академском окружењу. У програмима већег обима у којима постоји само делимични плагијаризам, ове технике тешко могу да послуже за ближу идентификацију сличних сегмената, јер по природи ствари оперишу над комплетним кодовима.

3.4. Структурално поређење

Да би се умањио утицај не само лексичких, већ и структуралних измена, већина техника структурно оријентисаног поређења узима у обзир и синтаксу програмског језика на коме је код написан, како би се добиле додатне информације које могу да се узму у обзир у процесу поређења. Стога се улазни програмски код парсира, а након тога мора проћи одређене трансформације пре фазе одређивања сличности. Трансформације се врше како би се елиминисао утицај лексичких измена у коду, а то се најчешће ради кроз процес токенизације програмског кода [59].

Већина техника за структурно оријентисано поређење се заснива на техникама упаривања стрингова који се добијају поменутим трансформацијама улазног кода. Програмски кодови се тада представљају помоћу два низа знакова (токена) који се на одређени начин пореде [26, 60]. Постоје и друге алтернативе које још боље описују структуру кода, а заснивају се на апстрактним стаблима парсирања [61] и анализи графа

зависности унутар програма [62]. Ипак, због велике сложености и рачунске захтевности ових поступака, они се чешће користе у детекцији софтверских клонова, када се пореди неколико програма већег обима. У наредним потпоглављима су стога ближе описане технике засноване на токенизацији, апстрактним стаблима парсирања и графовима зависности, а затим је у наставку текста посвећена пажња алгоритмима за детекцију сличности базираним на токенизацији и упаривању стрингова.

3.4.1. Токенизација

Велики број приступа за структурално поређење програмског кода користи токенизацију улазног кода као једну од фаза претпроцесирања у детекцији сличности. Улога токенизације је да се програмски код претвори у секвенцу смислених симбола. Ти симболи се називају токени и секвенца токена се доставља алгоритму за детекцију сличности као улаз.

Токени се бирају тако да представљају суштинске конструкте у програмском коду, које је тешко мењати, а да се притом изврши редукција (апстрактизација) целокупне количине информација које програмски код садржи. На пример, токени се бирају тако да засебни токени представљају почетак и крај блока кода, променљиве, позиве функција, изразе, а да се са друге стране уклоне коментари, форматирања и слични лексички елементи [24, 59]. Са друге стране, да би се елиминисао утицај замене петље еквивалентном, за петље се често користи исти токен.

У Табели 3.2. је дат пример токенизације једног једноставног потпрограма од неколико линија кода. Пример је конструисан на основу скупа токена коришћеног у секвенцијалној имплементацији експерименталног система описаног у Секцији 5.1. Као што се види, било какве лексичке модификације, попут измене имена функције или променљиве, промене библиотечке функције за испис алтернативном, неће променити токенизовани код, па стога неће бити могуће прикрити ни учињени плагијаризам.

Табела 3.2. Пример токенизације једноставног потпрограма

Пример кода	Нумерички код токена	Токен
`void pisi () {`	20	BEGIN_FUNCTION
`  int i = 0;`	1, 10	NUM_VARDEF, ASSIGN
`  i += 1;`	10	ASSIGN
`  printf("%d", i);`	11	APPLY
`}`	21	END_FUNCTION

Процес токенизације носи са собом и одређене проблеме који се пре свега огледају у губитку сувише велике количине информација. То је најчешће последица избора скупа токена, уколико је учињена превелика апстрактизација језичких конструката. У таквим случајевима се могу јавити лажно позитивни резултати, када се токенизоване репрезентације два програмска кода подударају у приличној мери, али ручна инспекција показује да сличност не постоји. У раду [24] су показани одређени примери који илуструју лоше изабран скуп токена и дају се смернице за решавање овог проблема које су искоришћене у њиховом алату. Са друге стране, алати базирани на токенизацији генерално пате од проблема са променом редоследа наредби и уметањем програмског кода чиме се прекидају потенцијалне секвенце за упаривање. О том проблему ће више речи бити у Поглављу 3.5.

Са техничког аспекта, процес токенизације се може спровести коришћењем већ постојећих алата за генерисање парсера који се користе у конструкцији програмских преводилаца. С обзиром да се за потребе поређења не генерише машински код, довољно је користити поједностављен, мање прецизан скуп токена, па је самим тим поједностављено и ажурирање алата приликом додавања нових програмских језика.

3.4.2. Апстрактна синтаксна стабла

Апстрактна синтаксна стабла (енг. *abstract syntax trees*) или апстрактна стабла парсирања (енг. *abstract parse trees*), како се понекад називају у литератури, су познате структуре података које се користе приликом превођења програма. Често се користе у анализи и трансформацијама програмског кода. Апстрактно стабло парсирања садржи информације о синтаксној структури неког програмског кода представљене помоћу *m*-арног стабла које се генерише помоћу одговарајућег генератора парсера, као што је *ANTLR* [63]. Чворови стабла представљају конструкте који се јављају у програмском коду, а синтакса је апстрактна, јер се неки детаљи из синтаксе програмског језика приказују имплицитно. Такав је случај са представљањем заграда у изразима које се подразумевају на основу структуре стабла. Слично, поједини конструкти, попут контролних структура, се могу представити на поједностављен начин. Идентификатори се такође често занемарују код примена у детекцији сличности, како би се елиминисао утицај преименовања идентификатора. Апстрактна синтаксна стабла се накнадно могу процесирати тако да им се придруже и додатне семантичке информације.

У смислу детекције сличности у програмском коду, кодови који се пореде се најпре парсирају тако да се добију поменута стабла, а затим се врши упаривање њихових подстабала да би се открила сличност. Процес упаривања захтева да се стабла на неки начин трансформишу у облик погодан за поређење. Уобичајено се за те намене користе језгра стабала парсирања (енг. *parse tree kernels*) која су дизајнирана тако да мапирају апстрактно синтаксно стабло у одговарајући вектор који садржи сва могућа подстабла која могу да се појаве у стаблу парсирања [61]. Затим се на основу ових вектора врши одређивање мере сличности.

Главни недостаци овог метода су висока рачунска комплексност, али и осетљивост на поједине структурне промене, као што су замена редоследа наредби, промена контролних структура и занемаривање тока података [41]. Програми који се парсирају морају бити синтаксно исправни да би се парсирали и поредили, што није увек случај са студентским радовима. Конструкција парсера такође може бити проблем, пошто додавање новог језика у систем за детекцију сличности захтева имплементацију комплетног парсера одговарајућег програмског језика. Иако за већину популарних програмских језика постоје генератори парсера и доступни описи граматика, то није увек случај, па њихова израда може представљати приличан програмерски напор.

3.4.3. Графови зависности програма

Да би се што вернije одсликала структура програмског кода, као и везе које постоје између појединих делова кода често се користе графови зависности програма (енг. *program dependence graphs*). Граф зависности програма представља репрезентацију контролних зависности и зависности по подацима у оквиру једне процедуре у виду усмереног графа [64]. Чворовима графа се представљају основни конструкти и наредбе у програму, а контролне зависности и зависности по подацима се репрезентују помоћу грана графа. Сматра се да граф зависности добро одсликава логику програма, а самим тим и начин размишљања програмера. У том смислу, добијени граф је инваријантан у односу на велики број техника плагијаризма у програмском коду, па се са великом поузданошћу може вршити детекција сличности [62].

Процес детекције сличности помоћу ове технике заснива се на поређењу скупова подграфова насталих од процедура из парова програма који се пореде. Поређење се базира на утврђивању изоморфизма подграфова. Уколико је неки подграф из скупа подграфова једног програма изоморфан неком подграфу из скупа подграфова другог програма, сматра се да су

процедуре које они представљају сличне. С обзиром да је тестирање изоморфизма два графа комплексан проблем са великом рачунском сложеношћу, прибегава се различитим техникама оптимизације којима се смањује број подграфова за поређење.

Главна предност овог приступа се заснива на отпорности на структуралне промене у програмском коду, као што су промена редоследа наредби, уметање делова кода, замена контролних структура еквивалентним и сл. Са друге стране, највеће мане ових система су њихова скалабилност, како у односу на величину програма који се пореде, тако и у односу на број програма који се пореди. Утврђивање изоморфизма подграфова се у том смислу издваја као елемент са највећом сложеношћу и поред оптимизација које је могуће применити. Такође, имплементација овог приступа захтева имплементацију парсера за одговарајући програмски језик како би се генерисао граф зависности.

3.5. GST алгоритам

Алгоритам поређења представља срж сваког система за детекцију сличности базираном на токенизацији. Перформансе, ефикасност рада и време извршавања целог система у великој мери зависе од алгоритма поређења. Стога се уместо алгоритама који врше егзактну претрагу у стринговима који су описани у Поглављу 3.2.1. често користе алгоритми који врше апроксимативну претрагу, односно примењују одговарајуће хеуристике како би смањили време претраге.

У контексту детекције сличности у програмском коду, програми који се пореде се представљају помоћу два низа токена, а циљ претраге је проналажење свих подударајућих подстрингова у оквиру два стринга са токенима који се посматрају. На основу претходно изложених идеја осмишљен је *Greedy-String-Tiling* (GST) алгоритам који је дао Вајс [53, 54]. Првобитна намена алгоритма је била поређење биолошких секвенци попут ДНК ланаца у биоинформатици, али је од стране истог аутора убрзо искоришћена и за потребе детекције сличности у програмском коду [25]. У овом раду је изложена варијанта алгоритма која је коришћена у систему за детекцију сличности JPlag [26].

Као што је речено, циљ GST алгоритма је проналажење свих подстрингова који су једнаки приликом упоређивања два стринга. Притом, алгоритам је хеуристички, јер не постоје гаранције да ће алгоритам увек пронаћи максималан скуп подударајућих подстрингова. Увођење таквих гаранција би претрагу учинило сувише неефикасном и скупом [26].

Да би се олакшало разумевање алгоритма, у наставку су дате три дефиниције из [53]. Алгоритам подразумева постојање два стринга, *A* и *B* дужине *m* и *n*, респективно.

Дефиниција 3.1. Максимално подударање (енг. *maximal match*) представља најдуже могуће подударање подстринга A_a који почиње од индекса *a* у стрингу *A* са подстрингом B_b који почиње од индекса *b* у стрингу *B*. Поклапање се прекида првим знаком који се не поклапа, крајем једног од стрингова или наиласком на маркирани знак (елемент).

Дефиниција 3.2. Плочица (енг. *tile*) представља обележено (маркирано), максимално подударање подстринга A_a са подстрингом B_b. Маркирани подстрингови се не могу користити у наредним итерацијама упаривања.

Дефиниција 3.3. Минимална дужина поклапања (енг. *minimum match length*) се дефинише тако да се максимална подударања мања од ове дужине игноришу у алгоритму. У даљем тексту ће овај параметар бити реферисан као MML.

Максималан скуп подударајућих подстрингова је резултат рада алгортима и он према дефиницијама мора да задовољи три есенцијална услова која су сумирана у [24]:

4) Сваки токен првог стринга може бити упарен са највише једним токеном другог стринга. Због првог услова, алгоритам не може да упари делове изворног кода који су евентуално дуплицирани у једном од кодова који се пореде.
5) Подстрингови морају бити пронађени, без обзира на њихову позицију у стрингу. Према овом услову, промена редоследа делова изворног кода није ефективна техника за сакривање плагијаризма. Овај услов важи само уколико су премештани делови кода већи од минималне дужине поклапања.
6) Дужим подударањима подстрингова се даје предност у односу на кратка подударања. Сматра се да су дужа подударања поузданија и да боље указују на делове сличног кода него краћа подударања која могу бити плод случајности.

Сам GST алгоритам се састоји из две основне фазе које су приказане у оквиру псеудокода датог у Табели 3.3. Линије 5-18 чине прву фазу алгоритма, док се друга фаза врши у линијама 19-25.

Табела 3.3. Псеудокод GST алгоритма [26]

```
0       Greedy-String-Tiling(String A, String B, int MML) {
1         tiles = {};
2         do {
3           maxmatch = MML;
4           matches = {};
5           forall unmarked tokens A_a in A {
6             forall unmarked tokens B_b in B {
7               j = 0;
8               while( A_a+j == B_b+j &&
9                      unmarked(A_a+j) && unmarked(B_b+j))
10                j++;
11                if ( j == maxmatch )
12                  matches = matches xor match (a, b, j);
13                else if ( j > maxmatch) {
14                  matches = {match(a, b, j)};
15                  maxmatch = j;
16                }
17              }
18            }
19            forall match(a, b, maxmatch) in matches {
20              for  j = 0...(maxmatch - 1) {
21                mark(A_a+j);
22                mark(B_b+j);
23              }
24              tiles = tiles + match(a,b,maxmatch);
25            }
26          } while (maxmatch > MinimumMatchLength  );
27          return tiles;
28      }
```

Прва фаза алгоритма врши претраживање два стринга у циљу проналажења најдужег максималног подударања користећи три угнеждене петље. Спољна петља пролази кроз све токене стринга *A*, док се у другој петљи врши поређење фиксираног токена првог стринга са свим токенима стринга B. Уколико су почетни токени једнаки, трећа петља пореди сукцесивне токене оба стринга тако да се пронађе најдуже могуће поклапање у датој итерацији. Након завршетка прве фазе су пронађена најдужа максимална поклапања и њихова дужина.

У другој фази се ради маркирање свих токена у оквиру максималних подударања издвојених у првој фази. Притом, обележавају се само они токени у оквиру подударања која се не преклапају са већ раније маркираним подударањима. Тиме се поштује први услов да се сваки токен може упарити само једном. Алгоритам се извршава док год у првој фази може да се пронађе максимално подударање дуже од MML. С обзиром да се максимална дужина поклапања смањује у свакој итерацији, алгоритам се гарантовано завршава.

Просечна сложеност GST алгоритма је реда величине $O(n^3)$, док се у најбољем случају када су стрингови потпуно различити она може смањити на $O(n^2)$. Као што се може видети из псеудокода алгоритма, прва фаза алгоритма је знатно сложенија од друге и она доминира у оквиру сложености, с обзиром да се у оквиру ње може пронаћи највише $(m - MML)\cdot(n - MML)$ подударања. У другој фази је у најгорем случају потребно обележити све токене, што се може урадити проласком кроз стрингове у линеарном времену.

Најгори могући случај се може произвести поређењем два једнака стринга или два стринга од којих се један садржи у другом. У таквој ситуацији ће се у свакој итерацији алгоритма пронаћи по једно максимално подударање, а алгоритам ће се завршити када се дужина подударања смањи испод MML, ефективно извршавајући све три угнеждене петље у максималном броју итерација са сложеношћу $O(n^3)$. Ако се пореде два различита стринга добија се најбољи случај извршавања. У том случају алгоритам мора да пореди сваки токен из стринга A са свим токенима из стринга B, што производи сложеност реда величине $O(n^2)$.

3.5.1. Модификација коришћењем Karp-Rabin-овог алгоритма

Иако доноси побољшања у односу на друге приступе, GST алгоритам и даље има релативно високу сложеност, као што је наведено у претходном поглављу. Стога се у пракси користи оптимизација GST алгоритма базирана на идејама из *Karp-Rabin*-овог алгоритма [65] за претраживање стрингова. Ова оптимизација не може да смањи временску сложеност у најгорем случају, али се сложеност значајно смањује у просечном случају.

Karp-Rabin алгоритам заснива претраживање кратког стринга, шаблона P у дужем стрингу, тексту T, на коришћењу хеш функција. Стога алгоритам најпре врши рачунање хеш вредности свих подстрингова дужине n у стрингу T дужине m. Хеш вредност шаблона P се такође рачуна. Тиме се подстрингови у тексту који се претражује претварају у n-цифрене целе бројеве који се могу лакше и брже упоређивати у односу на поређење појединачних знакова у оквиру посматраних подстрингова. Рачунање хеш вредности подстриногова је могуће урадити у линеарној временској сложености уколико се користе тзв. *rolling hash* функције.

Главна особина *rolling* хеш функција је да се на основу познатих вредности $h(T_{t-1}T_t...T_{t+n-2})$, T_{t-1} і T_{t+n-1} лако може израчунату вреднст $h(T_tT_{t+1}...T_{t+n-1})$ коришћењем основних аритметичких операција [60]. Код *rolling* хеш функције се подстринг $T_1...T_n$ третира као n-цифрени број у основи b, а наредна хеш функција се рачуна секвенцом аритметичких операција по модулу M:

$$h(T_1...T_n) = (T_1 \cdot b^{k-1} + T_2 \cdot b^{k-2} + ... + T_{n-1} \cdot b + T_n) \bmod M \quad (3.1)$$

Да би се срачунала хеш вредност подстринга $T_2...T_{n+1}$, потребно је одузети део резултата који се односи на цифру највеће тежине, помножити резултат са b, и додати нову цифру најмање тежине:

$$h(T_2...T_{n+1}) = ((\ h(T_1...T_n) - T_1 \cdot b^{k-1}) \cdot b + T_{n+1}) \bmod M \quad (3.2)$$

Основа се најчешће бира тако да буде прост број. У неким применама се рачунање остатка по модулу M се често изоставља из формуле, уколико постоје гаранције да са веома малом вероватноћом може да дође до прекорачења.

Након израчунавања, све хеш вредности се затим упоређују са хеш вредношћу стринга P. Уколико су хеш вредности једнаке, врши се поређење појединачних карактера подстрингова, да би се потврдила појава P у T, јер се због природе хеширања као поступка може догодити да се два различита подстринга пресликају у две једнаке хеш вредности. Пракса показује да је сложеност овог алгоритма скоро линеарна. У наставку је у Табели 3.4 дат псеудокод *Karp-Rabinov*-ог написан по узору на [45].

Табела 3.4. Псеудокод *Karp-Rabinov*-ог алгоритма

```
0       Karp-Rabin(T, P)
1       m := length(T);
2       n := length(P);
3       p := hash(P);
4       t := hash(T[1..m]);
5       for s := 1 to n − m + 1 do
6         if p = t then
7           if P[1..n] = T[s..s+n-1] then
8             print "Match at: ", s
9         end_if
10        if s < m − n + 1 then
11          t := rolling_hash(t, T[s+1..s+n])
12        end_if
13      end_for
```

Коришћењем идеја из *Karp-Rabin*-овог алгоритма да се подстрингови пореде по својој хеш вредности, основни GST алгоритам је могуће модификовати тако да се сложеност у одређеним случајевима смањи на испод $O(n^2)$. Те модификације се имплементирају на следећи начин [26]:

- На почетку се рачунају хеш вредности свих подстрингова дужине MML из стрингова *A* и *B*.
- Свака хеш вредност из *A* се пореди са сваком хеш вредношћу из *B*. Ако су две хеш вредности једнаке, подударање се верификује поређењем самих токена у оквиру подстрингова. Међутим, поређење се не зауставља након дужине MML, већ се наставља док год постоји поклапање или се не дође до краја једног од стрингова.
- Ради ефикасног идентификовања подстрингова из *B* који имају исту хеш вредност као посматрани подстринг из *A* користи се хеш табела. Тиме се практично елиминише квадратни број поређења хеш вредности који би у супротном био потребан.

Описана модификација GST алгоритма коришћењем *Karp-Rabin*-овог алгоритма је у литератури позната под именом *Running Karp Rabin Greedy String Tiling* (RKR-GST) [53].

3.6. Winnowing алгоритам

Поређење два токенизована програмска кода се не мора нужно обавити алгоритмима за проналажење шаблона у стринговима. Најчешће алтернативе су базиране на поређењу њихових дигиталних отисака (енг. *fingerprints*) који се на одређени начин креирају и селектују из оригиналних кодова. Ови приступи су генерално ефикаснији у смислу перформанси извршавања и скалабилности у односу на број кодова који се пореде, али је цена која се плаћа нешто мања прецизност као последица њихове апроксимативне природе.

Једну такву алтернативу представља *Winnowing* алгоритам [60] који се успешно примењује у алату Moss. Код ове технике, програмски код се дели на подстрингове дужине *k* који се називају *k-gram*-ови, а параметар *k* бира корисник. Сваки знак у стрингу, осим последњих *k-1*, представља почетак једног *k-gram-a*. Да би се оформио дигитални отисак, сви добијени *k-gram*-ови се најпре хеширају, а затим се селектује један подскуп хешева *k-gram*-ова, као дигитални отисак. Притом, тај подскуп хешева је знатно мањи од укупног броја *k-gram*-ова. Хеширање се врши на сличан начин као код *Karp-Rabinov*-ог алгоритма

коришћењем *rolling* хеш функција. Програмски кодови се пореде на основу добијених дигиталних отисака.

Главни проблем оваквог приступа представља избор одговарајућих хешева *k-gram*-ова који ће ући у отисак за поређење. Стога [60] дефинише три главна својства која алгоритам за детекцију сличности мора да испуни:

- Неосетљивост на белине, разлику између малих и великих слова, промену имена променљивих,
- Потискивање шума које подразумева да се свако откривено подударање мора бити довољно дугачко, како се не би сматрало случајним,
- Позициону независност која подразумева да промена редоследа, уметање и уклањање садржаја из документа не утиче на детекцију сличности.

У контексту детекције сличности у програмском коду, прво својство се задовољава процесом токенизације. Друго својство се задовољава избором параметра k који треба да буде довољно велики да се избегну кратка, случајна поклапања. Овај параметар је аналоган параметру MML код GST алгоритма. Обезбеђивање позиционе независности детекције је најпроблематичније, а код *Winnowing* алгоритма је задовољено кроз начин на који се врши селекција отисака за поређење.

Алгоритам горњим својствима додаје и правило да уколико постоји подударање чија је дужина бар једнака дужини гарантованог прага t (eng. *threshold*), онда то подударање мора бити детектовано. Описане константе k и t су параметри алгоритма, за које мора да важи $k < t$. Ове константе морају бити пажљиво изабране, јер од њих зависи осетљивост система. Веће вредности обезбеђују да подударање није случајно, али исто тако доводе и до смањења осетљивости на премештања садржаја докумената, јер се не могу детектовати релокације подстрингова дужине мање од k.

Winnowing користи концепт преклапајућих прозора код селекције отисака како би обезбедио да се детектује бар део неког довољно дугачког поклапања. Прозор се дефинише као низ узастопних *k-gram*-ова дужине $w = t - k + 1$. Ако се посматра секвенца хешева $h_1 ... h_n$ и притом важи $n > t - k$, онда бар један од хешева h_i мора бити изабран да се задовољи услов да се сва подударање дужа од t морају детектовати. То онда намеће описану дужину прозора.

Ако се посматра секвенца хешева $h_1...h_n$, који представљају један документ, свака позиција $1 \leq i \leq n - w + 1$ дефинише прозор $h_1...h_{i+w-1}$. Алгоритам обезбеђује да се из сваког прозора изабере најмање по један отисак и то тако што се селектује минимална вредност у посматраном прозору. Ако постоји више минималних вредности, онда се бира она која се налази најдесније у прозору. Избор минималне вредности је наметнут чињеницом да постоји велика вероватноћа да је минимална вредност из једног прозора такође минимална вредност и у наредном прозору. Притом, прозори се међусобно преклапају.

3.7. Одређивање мере сличности

Детекција сличности у програмском коду мора да произведе неки квантитативни показатељ који говори о мери сличности два посматрана програмска кода, а који може да се изведе на основу резултата које продукују алгоритми за поређење. У том смислу, најчешће се израчунава проценат сличности који говори о томе колики део програма који се пореде је покривен подударањима.

У овом раду је усвојена методологија за израчунавање сличности из [26] која се састоји од следеће две формуле. Најпре се дефинише укупна покривеност токена подударањима, односно укупна дужина свих пронађених и верификованих подударања (енг. *tiles*):

$$covarage(tiles) = \sum_{match(a,\ b,\ length)\ in\ tiles} length \tag{3.3}$$

где су A и B стрингови који се пореде, а m и n њихове дужине респективно, као што је дефинисано у Поглављу 3.5 . Затим се проценат сличности израчунава на следећи начин:

$$sim(A, B) = (2 \cdot covarage(tiles)\ ) / (m + n) \tag{3.4}$$

Ефективно, сличност два изворна кода је пропорционална броју линија заједничког дела кода подељеног са укупним бројем линија у оба кода који се пореде.

4. Анализа система за детекцију сличности у програмском коду

Детекција плагијаризма би била знатно компликованије без помоћи и подршке коју пружају системи за детекцију сличности у програмском коду. Ови системи не само да морају да буду отпорни на различите модификације које плагијатори користе да прикрију акт плагијаризма, већ морају понудити и низ других опција које утичу на функционалност и употребљивост система, како би били прихваћени од стране корисника.

У овом поглављу ће бити извршена упоредна анализа система који су приказани у отвореној литератури. На основу те анализе ће бити дефинисане кључне карактеристике које један систем за детекцију сличности треба да поседује да би био у широј употреби за детекцију сличности у програмском коду. Затим ће бити извршена и експериментална евалуација одабраних система.

4.1. Карактеристике система

Системи за детекцију сличности у програмском коду се могу сматрати сложеним софтверским системима, с обзиром да различите карактеристике директно утичу на њихову употребљивост од стране корисника. У отвореној литератури се може наћи више студија [24, 28, 66, 67] у којима је вршена упоредна анализа ових система.

Анализе и поређења се спроводе на квантитативном нивоу кроз евалуацију перформанси у смислу прецизности, поузданости и брзине извршавања о чему ће бити више речи у делу поглавља које се бави експерименталном евалуацијом одабраних алата. Анализа у оквиру наведених студија показује да ниједан алат значајно не превазилази друге на квантитативном нивоу, па је стога потребно упоређивати их и на квалитативном нивоу.

Системи за детекцију сличности у програмском коду се пореде и на квалитативном нивоу кроз поређење функционалности које пружају кориснику. Квалитативна поређења су углавном дескриптивна, а у овом раду ће бити размотрене карактеристике дефинисане по узору на критеријуме за поређење изложене у [24, 66]:

- **Подржани програмски језици** – директно утичу на употребљивост алата и дијапазон примене. Такође, одређени алати траже да кодови за поређење буду потпуно синтаксно коректни, док други дозвољавају и поређење кодова који се не парсирају у потпуности [67].
- **Проширљивост** – уколико је алат модуларно написан, онда се лако може адаптирати да подржи неки други програмски језик простом надоградњом одговарајућег лексера и парсера који представљају *front end* система. Са корисничког становишта у академском окружењу, мало је вероватно да ће исти наставници користити различите алате уколико у оквиру курсева користе различите програмске језике, па је погодно имати могућност проширења система за другачије примене [66].
- **Коришћени алгоритми за детекцију сличности** – алгоритми за детекцију сличности у програмском коду имају директан утицај на прецизност, поузданост и брзину извршавања система.
- **Презентација резултата** – утиче у великој мери на касније стадијуме у којима се врши ручна инспекција сличних кодова ради потврђивања плагијаризма. Три кључна елемента се издвајају [66]: сумаризација резултата, приказ парова сличних кодова уређен нерастуће по проценту сличности или груписан по кластерима сличних радова и могућност упоредног прегледа парова уз означавање сличних делова кода бојом.

- **Кориснички интерфејс** – одређени број алата има графичке корисничке интерфејсе, док се други користе из командне линије, што утиче на њихово прихватање међу корисницима и једноставност употребе.
- **Приступ сервису** – одређени алати се могу преузети и извршавати локално, док су други имплементирани у виду веб сервиса.
- **Сигурност** – ова особина је битна код алата који су имплементирани у виду веб сервиса. Начин коришћења сервиса у овом смислу може утицати на нарушавање приватности информација, па се морају предузети мере заштите у односу на неовлашћен приступ репозиторијумима програмских кодова.

Поред наведених особина, корисничком искуству доприносе и следеће особине:

- **Могућност изостављања шаблонског кода** – наставници често студентима стављају на располагање одређене алгоритамске скелете, већ реализоване делове кода које треба надоградити или делове кода који су изложени у оквиру предавања. Такви делови кода се могу наћи и представљати подударање у великом броју студентских радова, али свакако не представљају плагијаризам. Изостављањем таквог кода од стране алата пре самог поређења се може смањити број лажно позитивних резултата [66].
- **Могућност изостављања малих датотека** – ова особина може бити од користи код поређења програмског кода који садржи специфичан код, као што су *Java* зрна (енг. *beans*). Датотеке које садрже овакве кодове могу да произведу велику сличност, па се може вршити њихова елиминација.
- **Поређење са историјом** – на одређеним курсевима се због саме материје дају слични домаћи задаци. У том случају је погодно поредити не само програмске задатке из текућег циклуса, већ и оне из претходних циклуса. Уколико алат имплементира неку врсту складишта и чува податке о претходним поређењима, онда може да има ову могућност.
- **Начин обрачунавања процента сличности** – неки алати обрачунавају проценат сличности за сваку предату датотеку у оквиру једног студентског рада, док други обрачунавају проценат сличности на нивоу целог рада.
- **Отвореност кода** – неки системи су отвореног кода што омогућава једноставнију проширљивост.

4.2. Поређење постојећих система

У овом поглављу је направљен преглед и поређење релевантних система за детекцију сличности у програмском коду. У преглед су уврштени како често коришћени и етаблирани системи, као што су Moss и JPlag, тако и одређени експериментални системи, попут GPLAG и SID, који заврђују пажњу због концепата које имплементирају.

4.2.1. MOSS

Measure of software similarity (Moss) је систем за детекцију сличности у програмском коду који се развија на Универзитету Стенфорд од 1997. године. Moss омогућава поређење програмског кода за преко 20 програмских језика. Имплементиран је у виду веб сервиса којем се приступа путем *Linux bash* скрипта или апликација са графичким корисничким интерфејсом који су развила трећа лица. Проширивање система је могуће, али аутори нису отворили код апликације, већ се она могу радити на захтев.

Сам алат користи робусну верзију раније описаног *Winnowing* алгоритма [60]. Алгоритам је робуснији у смислу да селектује за поређење мање дигиталних отисака у односу на верзију која се користи за поређење текстуалних докумената. Како би избегао квадратан број поређења сваке датотеке са сваком, систем креира индексну базу (инвертовани индекс) која мапира отиске у њихове локације у свим документима у којима се појављују. На основу те базе се у следећем кораку врши упаривање појединачних докумената са свим отисцима које он садржи, па самим тим и документима који садрже те отиске. Moss затим формира листу подударајућих докумената и приказује кориснику најсличније резултате у виду скупа HTML страница.

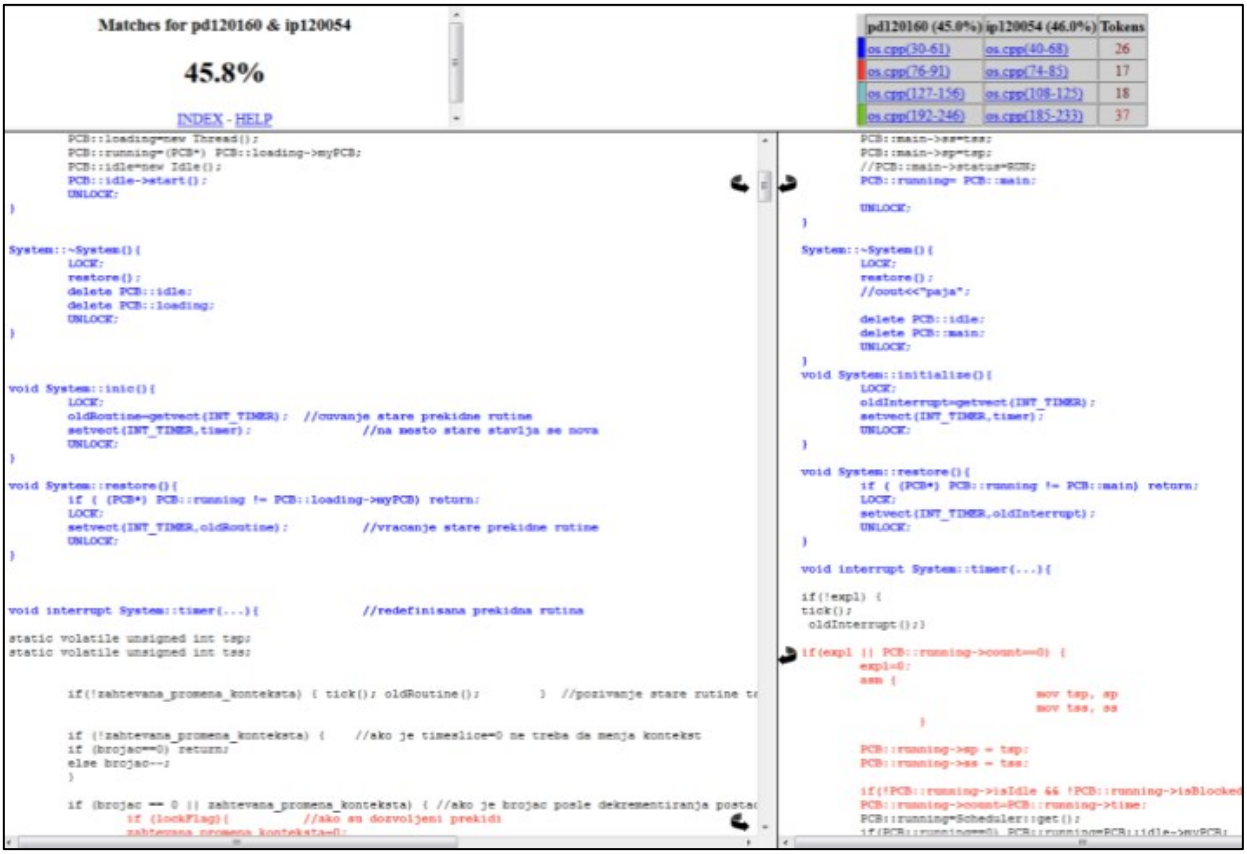
Сличност два програмска кода се дефинише на основу броја *k-gram*-ова који се поклапају, а коначни резултат укључује број токена који се подударају, број линија које се подударају и проценат преклапања програмског кода између детектованих парова датотека. Moss омогућава изостављања шаблонског кода, чиме се знатно смањује број лажно позитивних резултата.

4.2.2. JPlag

JPlag [26, 68] је систем за детекцију сличности развијен 2000. године на Универзитету у Карлсруеу, а базиран је на техникама за упаривање стрингова које су описане у претходном поглављу. Алат користи *Running Karp Rabin Greedy String Tiling* (RKR-GST) методу за

одређивање мере сличности између два програмска сегмента. Првобитно је алат био доступан само у виду веб сервиса, али је од 2015. године код постао јавно доступан, а новим корисницима је омогућено коришћење искључиво локалних инсталација. Додавање подршке за нове програмске језике је доступно кроз измену постојећих *front end* пакета.

JPlag поседује врло интуитиван кориснички интерфејс, који омогућава једноставан преглед резултата поређења који су приказани у виду хистограма, а слични радови су груписани у кластере. Сваки упоређени пар је могуће појединачно погледати, а слични делови кода ће бити обележени одговарајућом бојом, као на Слици 4.1. Кориснички интерфејс JPlag алата је у каснијим верзијама преузео и систем Moss.



Слика 4.1 Приказ сличности два задатка у систему JPlag

4.2.3. SPD

Sherlock Plagiarism Detector (SPD) је једноставан алат развијен на Универзитету у Сиднеју за проналажење сличности у текстуалним документима [69]. Алат је је базиран на генерисању дигиталног отиска и није специфично намењен за коришћење са програмским кодовима, већ се могу користити и други типови докумената, па самим тим не зависи од синтаксе било ког конкретног програмског језика. Извршава се локално на рачунару.

Дигитални отисак се формира хеширањем речи улазног текста. Корисник има контролу над бројем речи које улазе у један дигитални отисак, чиме се контролише грануларност поређења. Алат пореди сваку датотеку са сваком и као резултат враћа сличност парова датотека у процентима, без могућности инспекције појединачних датотека или делова кода. У ову анализу је SPD укључен као пример веома једноставног алата који није базиран на структурно оријентисаном поређењу, већ само на принципу креирања дигиталног отиска.

4.2.4. Sherlock

Систем за детекцију сличност у програмском коду, *Sherlock*, са Универзитета Ворвик у Великој Британији је међу првима увео интуитиван кориснички интерфејс и могућности приказа резултата у виду графа [18]. Програми који се пореде представљају чворове графа, а гране представљају релацију сличности. Притом, сличнији радови су повезани краћом линијом. Резултати детекције сличности се прослеђују неуралној мрежи која формира слику на којој се приказују резултати сличности међу програмима. Извршава се на локалном рачунару, али постоји и веб верзија која је интегрисана у оквиру BOSS система [70] за предају и аутоматско оцењивање домаћих задатака.

Алгоритам за детекцију је заснован упаривању стрингова које се врши у пет корака. Програми се пореде: у оригиналној форми, са уклоњеним белим знацима, са уклоњеним коментарима, са уклоњеним белим знацима и коментарима и токенизованом облику. Већи број корака се користи, јер су аутори сматрали да се токенизацијом губи сувише информација које се могу искористити за потврду плагијаризма. Програм се већ дуже време не развија активно.

4.2.5. SID

Software Integrity Diagnosis (SID) [44] је веб базирани систем за детекцију сличности између два програмска кода који користи метрику засновану на ентропији и комплексности Колмогорова која се користи у теорији информација. У теорији информација, комплексност Колмогорова неког објекта се дефинише као дужина најкраћег програма у одговарајућем програмском језику који производи тај објекат као резултат свог рада. Сматра се да комплексност Колмогорова није израчунљива, али постоје доста добре апроксимације засноване на компресији података која се у алату и користи.

Улазни програмски кодови је најпре токенизују, а затим се врши компресија секвенци токена. Компресија података користи модификовани *Lempel-Ziv* (LZ) алгоритам уз апроксимативно кодирање сличних секвенци на бази Левенштајнове дистанце и функције прага сличности. За разлику од стандардних алата за компресију који узимају у обзир и време извршавања и меморијске захтеве, у овом случају се тежи максималном степену компресије.

Аутори потврђују ефикасност оваквог приступа на већем броју узорака и поредећи га са алатима Moss и JPlag. Такође, као највећа предност овог система се истиче његова отпорност на уметање програмског кода. Међутим, иако ефикасан, највећи проблем овог система јесте поузданост сервиса, који често не заврши свој рад за предати материјал [71]. У тренутку писања овог рада, сервис више није био доступан на интернету.

4.2.6. GPLAG

GPLAG је алат који пореди графове зависности програма да би утврдио сличност два програмска кода, као што је описано у Поглављу 3.4.3. Аутори су утврдили пет група промена у програмском коду на које алат треба да буде отпоран и у складу са тим су га пројектовали. То су: промена форматирања, преименовање идентификатора, измена редоследа наредби, замена контролних структура и убацивање кода.

Алат анализира задати програмски код, издваја потпрограме и за сваки потпрограм креира граф зависности, чиме се ефективно ствара скуп графова зависности за појединачни програм. Два програмска кода се сада упоређују утврђивањем изоморфизма између појединачних графова зависности из тих скупова.

С обзиром на потенцијално велики број поређења и комплексност утврђивања изоморфизма два графа, GPLAG уводи одређене оптимизација како би смањио простор претраживања. Аутори наводе да само мали број графова треба поредити на изоморфизам и да велики број поређења може да се одбаци већ на високом нивоу. Стога су имплементирали статистички филтер са губитком (енг. *statistical lossy filter*) којим се значајно смањује простор поређења.

4.2.7. PTK

Као што је поменуто у Поглављу 3.4.2., постоје алати који поређење програмских кодова заснивају на коришћењу апстрактних стабала парсирања. Један такав алат је приказан у [61] и користи описани концепт за поређење програма написаних на програмском језику Java. Поред самог алата, у раду је представљена и конкретна техника поређења апстрактних синтаксних стабала помоћу функција-језгара, па је стога и сам алат у овом раду означен именом PTK (*Parse Tree Kernel*).

Функције-језгра које су коришћене за поређење су познате из области процесирања природних језика (енг. *natural language processing* - NLP), али су морале бити измењене због другачије природе програмских језика. У NLP применама, синтаксна стабла су мање дубине, па евентуалне промене у корену стабла мање утичу на резултат поређења. Такође, у NLP применама је битан поредак подстабала, док код програмских језика то нема значај и систем за поређење треба да буде отпоран на промену редоследа блокова у коду. Аутори тврде да имплементирани систем даје одличне резултате на синтетичким тестовима и приликом поређења са JPlag системом за детекцију плагијаризма и CCFinder системом за детекцију софтверских клонова у програмском коду.

4.2.8. Остали системи

У претходним поглављима су описани системи за детекцију сличности у програмском коду који су због своје зрелости, широке распрострањености или значајних концепата заслужили да подробније буду размотрени. Међутим, постоји низ система за детекцију сличности који се помињу у отвореној литератури и имали су одређено место у историји или нуде интересантне концепте који још увек нису потпуно заживели у пракси. Стога ће они бити наведени у овом поглављу.

YAP3 [25] је последња верзија YAP (*Yet Another Plagiarism detection system*) система за детекцију сличности коју је 1996. развио Мајкл Вајс, аутор RKR-GST методе за детекцију сличности у програмском коду. Алат користи оригинални RKR-GST алгоритам [54] намењен за упаривање секвенци нуклеотида. Plaggie [72] је систем за детекцију отвореног кода, настао из потребе за сигурним, локалним решењем за детекцију плагијаризма. Направљен је по узору на JPlag који се у тренутку писања алата Plaggie извршавао само у виду веб сервиса. SIM (*software SIMilarity tester*) [73] је алат развијан од 1989. како за детекцију сличности у

програмском коду, тако и за детекцију софтверских клонова. Алат најпре врши токенизацију програмског кода, а затим проналази најдуже заједничке непреклапајуће подстрингове.

Један од главних проблема система за детекцију сличности је њихова скалабилност у односу на број кодова које треба упоредити. FPDS (*Fast plagiarism detection system*) [59] користи технику поређења названу параметризовано упаривање. Он за претраживање датотека са сличним програмским кодом користи индексну структуру у облику низа суфикса чиме смањује број поређења међу датотекама уз незнатан губитак прецизности. Други приступ за смањење броја поређења је примењен код система EPlag [74] који користи двостепену обраду. У првом кораку се кодови грубо пореде на основу метрика базираних на бројању атрибута, а затим се за други корак у коме се користи RKR-GST метода за поређење бира само одређени број кодова.

SCSDS (*Source Code Similarity Detector System*) је софтверски систем развијен са циљем да исправи одређене недостатке старијих система за детекцију сличности у програмском коду попут JPlag и Moss. Систем користи модификовани, проширени скуп токена како би повећао отпорност на модификације програмског кода. За одређивање сличности се користе и RKR-GST и *Winnowing* алгоритам, а проценат сличности програмског кода се рачуна на основу посебне дефинисане формуле која у обзир узима проценте сличности које дају алгоритми за детекцију са различитим тежинама. Алат је такође дизајниран да буде лако проширив и да подржава искључивање шаблонског кода.

Релативно мали број система за детекцију сличности у програмском коду поред виших програмских језика подржава и поређење асемблерских кодова. Један такав алат је pk2 (*pecados*, од шпанске речи која означава грехе). Он омогућава поређење кодова на програмском језику C и *Motorola* асемблерском језику коришћењем токенизације и скупа посебно дефинисаних метрика.

4.2.9. Класификација система

У овом поглављу је направљен упоредни преглед система за детекцију сличности описаних у поглављима 4.2.1–4.2.8. Преглед је дат табеларно, а као главни критеријуми поређења су издвојени подржани програмски језици, алгоритам за детекцију сличности у програмском коду, кориснички интерфејс, начин презентације и форма резултата и сигурност. Резултат поређења је дат у Табели 4.1.

За поређење алата су коришћени подаци из отворене литературе, а приказана табела представља проширење табеле приказане у ауторовом раду [28]. Поред имена сваког алата су наведене референце из којих су прикупљени подаци, као и линк ка веб страници система уколико је доступна. Тамо где подаци нису били доступни или функционалност није подржана, стављен је знак „/“.

Као што се види из табеле, највећи број система за детекцију подржава C, C++ и Java програмске језике. Алати Moss, JPlag и SIM се издвајају по броју подржаних програмских језика. Већина алата су имплементирани као независне апликације које се извршавају локално, али неки имају могућност да се извршавају као веб сервиси. Алати који су имплементирани као веб сервиси по правилу нуде неки вид аутентификације да би се избегле злоупотребе.

Табела 4.1. Класификација система за детекцију сличности у програмском коду

Систем	Програмски језик	Алгоритам за детекцију	Кориснички интерфејс	Презентација резултата	Сигурност
Moss [60, 75]	C, C++, C#, Java, Python, VB, Pascal, ...	Winnowing	Командна линија, независни веб клијенти	HTML са процентима сличности, обележене линије кода	Кориснички број, e-mail
JPlag [26, 76]	C, C++, C#, Java	RKR-GST	Веб клијент	HTML са процентима сличности, обележене линије кода	Корисничко име, лозинка
SPD [69]	/	Бројање атрибута	Командна линија	Текст, проценти сличности	/
Sherlock [18, 77]	C++, Java	Токенизација	Java клијент	Граф сличности, проценат сличности	/
SID [44]	C, C++, Java	Комплексност Колмогорова	Веб клијент	HTML са процентима сличности, обележене линије кода	Корисничко име, лозинка
GPLAG [62]	C, C++, Java	PDG	/	Упоредни приказ са процентом сличности	/
PTK [61]	Java	Језгро стабла парсирања	/	Упоредни приказ са процентом сличности	/
YAP3 [25]	C, Pascal, Lisp	RKR-GST	Командна линија	Tekst, procenti sličnosti	/
Plaggie [72, 78]	Java	Токенизација, GST	Командна линија, конфигурациона датотека	Текст, HTML са процентима сличности	/
SIM [73, 79]	C, Java, Pascal, Modula-2, Miranda, Lisp	Токенизација	Командна линија	Текст, број поклапања токена	/
FPDS [59]	Java	Параметризовано упаривање	Java клијент	Граф сличности, проценат сличности	/
EPlag [74]	Java	Бројање атрибута, RKR-GST	Веб клијент	Упоредни приказ са процентом сличности	Корисничко име, лозинка
SCSDS [24]	Java, C++, C#	RKR-GST + Winnowing	Командна линија, веб клијент	Упоредни приказ са процентом сличности	/
pk2 [32]	C, Motorola ASM	Токенизација	Командна линија	Текст датотека са процентима сличности	/

Презентација резултата је најчешћа кроз текстуални или HTML излаз унутар којих се за сваки пар радова наводи мера сличности. Главна предност HTML излаза је што омогућава једноставну навигацију кроз сличне радове и њихов упоредни приказ уз означавање сличних линија, као што је то приказано за алат JPlag на Слици 4.1. Неки алати овакав упоредни приказ остварују кроз одговарајуће прозоре унутар графичког корисничког интерфејса. Два алата нуде рудиментаран приказ резултата у облику графа.

Због својих особина, системи JPlag, Moss и SPD су коришћени за експерименталну евалуацију система за детекцију сличности. JPlag и Moss су широко прихваћени, бесплатни и поуздани алати за детекцију сличности са подршком за велики број програмских језика. Корисницима нуде велики број опција за поређење. Са друге стране, SPD је веома једноставан алат базиран на хеширању и креирању дигиталног отиска. У овом истраживању је, пре свега, коришћен као репер за поређење са системима који користе напредније приступе за детекцију сличности. Студенти који плагирају програмски код користе релативно грубе методе прикривања, пре свега због времена потребног да би се извршиле сложеније модификације, па је понекад погодно користити једноставне алате да би се брзо добиле информације о постојању плагијаризма.

4.3. Интеграција са другим системима

Детекција плагијаризма у програмском коду углавном намеће наставном особљу додатан посао. Тај додатан посао може да захтева значајну количину времена, јер укључује пуно физичког посла због сакупљања, предобраде и поређења студентских радова помоћу одабраног алата, а касније и ручне инспекције добијених резултата.

Наставници користе различите начине да сакупе студентске радове. Радови се сакупљају путем електронске поште, предајом путем различитих физичких медијума (оптички дискови, USB меморије), снимање у наменске директоријуме на за то предвиђене лабораторијске рачунаре, али све чешће и путем наменских система за предају и обраду домаћих задатака. Након прикупљања радова, они морају проћи извесну предобраду, како би се евентуално уклониле сувишне датотеке и унифицирали по формату који захтевају алати за детекцију сличности. Тек након ове фазе може се извршити обрада помоћу одговарајућег алата.

Из претходно наведених разлога наставници све више прибегавају аутоматизацији поступка предаје, складиштења и обраде програмских кодова које студенти предају као своје радове. Аутоматизација овог поступка се спроводи кроз наменске системе за предају и оцењивање домаћих задатака или кроз платформе за електронско учење путем којих се комплетна администрација курса врши на једном месту.

4.3.1. Системи за електронско учење

Системи за електронско учење и оцењивање су са развојем рачунарских технологија и Интернета заузели значајно место у образовном процесу. Велики број ових система se користи у извођењу наставе и омогућава да студенти стекну практично искуство из одређеног дела градива. Са друге стране, одређен број система се користи као подршка процесу извођења наставе, као што су платформе за електронско учење (*Learning Management Systems*).

Платформе за електронско учење омогућавају спровођење и администрацију читавог курса. Оне омогућавају дистрибуцију наставних материјала, обавештавање и комуникацију са студентима, проверу знања и вођење евиденције о активностима и резултатима студената. Познатије платформе за електронско учење су комерцијална платформа Blackboard [80] и платформа отвореног кода Moodle [81]. У отвореној литератури постоји више студија које показују позитиван утицај употребе ових платформи у наставном процесу [82], а Moodle се издваја као једна од најефективнијих платформи [83], што показују и резултати које је на једном од својих курсева остварио и аутор овог рада [84].

С обзиром на могућност свеобухватног управљања наставним процесом и проширљивост Moodle платформе за електронско учење, неколико пројеката је имало за циљ интеграцију алата за детекцију сличности у програмском коду са овом платформом. Оваква интеграција најчешће подразумева коришћење постојећих решења у оквиру Moodle платформе за управљање корисницима, задавање задатака, предају и складиштење радова и приказ и складиштење резултата. За поређење радова се најчешће користе системи JPlag и Moss.

У отвореној литератури је идентификовано неколико проширења за Moodle која омогућавају поређење предатих програмских кодова. *Programming Code Plagiarism Plugin* [85] интегрише алате JPlag и Moss у Moodle, омогућава аутоматско поређење предатих студентских радова након задатог рока. Резултати се приказују интерактивно, уз могућност

филтрирања и груписања резултата по сличности. Омогућено је и давање коментара студентима како би се подигла њихова свест о томе шта представља плагијаризам, а шта не. Други интересантан додатак за Moodle је представљен у оквиру [86] и представља комплетно решење за предају, складиштење, оцењивање и поређење програмских кодова. Додатак користи GCC преводилац и Moss систем за детекцију сличности. Ефективност оваквог приступа је потврђена кроз курс који се бави структурама података, где су аутори приметили значајно смањење времена потребног за оцењивање домаћих задатака уз смањење броја плагираних радова.

4.3.2. Системи за предају и обраду студентских радова

Поред општенаменских платформи за електронско учење које могу да се користе за предају и аутоматско оцењивање програмских кодова, развијен је и већи број специјализованих система за ову намену [31, 32, 70, 87, 88]. Акценат ових система је пре свега на подршци за аутоматизовано оцењивање студентских радова, али већи број ових система омогућава и детекцију сличности као додатну опцију.

Један од првих система за предају и оцењивање студентских радова путем Интернета који интегрише и детекцију сличности је BOSS (енг. *BOSS Online Submission System*) [70] са универзитета Ворвик који користи алат Sherlock са истог универзитета. Сличан систем за интерне потребе је развијен на Техничком универзитету у Мадриду [32] и користи *pk2* алат за детекцију сличности. У раду [31] је такође приказан један систем за електронску предају студентских радова који пружа рудиментарну подршку за поређење и приказ сличних радова коришћењем претраге текста и који је успешно примењен у пракси.

Аутор овог рада је учествовао у изради једног веб система („Веб домаћи“) за предају и аутоматско поређење предатих домаћих задатака коришћењем алата Moss чији су детаљи изложени у [88], а кратак опис ће бити изнет у наставку. Апликација је написана у PHP-у и обезбеђује интерфејс ка Moss систему за детекцију сличности. Наставници могу да управљају својим курсевима и задају домаће задатке, док је студентима омогућена једноставна предаја и преглед домаћих задатака. Циљ апликације је да олакша откривање плагијаризма приликом израде студентских радова, уз минимизацију напора наставника који се тиче прикупљања, складиштења, обраде и поређења предатих радова. Наставник тада на једноставан начин добија информацију о сличним радовима и може више времена да посвети ручној провери оних радова за које постоји оправдана сумња да су плагијати.

Постоје три врсте корисника система – студенти, наставници и администратор система. Студент и наставник су регистровани корисници који након пријаве могу да врше одређене промене у систему. Администратор је корисник који је јединствен за систем и има посебне привилегије за управљање системом. Пре првог коришћења система нови студенти треба да се региструју. На тај начин се омогућава поуздана аутентификација корисника, а коришћен је LDAP сервис како би омогућила интеграција са постојећим студентским сервисима на Електротехничком факултету.

Да би предао домаћи задатак, студент мора да се пријави на курс. Притом се врши провера да ли је курс у току и да ли се студент налази на списку пријављених на курс који приликом креирања курса доставља наставник. Након пријаве на курс, студент има могућност да предаје домаће задатке на том курсу до одређеног рока. Уоквиру сваког домаћег задатка студенту су видљиви материјали који се могу користи приликом његове израде. Домаћи задатак се може предати као појединачна датотека или у облику *zip* архиве путем форме на Слици 4.2.

Слика 4.2. Форма за предају домаћег задатка у оквиру система „Веб домаћи“

Наставник може да дода нови курс у систем, а након тога и домаће задатаке и материјале за тај курс. Податке о курсу и домаћем задатку могуће је променити у сваком тренутку. При постављању новог курса, неопходно је да наставник постави списак студената који могу да се пријаве на курс, што се користи код регистрације корисника. Наставник дефинише рок за предају домаћег задатка након кога студенти више немају могућност

предавања радова. Наставник може ручно да продужи рок за предају. Такође, наставник може да брише курсеве и преузима архиве домаћих задатака.

Наставник у сваком тренутку може да покрене и изврши проверу предатих задатака путем форме на Слици 4.3, без обзира да ли је рок за предају истекао. Након покретања провере, приказује се табела са свим сличним паровима радова који су пронађени. При томе се приказује број спорних линија кода, као и проценат који те линије чине у задатку. Избором било ког сличног пара радова, отворе се страница за детаљан упоредни преглед докумената, коју генерише алат Moss. Наставник даље ручном инспекцијом кода доноси закључак да ли су радови плагирани или не.

Администратор представља наставника са посебним додатним функционалностима које укључују додавање новог наставника у систем или на постојећи курс, као и преглед свих корисника система. Такође, администратор може да мења све податке у систему.

Слика 4.3. Форма за проверу домаћих задатака у оквиру система „Веб домаћи“

4.4. Експериментална евалуација одабраних система

У овом поглављу је представљена експериментална евалуација три одабрана система за детекцију сличности у програмском коду – JPlag, Moss и SPD која је као студија случаја изложена у оквиру ауторовог рада [28]. Експериментална евалуација система је извршена коришћењем два скупа програмских кодова. Над првим скупом програмских кодова су у контролисаним условима вршене измене, тако да он представља симулиран, вештачки плагијаризам. Други скуп програмских кодова којим су алати евалуирани представља реална студентска решења са неколико предмета на Електротехничком факултету.

Стога су најпре описани детаљи предмета из којих су преузети програмски кодови коришћени у експериментима и мотивација за употребу система за детекцију сличности у реалним условима из које је и произашла идеја за експерименталном евалуацијом различитих система. Затим је описана методологија евалуације. Потом су изложени детаљи експеримената у вези са симулираним плагијаризмом, а на крају са студентским кодовима из праксе.

4.4.1. Опис предмета и праксе испитивања

За експерименталну евалуацију система за детекцију сличности у програмском коду су коришћени програмски кодови са три предмета на Електротехничком факултету: Практикум из програмирања 1 (PP1), Практикум из програмирања 2 (PP2) и Оперативни системи 1 (OS1). Преглед предмета је приказан у Табели 4.2. PP1 и PP2 су практични предмети са прве године основних студија, који допуњују предмете Програмирање 1 и Програмирање 2 који покривају знатно шире области и имају класичне завршне испите [3, 89]. PP1 покрива асемблерски језик (ASM) и програмски језик Pascal, док PP2 покрива програмирање у језику C. У потпуности се спроводе кроз лабораторијске рад и пет програмских задатака, задатих редом од лакших ка тежим, а који чине 70% коначне оцене. Сваки задатак покрива неку битну програмерску тему, попут манипулације низовима и стринговима, улазно-излазних операција и рада са датотекама, сортирања података, структура података (нпр. уланчане листе), итд. Због велике фреквенције програмских задатака који се задају на недељној бази у другој половини семестра, неки студенти прокушавају да плагирају своја решења.

Табела 4.2. Преглед предмета

Предмет	**PP1**	**PP2**	**OS1**
Семестар	I	II	IV
Програмски језик	ASM, Pascal	C	C++
Број задатака	5	5	1
Величина групе студената	150-200	150-300	200-300
Просечан број линија кода у решењу	50-200	100-300	700-1000
Процентуално учешће програмских задатака у финалној оцени	70%	70%	30%
ECTS бодови	2	2	6

Предмет Оперативни системи 1 се предаје у четвртом семестру на рачунарским одсецима и покрива основне принципе функционисања и имплементације оперативних система. Суштински део курса је већи програмски задатак (пројекат) који покрива основне концепте који се излажу на предмету. У основи, студенти треба да напишу језгро једноставног оперативног система. Овај пројекат је први захтевнији и тежи задатак који студенти треба да ураде током својих студија. Из тог разлога је он честа мета плагијаризма.

Пројекти и домаћи задаци се обично бране у рачунарским лабораторијама пред асистентима и демонстраторима (испитивачима). Испитивачи треба да утврде да ли су студенти заиста решили задатак самостално и да ли могу да изврше захтеване измене у коду. На основу тестова и прегледа кода, испитивачи понекад примете значајне сличности у решењима различитих студената. Међутим, дешава се да неколико испитивача ради са студентима истовремено. У таквим случајевима је очигледно потребан систематичнији, аутоматизовани приступ да би се открила слична програмска решења. Стога се на предметима дужи низ година врши провера предатих домаћих задатака коришћењем система Moss, а студентима коригују освојени поени, уколико се утврди плагијаризам. Против несавесних студената се покрећу и дисциплинске пријаве пред Дисциплинском комисијом Факултета.

4.4.2. Методологија евалуације

Да би се упоредила ефикасност алата за детекцију плагијаризма у програмском коду спроведена су два типа експеримената. Први тип експеримената је заснован на симулираном плагијаризму. Узета су три различита програмска кода са описаних предмета и вршене измене над њима у покушају да се плагирају. Програмски кодови су писани у једноставном асемблерском језику за школски рачунар *picoComputer* (PP1), програмском језику C (PP2) и програмском језику C++ (OS1).

У асемблерском коду (PP1 код) од студената је захтевано да напишу програм који уклања све елементе низа чија бинарна репрезентација садржи више бинарних нула него бинарних јединица. Решење задатка покрива теме попут манипулације низовима, петље и потпрограма у асемблерском програмирању. Почетна верзија састоји се од око 50 линија кода.

Други пример програма састојао се од 200 линија кода писаног на програмском језику C (PP2 код). Код представља решење последњег (петог) задатка на предмету. Задатак је био да се напише програм који конвертује задати *DivX* титл фајл из једног формата у други са опционом корекцијом времена приказивања титлова. Укључене су напредније теме из програмирања на програмском језику C, попут манипулације датотекама, парсирања текстуалног формата, двоструко уланчаних листи, функционалне декомпозиције проблема, условног превођења, итд.

Последњи пример је узет са предмета OS1 и чини га програмски код написан на програмском језику C++ (OS1 код). Узели смо једно решење програмског пројекта који покрива теме попут управљања нитима, промене контекста, синхронизације, итд. Циљ је да се имплементира једноставан, али функционалан оперативни систем у C++ програмском језику са задатим интерфејсом. Решење обично садржи око 1000 линија програмског кода.

Описани програмски кодови су коришћени као основа, а затим су над њима вршене различите измене, искључујући писање кода од почетка. Као резултат измена су направљене четири нове верзије кода, после 1, 2, 4 и 8 сати рада. Свака следећа верзија кода је настала додавањем нових измене на претходну верзију. На овај начин је покушана симулација реалне ситуације у којој студент на неки начин прибави комплетно, тестирано решење програмског задатка и онда покуша да прикрије плагијат пре предаје задатка вршењем измена. У зависности од расположивог времена, студент може да измени код до одређене мере.

Све модификације које су вршене над програмским кодовима су забележене и класификоване у складу са поделом дефинисаном у Поглављу 2.3.3. Све модификације су груписане као лексичке или структуралне измене. Лексичке измене се могу једноставно извршити коришћењем опције „*Find & Replace*“ у стандардним едиторима текста (као што су измене L4, L6, L7) или коришћењем алата за форматирање изворног програмског кода у специјализованим едиторима (измене L1, L2). Структуралне измене захтевају напредније програмерске вештине (измене S1-S3), познавање процедуралног (измена S5) или објектно-оријентисаног програмирања (измене S7-S8), па самим тим је потребно уложити знатно више времена у њихово спровођење.

После експеримената са симулираним плагијатима, вршено је поређење JPlag, Moss и SPD алата уз помоћ скупа студентских задатака из праксе. Коришћена су решења која су студенти предавали током школских година 2012/2013. и 2013/2014. на сва три предмета. Најпре су приказани број потврђених случајева плагијаризма и укупан број задатака коришћених у анализи. Потврђени случајеви плагијаризма су изведени из резултата Moss алата након ручне провере.

Затим су алати поређени према њиховим ранг листама сличности. Свака листа састоји се од парова решења чија је сличност изнад одређеног прага сличности за сваки алат. Листе су сортиране у опадајућем редоследу по процентима сличности. Упоређено је највиших 20% парова из наведених листа, а резултати су приказани графички..

4.4.3. Симулирани плагијаризам

У овом експерименту симулирано је понашање студената и покушано плагирање постојећег решења у ограниченом временском року. Као што је претходно описано, за сва три задатка су креиране четири различите верзије, након 1, 2, 4 и 8 сати рада. Извршене измене су резимиране у Табели 4.3. С обзиром да је измене у кодовима вршило стручно лице, може се сматрати да начињене измене представљају горњу границу количине посла која се може извршити за задато време, а да је просечан студент у стању да уради мање иземена.

Табела 4.3. показује да се у кратком времену (1 сат) може извршити већина лексичких измена, а само неке структуралне измене. Да би се задржала оригинална функционалност кода, свака измена је захтевала поновно превођење и тестирање кода. ASM код је једноставан (око 50 линија), па већина измена може да буде урађена за релативно кратко време. Чак и сложеније реструктурирање кода може да се заврши за два сата.

Међутим, C код је захтевао један сат рада само за вршење лексичких измена и промену редоследа метода. Да би се извршиле ове измене, коришћена је „*Find & Replace*“ функционалност у едитора програмског кода и регуларни изрази за уклањање свих коментара. Већина структурних измена урађена је током другог сата, попут измене контролних структура, замена позива функције телом функције, рефакторисања метода и сл. У последња четири сата, извршене су неке напредније измене попут употребе алтернативних библиотечких функција, реструктурирање улазних и излазних фаза рада програма, итд. Напредније измене захтевале су више времена и честа превођења модификованог кода, а генерално су биле подложније грешкама, како синтаксним, тако и семантичким.

Табела 4.3. Извршене измене у коду током времена (L – лексичке измене, S – структуралне измене)

Задатак	**ASM**				**C**				**C++**			
Број линија кода у полазном решењу	**50**				**212**				**999**			
Измене / Време (сати)	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**	**1**	**2**	**4**	**8**
промена форматирања у програмском коду (L1)	+				+				+			
додавање, измена или брисање коментара (L2)	+				+				+			
измена или другачије форматирање излаза програма (L3)					+							
преименовање идентификатора (L4)	+				+	+			+			
подела или спајање декларација променљивих (L5)	+				+	+			+			
додавање, измена или брисање модификатора (L6)									+			
промена константних вредности и знаковних низова (L7)	+			+	+				+			
промена редоследа и измена оператора и операнада (S1)		+								+		
промена редоследа наредби у оквиру блока кода (S2)		+				+	+	+	+	+		
промена редоследа блокова кода (S3)	+			+	+	+				+		
додавање редундантних наредби или променљивих (S4)		+										+
измена или замена контролних структура еквивалентним (S5)			+			+	+				+	
измена типова и структура података (S6)						+						
рефакторисање функције и замена позива функције телом (S7)		+				+		+		+		
додавање редундантног кода (S8)			+								+	
додавање привремених променљивих и подизраза (S9)										+		
структурални редизајн програмског кода (S10)		+				+	+					+
промена досега идентификатора (S11)												+
коришћење алтернативних библиотечких функција (S12)							+					
превођење макроа у функцију и обратно (S13)								+				+
коришћење еквивалентних наредби (S14)		+					+	+				
промена декларације (потписа) функције (S15)						+						
Резиме измена	**6**	**12**	**14**	**16**	**7**	**16**	**21**	**25**	**6**	**11**	**13**	**17**
Број линија кода у коначном решењу	**50**	**45**	**54**	**51**	**239**	**225**	**222**	**224**	**1245**	**1246**	**1265**	**1260**

Модификације у C++ коду у току првог сата укључују лексичке измене и промену редоследа атрибута и метода у декларацијама класа. Само измене L1 і L2 су извршене екстензивно у целом коду. Измене L1 су разлог значајног повећања дужине кода у измењеним верзијама. Остале лексичке измене су рађене спорадично, јер не могу бити извршене за кратко време у коду ове дужине. На пример, доследна промена имена свим променљивама може трајати неколико сати. Структурне измене су рађене наредних седам сати. Све структурне измене су извршене без коришћења икаквог знања о систему који се развија. После осам сати рада, преостале су многе додатне могућности за даље измене.

Добијена решења су упоређена коришћењем алата SPD, Moss и JPlag. Резултати су приказани у Табели 4.4. JPlag нема подршку за асемблерски језик, па је програмски код поређен као текст. Исти приступ је покушан са Moss алатом, али резултати нису показали никакву сличност између кодова. Како алат Moss подржава MIPS и 8086 ассемблер, али не школски псеудо-асемблер *picoComputer*, код је поређен као MIPS асемблерски језик на који *picoComputer* асемблер донекле подсећа. Ипак, ни то није побољшало резултате поређења. Као што је приказано у Табели 4.4., ниједан од алата није открио значајну сличност, чак ни са верзијом кода креираном после једног сата. Ово је разумљиво, с обзиром да су кодови поређени као обичан текст (SPD, JPlag) или асемблерски језик сличан циљаном (али не идентичан), па су просте лексичке измене биле довољне да сакрију плагијаризам.

Са C кодом су резултати другачији. Пошто су током првог сата рада извршене само лексичке измене, Moss и JPlag су показали високу сличност добијеног кода са полазном верзијом. Сличност знатно опада са увођењем структуралних измена. Посебно се ефикасним показала промена редоследа исказа у блоковима кода, јер је познато да овај тип модификација може да збуни алгоритме за упаривање [24]. Код верзије добијене после осам сати, примећени су само ниски проценти сличности. Пошто су PP1 и PP2 програмски задаци намењени решавању у року од 3-8 сати, ово је разумљиво, јер потпуно ново решење може бити урађено за то време. SPD алат је у потпуности подбацио у овом случају, јер су лексичке измене извршене током првог сата потпуно промениле дигитални потпис кода.

Резултати детекције сличности за C++ код показују константан степен сличности код поређења SPD алатом, јер је код довољно велики да задржи неке карактеристике почетног кода, чак и после лексичких и струкурних иземена. Moss и JPlag су лексичке измене у потпуности открили као плагијаризам. Сличност верзије добијене после два сата са почетном опада знатно код поређења Moss алатом. Чини се да промена редоследа тела метода може да омете детекцију сличности код одређених метода краће дужине. Те неоткривене методе нису структурно измењене. Верзија добијена после четири сата садржи измене контролних структура које не могу бити откривене овим алатима и те измене највише доприносе паду сличности са основном верзијом. Сличност коначне верзије са основном је довољно ниска да плагијаризам можда не буде детектован ни од стране испитивача.

Табела 4.4. Сличност измењених верзија са полазном верзијом кода

Задатак	ASM			C			C++		
Време (сати) / Алат	SPD	Moss	JPlag	SPD	Moss	JPlag	SPD	Moss	JPlag
1	5%	3%	6.3%	21%	80%	78.4%	33%	94%	97%
2	4%	7%	3.9%	18%	29%	37.4%	32%	58%	81%
4	5%	10%	3.6%	18%	19%	32.9%	29%	40%	53%
8	4%	11%	3.7%	17%	12%	15.8%	26%	26%	32%

4.4.4. Студентски кодови – примери из праксе

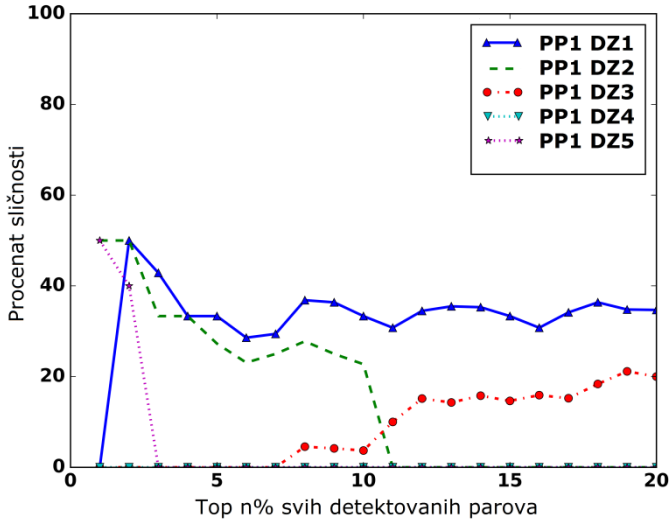
У овом поглављу су представљени резултати поређења JPlag, Moss и SPD алата на скупу стварних студентских задатака. За илустрацију распрострањености плагијаризма, Табела 4.5. приказује резултате детекције сличности на сва три предмета. Генерално, значајне сличности које могу бити сматране плагијаризмом детектоване су у 5-10% свих студентских решења, што је сагласно са претходним истраживањима аутора [28, 90].

Табела 4.5. Резултати детекције плагијаризма (потврђени случајеви / укупан број задатака)

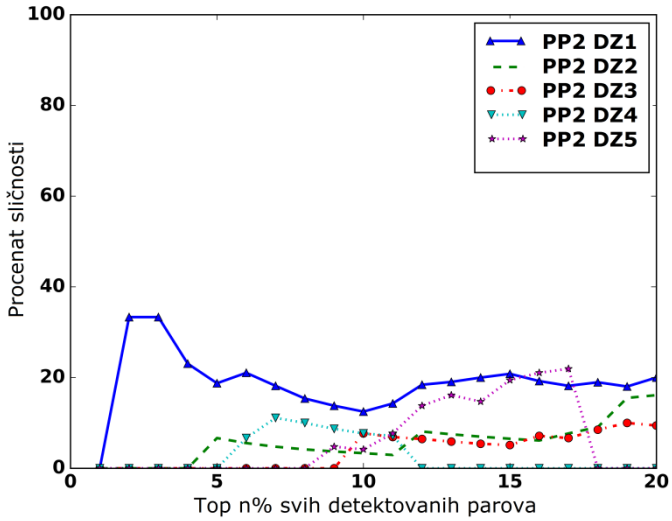
Предмет	PP1					PP2					OS1
Школска година / Домаћи задатак	**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**	**1**
2012/2013	9 / 110	0 / 94	8 / 117	8 / 85	6 / 100	3 / 140	2 / 130	6 / 101	8 / 104	4 / 101	11 / 190
2013/2014	2 / 135	12 / 162	3 / 159	6 / 129	6 / 154	6 / 185	4 / 180	10 / 164	0 / 157	10 / 142	6 / 178

На Слици 4.4. и Слици 4.5, упоређени су алати према њиховим ранг листама сличности. За сваки алат је узета његова ранг листа парова сличних задатака, а затим је посматрана сагласност ове три листе за првих *n* процената сличних парова. Хоризонтална оса на сликама које следе приказује проценат парова са врха листе који је узет у разматрање, док вертикална оса приказује сагласност резултата за сва три коришћена алата или парове алата, што је наглашено на одговарајућим сликама. Ова анализа је примењена како би се одмерио утицај коришћеног приступа за детекцију сличности у програмском коду на крајњи резултат и установиле разлике између тестираних алата.

За PP1 и PP2 предмете, резултати су конзистентни за све задатке, јер алати показују слично понашање у око 20-40% случајева. Нижи степен слагања сва три алата је из разлога неефикасности SPD алата при поређењу кратких кодова. Они се знатно лакше мењају у односу на дуже програмске кодове у краћем времену, па је мањи број линија идентичан оригиналном решењу.



(а) Предмет PP1

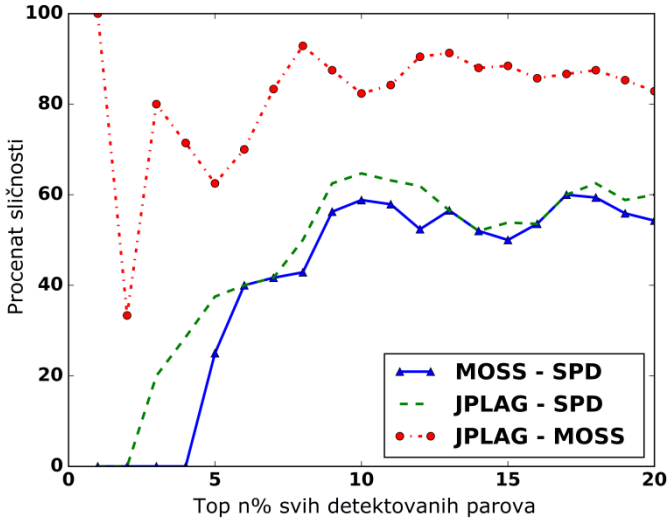


(б) Предмет PP2

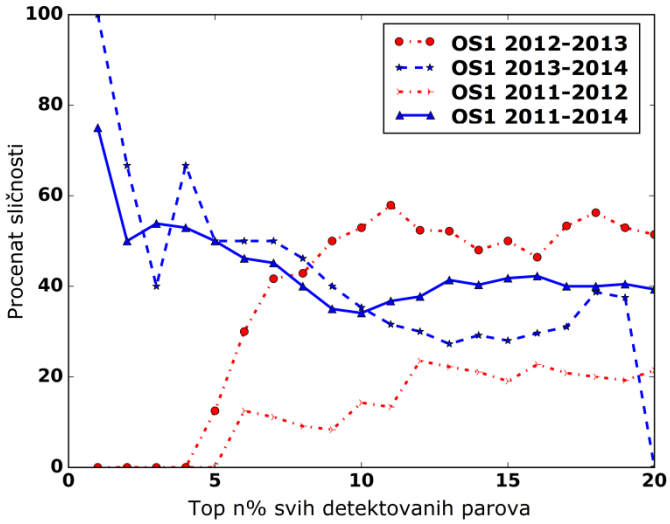
Слика 4.4. Поређење највиших 20% сличних парова за сва три алата за различите домаће задатке

На предмету OS1 резултати су мало бољи јер се алати генерално слажу у више од 50% случајева (Слика 4.5а). Програмски задатак из OS1 има обиман код, па је већи број линија остао идентичан у плагираним решењима. Додатно, OS1 код садржи део са интерфејсом (костур код) који је заједнички за све задатке, што повишава проценат сличности код SPD алата. Moss и JPlag се слажу у више од 80% случајева за 20% најбоље рангираних резултата поређења, што потврђује и [66] за Java програме.

Очигледно је да два алата могу да дају добро усмерење у испитивању плагијаризма. Слика 4.5б. приказује поређење за 447 предатих решења од 2011. до 2014. са сва три алата. Иако су програмски задаци имали неких разлика у спецификацији, срж задатка и интерфејсни део су били слични, па су студенти могли да користе решења задатака из предходних година, измене их и предају. Оцене сличности остају високе и алати су могли да детектују плагијате.



(а) парови алата за школску годину 2012/2013



(б) сва три алата за све задатке

Слика 4.5. OS1 предмет – поређење највиших 20% сличних радова

4.4.5. Дискусија добијених резултата

Резултати спроведених експеримената показују да су JPlag и Moss алати далеко супериорнији по својим карактеристикама и резултатима у односу на алате базиране на једноставнијим метрикама, попут SPD. Такође, и у симулираним и у реалним условима, оба алата показују веома сличне карактеристике. То је конзистентно са резултатима поређења која су вршена за Java програме [66], као и приликом поређења нових алата са JPlag и Moss [24, 44, 59].

Експерименти са симулираним плагијаризмом показују да само мањи број структурних измена значајно утиче на смањење сличности између кодова који се пореде. Те измене се пре свега односе на измену контролних структура, уметање појединачних наредби и промена редоследа наредби унутар блока кода. Ове измене нешто више утичу на смањење детектоване сличности код алата Moss, него код алата JPlag. JPlag неће детектовати ове измене само уколико се раде над блоковима дужине мање од MML, што је разматрано у Поглављу 3.5. приликом дискусије о GST алгоритму који алат користи. Притом, аутори система JPlag наводе још низ примера појединачних измена [26] које могу да збуне алат и смање проценат сличности посматраних програмских кодова. Међутим, аутори исто тако тврде да се наведене измене морају применити систематично у комплетном плагираном коду, да би алат значајно смањио проценат сличности.

Приликом тестирања са реалним скупом студентских радова, примећено је да студенти углавном користе основне технике за прикривање плагијата, попут лексичких измена и промене редоследа блокова кода. Само је неколико студената користило напредније технике, попут промене редоследа операнада или измене контролних структура. На PP1 и PP2 предметима већина плагијата добијена је копирањем делова радова или изменом комплетних радова колега. JPlag и Moss су веома погодни за ове случајеве, јер смањују утицај лексичких измена препроцесирањем токенизацијом. SPD дигитални потписи се лако мењају, па он нема могућност да открије те случајеве. Потпуне копије су открила сва три алата, јер су оне имале само минималне измене.

4.5. Недостаци постојећих решења

Као што је објашњено у Поглављу 3.1. детекција плагијаризма у програмском коду је вишестепени процес, а детекција сличности у програмском коду је само једна од фаза која се мора спровести на путу до коначне потврде плагијаризма у конкретним кодовима. Највећи

недостатак постојећих решења се, пре свега, огледа у томе што су она углавном концентрисана на фазу детекцију сличности у програмском коду и њена побољшања. Остали аспекти који се односе на анализу добијених резултата детекције сличности и укупно корисничко искуство су уобичајено у другом плану.

Системи за детекцију сличности у програмском коду најчешће не поседују никакву даљу логику која омогућава анализу добијених резултата. Заправо, већина система генерише резултате поређења у облику текстуалних или HTML датотека и тиме завршава свој рад. Приказ резултата ретко омогућава груписање и кластеризацију сличних радова, већ је најчешће омогућен само преглед парова радова. Само два анализирана алата омогућавају рудиментиран приказ резултата у виду графа који би омогућио детаљнији увид у колаборације појединачних студената и група студената. Притом, ниједан од њих не дозвољава контролу начина приказа. Ниједан анализирани алат не омогућава анализу оваквих колаборативних графова (мрежа) методама за анализу социјалних мрежа.

Кориснички интерфејси анализираних алата су такође на ниском нивоу. Већина алата се покреће из командне линије, док само мањи број поседује графичке корисничке интерфејсе који олакшавају рад са њима. Иако су ови алати намењени корисницима-професионалцима на пољу рачунарства, ипак је већи број алата слабо прихваћен и коришћен због великог напора који је потребан за њихово овладавање. Поред тога, одређени алати су платформски зависни или се лакше извршавају на одређеним платформама, попут алата Moss који се лакше извршава под *UNIX/Linux* окружењима.

Скалабилност система за детекцију сличности у програмском коду са повећањем броја кодова који се пореде такође представља значајан проблем. Стога се поједини приступи за детекцију сличности, попут коришћења графова зависности не користе за поређење великог броја кодова, какав је случај са студентским радовима. Са друге стране, код система који користе GST алгоритам, број поређења расте квадратно са бројем кодова који се пореде.

Иако постоје технике као што су коришћење инвертованих индекса за претрагу или суфиксна стабла којима се може смањити број поређења, то се ради уз одређени губитак прецизности. Стога се намеће питање да ли се смањивање времена обраде већег броја радова може постићи паралелизацијом система за детекцију сличности и коришћењем доступних хардверских ресурса (језгара) централних и графичких процесора.

У раду ће кроз наредне главе управо бити испитана три главна правца за побољшање система за детекцију сличности у програмском коду како би они постали што ефектнији у детекцији плагијаризма. Смањивање времена обраде већег броја радова ће бити омогућено паралелизацијом система на нивоу организације послова и алгоритама за детекцију сличности. Анализа колаборација ће бити спроведена методама за анализу социјалних мрежа, које се ослањају на теорију графова применљиву у анализи социјалних мрежа. Визуелизација резултата ће омогућити једноставније уочавање и инспекцију сличних парова радова.

5. ПАРАЛЕЛИЗАЦИЈА АЛГОРИТАМА ЗА ДЕТЕКЦИЈУ СЛИЧНОСТИ У ПРОГРАМСКОМ КОДУ

У протеклих десетак година је забележен велики напредак на пољу рачунарског хардвера. Са достизањем ограничења у повећавању радне фреквенције (енг. *frequency wall*) код једнојезгарних централних процесора, произвођачи хардвера су се окренули подизању перформанси кроз паралелизам на нивоу функционалних јединица. Наиме, добици у перформансама који се могу остварити кроз подизање радне фреквенције су значајно смањени због три значајна фактора [91]:

- све већег раскорака у брзинама процесора и оперативне меморије чиме меморијски пропусни опсег постаје уско грло система (енг. *memory wall*),
- тешкоћа да се пронађе довољно паралелизма на нивоу инструкција (енг. *ILP wall*),
- експоненцијално растуће потрошње енергије и дисипације топлоте са порастом радне фреквенције (енг. *power wall*).

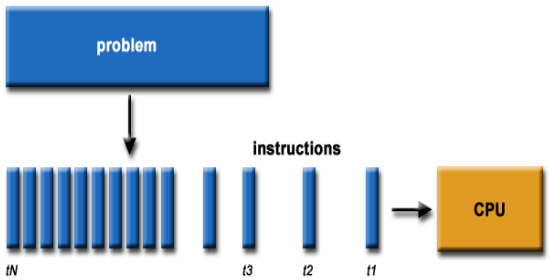
Модерни рачунарски системи се састоје од вишејезгарних централних процесора (*Central Processing Units* - CPU) и многојезгарних графичких процесора (*Graphic Processing Units* - GPU). Графички процесори су у последње време прешли велики пут од специјализованих процесора на пољу рачунарске графике до процесора опште намене који се све више користе у различитим научним и комерцијалним применама.

У овом поглављу ће бити описане технике које се могу применити за паралелизацију система за детекцију сличности у програмском коду на централним и графичким процесорима. Фокус ће бити на паралелизацији детекције сличности у програмском коду, као најзахтевнијем делу оваквих система. Биће размотрена паралелизација на нивоу алгоритма за детекцију и на нивоу организације послова и изложени резултати евалуације.

5.1. Хетерогено рачунарство

До почетка овог века, клијентски, кориснички рачунари су углавном били једнопроцесорски системи, док су вишепроцесорски системи углавном били намењени за серверско тржиште и релативно уске научне кругове у оквиру суперрачунарских лабораторија. Једнопроцесорски системи извршавају код углавном детерминистички, секвенцијално, у поретку у којем се оне јављају у програмском коду (енг. *in order* извршавање). Да би се решио неки проблем, одговарајући алгоритам се раздваја на низ дискретних инструкција, које се концептуално извршавају у програмском поретку, као на Слици 5.1. У једном тренутку се може извршити само једна инструкција, мада су у модернијим једнопроцесорским системима уграђивани различити механизми којим се врши имплицитна паралелизација извршавања на нивоу инструкција, мењање редоследа извршавања инструкција (енг. *out of order*), као и спекулативно извршавање.

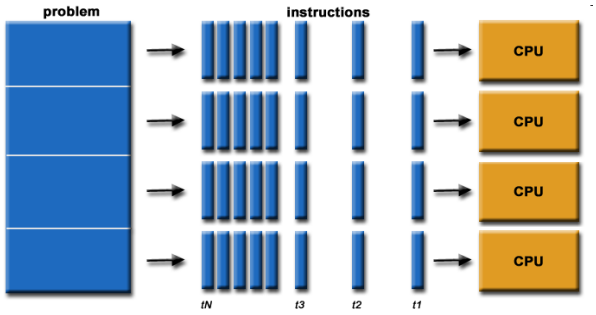
Као што је раније поменуто у уводном делу поглавља, данашњи рачунари углавном представљају паралелне рачунарске системе. У таквим системима може постојати више процесних чворова са рачунским ресурсима који могу да обављају различите послове, а паралелно рачунарство (енг. *parallel computing*) представља истовремено коришћење више таквих рачунских ресурса да би се решио неки проблем. Паралелно рачунарство се пре свега користи, јер доноси уштеду на времену и омогућава решавање већих проблема, али и због коришћења слободних нелокалних ресурса и превазилажења меморијских ограничења.



Слика 5.1. Секвенцијално извршавање програма [92]

Код паралелних рачунарских система софтвер се типично извршава на више процесних јединица. Проблем се раздваја на више делова који се могу решавати конкурентно, у паралели. Сваки део проблема се даље раздваја на низове дискретних инструкција. Инструкције из сваког од делова се извршавају истовремено на различитим рачунским ресурсима (процесним елементима), као на Слици 5.2. За проблем који се решава битно је да поседује делове који се могу решавати истовремено и да је притом време које се утроши на решавање проблема коришћењем више ресурса значајно мање од времена потребног да се реши на рачунару са једном процесорском јединицом.

Карактеристична су два начина за поделу проблема на делове који се могу решити у паралели. Код функционалне декомпозиције проблема, проблем се дели на карактеристичне и заокружене функције (послове) који се онда додељују процесорима на обраду. Код декомпозиције домена, уобичајено постоји просторни паралелизам или паралелизам на нивоу података (енг. *data parallelism*), па се подаци на одређени начин расподељују процесорима на обраду. Таква расподела укључује блоковску и цикличну расподелу која се може обавити у више димензија, све у зависности од потреба проблема који се решава.



Слика 5.2. Паралелно извршавање програма [92]

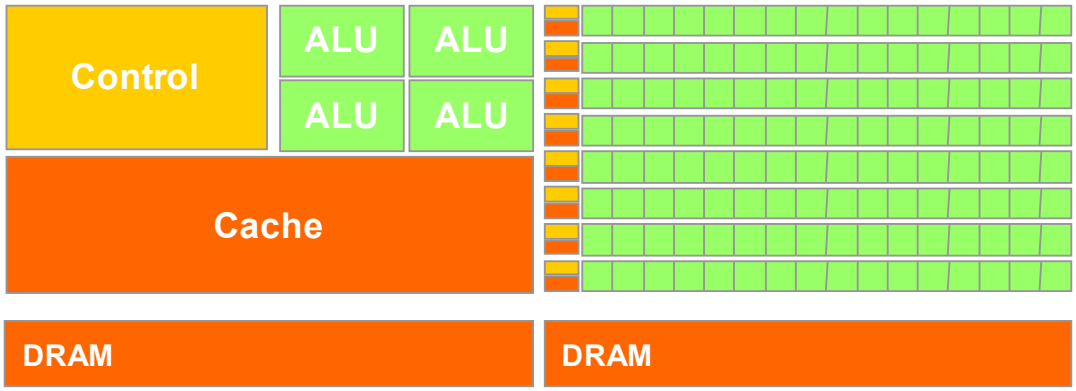
Рачунски ресурс који се користи за решавање проблема може бити појединачни рачунар са више процесора, различите акцелераторске картице, више рачунара увезаних рачунарском мрежом или нека комбинација. У модерним рачунарима су присутни вишејезгарни (енг. *multicore*) централни процесори, као и многојезгарни (енг. *manycore*) графички процесори. Такође, постоје копроцесорске картице опште намене са великим бројем језгара, попут *Intel Xeon Phi* или наменски FPGA (енг. *Field Programmable Gate Array*) процесори који се конфигуришу за специјализоване послове.

Због присуства процесних елемената различитог типа паралелни рачунарски системи који садрже више рачунарских ресурса различите архитектуре често се називају хетерогени рачунарски системи, а решавање проблема коришћењем ових система хетерогено рачунарство. Сматра се да се код оваквих рачунарских система постижу боље перформансе и енергетска ефикасност у односу на традиционалне рачунарске системе, пре свега додавањем различитих копроцесорских јединица са специјализованим функцијама [93].

Централни процесори су уобичајено оријентисани ка извршавању послова и погодни за случајеве код којих постоји јасна функционална декомпозиција проблема. Обично садрже неколико моћних аритметичко-логичких јединица (енг. *arithmetic logic unit* - ALU) и сложену контролу тока извршавања програма (Слика 5.3а). Уобичајено се програми на вишејезгарним централним процесорима извршавају помоћу нити. Нити су добро познат концепт у паралелном рачунарству и представљају конкурентне и независне токове контроле унутар једног процеса. За потпуну искоришћеност једног CPU-а је потребно покренути тек неколико нити. Такође, архитектура централних процесора је оријентисана ка смањивању кашњења у приступу меморији кроз хијерархију кеш меморија.

Са друге стране, графички процесори су оријентисани ка обради што веће количине података у јединици времена. Састоје се од већег броја једноставних и енергетски ефикасних ALU јединица са поједностављаном контролом тока и кеш меморијама релативно малог капацитета (Слика 5.3б). Погодни су за проблеме који изражавају високу регуларност и просторни паралелизам, као што су обрада слике, видео сигнала, различите научне симулације и сл. Скупи приступи оперативној меморији се не сакривају хијерархијом кеш меморија, већ интензивном употребом великог броја лаких нити. Тај број уобичајено зависи од димензије проблема који се решава и креће се од неколико хиљада до неколико милиона нити.

Хетерогени рачунарски системи стога најчешће раде у режиму копроцесирања, где сваки процесор обавља посао за који је најпогоднији. Оперативни систем, секвенцијални делови програма, као што су улаз и излаз података и сл., се типично извршавају на CPU, док се рачунски интензивни делови поготово ако поседују просторни паралелизам пребацују на GPU да би се постигла већа убрзања.



(а) CPU **(б) GPU**

Слика 5.3. (а) Архитектура централног процесора, (б) архитектура графичког процесора [94]

У пракси се хетерогени рачунарски системи програмирају кроз различите програмске моделе ниског и високог нивоа. Највише коришћени програмски модели ниског нивоа за хетерогено рачунарство су OpenCL (*Open Computing Language*) и CUDA (*Compute Unified Device Architecture*). OpenCL је отворени стандард намењен за програмирање најширег дијапазона уређаја од вишејезгарних централних процесора, преко FPGA копроцесора до графичких процесора. CUDA је власничка технологија компаније NVIDIA за програмирање хетерогених система који користе њихове графичке процесоре. Друге библиотеке за паралелно програмирање, попут MPI (eng. *Message Passing Interface*) се такође све више прилагођавају за употребу у хетерогеним системима.

Иако програмски модели ниског нивоа омогућавају постизање веома добрих перформанси извршавања програма, њих такође одликује и стрма крива учења и релативно велики прогамерски напор да би се написао и оптимизовао код. Стога све популарнији, нарочито у научној заједници, постају програмски модели високог нивоа засновани на директивама, као што су OpenMP и OpenACC [95]. OpenMP је добро познат програмски модел намењен пре свега вишејезгарним централним процесорима. Програмер користи претпроцесорске директиве да означи регионе кода које преводилац аутоматски треба да паралелелизује. Типично, паралелизују се петље, а то захтева да њихове итерације буду међусобно независне по подацима. Сличан приступ је примењен и код програмског модела OpenACC којим се омогућава паралелизација директивама на графичким процесорима. Оба програмска модела омогућавају остваривање релативно добрих убрзања у односу на моделе ниског нивоа за доста краће време. Стога се често користе за развој прототипа паралелне верзије програма, како би се проценило да ли се паралелизација исплати или не.

5.2. Коришћене технологије

У овом поглављу је дат детаљнији преглед технологија које су коришћене за паралелизацију алгоритама за детекцију сличности у програмском коду. Паралелизација је у овом раду спроведена на централном и графичком процесору, како би се истражиле њихове могућности за примену у детекцији сличности. Разлике у архитектури диктирају коришћени програмски модел и начине за паралелизацију.

Као што је раније поменуто, паралелизација на централним процесорима се уобичајено врши коришћењем нити. Постоји већи број програмских модела и технологија за паралелизацију нитима, као што су OpenMP, Intel *Cilk/Cilk++* и TBB (*Thread Building Blocks*), *Pthreads* и сл. OpenMP је програмски модел за паралелизацију директивама, где програмер анотира делове програмског кода одговарајућим директивама препуштајући преводиоцу да изврши паралелизацију. Intel *Cilk/Cilk++* и TBB уводе нове паралелне конструкте у програмске језике C и C++ којима се омогућава једноставна паралелизација. Ипак, за паралелизацију на централним процесорима је одабран програмски модел *Pthreads* нити, јер омогућавају програмеру експлицитну и потпуну контролу над различитим аспектима извршавања нити. Такође, различита истраживања [96, 97] показују одређену предност ове над другим технологијама, нарочито уколико се жели паралелизација на нивоу послова.

На пољу графичких процесора је избор технологије за паралелизацију нешто једноставнији. NVIDIA CUDA и OpenCL су логични избори за програмирање ових процесора уколико се жели коришћење API-ја (енг. *Application Programming Interface*) ниског нивоа уз потпуну контролу над свим детаљима оптимизације, док OpenACC пружа могућност за паралелизацију директивама. С обзиром да је за рад био доступан графички хардвер компаније NVIDIA, логичан избор је био коришћење NVIDIA CUDA технологије која је и најзрелија технологија на овом пољу [98]. Истраживања [99, 100] такође показују упоредиве перформансе CUDA и OpenCL апликација на NVIDIA графичким процесорима, уз нешто компликованије програмирање OpenCL апликација због генералнијег програмског модела. Стога је коришћење CUDA програмског модела сасвим оправдано.

5.2.1. *Pthreads* нити

Вишејезгарни централни процесори се уобичајено програмирају коришћењем нити које представљају стандардан концепт дуго познат у оперативним системима и у системима са дељеном меморијом. Као што је раније поменуто потребно је покренути неколико нити да би се искористио пун потенцијал данашњих CPU-ова и запослили сви хардверски ресурси.

У овом раду је коришћена POSIX *threads* (*Pthreads*) имплементација нити, заснована на IEEE POSIX 1003.c стандарду која је присутна као стандардни део фамилије UNIX/Linux оперативних система [101]. Фамилија POSIX стандарда је развијена као API подршка за софтвер који се пише за UNIX оријентисане оперативне системе, али их је могуће применити и на друге оперативне системе. Стандард је пре свега развијен да омогући што бољу портабилност (преносивост) софтвера са једне платформе на другу.

Овај програмски модел је експлицитан, а корисник је задужен за стварање, управљање и синхорнизацију рада нити. Модел је нарочито погодан за паралелизацију на нивоу задатака. POSIX нити представљају апликативни програмски интерфејс имплементиран кроз скуп корисничких типова и функција написаних на програмском језику C, описаних у заглављу *pthreads.h* и реализованих у пратећој библиотеци. Програмери могу користити ову библиотеку да креирају, манипулишу и управљају нитима, као и да врше синхронизацију међу нитима користећи међусобно искључење (енг. *mutex*), условне променљиве и семафоре. Иако су *Pthreads* нити карактеристичне за UNIX/Linux оријентисане оперативне системе постоји и имплементација за *Microsoft Windows* оперативни систем.

Нити се користе за унапређење перформанси програма, јер рад са нитима захтева много мање режијске трошкове него рад са процесима. Нити постоје унутар контекста једног процеса и користе све његове ресурсе. Режијски трошкови њиховог стварања су мали, јер се реплицирају само ресурси неопходни за њихово извршавање. Такође, због дељења ресурса и истог адресног простора, комуникација између нити је много ефикаснија од комуникације између процеса, а могуће је и преклапање обраде и улазно-излазних операција. Док једна нит чека на спору периферију, остале нити могу да интензивно користе процесор.

Са друге стране, пошто деле исти адресни простор, оно што једна нит уради над дељеним ресурсима ће бити видљиво у свим осталим нитима тог процеса. Могућност истовременог приступа подацима захтева експлицитну синхронизацију у програмском коду. Стога је програмер одговоран за заштиту интегритета дељених података, као и несметан приступ до њих. Такође, одговорност програмера је и правилна употреба постојећег програмског кода, попут библиотечких позива, за који често није познато да ли је безбедан за извршавање у вишенитном окружењу. Добар пример таквог кода су библиотечке функције програмског језика C за генерисање псеудослучајних бројева које користе глобални, дељени објекат за чување свог стања.

Типови података и функције из *Pthreads* API се могу поделити у неколико целина, од којих су најважније следеће четири:

- Управљање нитима које омогућава њихово стварање, уништавање, растављање и спајање
- Синхронизација путем међусобног искључивања кроз неколико типа брава
- Условне променљиве које омогућавају синхронизацију у зависности од остварења задатог услова
- Синхронизациони механизми вишег нивоа као што су баријере, браве за читање и упис, и сл.

За све наведене функционалности су дефинисани одговарајући типови података и функције за иницијализацију и манипулацију објектима. Све функције раде над нетранспарентним објектима (нитима, бравама, и сл.) и не може се директно приступати њиховим пољима. Стварање и промена атрибута објеката се врши искључиво позивањем API функција. Креирани и иницијализовани објекти се морају уништити када више нису потребни.

Нити се стварају над функцијама које имају строго дефинисано заглавље. Нит извршава тело функције, а аргументи се могу проследити коришћењем корисничке структуре приликом њеног стварања. Програмер може имати утицаја на одређене атрибуте нити као што су приоритет извршавања, могућност спајања (енг. *join*) са другим нитима, величина стека и сл. Једна нит може креирати произвољан број других нити, а укупан број нити је ограничен само од стране оперативног система. Не постоји хијерахијски однос међу нитима, већ су све нити су међусобно једнаке (енгл. *peers*).

У смислу извршавања посла, три су типична односа међу нитима:

- *Pipeline* модел код кога је посао подељен на мање, логички независне целине које могу бити одвојено завршене.
- *Manager/worker* модел код кога једна нит (главна нит) има улогу координатора и додељује послове осталим нитима. Типично се имплементира кроз статички или динамички створен базен (депо) нити.
- *Peer* модел у којем главна нити има одређена посебна задужења у вези са улазно-излазним операцијама, али равномерно учествује у послу као и остале нити.

Избор модела употребе нити је најчешће диктиран потребама проблема који се решава. Код проблема који имају уочљиву небалансираност у оптерећењу се чешће користи *manager/worker* модел, док се *peer* модел чешће користи код обрада са равномерно распоређеним оптерећењем. *Pipeline* модел се обично користи код функционалне декомпозиције проблема.

5.2.2. CUDA

CUDA (енг. *Compute Unified Device Architecture*) је паралелна рачунарска архитектура развијена од стране компаније NVIDA која ефикасно подржава *data-parallel* програмски модел. CUDA је архитектура специфична за графичке процесоре и намењена је за решавање општих рачунских проблема на графичким процесорима (енг. *General-Purpose computing on Graphics Processing Units*).

Од свог почетка, графички процесори су били намењени за специјализована израчунавања на подручју графике, али су временом еволуирали у програмабилне, високопаралелне, многојезгарне процесоре са великом рачунском снагом и великим пропусним опсегом (енг. *bandwith*). Ови процесори су поготову погодни за проблеме који изражавају регуларност са високим аритметичким интензитетом израчунавања и данас имају примену на различитим пољима, од обраде сигнала до рачунских симулација из физике, хемије и биолошких наука. Графички процесори суштински експлоатишу паралелизам на нивоу података. *Data-parallel* модел обраде података се на графичком процесору имплементира покретањем великог броја лаких нити, а потребно је покренути више хиљада до неколико стотина хиљада нити да би се постигле максималне перформансе.

Програмски модел CUDA омогућава програмирање коришћењем наменског апликативног програмског интерфејса и екстензије програмског језика C, као и великог броја доступних библиотека. За рад су доступни и алати за откривање грешака (енг. *debugging*) и оптимизацију (енг. *profiling*) програмског кода.

Програми се на овој платформи извршавају у режиму копроцесирања, што је типично за хетерогене системе. Део кода се извршава на централном процесору и он се користи за подешавање обраде, припрему података, иницирање меморијских трансфера и сл., док се рачунски интензивни делови програма извршавају на графичком процесору у виду посебних функција – језгара (енг. *kernel*), која се извршавају на графичком процесору од стране великог броја нити. Језгро представља програмски код који извршава једна нит. Због скалабилности извршавања језгара на различитом графичком хардверу, нити се организују у блокове, а блокови се организују унутар решетке (енг. *grid*).

Блокови нити се извршавају на мултипроцесорским јединицама унутар графичког процесора (енг. *streaming multiprocessor*) које се састоје од одређеног броја скаларних процесора. У једном тренутку сви скаларни процесори унутар SM-а извршавају исту инструкцију што одговара SIMD (енг. *Single Instruction Multiple Data*) arhitekturi. Блокови нити се независно распоређују на мултипроцесоре који постоје унутар система, тако да не постоји могућност за глобалну синхронизацију нити. Нити се могу међусобно синхронизовати само унутар истог блока.

Блокови нити и решетка могу бити организовани у више димензија (1D, 2D или 3D), како би се што боље подржали различити обрасци приступа меморији, у зависности од проблема који се решава. Свака нит која се извршава на једном мултипроцесору може кроз скуп уграђених идентификатора да одреди где се налази унутар блока и решетке и да затим једнозначно одреди над којим подацима треба да ради.

Због великог броја нити које се извршавају у паралели, меморијска хијерархија графичких процесора је нешто другачија него код централних процесора. Све нити могу равноправно приступати глобалној, оперативној меморији на нивоу целог процесора. Међутим, приступ овој меморији је изразито спор. Стога на нивоу мултипроцесора постоји дељена меморија малог капацитета која се може користити као софтверски управљана кеш меморија како би се поправиле перформансе извршавања програма. Кеш меморија првог нивоа је малог капацитета и постоји на нивоу мултипроцесора, док је кеш меморија другог

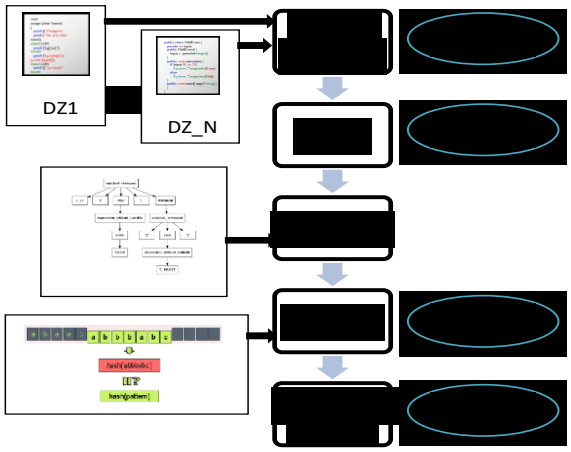
нивоа заједничка за све мултипроцесорске јединице. CPU и GPU најчешће поседују одвојене оперативне меморије и одвојене адресне просторе, тако да је потребно вршити меморијске трансфере приликом премештања података са једног на други процесор.

Иако није тешко написати програм који се коректно извршава на графичком процесору, оптимизација перформанси је задатак који захтева одређени ниво познавања архитектуре. Перформансе могу значајно да варирају зависно од ограничења постављених расположивим ресурсима конкретног графичког процесора, а на програмеру је да покуша да искорити сав доступан паралелизам. Више информација о CUDA технологији и начину програмирања се може наћи у [94, 102].

5.3. Секвенцијална имплементација

Као полазна основа за паралелизацију је искоришћена секвенцијална имплементација експерименталног система за детекцију сличности који је базиран на токенизацији и RKR-GST алгоритму, а која је представљена у раду [103]. Систем је имплементиран по узору на систем JPlag, уз незнатно модификован скуп токена.

Систем подржава поређење изворних кодова програма написаних у програмском језику C++, а подржана је и већина конструката програмског језика C. Систем је имплементиран тако да буде модуларан и лако проширљив, па је стога његов рад подељен у пет фаза: лексичку анализу (коју обавља лексер), синтаксну анализу (коју обавља парсер), токенизацију, хеширање и одређивање сличности. Фазе рада имплементираног секвенцијалног система су приказане на слици 5.4. Поред сваке фазе је назначен начин имплементације уз навођење коришћеног алата или алгоритма.



Слика 5.4. Фазе рада имплементираног система

5.3.1. Лексер, парсер и токенизтаор

Прва фаза рада система за детекцију сличности је лексичка анализа у којој се посматрани програмски код дели на низ симбола (токена) који се прослеђују у фазу парсирања. Идентификација токена се врши коришћењем регуларних израза, а они суштински представљају кључне речи и операторе језика C/C++. За сваки токен се памти број линије у којој се он налази, пошто је та информација битна приликом приказивања резултата поређења.

У фази лексичке анализе се врши и одређено процесирање улазног кода како би се уклонили коментари, модификатори променљивих, декларације простора имена и претпроцесорске директиве. Такође, врше се и одређена поједностављења у коду. Пуна имена променљивих (енг. *fully qualified name*) код класних и сличних типова се детектују и третирају се на исти начин као и променљиве простих имена, а парсеру се прослеђује исти токен. Операторске функције се третирају на исти начин као и обичне функције. Типови података, чији се називи састоје од више од једне речи, добијају исти токен као и типови са називом од само једне речи.

Резултат рада лексера се прослеђује парсеру који врши синтаксну анализу на основу дефинисане граматике. У овој фази се препознају исправне језичке конструкције и генерише стабло парсирања. Парсер током свог рада комуницира са токенизатором коришћењем акција парсера. Скуп токена коришћен у секвенцијалној имплементацији је базиран на скупу токена који користи систем JPlag уз незнатне измене које се пре свега односе на употребу језика C/C++ уместо језика Java.

Према [68], скуп токена је тако дефинисан да карактерише структуру програма, али да буде робустан у односу на могуће модификације које плагијатори могу да учине. Постоји посебан токен за почетак и крај блокова кода, подржани су различити конструкти за контролу тока програма, променљиве се замењују токеном који одговара типу променљиве. По узору на JPlag, аритметичке операције су апстраховане коришћењем једног заједничког токена.

Сам токенизатор је имплементиран кроз три класе које имплементирају саму табелу симбола, досег у којем је токен дефинисан и токене. У класи *Token* се чува идентификатор токена, редни број линије у програмском коду у којој се токен појављује и поље које означава да ли је токен маркиран у фази поређења. Класа *Scope* представља досег дефинисан

блоком кода и садржи листу токена и показивач на објекат исте те класе који представља спољашњи досег. Класа *Table* је имплементирана као статичка класа над којом промене врши парсер.

Имплементирани парсер и лексер су базирани на стандарду C++98 и тестирани коришћењем задатака из збирке [104]. Сви задаци из наведене збирке се успешно тестирају, али је примећено да постоји проблем са процесирањем кода који садржи макрое, пошто се претпроцесорске директиве игноришу. Самим тим долази до незавршених макроекспанзија. Такође, коришћена граматика је у неким случајевима двосмислена, што је статички разрешавано давањем приоритета декларацијама. С обзиром да је актуелни стандард C++14, све изложено представља простор за унапређења секвенцијалне имплементације.

5.3.2. Хеширање и детекција сличности

Као што је раније поменуто, секвенцијална имплементације за детекцију сличности користи GST алгоритам оптмизован коришћењем *Karp-Rabin* алгоритма. Стога наредна фаза у раду система за детекцију представља хеширање које се спроводи у циљу побољшања перформанси. Коришћењем одговарајуће *rolling hash* функције, врши се хеширање над низом од MML узастопних токена. Недостатак коришћене хеш функције је што добијене хеш вредности могу бити прилично велике. Како би се избегли велики бројеви у [60] се предлаже коришћење операција сабирања и множења по модулу.

Сваки изворни код за поређење је представљен објектом класе *Program* који садржи податке о имену датотеке у којој се код налази, вектор токена, вектор хеш вредности и хеш мапу која пресликава срачунату хеш вредност MML узастопних токена на позицију првог од тих MML токена у вектору токена. У овој фази рада система се врши израчунавање вектора хешева и креирање хеш мапе за сваки од улазних програмских кодова који се пореде. Треба напоменути да се због ефикасног смештања и претраживања хешева у имплементацији користи готова *std::multimap* структура података из C++ STL библиотеке. Ова структура омогућава дохватање почетних позиција свих токена из једног програма који одговарају хеш вредности из другог програма за поређење, што значајно смањује број потребних поређења и тиме унапређује ефикасност извршавања.

Детекција сличности у програмском коду у потпуности прати изложену модификацију GST алгоритма из Поглавља 3.5. За сваки пар програмских кодова који се упоређују се памти вектор објеката који садрже податке о потврђеним подударањима (енг. *tiles*). За свако потврђено подударање се чувају подаци о почетном индексу у вектору токена првог програма, почетном индексу у вектору токена другог програма и дужини подударања. Мера сличности се израчунава по формули изложеној у Поглављу 3.7.

5.3.3. Евалуација секвенцијалне имплементације

У овом поглављу је изложена кратка евалуација секвенцијалног система у односу на системе JPlag и Moss. При тестирању система коришћени су пројектни задаци са предмета Оперативни системи 1 са Електротехничког факултета у Београду. Било је доступно укупно 125 радова написаних у програмском језику C++ који се успешно парсирају. Мањи делови кода у овим пројектним задацима су написани на асемблеру или коришћењем макроа. Радови у просеку садрже око 1000 линија изворног кода. Резултати рада система су поређени са резултатима система JPlag и Moss са подразумеваним параметрима.

Имплементирани секвенцијални систем (у даљем тексту BASE) и JPlag рачунају проценат сличности два програмска кода по истој методологији. Међутим, систем Moss за сваки упоређени програмски код даје проценат садржаја програма који је сличан другом програму, као и укупан број линија кода у програмима који се преклапа. Да би се сва три система равноправно упоредила, потребно је резултат који даје Moss адаптирати, тако што се број подударајућих линија који врати Moss множи са два, па се производ подели збиром броја линија улазних програмских кодова који се пореде.

Коришћени скуп задатака садржи 125 студентских радова. Како се сваки програмски код пореди са свим осталима, а релација поређења је симетрична, то даје 7750 парова улазних програмских кодова који се могу упоредити. У Табели 5.1. је приказан број програмских кодова за BASE алат и JPlag у односу на проценат сличности у опсезима од 10%. Из ове анализе је изостављен систем Moss, јер иако пореди све предате програмске кодове, овај алат враћа меру сличности за највише 250 најсличнијих парова.

Табела 5.1. Поређење резултата имплементираног система (BASE) и система JPlag у опсегу од 10%

Проценат сличности	0-10%	10-20%	20-30%	30-40%	40-50%	50-60%	60-70%	70-80%	80-90%	90-100%
JPlag	5931	1329	325	133	27	2	2	0	0	1
BASE	6886	655	136	54	14	3	1	0	1	0

Резултати приказани у Табели 5.1. показују релативно слично понашање оба тестирана алата за сличности кодова изнад 50%. У доњој половини опсега BASE систем даје нешто конзервативније резултате. Претходна истраживања [105, 106] показују да проценат сличности преко 50% готово извесно указује на плагијат, док се нешто ниже вредности могу користити уколико постоји сумња на делимични плагијаризам. У том смислу, имплементирани алат даје упоредиве резултате као и JPlag, а сви кодови са сличношћу од преко 50% се испостављају као плагијати.

Табела 5.2. *Top-10* поређење сва три алата

Парови студената	JPlag	Moss	BASE
am-tt	94	81	86
ln-mi	60	56	59
sb-tm1	65	53	62
pp-tm2	51	40	56
sm-tm2	54	44	55
jv-oo	50	49	49
gd-tm1	48	29	36
gd-vs	48	23	32
tm1-ua	47	37	34
ip-pd	46	45	46
rm-so	42	46	50
ka-tm1	40	37	32
jv-sb	31	25	47
sm-pp	43	32	46

Друга анализа је спроведена по узору на [66] и представља поређење *top-n* најсличнијих парова улазних кодова. Овде је коришћено *top-10* поређење које подразумева поређење најсличнијих 10 парова програма за сваки од система. Такво поређење даје најмање 10, а највише 30 парова програма чијом ручном инспекцијом даље треба утврдити постојање плагијаризма. Резултати поређења су дати у Табели 5.2., а састоје се од 14 парова програмских кодова. Парови су именовани на основу иницијала студената чији се програмски кодови упоређују. Вредности у табели представљају проценат сличности који је сваки систем вратио за дати пар улазних програмских кодова. Примећује се да су резултати рада BASE система јако слични резултатима рада преостала два система. Једина значајна разлика се примећује код парова *gd-tm*, *gd-vs* и *jv-sb.* Уочена разлика је пре свега последица имплементационих одлука које се односе на игнорисање претпроцесорских директива и третирање пуних имена променљивих које се третирају као промељиве простих имена.

Тестирање сагласности резултата које дају алати се може обавити кроз корелациону анализу њихових листа сличности. Овде је коришћен Спирманов коефицијент корелације који се дефинише на следећи начин:

$$\rho = 1 - \frac{6*\sum d_i^2}{n*(n^2-1)} \tag{5.1}$$

где $d_i=x_i-y_i$ представља разлику рангова посматраних парова на листама сличности алата. При томе, уколико постоје дупликати у оригиналним вредностима X_i і Y_i, рангови x_i и y_i представљају средње вредности позиција оригиналних вредности у растућем поретку. Коефицијент је срачунат за листе које су добијене *top-10*, *top-20* и *top-50* поређењем на претходно описани начин за парове алата.

Табела 5.3. Корелациона анализа за *top-10*, *top-20* и *top-50*

Поређење	BASE – JPlag	BASE - Moss	JPlag - Moss	Број парова
Top-10	0,70	0,81	0,65	14
Top-20	0,64	0,76	0,71	60
Top-50	0,10	0,31	0,27	94

Резултати корелационе анализе су приказани у Табели 5.3. Резултати показују да постоји значајна сагласност листа сличности за сва три алата које су добијене узимањем првих 10 или 20 сличних парова за сваки од алата. Резултати имплементираног алата показују нешто бољу сагласност са алатом Moss, него са алатом JPlag. Међутим, јако слични, плагирани радови добијају велики проценат сличности од стране сва три алата, што је и утврђено ручном инспекцијом. Ручна инспекција ових листа показује да се оне састоје од радова са сличношћу већом од 40%, што емпиријски указује на плагијаризам. Ширењем листе на 50 сличних парова за сваки од алата, примећује се велики број парова са блиским, релативно ниским процентом сличности. Такве парове због имплементационих разлика алати другачије рангирају, па стога коефицијент корелације значајно опада.

Пошто постоје комплетне листе сличности, за корелацију резултата алата JPlag и секвенцијалне имплементације се може користити и класичан коефицијент корелације (тзв. Пирсонова корелација), како би се утврдила сагласност детектованих степена сличности:

$$Correl(X,Y) = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}} \tag{5.2}$$

Добијају се високи коефицијенти корелације између 0.72 и 0.96, у зависности од тога колико елемената са врха ранг листе сличности се користи за рачунање, што је приказано на Слици 5.5.

Слика 5.5. Корелација резултата JPlag и референтног BASE алата

Евалуације говори да се секвенцијални систем понаша јако слично системима JPlag и Moss. Мање разлике постоје углавном због имплементационих одлука спроведених у прве три фазе система које су образложене у Поглављу 5.3.1. Као што је објашњено, ове одлуке утичу како на смањење, тако и на повећање процента сличности.

5.3.4. Могућности за паралелизацију

Постоји неколико места у оквиру секвенцијалне имплементације која су погодна за паралелизацију. Најпре се може уочити да се прве три фазе имплементираног система морају извршити за сваку улазну датотеку са програмским кодом. Притом, лексичка анализа, синтаксна анализа и токенизација се могу извршити независно за сваки изворни код у односу на друге. Самим тим, постоји могућност за паралелизацију на нивоу послова (енг. *task.*) на централном процесору.

Погодно је искористити концепт депоа (базена) послова (енг. *task pool*). Један програмски код би представљао један посао, а послови би били обрађивани од стране нити које би их равноправно обављале. Распоређивање послова би било динамичко, због неједнаког оптерећења у обради појединачних програмских кодова (нпр. због различите дужине). Нит која заврши посао раније би резултате сместила у одговарајући бафер и узела нови посао из депоа.

Алгоритам за детекцију сличности представља друго значајно место у оквиру кога би се могло остварити побољшање перформанси паралелизацијом. С обзиром да се сваки рад мора упоредити са сваким другим, уколико се желе резултати без губитка тачности, такође се може искоритити концепт депоа послова. Један посао би представљало поређење два програмска кода, а нит би била задужена да спроведе RKR-GST алгоритам над паром програмских кодова.

Алтернативни приступ паралелизацији на нивоу послова би била паралелизација на нивоу алгоритма. Уколико се погледа псеудокод GST алгоритма који је дат у Табели 3.3., уочава се да се унутрашње петље понављају секвенцијално док год се проналазе подударања дужа од минималне дужине подударања. Стога алгоритам мора да претрпи одређено реструктурирање, како би се омогућила паралелизација најзахтевнијег дела, а то је проналажење стрингова који се подударају, што ће бити детаљно описано у наредним поглављима.

5.4. Паралелна имплементација

У овом раду су приказана два могућа приступа за паралелизацију система за детекцију сличности у програмском коду изложена у [107]. Први приступ на нивоу организације послова је коришћен у имплементацији на централном процесору, с обзиром да се секвенцијална имплементација могла на једноставан начин модификовати да подржи овакав тип паралелизације. Паралелизација на графичком процесору је обављена на нивоу алгоритма, с обзиром да извршавање послова није добро подржано на овој платформи. У наставку су приказани детаљи у вези једне и друге имплементације, као и спроведена експериментална евалуација.

5.4.1. Реструктурирани GST алгоритам

GST алгоритам описан у Поглављу 3.5 на основу ког је реализована секвенцијална имплементација није директно погодан за паралелизацију, па је стога извршено одређено реструктурирање алгоритма. Наиме, у оквиру прве фазе алгоритма који чине три угнеждене петље се врши секвенцијално понављање унутрашњих петљи докле год се проналазе подударања подстрингова која су дужа од MML знакова. Алгоритам је тако имплементиран да се подударања обавезно проналазе од најдужих ка најкраћим. Такво извршавање не може да се паралелизује због постојања зависности, па се оне морају отклонити.

Стога је алгоритам преуређен тако да се одвија у две фазе. У свом првом делу, алгоритам проналази и формира скуп свих подударајућих подстрингова дужих од MML. За свако пронађено подударање се памте индекси почетка у првом и другом стрингу који садрже токене, као и дужина подударања, а резултати се смештају у једноструко уланчану листу. Резултујућа подударања су сортирана опадајуће по дужини у једноструко уланчаној листи.

У другом делу се кроз итеративни поступак врши селекција најпогоднијих резултата подударања из формираног скупа. Један главних услова GST алгоритма захтева да се један токен може само једном упарити са неким другим токеном. Стога се у сваком кораку бира најдуже подударање које се не преклапа са неким претходно селектованим подударањем. Затим је неопходно извршити ажурирање преосталих подударања у скупу необрађених, како би се задовољио наведени услов.

Ажурирања сваког појединачног подударања може довести до његовог скраћивања. Уколико је након провере подударање скраћено, а дужина се смањила на мање од MML знакова, онда такво подударање треба уклонити из скупа. У случају да је подударање скраћено, али је дужина остала већа или једнака MML знакова, онда то подударање треба преместити на ново место у једноструко уланчаној листи, тако да она остане сортирана. Псеудокод модификованог алгоритма се налази у Табели 5.4.

Табела 5.4. Псеудокод модификованог GST алгоритма

```
0       Greedy-String-Tiling(String A, String B, int MML) {
1         /* Faza 1 */
2         maxmatch = MML;
3         matches = {};
4         forall tokens A_a in A {
5           forall tokens B_b in B {
6             j = 0;
7             while( A_a+j == B_b+j )
8               j++;
9             if ( j >= MML ) {
10              matches = matches + match(a,b,j);
11              if ( j >= maxmatch )
12                maxmatch = j;
13            }
14          }
15        }
16        /* Faza 2 */
17        len = maxmatch;
18        forall match(a,b,len) in sorted matches {
19          new_len = updateAndCheckTiles(matches, match(a,b,len));
20          switch ( new_len ) {
21            case UNCHANGED:
22              tiles = tiles + match(a,b,len);
23            case LESS_THAN_MML:
24              matches = matches - match(a,b,len);
25            case SHORTENED:
26              relocate(matches,match(a,b,new_len))
27          }
28          len = next_match_len(matches);
29        }
30        return tiles;
31      }
```

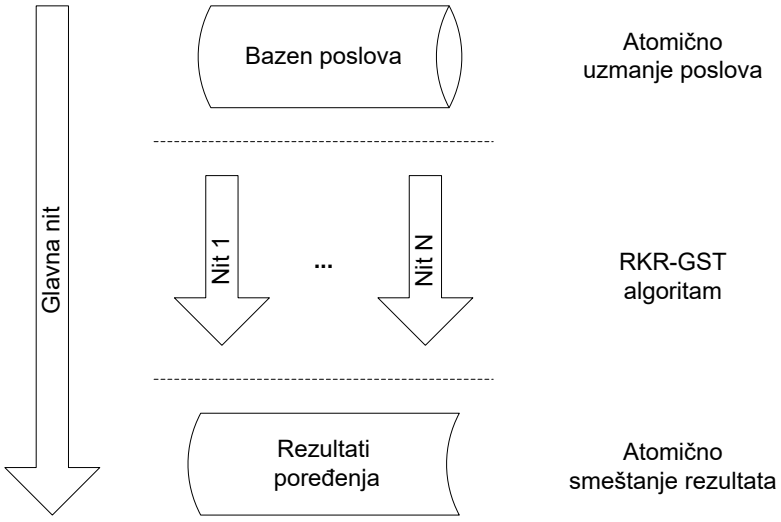
5.4.2. Имплементација на централном процесору

Као што је већ поменуто, на централном процесору је обављена паралелизација на нивоу задатака (послова) коришћењем нити. Број нити се одређује динамички у време извршавања програма на основу броја досупних хардверских ресурса. За ту намену се користе одговарајуће функције из *Pthreads* API које дохватају број расположивих језгара унутар процесора.

Нити се покрећу од стране главне нити која не учествује у обради. Њени једини задаци су да обави иницијализацију неопходних структура података и синхронизационих објеката и стварање нити. Синхорнизациони објекти се односе на браве и условне променљиве које се користе за заштиту дељених структура података (бафера) за расподелу посла и смештање резултата. Након стварања нити, главна нит им сигнализира да могу да почну са радом. Затим главна нит чека на завршетак рада нити-радника и прослеђује резултате у позивајуће окружење.

Након сигнализације да може да почне извршавање, свака нит узима један пар програма за упоређивање и спроводи комплетан RKR-GST алгоритам. Затим се резултати атомично уписују у одговарајућу структуру за смештање резултата, а нит узима наредни посао за поређење. Послови се расподељују на динамичкој бази по систему FCFS (енг. *First Come First Served*).

Мотивација за динамичко распоређивање се базира на изразитој неједнакости времена упоређивања парова задатака. С обзиром да се радови пореде сваки са сваким, пракса показује да се значајна подударања јављају само у малом проценту радова, па је у таквим случајевима време извршавања алгоритма дуже него код радова који нису слични чије поређење се брже завршава. Низ узима нове послове на извршавање док год се не исцрпи базен послова, када се заблокира и чека сигнал главне нити да може да финализује своје извршавање. Слика 5.6. приказује описани начин извршавања нити и организације послова на централном процесору.



Слика 5.6. Организација извршавања нити на нивоу послова на централном процесору

5.4.3. Имплементација на графичком процесору

Имплементација на графичком процесору ближе прати реструктурирани GST алгоритам описан у Поглављу 5.4.1. За паралелизацију је пре свега погодан први део прве фазе реструктурираног алгоритма. У њему се врши проналажење свих подударајућих подстрингова дужих од *MML* за један пар програма и он представља најинтензивнији део алгоритма. Може се приметити да тај део алгоритма није рачунски интензиван, јер се само врше приступи меморијским локацијама и поређења.

Са друге стране, проблем проналажења подударајућих подстрингова се свакако може паралелизовати, јер једно конкретно подударање може почети од било ког знака у једном низу токена за поређење и било ког знака у другом низу токена за поређење. Након проналажења свих подударајућих подстрингова на GPU врши се трансфер резултата на CPU, где се врши друга фаза алгоритма и финална обрада.

С обзиром да CPU и GPU имају одвојене оперативне меморије и адресне просторе, имплементирани алгоритам есенцијално представља хетерогени, хибридни алгоритам. Стога је пре почетка рада потребно извршити иницијализацију извршавања и алокацију неопходне меморије на страни GPU, као и иницијалне трансфере са CPU на GPU. У том циљу се вектори токена и хешева токена парова програма копирају на GPU.

Само поређење парова програма се имплементира у виду одговарајућег језгра 2D организације. Величина решетке је притом једнака производу димензија стрингова које су умањене за MML. Свака нит на основу својих координата у дводимензионалној решетки језгра израчунава индекс у вектору токена за оба програма који се пореде. Израчунати индекси означавају почетне токене потенцијалних подударајућих подстрингова. У складу са *Karp-Rabin*-овом модификацијом основног алгоритма, нити најпре пореде хеш вредности. Уколико су хеш вредности једнаке, тек је онда потребно извршити проверу да ли се заиста подударају стрингови на основу којих су израчунате хеш вредности.

Пример организације једног језгра је приказан у Табели 5.5. за два низа токена дужине 8, блок нити димензија 8x8 и MML једнако 3. Сивом позадином су обележене нити које ће детектовати подударање хешева на одговарајућим локацијама и које ће морати да обаве додатну проверу.

Табела 5.5. Пример једног блока димензија 8x8 за језгро 2D организације

		Низ токена А									
		A	D	M	A	T	C	H	R	B	H
Низ токена Б	B	$T_{0,0}$	$T_{1,0}$	$T_{2,0}$	$T_{3,0}$	$T_{4,0}$	$T_{5,0}$	$T_{6,0}$	$T_{7,0}$		
	S	$T_{0,1}$	$T_{1,1}$	$T_{2,1}$	$T_{3,1}$	$T_{4,1}$	$T_{5,1}$	$T_{6,1}$	$T_{7,1}$		
	M	$T_{0,2}$	$T_{1,2}$	$T_{2,2}$	$T_{3,2}$	$T_{4,2}$	$T_{5,2}$	$T_{6,2}$	$T_{7,2}$		
	A	$T_{0,3}$	$T_{1,3}$	$T_{2,3}$	$T_{3,3}$	$T_{4,3}$	$T_{5,3}$	$T_{6,3}$	$T_{7,3}$		
	T	$T_{0,4}$	$T_{1,4}$	$T_{2,4}$	$T_{3,4}$	$T_{4,4}$	$T_{5,4}$	$T_{6,4}$	$T_{7,4}$		
	C	$T_{0,5}$	$T_{1,5}$	$T_{2,5}$	$T_{3,5}$	$T_{4,5}$	$T_{5,5}$	$T_{6,5}$	$T_{7,5}$		
	H	$T_{0,6}$	$T_{1,6}$	$T_{2,6}$	$T_{3,6}$	$T_{4,6}$	$T_{5,6}$	$T_{6,6}$	$T_{7,6}$		
	P	$T_{0,7}$	$T_{1,7}$	$T_{2,7}$	$T_{3,7}$	$T_{4,7}$	$T_{5,7}$	$T_{6,7}$	$T_{7,7}$		
	P										
	T										

Само оне нити које пронађу подударања веће или једнаке дужине од MML могу да свој резултат упишу у резултујући вектор коришћењем одговарајуће уграђене (енг. *intrinsic*) функције за атомично инкрементирање меморијске локације на GPU. Атомичне функције које подржава CUDA извршавају тзв. читај-промени-упиши (енг. *read-modify-write*) операцију над једном 32-битном или 64-битном речи која се може налазити у глобалној или дељеној меморији. Притом се гарантује да ће операција бити извршена унутар једне трансакције и да друге нити неће моћи да приступају одговарајућој адреси док се она не заврши [94]. Атомичне функције које раде над локацијама из глобалне меморије графичког процесора захтевају да се такве локације не кеширају. Стога употреба оваквих операција може да донесе и знатна убрзања, али и значајна успорења програмског кода, све у зависности од случаја коришћења.

У случају који се овде разматра коришћење *atomicAdd* операције се показује као сасвим оправдано, с обзиром да само мали број нити који пронађе одговарајућа подударања дужа од MML знакова заправо уписује резултат. Алтернатива употреби атомичних операције је било коришћење додатног вектора за смештање резултата рада сваке од нити. То се показује као непрактично због тога што велики број резултата поређења није од интереса, а режијски трошкови меморијских трансфера таквог резултујућег низа у CPU контекст се показују као релативно велики.

Као додатна мера оптимизације, резултати се преносе назад у CPU контекст тек након поређења једног улазног програма са свим преосталим програмима, чиме се штеди време које троше меморијски трансфери. Након обраде свих парова програма почиње на CPU друга фаза модификованог GST алгоритма у којој се дефинишу коначни скупови подударајућих подстрингова за све обрађене програмске кодове, као што је описано у поглављу 5.4.1.

5.5. Експериментални резултати и дискусија

Паралелна имплементација експерименталног система за детекцију сличности у програмском коду је евалуирана коришћењем одговарајуће хардверске и софтверске платформе. Резултати тестирања су затим приказани и дискутовани у односу на перформансе секвенцијалне имплементације.

5.5.1. Методологија анализе

У овој секцији је изложена методологија извршене евалуације. Описани су хардверска и софтверска платформа и подаци коришћени за тестирање.

i) Хардверска и софтверска платформа

Евалуација имплементираних алгоритама је урађена на процесору Intel Core i7-5820K 3.30GHz са шест физичких језгара и 16GB RAM меморије. Процесор хардверски подржава покретање до 12 нити захваљујући подршци за симултано вишенитно извршавање на једном језгру (енг. *Simultaneous Multithreading*). На серверу су уграђена два NVIDIA графичка акцелератора: GTX Titan X са 12GB RAM, 3072 CUDA језгра, новије *Maxwell* архитектуре, и Tesla K40c са 12GB RAM, 2880 CUDA језгра, старије *Kepler* архитектуре. На серверу се извршава *Ubuntu Linux* 14.04 LTS оперативни систем, а инсталиран је CUDA *Toolkit* 7.5.

ii) Тест окружење

Тестирање је извршено коришћењем два реална скупа радова са Електротехничког факултета у Београду. Први скуп радова се састоји од 125 студентских радова из предмета Оперативни системи 1 који су написани на програмском језику C++. Радови у просеку садрже око 1000 линија изворног кода. Други скуп радова се састоји од 104 студентска домаћа задатка са предмета Практикум из програмирања 2 написана на програмском језику C. За тестирање су узети радови са четвртог домаћег задатка. Задаци у просеку садрже нешто више од 200 линија програмског кода, а успешно се парсира 99 задатака.

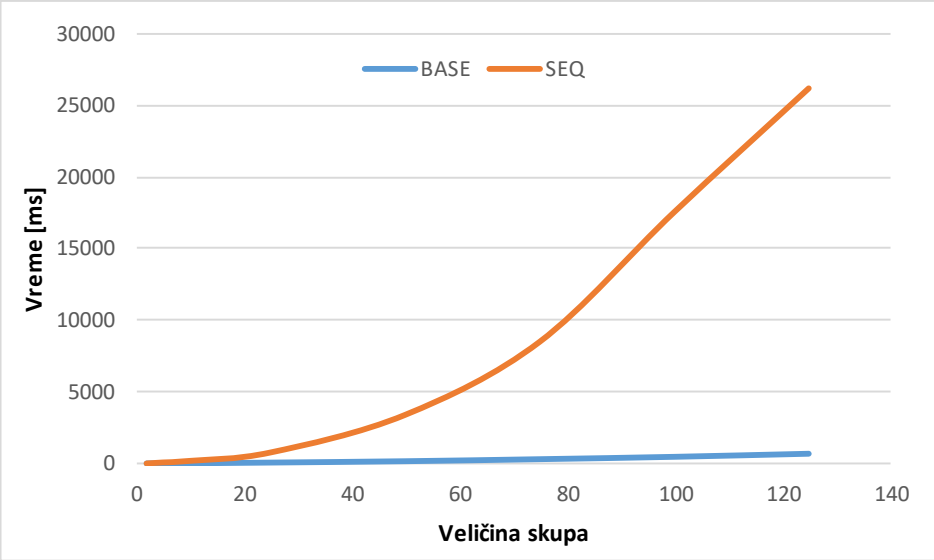
Већина резултата тестирања из оба тест скупа спада у групу од 0-20% сличности при поређењу, што значи да ти радови нису слични или су врло мало слични. Овакав избор тест примера можда није најбољи за тестирање перформанси алгоритма под пуним оптерећењем, јер алгоритам није сувише оптерећен, али добро одражава реалан случај употребе. Стога је тестирање извршено са величинама скупа радова од 2, 10, 25, 50, 75, 100 и 125 (само код OS1), а радови су убацивани у скуп за тестирање од сличнијих ка мање сличним, водећи се резултатима детекције сличности које је дала референтна, секвенцијална имплементација. Такође, изабрана су два скупа радова различите просечне дужине, како би се упоредиле перформансе паралелних имплементација у зависности од просечне дужине улазних датотека.

5.5.2. Анализа резултата

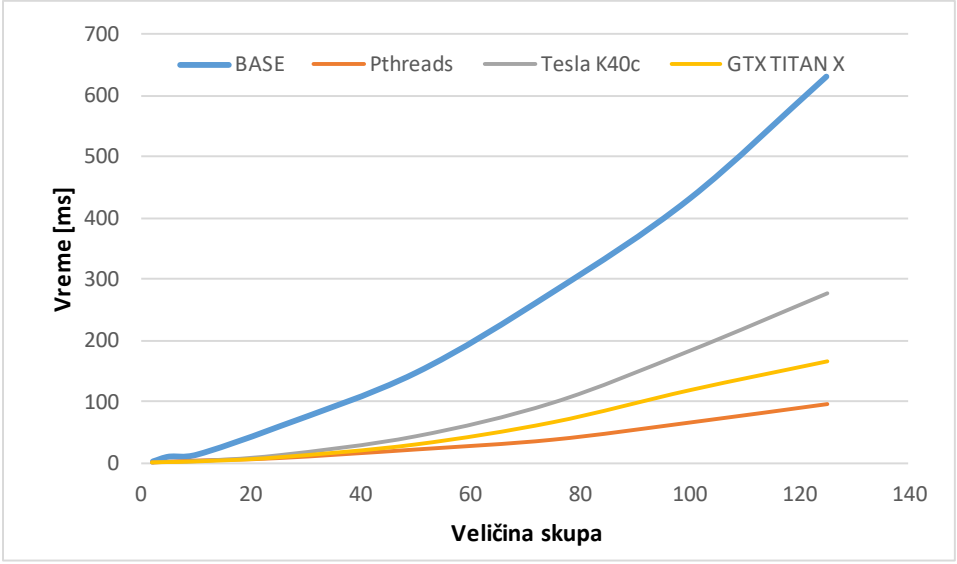
Да би се установиле евентуалне разлике у извршавању и њихов каснији утицај на паралелне имплементације, евалуација је најпре спроведена над референтном имплементацијом и модификацијом RKR-GST алгоритма која је учињена да би алгоритам могао да се паралелизује. Коришћен је OS1 скуп тестова. Резултати поређења су приказани на Слици 5.7.

Познато је да реструктурирање секвенцијалног алгоритма у циљу паралелизације често може да доведе до значајног успорења секвенцијалног извршавања. Такво понашање је уочљиво на Слици 5.7., где време извршавања реструктурираног алгоритма значајно расте са порастом броја парова кодова за поређење. Овакво понашање је, пре свега, резултат поделе GST алгоритма у две фазе.

Због тога се појављује додатна количина посла у другој фази алгоритма, јер ће у скуп подударања у првој фази бити додата и нека подударања која се преклапају и која касније треба да се елиминишу. Међутим, и друга истраживања показују да оптимизовани секвенцијални алгоритми често нису погодни за паралелизацију, па се морају користити једноставнији основни алгоритми [108].



Слика 5.7. Поређење времена извршавања референтне (BASE) и модификоване секвенцијалне (SEQ) имплементације за OS1 скуп тестова



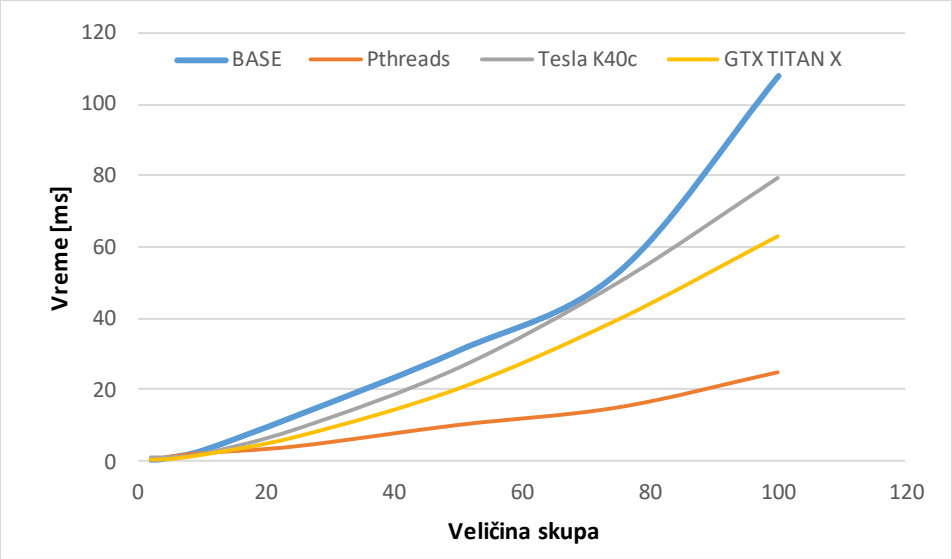
Слика 5.8. Поређење времена извршавања секвенцијалне (BASE), Pthreads и CUDA имплементације за OS1 скуп тестова

Са Слике 5.8. се јасно види да коришћење реструктурираног RKR-GST алгоритма представља оправдан приступ за паралелизацију. *Pthreads* имплементација на централном процесору се за задати скуп улазних података показује и до 7 пута бржом од секвенцијалне имплементације. Добијена убрзања на графичком процесору су нешто мања и добија се у просеку до 4 пута брже извршавање. Примећује се такође да се имплементирани алгоритам брже извршава на новијој, *Maxwell* генерацији графичких картица, пре свега због иновација у архитектури и нешто већем броју доступних CUDA језгара.

Слични резултати се могу приметити и на Слици 5.9. која даје упоредни приказ времена извршавања имплементираних алгоритама за PP2 скуп тестова. Као што је напоменуто, за овај скуп тестова је карактеристично што они представљају кратке програмске кодове. Стога су примећена мања убрзања у односу на претходни случај, превасходно због већег удела режијских трошкова у односу на само поређење. *Pthreads* имплементација се поново показује бржа у односу на CUDA имплементацију.

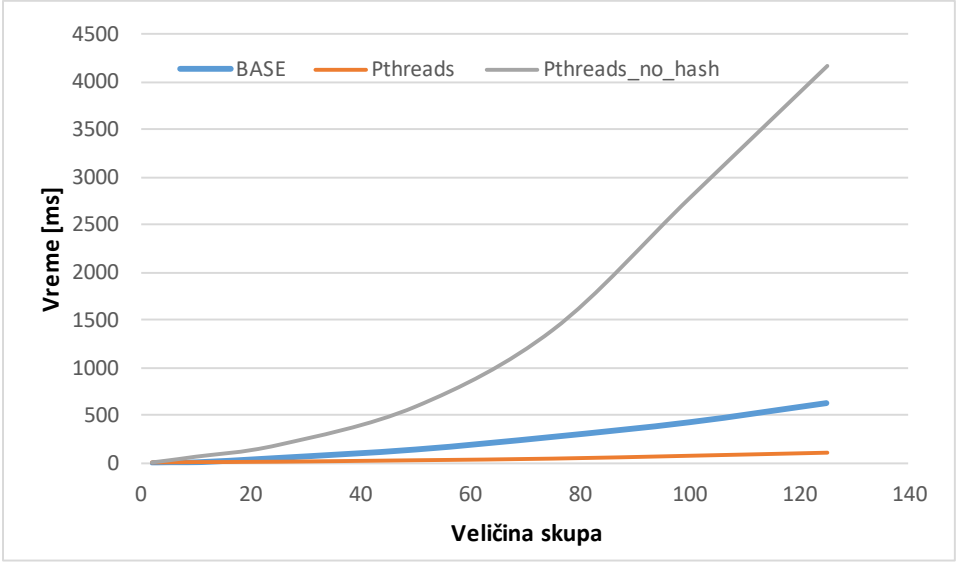
Главни разлог за брже извршавање паралелног алгоритма на централном процесору представљају режијски трошкови извршавања на графичком процесору који се пре свега односе на меморијске трансфере. Као што је раније објашњено, GPU имплементација заправо представља хибридно решење које се делимично извршава на CPU, а делимично на GPU. Због тога је потребно вршити трансфер података, јер централни и графички процесор имају своје одвојене меморијске просторе.

Са друге стране, сам проблем поређења токена нема изражен рачунски интензитет, па не постоји довољна количина рачунског посла у односу на број приступа спорој, глобалној меморији графичког процесора којим би се покрила кашњеа у приступу меморији. Такође, примећено је да је режијско време потребно за позивање језгра једнако времену извршавања језгра, што представља простор за побољшање имплементације ефикаснијим упошљавањем нити.



Слика 5.9. Поређење времена извршавања секвенцијалне (BASE), Pthreads и CUDA имплементације за PP2 скуп тестова

Слика 5.10. приказује утицај коришћења хеширања на време извршавања RKR-GST алгоритма. Упоређене су секвенцијална имплементација и две *Pthreads* имплементације на централном процесору, једна која користи хеширање и друга која га не користи. Резултати ове анализе јасно показују значај хеширања и *Karp-Rabinov*-е оптимизације на ефикасно извршавање GST алгоритма, јер се помоћу хеширања значајно смањује број додатних поређења.



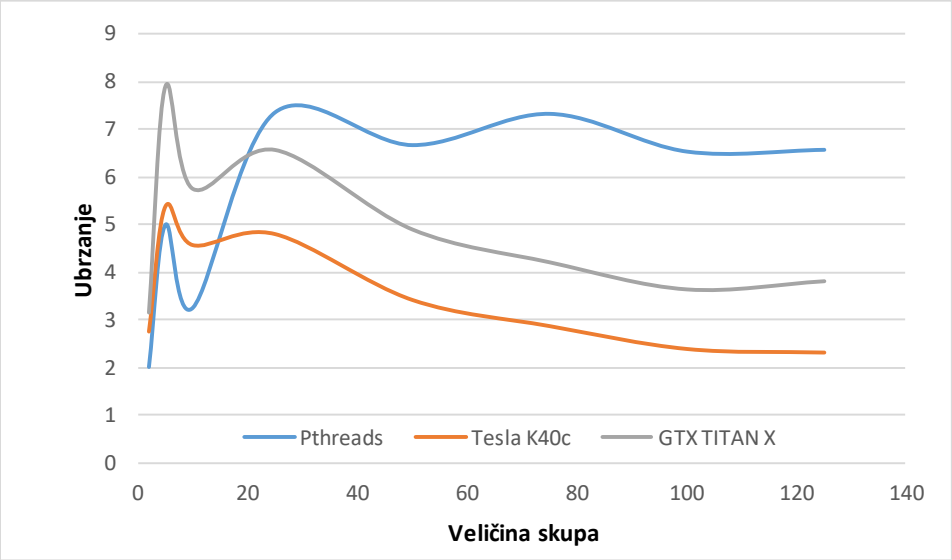
Слика 5.10. Поређење времена извршавања секвенцијалне (BASE), Pthreads и Pthreads имплементације без хеширања за OS1 скуп тестова

Због употребе STL структура података код хеширања, имплементација на централном процесору је ефикаснија у овом сегменту у односу на GPU имплементацију. У тренутку писања овог рада, стабилна имплементација хеш мапа није доступна за графичке процесоре, већ постоје само експерименталне имплементације, попут [109]. Хеш вредности су стога смештане у обичне векторе То је, такође, донело побољшање перформанси, а извршавање језгра је у просеку убрзано за нешто више од 10 процената. Ипак, ефикаснија имплементација корака GPU алгоритма који укључује хеширање оставља простор за даља истраживања.

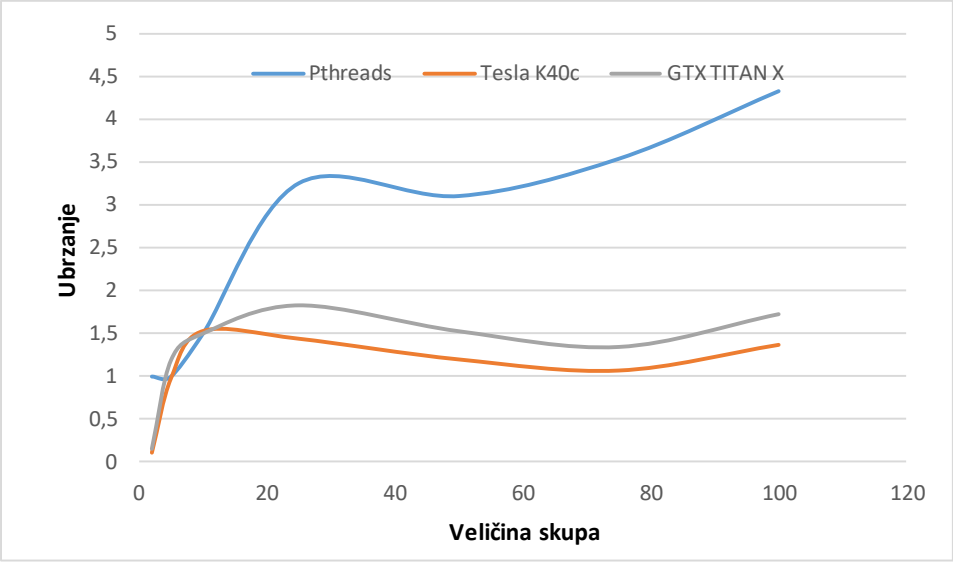
Коначно, на Слици 5.11. и Слици 5.12. су приказана добијена убрзања паралелних имплементација у односу на секвенцијалну имплементацију за оба скупа тестова. Може се приметити да су убрзања релативно константа за већи број парова за поређење. На оба графика се такође примећују варијације убрзања за величину скупа до 25 тестова. Тај узорак тестова није сасвим репрезентативан у смислу равномерности оптерећења, с обзиром да су тестови тако додавани у скуп да они парови са највећом сличношћу буду додати на почетку. Таквих програмских кодова је и у једном и у другом случају релативно мало у односу на укупан број парова за поређење. У PP2 скупу тестова само пет парова програмских кодова показује сличност изнад 30%, док је у OS1 скупу тестова тај број нешто већи са око 30 парова програмских кодова који показују сличност изнад 30%. Сличност велике већине тих парова ипак не прелази 50%.

Имплементација на централном процесору је реализована на нивоу организације послова, па се стога и не могу очекивати велика убрзања када је скуп послова мали. У случају скупова од 2 и 5 кодова, број послова је релативно мали, па чак све нити ни не успеју да добију бар по један посао. Убрзање се стабилно одржава на нивоу од око 7 пута за тест скупове изнад 25 тестова.

Што се тиче имплементације на графичком процесору, она је реализована на нивоу алгоритма. Примећује се такође варијација убрзања са мале скупове тестова, али из другачијег разлога. Главни део посла у оквиру алгоритма се одвија у језгру које је задужено да пронађе сва подударања дужине веће од MML. Резултати профајлирања показују да то језгро конзумира најзначајнији део времена због потребе за приступима спорој, глобалној меморији. Удео времена који конзумирају позиви овом језгру расте са око 70% за два пара програма до 90% за скупове са преко 25 тестова. Након тога убрзање се одржава на приближно константном нивоу од око 4 пута у односу на секвенцијалну имплементацију.



Слика 5.11. Примећена убрзања паралелних имплементација у односу на секвенцијалну имплементацију за OS1 скуп тестова



Слика 5.12. Примећена убрзања паралелних имплементација у односу на секвенцијалну имплементацију за PP2 скуп тестова

6. Примена метода за анализу социјалних мрежа у детекцији плагијаризма

Теорија мрежа је део теорије графова која се специфично бави проучавањем усмерених и неусмерених графова којима се моделују односи између дискретних објеката (ентитета). Једна мрежа се може представити помоћу скупа чворова и грана, где дискретни објекти представљају чворове, а односи се моделују гранама.

Анализа мрежа има велику примену у природним и друштвеним наукама, па се тако користи у физици, хемији, рачунарству, електротехници, биологији, економији, социологији [110, 111]. Са развојем друштвених (социјалних) мрежа на Интернету, као што су *Facebook*, *Twitter* или *Instagram*, све већи значај добијају различите методе за анализу социјалних мрежа. Уобичајено, корисници оваквих мрежа се моделују чворовима, а релације међу корисницима гранама мрежног графа. У случају *Facebook*-а, релација пријатељства је двосмерна, док су на *Twitter*-у и *Instagram*-у релације једносмерне, јер се заснивају на праћењу корисника. За анализу оваквих мрежа се уобичајено користи већи број графовских метрика.

У контексту детекције плагијаризма у програмском коду, резултати се природно могу представити у виду неусмереног, тежинског графа, јер се плагијаризам код универзитетских курсева може посматрати као друштвени феномен који може укључивати већи број актера. Граф као апстракција логички одговара проблему. Сваки студент и његов рад представљају један чвор, док се проценат сличности радова два студента додељује грани која их спаја.

Стога ће ово поглавље бити посвећено примени метода за анализу социјалних мрежа у детекцији плагијаризма у програмском коду. Као што је објашњено у Поглављу 3.1., Калвин и Ланкастер [34] су процес детекције плагијаризма дефинисали у четири фазе: сакупљање програмских задатака, детекција сличности (анализа), потврда сличности и истраживање плагијаризма. У претходним поглављима је нагласак превасходно био на другој фази, јер од квалитетно обављене детекције сличности умногоме зависи крајњи резултат.

У овом поглављу ће нагласак бити стављен на последње две фазе, а социјална мрежа којом се представљају резултати детекције сличности ће бити анализирана одговарајућим методама како би се боље карактеризовале релације између појединачних студената и група студената. Биће размотрене различите метрике које се могу користити да се детектују потенцијални недозвољени облици сарадње међу студентима, као и софтверски алати који омогућавају да се ове метрике израчунају. Осмишљена методологија за детекцију плагијаризма ће бити верификована кроз одговарајуће примере.

6.1. О социјалним и другим мрежама

Иако се у свакодневном говору под термином социјална мрежа углавном мисли на интернет сервисе попут *Facebook*-а, у најширем смислу социјалне мреже моделују односе и интеракције између различитих ентитета (актера). Актери су у том смислу ентитети који представљају појединачне особе, друштвене групе, организације, научне радове и сл. [112]. Често се овакве мреже називају колаборационим мрежама, јер приказују начине сарадње појединих актера.

Корени изучавања социјалних мрежа сежу до тридесетих година двадесетог века, а оно постаје доминантно током осамдесетих година. Типичне примене су у одређивању круга пријатеља или сарадника неке особе, проналажење особа од утицаја на неком пољу, али и откривање начина на који се информације шире унутар неке мреже. Међутим, мрежама се могу моделовати и многи други односи. Неке од карактеристичних примена су:

- одређивање интеракција између различитих честица у физици честица и статистичкој физици [113],
- анализирање протеинских и других структура у биолошким наукама, хемији и фармакологији [113, 114],
- моделовање гена и њихових међусобних повезаности у генетици [114],

- праћење и контрола ширења заразних болести у епидемиологији [115],
- планирање мреже путева, транспорт и логистика [116, 117],
- истраживање и оптимизација рачунарских и телекомуникационих мрежа [118, 119],
- библиометријске и наукометријске анализе којима се анализира научна продукција појединих научника или организација [120, 121].

Анализом мреже може се доћи до информација о организацији саме мреже, односима између њених ентитета, начину ширења информација унутар ње, као и појединачним улогама које имају одређени чворови [122]. Анализа се углавном спроводи коришћењем различитих математичких и статистичких метода које су развијене за наведене потребе. Велику улогу у анализи социјалних мрежа имају мере централности, као што су степен централности, централност по блискости и релациона централност, које се могу користити за одређивање позиције појединих актера унутар мреже. Постоји и велики број других метрика које ће бити детаљније описане у Поглављу 6.2.

Рачунарском анализом оваквих мрежа данас се врше интензивна истраживања како би се остварио напредак на пољима која захтевају обраду велике количине података. Развиле су се и нове научне дисциплине, у којима мреже представљају основну структуру проучавања, попут геномске информатике. У том смислу, графови који представљају модерне социјалне мреже, попут *Facebook*-а или *Twitter*-а, садрже и по неколико стотина милиона чворова, тако да ефикасна имплементација метода за њихову анализу добија на значају, како са становишта временске сложености, тако и са становишта просторне сложености.

Графови који моделују мреже се уобичајено представљају матричном или уланчаном репрезентацијом [123]. Матрична репрезентација је боља у ситуацијама када је потребна динамичка промена броја грана и када је граф густ. Уланчана репрезентација је просторно ефикаснија за ретке графове, а показује се погоднијом када је потребно динамички мењати скуп чворова. Уколико се скуп чворова и грана не мења, често се уланчана репрезентација имплементира коришћењем линеаризованих (статичких) листа суседности.

Што се тиче временске сложености, она углавном зависи од конкретних операција, па је понекад боља једна, а понекад друга репрезентација. Матрице суседности су погодније у алгоритмима који често проверавају постојање одређене гране у графу, док листе суседности дају боље резултате када постоји потреба да се приступи свим суседима одређеног чвора [123]. У контексту анализе социјалних мрежа, алати који се користе за ту намену

употребљавају сопствене формате за складиштење графова (мрежа) на спољашње меморије, а интерна репрезентација се прилагођава потребама анализе.

6.2. Метрике и методе за анализу социјалних мрежа

У овом поглављу ће бити детаљније описане појединачне метрике које се могу користити за анализу социјалних мрежа. Најпре ће бити дате опште мере, а затим ће више пажње бити поклоњено мерама централности и методама за кластеризацију мрежа.

6.2.1. Опште мере

Постоји већи број метрика које се користе да би се описала основна структура графа. Неке од њих као што су величина и густина мреже говоре о повезаности чворова унутар мреже, док друге говоре о положају и утицају појединачних чворова. Такође, одређене метрике се рачунају на нивоу сваког појединачног чвора, док се неке могу израчунати и на нивоу целе мреже. С обзиром да ће ове мере бити примењиване у контексту мрежа добијених детекцијом сличности у програмском коду, углавном ће бити разматране варијанте метрика за неусмерене графове по дефиницијама које су дате у [122, 123].

- **Степен чвора** се односи на број грана које су инцидентне (суседне) посматраном чвору и у директној је вези са централношћу по степену.
- **Достижност** у мрежи показује да ли се из неког чвора мреже могу досегнути преостали чворови. Уколико неки чворови нису достижни у мрежи, то може указивати на одређене поделе унутар ње.
- **Повезаност** чворова дефинише број чворова који треба уклонити на путањи од једног до другог посматраног чвора да би они постали неповезани. Уколико постоји већи број путева између два актера у мрежи, тада су они повезанији.
- **Величина мреже** се односи на укупан број чворова у мрежи. На основу ње се може дефинисати максимални могући степен сваког чвора.
- **Густина мреже** се рачуна као количник укупног броја грана и максималног могућег броја грана у мрежи. Густина мреже указује на степен повезаности мреже, па тако у гушћим мрежама постоји знатно више веза међу чворовима, чиме се омогућава бољи проток информација. Стога, у ретким мрежама боље позиционирани појединци могу да контролишу ширење информација и сл.

- **Фрагментација мреже** се односи на пропорцију парова чворова у мрежи који нису достижни у односу на укупан број парова.

Концепт растојања (дистанци, удаљености) између чворова унутар мреже је такође значајан са становишта разумевања позиције појединих чворова. Из њега се изводи неколико метрика [122, 123].

- **Геодезијска дистанца** представља број грана на најкраћој путањи између два актера у мрежи. Уколико су два актера суседи унутар мреже, онда ова дистанца има вредност један. Код тежинских графова, ова дистанца се може рачунати и као најкраће растојање на путу између два актера у мрежи.
- **Ексцентричност чвора** представља највећу геодезијску дистанцу чвора и говори о удаљености у односу на најдаљи чвор са којим је он повезан.
- **Дијаметар** мреже служи као индикатор удаљености чворова, а једнак је најдужој од свих најкраћих путања у мрежи између парова чворова.

6.2.2. Мере централности

Као што је већ поменуто, мере централности се често користе приликом анализе мрежних графова. Оне су значајне са становишта увида у положај појединачних чворова у односу на њихово окружење или целокупну мрежу, а такође откривају и који су чворови битни за одржавање повезаности мреже. У овом раду ће бити описане три основне мере централности које је у свом раду дефинисао Фримен [124], као и неке њихове варијације које се могу искористити у контексту детекције плагијаризма у програмском коду.

Централност по степену (енг. *degree centrality*) се одређује као број директних суседа сваког чвора у мрежи [124]. Суштински, ова мера одговара степену чвора код неусмерених графова и једнака је броју инцидентних грана том чвору. Математички се дефинише на следећи начин, под претпоставком да је мрежа представљена матрицом суседности A:

$$C_D(p_i) = \sum_{k=1}^{N} A(p_i, p_k) \tag{6.1}$$

где је N укупан број чворова у графу. Уколико је мрежа тежинска, тежине појединачних грана се могу узимати у обзир приликом рачунања централности по степену, чиме више добијају на значају они чворови који имају релације са већом тежином. Уобичајено се сматра да чворови у мрежи са већим бројем релација имају више алтернативних начина да задовоље своје потребе, јер су мање зависни у односу на друге актере. Често се ради поређења са

другим мрежама користе нормализоване вредности централности по степену, процентуално исказане у односу на величину мреже умањену за један (посматрани чвор).

Централност по својственом вектору или Боначићева централност (енг. *eigenvector centrality*) је метрика која узима у обзир и суседство посматраног чвора и представља варијанту централности по степену [125]. Метрика уводи појмове утицајности и моћи. Сматра се да је неки чвор утицајнији уколико његови суседи такође имају велики број суседа, јер порука коју он пошаље може брзо да стигне до великог броја актера. Са друге стране, сматра се да је неки чвор моћнији уколико његови суседи немају велики број својих суседа, јер су зависни од посматраног чвора. Да би се израчунала централност по својственом вектору, мора се најпре одредити својствени вектор који ће садржати скорове релативне централности за сваки чвор:

$$x_i = \frac{1}{\lambda}\sum_{k\epsilon M(i)} x_k = \frac{1}{\lambda}\sum_{k\epsilon G} A(p_i, p_k)x_k \tag{6.2}$$

где је λ нека константа, а A матрица суседности за посматрану мрежу, а $M(i)$ скуп суседа чвора p_i. Сама мера централности се добија проналажењем највеће одговарајуће сопствене вредности λ, што се врши даље итеративним поступком [126].

Бета централност [122] је варијанта централности по својственом вектору која уводи додатан бета параметар за пондерисање који узима вредности у опсегу [-1, 1]. Позитивне вредности бета параметра истичу чворове чији суседи такође имају велики број суседа и тиме их чине утицајнијим, док негативне вредности параметре истичу чворове чији суседи сами немају много суседа. Проблем ове метрике је релативно комплексно срачунавање својствених вредности и финалних скорова за сваки чвор.

Централност по блискости (енг. *closeness centrality*) се израчунава за сваки чвор као просечна удаљеност чвора од свих осталих чворова у графу са којима је повезан [124]. Уобичајено се дефинише као реципрочна вредност суме најкраћих растојања од посматраног чвора до осталих са којима је повезан која је нормализована величином мреже умањеном за један:

$$C_C(p_i) = \frac{N-1}{\sum_{k=1}^{N} d(p_i, p_k)} \tag{6.3}$$

где $d(p_i, p_k)$ представља најкраће растојање између посматраног чвора и неког од преосталих у мрежи са којима је повезан, а N укупан број чворова у графу. Сматра се да су чворови са

већом вредношћу ове мере на централнијим позицијама у мрежи, јер им је у просеку потребно мање корака да дођу до преосталих чворова у мрежи.

Релациона централност (енг. *betweenness centrality*) се одређује за сваки чвор и пропорционална је броју најкраћих путева између свих осталих парова чворова на којима се посматрани чвор налази у односу на укупан број таквих најкраћих путева [124].

$$C_B(p_i) = \sum_{j=1}^{N} \sum_{k=1}^{j-1} \frac{g_{jk}(p_i)}{g_{jk}} \tag{6.4}$$

где g_{jk} представља укупан број најкраћих путева који повезују чворове p_j и p_k, $g_{jk}(p_i)$ представља број таквих путева који укључују чвор p_i, а N укупан број чворова у графу. У анализи социјалних мрежа је ова мера уведена како би се квантификовала контрола комуникације унутар мреже од стране појединих актера [124]. Чворови са вишим вредностима релационе централности се налазе на већем броју најкраћих путања и самим тим представљају „мостове“ или посреднике између осталих чворова у мрежи.

Централизација мреже се рачуна на нивоу целе мреже и може се израчунати за било коју меру централности. Она показује варијансу израчунате мере посматране мреже процентуално у односу на мрежу исте величине која има топологију звезде. Мрежа са топологијом звезде поседује један централни чвор са којим су повезани сви остали чворови и који немају других грана. У таквој мрежи, расподела моћи је највише неједнака, па већи проценат централизације неке мреже говори да у њој постоји одређен број чворова који су у бољој позицији у односу на остале.

6.2.3. Кластеризација мреже

У претходним поглављима су детаљније описане метрике које се могу користити за карактеризацију комплетне мреже или појединачних чворова у смислу њихове повезаности, позиције у мрежи и могућности за размену информација. Други приступ за анализу социјалних мрежа се заснива на проучавању њихове структуре и идентификацију одговарајућих подструктура (кластера, група). Основне јединице у оквиру мреже су дијаде и тријаде. Дијаду чини пар актера који могу бити повезани граном, док тријаду чине три актера и њихове могуће везе. За проучавање структуре мреже су од интереса дијаде и тријаде код којих постоји повезаност међу чворовима.

Проучавање структуре мреже омогућава издвајање гушће повезаних подструктура и самим тим боље разумевање понашања мреже у целини. Такође, оваква анализа омогућава

бољи увид у позицију појединачних чворова. Поједини чворови могу представљати мостове између различитих група, али исто тако бити повезани само са члановима своје групе. Са друге стране, одређени чворови могу бити потпуно изоловани од остатка мреже, што онда утиче на њихово понашање.

У литератури се често наводе две мере за процену структуре мреже. Прва мера је **коефицијент кластеризације** [127, 128] који може да се срачуна за појединачан чвор и комплетну мрежу. Коефицијент кластеризације за појединачан чвор се рачуна као густина мреже коју чине посматрани чвор, његови суседи и њихове међусобне везе. Таква мрежа се често назива *ego* мрежом посматраног чвора. Степен кластеризације комплетне мреже се рачуна на основу просека појединачних чворова.

Друга често коришћена мера за карактеризацију структуре мреже је **модуларност** [129]. Модуларност је мера квалитета партиционисања чворова мреже у одговарајуће кластере. Графови са већом модуларношћу су подељени на кластере који су, унутар себе, густо повезани, а немају велики број веза са остатком графа. Математички се дефинише као однос броја грана у одређеним кластерима у односу на укупан број грана умањен за исти такав однос који би се добио када би гране између чворова биле распоређене на случајан начин.

Постоји већи број приступа за идентификацију група у оквиру мреже. Они се грубо могу поделити на приступе који граде веће групе полазећи од мањих градивних јединица одоздо на горе (енг. *bottom up* приступ) и приступе који полазе од комплетне мреже и покушавају да изврше хијерархијску поделу одозго на доле (енг. *top down* приступ). Такође, коришћени приступ умногоме зависи од дефиниције подструктуре у оквиру мреже. Стога ће у наставку текста бити дате најчешће дефиниције подструктура.

Клика (енг. *clique*) је подграф посматраног графа који чине чворови који су много чвршће повезани међусобно, него што су повезани са остатком мреже. Према строгој дефиницији, једна клика се састоји од највећег могућег скупа чворова који су међусобно повезани сваки са сваким, односно представљају максималан комплетан подграф посматраног графа. У том смислу, у оквиру једног графа може бити идентификовано више клика, а оне се могу међусобно преклапати, односно један чвор може припадати већем броју клика. Тежине грана не играју улогу у одређивању клика.

За многе примене је дефиниција клике превише строга, јер намеће постојање директне гране између свих чланова клике. Стога се врше одређене релаксације наведене дефиниције, тако да се у групу укључи већи број чворова. ***N*-клика** је скуп чворова који су међусобно повезани било директно, било путевима дужине *N*, где *N* обично узима вредност два. Појам ***N*-клана** додатно прецизира да припадници клике морају бити повезани преко неког другог члана клике, што значи да унутар клике не постоји пут дужи од *N*.

Алтернативни приступ за релаксацију дефиниције клике је коришћен код ***K*-плексова** и ***K*-језгара**. *K*-плекс се дефинише као скуп чворова код којег појединачни чворови имају везе са свим чворовима, осим неких *K* чворова. Дефиниција *K*-језгра подразумева постојање најмање *K* веза ка осталим члановима групе и представља додатну релаксацију.

Повезана компонента графа представља максимални скуп чворова (подграф) таквих да постоји пут између било која два чвора. Овакве компоненте се често називају слабо повезане компоненте. Уколико постоје изоловани чворови, они такође представљају засебне компоненте. Код усмерених графова се разликују термини слабо и јако повезаних компоненти, где јако повезану компоненту чине чворови код којих постоји достижност између било која два чвора компоненте. **Број повезаних компоненти** се односи на укупан број слабо повезаних компоненти и понекад се користи као метрика за опис мреже.

Повезане компоненте се могу дефинисати и за тежинске мреже. Тада се подаци о тежини гране користе да би се формирале повезане компоненте. Алгоритми уобичајено полазе од грана највеће тежине, а онда итеративно смањују цену грана док год се не добије само једна компонента. Резултати се уобичајено приказују хијерахијски у виду дендограма. За разлику од до сада описаних подструктура, повезане компоненте се одређују приступом одозго на доле.

Сличан приступ за одређивање повезаних компоненти на основу тежина појединачних грана је дао Грановетер [130] у својој теорији о јаким и слабим везама у социјалним мрежама. Грановетер је код тежинских мрежа дефинисао три врсте веза: јаке везе (енг. *strong ties*), слабе везе (енг. *weak ties*) и неприсутне везе (енг. *absent ties*). Јаке везе су све оне везе изнад одређеног прага *s*, неприсутне везе све оне испод неког прага *w*, а слабе све оне између. Праг *w* дефинише истраживач, док се праг *s* одређује алгоритамски.

Сам приступ за одређивање компоненти се заснива на особини слабе транзитивности. Сматра се да је тријада чворова слабо транзитивна уколико постојање јаких веза између

парова чворова *x, y* и *y, z* имплицира постојање слабе везе *x, z*. На основу описаних особина се одређује вредност *s* као најмања тежина јаке везе, таква да постоји слабо транзитивна тријада код које слаба веза има тежину већу од *w*. Затим се граф претвара у бинарни на основу вредности *s* и одређују повезане компоненте.

На крају, у контексту одређивања кластера, треба поменути и алгоритме за детекцију комуна у социјалним мрежама који итеративним поступком долазе до одговарајућих кластера. Најпознатији хијерархијски метод представља *Girwan-Newman* алгоритам [131] који прогресивно уклања гране у графу на основу њихове релационе централности и као крајњи резултат производи дендограм са поделом. Алтернатива овом приступу су алгоритми засновани на оптимизацији модуларности унутар кластера [129, 132]. Један такав алгоритам, *Louvain* метод [133], је основа за одређивање кластера у програмском пакету Gephi који је коришћен за визуелизацију у оквиру овог рада.

6.3. Софтверски алати за анализу социјалних мрежа

С обзиром на сложеност израчунавања описаних метрика за анализу социјалних мрежа, њихова анализа се уобичајено врши кроз одговарајуће алате. У овом поглављу ће бити представљено неколико таквих алата, а њихов упоредни преглед је дат у Табели 6.1. Детаљна упоредна анализа алата за анализу социјалних мрежа по различитим критеријумима је извршена у [134], али значајни алати као што су UCINET и NodeXL нису размотрени.

Табела 6.1. Преглед функционалности разматраних алата за анализу социјалних мрежа

Алат	Плат-форма	Про-ширив	Улазни формати	Излазни формати	Методи анализе	Кластеризација	Лиценца
UCINET	Windows	Не	DL, Excel, VNA, Pajek .NET, Text	DL, Excel, Pajek .NET, Mage, Metis, VNA	Мере централности, *ego* мреже, кохезија	Хијерархијска *Girwan-Newman*, *Markov*, анализа компоненти	Комерц.
Pajek	Windows	Не	Pajek .NET, .DAT	Pajek .NET	Мере централности, *ego* мреже, кохезија, детекција комуна	*Louvain, VOS*	Беспл.
Gephi	Java	Да	Gephi, CSV, GEXF, GDF, GML, GraphML, Pajek .NET, DOT, DL, Tulip	Gephi, CSV, GEXF, GDF, GML, Pajek .NET, VNA DL	Основне мере централности и графовске статистике	*Louvain* метод	Беспл.
NodeXL	MS Excel	Да	GraphML, DL, Excel, *Facebook, Twitter, Youtube*	GraphML, Excel, CSV	Основне мере централности и графовске статистике, *Page rank*	*Girwan-Newman,Clauset-Newman-Moore, Wakita-Tsurumi*	Basic (беспл.), Pro (комерц.)

6.3.1. UCINET

Софтверски пакет *UCINET* [135] је један од најпознатијих алата за анализу социјалних мрежа. Алат нуди велики број алгоритама и метрика за анализу графова које укључују статистичку анализу, мере централности, идентификацију подгрупа и улога. Подржан је рад и са социоцентричним и егоцентричним подацима који се могу прочитати из великог броја улазних формата. Као додатак овом софтверу постоји алат *NetDraw* за визуелизацију графова. Из наведених разлога он је изабран као главни алат за анализу мрежа у оквиру овог рада.

Највећи недостатак му је то што је у питању комерцијални производ уз могућност ограничене пробне употребе у трајању од 90 дана. Алат није могуће самостално проширивати.

6.3.2. Pajek

Алат *Pajek* је бесплатна алтернатива за исцрпну анализу социјалних мрежа [136]. Аутори тврде да је програм способан да обрађује велике мреже са више милиона чворова и да је обрада само ограничена количином слободне оперативне меморије.

Pajek, такође, омогућава израчунавање великог броја метрика кроз рад са шест врста објеката: графовима, партицијама, векторима, кластерима, пермутацијама и хијерархијама. Међутим, велики број корисника сматра да алат има неинтуитиван кориснички интерфејс и да је компликован за коришћење. Такође, он има подршку за визуелизацију мрежа.

6.3.3. Gephi

Gephi је један од најпознатијих бесплатних алата за анализу и визуелни приказ графова, развијен на Универзитету у Компјењу, у Француској, 2009. године [137]. *Gephi* нуди велики број опција за визуелизацију графова и манипулацију њиховом структуром, а подаци се могу увозити из различитих графовских формата.

Анализа мреже се може спровести кроз различите алгоритме, као што су одређивање просечног степена чворова графа, централности чворова и повезаних компоненти, вршење кластеризације, рачунање различитих растојања, густине графа и сл. На основу атрибута графа и израчунатих вредности се може применити већи број филтера и вршити распоређивање чворова на екрану. Ипак, број и сложеност метрика које *Gephi* може да

израчуна су релативно ограничени, па је овај алат пре свега погодан за коришћење у домену визуелизације мрежа о чему ће бити више речи у Поглављу 8.

Gephi је писан у програмском језику Java, а нове могућности се додају са изузетном лакоћом, захваљујући опширној документацији доступној на интернету и великој заједници програмера. Готово сваки аспект програма може се модификовати, па се тако могу додати нове метрике, филтери, начини прегледа графа, подршка за увоз нових графовских формата, као и подршка за аутоматско генерисање графа на било који начин.

6.3.4. NodeXL

Алат *NodeXL* [138] је развијен као додатак за *Microsoft Excel*, првенствено за визуелизацију и анализу друштвених мрежа помоћу графова. Чињеница да се заснива на алату који већина корисника рачунара врло добро познаје и могућност за аутоматско повезивање са друштвеним мрежама су довеле до његове популарности. Граф може бити креиран ручним уношењем података у *Excel* табелу, али и аутоматизовано, увозом података из графовских формата или директно са друштвених мрежа као што су *Twitter*, *YouTube* и *Facebook*.

Анализа графа је могућа кроз већи број статистичких прорачуна као што су просечна тежина грана, просечан степен чворова, кластеризација графа, као и статистике уско повезане са друштвеним мрежама. Иако представља одличан спој алата за анализу и визуелизацију графова, у основној верзији која је бесплатна је могуће користити само мали подскуп метрика, што му уједно представља и највећи недостатак.

Како је цео алат развијен у програмском језику C#, могуће је писати одговарајућа проширења. Од проширења, могуће је стварати адаптере за прихватање нових улазних формата, нове начине за аутоматско распоређивање чворова на екрану, примењивати алгоритме за анализу графова и сл. Ипак, због интеграције са *Excel*-ом, ово није превише интуитивно.

6.4. Постојећи приступи за детекцију плагијаризма помоћу анализе социјалних мрежа

Начини и облици дозвољене и недозвољене сарадње међу студентима су анализирани са социјалног аспекта у неколико студија. Студија [43] се бави истраживањем релација између студената који размењују програмски код, а детекција плагијата је коришћена као

алат за откривање тих социјалних релација. Студија тврди да студенти не траже савете око решења насумично. У ситуацијама када се решења задатака не могу набавити на интернету или из сличних извора, студенти траже решења од својих колега са којима имају пријатељске односе, од оних у које имају поверење, али и од оних о којима имају добро стручно мишљење. Релације између студената су идентификоване помоћу анкете, на основу чега је даље конструисана социјална мрежа.

Детекција плагијаризма је извршена помоћу LSI методе која је позната код поређења текстова. Добијени резултати су међусобно корелисани, а аутори сугеришу да се познавање друштвених односа може употребити од стране наставника током процеса учења како би се успоставили правилни обрасци понашања и спречила појава плагијата. Међутим, аутори нису користили никакве метрике да карактеришу саму мрежу, већ су се више фокусирали на проучавање појединачних релација између студената и резултата анкете.

Слична студија је обављена и у Словенији [139, 140] у оквиру које су аутори покушали да мрежу која је добијена на основу резултата алата за детекцију плагијаризма обогате додатним семантичким информацијама. Аутори су имплементирали веб трагач и уз пристанак студената извршили прибављање и анализу релација познанстава и праћења корисника на друштвеним мрежама. Познанство је двосмерна релација, док је праћење једносмерна релација. Анализирани су познанства студената на социјалним мрежама *Facebook* и *LinkedIn*, праћење на мрежама *Twitter* и *Google+* и резултати са општенаменских претраживача, као што су *Google*, *Bing* и *Yahoo*.

На основу добијених података је семантички обогаћена социјална мрежа и извршена визуелизација резултата. Основна идеја аутора је да ће коришћењем информација из такве мреже спрегнутих са резултатима детекције сличности која се обавља неким од стандардних алата смањити обим посла у последње две фазе детекције плагијаризма. Такође, успешна анализа повезаности корисника на социјалним мрежама на пољима као што је откривање превара са аутомобилским осигурањем [141], сугерисала је да је таква анализа могућа и у детекцији плагијаризма.

Имплементирани алат агрегира све добијене информације и за сваки пар сличних радова генерише финални скор у који улазе и подаци о повезаности на социјалним мрежама. Статистичка евалуација у којој је учествовало 76 студената курса програмирања је показала успешност имплементираног приступа и његову ефикасност у детекцији плагијаризма.

Иако досадашњи приступи који користе технике за анализу социјалних мрежа у детекцији плагијаризма показују делотворност и корисност оваквог начина интерпретације резултата детекције сличности, ниједан од приступа није користио нумеричку карактеризацију самих мрежа, нити било какве метрике које се стандардно користе за евалуацију и анализу социјалних мрежа на другим пољима. Стога је у оквиру овог рада примењен комплементарни приступ који поред представљања резултата детекције сличности путем графа врши и одређену нумеричку карактеризацију саме мреже, како би се лакше открили и разумели шаблони колаборације и открио плагијаризам.

6.5. Експериментална евалуација

У овом поглављу је извршена експериментална евалуација изложених метрика и метода за анализу социјалних мрежа у детекцији плагијаризма у програмском коду. Да би се боље разумео контекст у коме анализа социјалних мрежа може помоћи у детекцији плагијаризма, издвојене су четири карактеристичне шеме колаборације уочене у постојећој дугогодишњој пракси међу студентима.

- Изоловани пар студентских радова (у даљем тексту: PAIR) чине два програмска кода са израженом детектованом сличношћу, без веза ка другим радовима. Ова шема колаборације се једноставно уочава и у изворним резултатима које дају алати за детекцију сличности у програмском коду.
- Група сличних радова са израженим централним чвором (у даљем тексту: STAR). Типично, у оваквим ситуацијама један студент представља извор плагијаризма, било слањем програмског кода другим колегама или јавним објављивањем кода на неки други начин. Често је централни чвор актер који је извор плагијаризма, али уколико је у таквој групи сличних радова једна од веза између чворова израженија и укључује централни чвор, онда то може бити и чвор на „периферији“. То је најчешће у случајевима када су сам оригинални извор и дистрибутер две различите, али интензивно повезане особе.
- Група сличних радова без израженог централног чвора (у даљем тексту: GROUP). Оваква ситуација често значи да извор плагијаризма није доступан из неког разлога. Један од разлога може бити заједнички рад студената, где сви онда предају то заједничко решење или решења са заједничким деловима. Са становишта академског поштења је много негативнија ситуација у којој треће лице са

надокнадом или без ње пише или уступа решења групи студената који их онда могу до неке мере модификовати и предати као своје дело. Уобичајено, такви радови ће бити повезани у кластер налик клики.

- Појединачан студент проналази и комбинује програмски код из више извора да би дошао до решења сопственог задатка (у даљем тексту: MIX). Типично, овакви чворови су релативно издвојени и налазе се на периферији својих кластера са тенденцијом повезивања са чворовима из различитих кластера.

Поставља се питање кроз које метрике и методе за анализу социјалних мрежа је могуће препознати наведене ситуације и како се оне карактеришу. Такође, поставља се питање да ли је у одређеним ситуацијама могуће одредити извор плагијаризма или сузити скуп чворова на које се сумња да могу бити извор, како би се четврта фаза детекције плагијаризма, истраживање плагијаризма, учинила ефектнијом.

6.5.1. Методологија анализе

За евалуацију је коришћена колекција радова студената са курсева Оперативни системи 1 и Практикум из програмирања 2 на Електротехничком факултету Универзитета у Београду. У првој колекцији се налазило 447 радова написаних на програмском језику C++ са просечно 1000 линија програмског кода, док се у другој колекцији налазило 142 рада са петог домаћег задатка са просечно 200 линија кода, написаних на програмском језику C. Радови из обе колекције су обрађени системом Moss и генерисане су HTML странице са резултатима.

Подсећања ради, систем Moss обрађује све парове улазних датотека, али као релевантне резултате враћа 250 најсличнијих парова радова, док остале одбацује. Самим тим и резултујуће мреже ће имати мање чворова од укупног броја обрађених радова. То не представља озбиљно ограничење у анализираним случајевима, јер велики број парова радова има релативно ниску сличност, што ће бити дискутовано у анализи резултата. На основу ових страница је вршено парсирање резултата и генерисање графова и одговарајућих улазних датотека за алате UCINET и Gephi који су примарно коришћени у анализи. У анализи су најпре наведене основне карактеристике анализираних мрежа, а затим евалуиране појединачне метрике. Посебан акценат је стављен на мере централности и њихово тумачење у контексту мрежа насталих на основу резултата детекције сличности.

6.5.2. Анализа резултата

Основни параметри разматраних мрежа су приказани у Табели 6.2. OS1 мрежу чини 190 чворова, док PP2 мрежу чини 110 чворова. Као што се може приметити, обе резултујуће мреже су доста ретке. То је сасвим логично, с обзиром да у реалним ситуацијама појава плагијаризма треба да буде изузетак, а не правило и у складу са тим не треба очекивати велики број сличних радова, нарочито не сваког рада са сваким другим. Ранија истраживања су показала [28, 90] да проценат ове појаве ретко прелази 5% од укупне студентске популације, па је у складу са тим и очекивано да таква мрежа буде ретка.

Фрагментираност мреже показује супротне тенденције за две посматране мреже. Код OS1 мреже се показује изразита фрагментираност, док је у PP2 мрежи она ниска. То показује и број детектованих слабо повезаних компоненти. У OS1 мрежи је детектовано 47 слабо повезаних компоненти, од чега се 33 компоненте састоје само од пара чворова. У PP2 мрежи постоје само 4 такве компоненте.

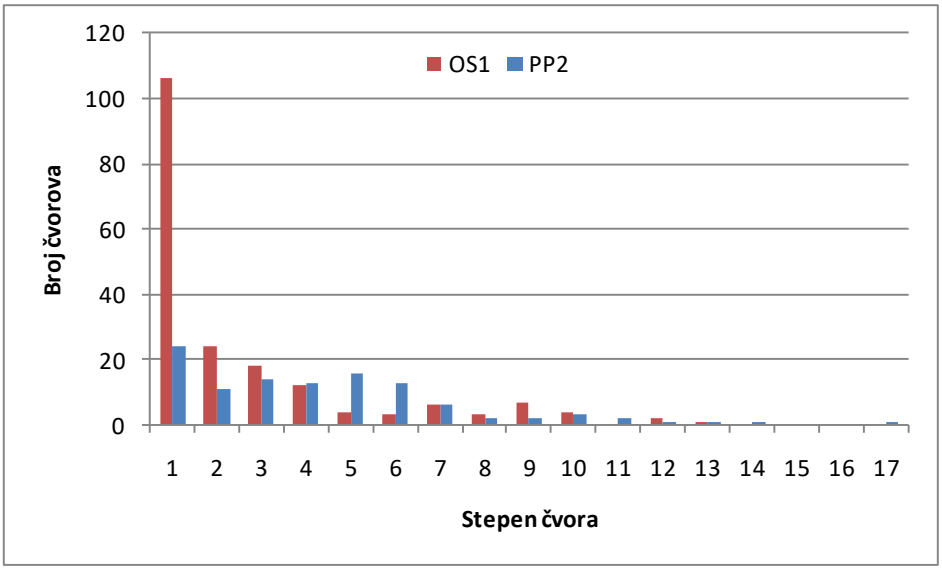
Генерално, у мрежама које су настале од резултата детекције сличности би се могло очекивати да буду изразито фрагментиране. С обзиром да је плагијаризам појава која најчешће обухвата мање, затворене групе сарадника, не може се очекивати да велики број чворова буде међусобно достижан. Међутим, у случају PP2 мреже показује се да је велики број грана са сличношћу испод 10 процената и да је та сличност последица исте тематике проблема који су студенти решавали (уланчане листе и датотеке). Стога постоје одређене, мале сличности због природе проблема и коришћења типског кода. Са друге стране, на сваком домаћем задатку из предмета PP2 има више група задатака, па су стога сличности генерално мање. Може се рећи да и од поставке задатка много зависи степен добијене сличности, с обзиром да наставници понекад дефинишу детаље имплементације, као што су костур решења, декларације функција, класни дијаграм и сл., које подижу сличност.

Табела 6.2. Основни параметри разматраних мрежа и њихових варијанти (SIZE – величина мреже, DENS – густина мреже, AVG DEG - просечан степен чвора мреже, MAX DEG – максимални степен чвора мреже, DIAM – дијаметар мреже, FRAG – фрагментација мреже, MIN EDG – минимална тежина гране у мрежи , MAX EDG – максимална тежина гране у мрежи, NWC – број слабо повезаних компоненти, CC – коефицијент кластеризације)

Скуп података	SIZE	DENS	AVG DEG	MAX DEG	AVG DIST	DIAM	FRAG	MIN EDG	MAX EDG	NWC	CC
OS1	190	0,014	2,63	13	4,29 ± 2,3	12	0,858	0,04	0,76	47	0,43
PP2	110	0,04	4,38	17	3,56 ± 1,27	9	0,106	0,02	0,27	4	0,32

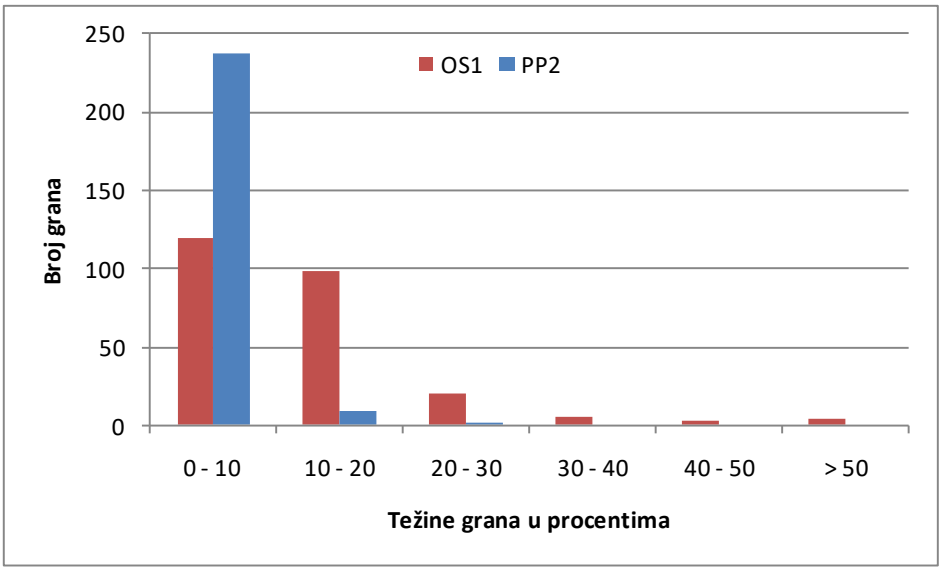
Коефицијент кластеризације код OS1 износи 0,43, што је, имајући у виду природу мреже, релативно висока вредност. Она говори да чворови са већим бројем веза теже ка стварању клика, што онда може указивати на затворене групе студената у оквиру којих се размењује код. Дијаметри прве и друге мреже су слични, а просечна геодезијска дистанца у смислу броја грана између парова чворова је око 4, што указује на релативно мали број корака потребних да би се из једног чвора дошло до другог у обе мреже.

Минимални степен чворова у оба графа је један и тривијално га је одредити, јер чворови који учествују у PAIR случају сличних радова морају имати једну грану. Максимални степен чвора је 13 у OS1 мрежи, док је 17 у PP2 мрежи, а просечан степен 2,63, односно 4,38, што указује на то да је највећи број чворова у вези са релативно мало преосталих чворова. Дистрибуција чворова по степену чвора је приказана на Слици 6.1.



Слика 6.1. Расподела чворова разматраних мрежа по степену чвора

Највећи степен сличности, који уједно представља и највећу тежину гране мреже, износи код OS1 мреже - 0,76, док најмања тежина гране износи 0,04. У PP2 мрежи је сличност у значајно мањем опсегу, између 0,02 и 0,27. Расподела тежина грана за обе мреже у опсезима од по 10% је дата на Слици 6.2. Може се приметити, да највећи број грана има тежину мању или једнаку 0,1 што најчешће указује на случајну сличност између чворова или сличност која је последица сличног нивоа знања, употребе дозвољеног, типског кода и сл. Такође, велики број грана има сличност у опсегу од 0,1 до 0,2.



Слика 6.2. Расподела грана мреже по тежини у опсезима од по 10%

С обзиром да претходна истраживања [28, 105] показују степен сличности од преко 50% указују на плагијаризам комплетног кода, а да ниже вредности у распону од 20 до 50 процената указују на делимични плагијаризам, од интереса је окарактерисати и мреже добијене од резултата детекције сличности уз примену одређеног прага сличности. Стога је таква анализа спроведена за OS1 скуп података уз праг сличности постављен на 10, 20 и 30 процената. За PP2 скуп радова коришћен праг сличности од 10 и 15 процената, јер грана највеће тежине има вредност 0,27. Резултати су приказани у Табели 6.3., а одговарајући тест скупови су именовани тако што је на име предмета додат и коришћени праг.

Може се приметити да густина мрежа добијених коришћењем прага сличности, иако нешто већа, и даље остаје изразито ниска. Коефицијент кластеризације има тенденцију раста, јер се уклањањем грана са ниском сличношћу и одговарајућих чворова, повећава могућност стварања клика, због мањег броја потребних веза међу чворовима. Примећује се да га за скупове OS1-30 и PP2-15 није могуће израчунати, јер у мрежама скоро да не постоје чворови степена два (осим по једног), већ само парови чворова. Фрагментација мрежа углавном расте, што је у складу са објашњењем датим на почетку поглавља. Преостале метрике, као што су величина мреже, просечан и максимални степен чворова, просечне дистанце и дијаметар значајно опадају.

Табела 6.3. Основни параметри разматраних мрежа и њихових варијанти (SIZE – величина мреже, DENS – густина мреже, AVG DEG - просечан степен чвора мреже, MAX DEG – максимални степен чвора мреже, DIAM – дијаметар мреже, FRAG – фрагментација мреже, MIN EDG – минимална тежина гране у мрежи , MAX EDG – максимална тежина гране у мрежи, NWC – број слабо повезаних компоненти, CC – коефицијент кластеризације)

Скуп података	SIZE	DENS	AVG DEG	MAX DEG	AVG DIST	DIAM	FRAG	MIN EDG	MAX EDG	NWC	CC
OS1	190	0,014	2,63	13	4,29 ± 2,3	12	0,858	0,04	0,76	47	0,43
OS1-10	120	0,018	2,16	11	4,64 ± 2,76	11	0,837	0,1	0,76	32	0,42
OS1-20	50	0,026	1,294	3	1,27 ± 0,54	3	0,966	0,21	0,76	21	0,59
OS1-30	25	0,043	1,04	2	1,07 ± 0,25	2	0,95	0,31	0,76	12	/
PP2	110	0,04	4,38	17	3,56 ± 1,27	9	0,106	0,02	0,27	4	0,32
PP2-10	14	0,11	1,42	4	1,375 ± 0,48	2	0.824	0,11	0,27	5	0,54
PP2-15	7	0.19	1,43	2	1,2 ± 0,4	2	0,76	0,16	0,27	3	/

За број повезаних компоненти закључак није сасвим очигледан. С обзиром на почетну високу фрагментираност OS1 мреже, увођење прага уклања PAIR случајеве са малом сличношћу, па стога број слабо повезаних компоненти пада. Међутим, због раније објашњеног ефекта типског кода, увођење прага код PP2 мреже разбија велику повезану компоненту, па се добија једна компонента више. С обзиром да је иницијални број повезаних компоненти био изузетно мали у односу на величину мреже, то је добро, јер даје финију кластеризацију радова на групе које треба испитати.

Са становишта онога ко спроводи детекцију плагијаризма, свакако је од значаја смањивање броја сличних парова за мануелну инспекцију. У складу са тим, увођење прага сличности може да помогне у томе, али и да истакне делове мреже који су од значаја за даље испитивање плагијаризма. Тешко је рећи који је праг сличности најпогоднији за употребу, али праћење карактеристика мреже, као што су величина мреже, просечан степен чвора, фрагментација и број повезаних компоненти, може да помогне.

Да се не би изгубиле информације о кластерима чворова који указују на групни плагијаризам (типови STAR, GROUP і MIX) просечан степен чвора не би требало да падне значајно испод 1,5. Да би се у мрежи задржале такве групе повезаних чворова, у њој мора остати и један број чворова са степеном већим од један. Фрагментација би требало да остане стабилно висока. Величина мреже свакако не би требало да падне на испод 5% од укупног броја обрађених радова, јер се сматра да је преваленција плагијаризма у академском

окружењу између 5 и 10 процената студената [28, 90], па се онда могу изгубити важне информације уклањањем превеликог броја чворова.

Мере централности такође могу да се искористе као помоћно средство у детекцији плагијаризма, а посебно да се ојачају аргументи у четвртој фази детекције, истраживању плагијаризма. Табела 6.4. и Табела 6.5. дају преглед коришћених метрика. За OS1 скуп података су приказани резултати за 26 највише рангираних чворова по паровима сличности, односно приказани су резултати и за све чворове који учествују у гранама са тежином већом или једнаком 0,2. За PP2 скуп су приказани резултати за 14 највише рангираних чворова чија бар једна грана има тежину већу или једнаку од 0,1.

У анализи је показано да има смисла користити централност по степену, релациону централност, централност по својственом вектору, као и бета централност са коефицијентом β=-1. Приказан је и коефицијент кластеризације за сваки од чворова. Резултати за OS1-30 и PP2-15 скупове података нису приказани, јер се због веома малог броја чворова и просечног степена губе скоро све мрежне карактеристике, а мрежа претежно своди на PAIR случај плагијаризма.

Једина мера која није коришћена у овој анализи је централност по блискости. С обзиром да она мери просечну удаљеност чвора од свих осталих чворова, показује се да она нема смисла у контексту детекције плагијаризма у програмском коду. Анализиране мреже су релативно малог дијаметра и просечне удаљености су мале, па самим тим и овај параметар веома мало варира од чвора до чвора и самим тим нема посебно значење.

У контексту детекције плагијаризма најпре треба поменути централност по степену, јер ова мера има јасну интерпретацију. Чворови са израженом централношћу по степену имају већи потенцијал за сарадњу, јер имају већи број конекција. У оба скупа података се може уочити по неколико чворова који задовољавају овај критеријум:

- OS1: tm120142, kb110573, ln070384, jv120136, sb120140, pp100300
- PP2: mb130613, bd130231, mv130683, nj130128

Табела 6.4. Мере централности и коефицијент кластеризације за OS1 скуп података за 26 по сличности најбоље рангираних чворова (DC – централност по степену, BC – релациона централност, EVC – централност по својственом вектору, BetaC – бета централност са $\beta = -1$, CC – коефицијент кластеризације појединачних чворова)

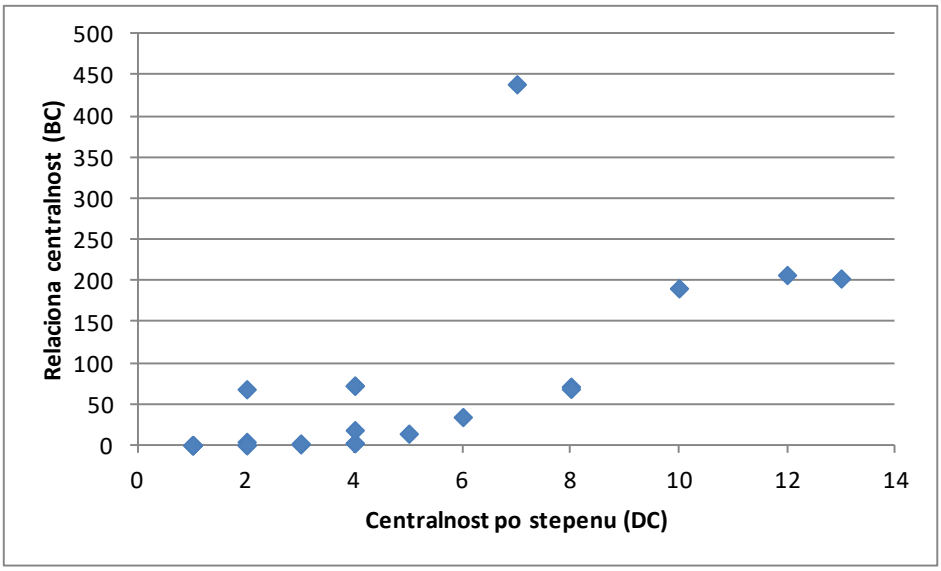
Скуп података	OS1					OS1-10					OS1-20				
Студент	DC	BC	EVC	BetaC	CC	DC	BC	EVC	BetaC	CC	DC	BC	EVC	BetaC	CC
am120206	1	0	0	0,43	/	1	0	0	0,43	/	1	0	0	0,43	/
bd120269	1	0	0	0,36	/	1	0	0	0,36	/	1	0	0	0,36	/
ip120054	2	0	0	0,25	1	2	0	0	0,25	1	1	0	0	0,17	/
jb110473	2	0	0	0,26	1	1	0	0	0,21	/	1	0	0	0,31	/
jv120136	8	68,41	0,03	0,61	0,54	6	151,97	0,25	0,50	0,33	2	3	0,34	0,39	/
kb110573	12	206,95	0,36	0,89	0,47	6	162,17	0	0,74	0,13	1	0	0	0,15	/
km120115	1	0	0	0,29	/	1	0	0	0,29	/	1	0	0	0,29	/
ln070384	10	190,76	0,32	0,95	0,53	4	386	0	0,71	0,17	1	0	0	0,34	/
ma120245	1	0	0	0,29	/	1	0	0	0,29	/	1	0	0	0,29	/
md090332	2	4	0	0,45	0	1	0	0	0,36	/	1	0	0	0,36	/
mfl10524	1	0	0	0,31	/	1	0	0	0,36	/	1	0	0	0,36	/
mg060250	3	1	0,09	0,26	0,67	2	0	0	0,25	1	1	0	0	0,13	/
mi070555	2	68	0,04	0,12	0	2	46	0	0,24	0	1	0	0	0,34	/
mv110185	4	2,5	0	0,54	0,67	3	2	0	0,53	0,33	1	0	0	0,31	/
oo120362	6	34,24	0,03	0,34	0,67	4	18,33	0,16	0,29	0,5	1	0	0,15	0,19	/
pd120160	3	2	0	0,68	0,33	3	2	0	0,68	0,33	2	1	0	0,55	/
pm120213	1	0	0	0,36	/	1	0	0	0,36	/	1	0	0	0,36	/
pp100300	7	438,95	0,15	0,57	0,33	4	442	0,08	0,57	0,5	3	2	0	0,59	0,33
sb120140	8	71,18	0,03	0,48	0,5	7	151,78	0,33	0,45	0,48	1	0	0	0,20	/
sd040376	4	18,27	0,09	0,34	0,5	2	0	0	0,33	1	2	1	0	0,58	/
sm100435	4	72,36	0,04	0,36	0,67	3	0	0,08	0,31	1	2	0	0	0,30	1
sn100408	4	2,5	0	0,38	0,67	2	0	0	0,28	1	1	0	0	0,24	/
st100508	5	14	0	0,43	0,3	2	0	0	0,28	1	1	0	0	0,24	/
tm100107	4	72,36	0,04	0,45	0,67	4	90	0,08	0,52	0,5	2	0	0	0,34	1
tm120142	13	202,69	0,04	1,15	0,369	11	252,28	0,43	1,1	0,31	2	1	0	0,54	/
tt120311	1	0	0	0,43	/	1	0	0	0,43	/	1	0	0	0,43	/

Табела 6.5. Мере централности и коефицијент кластеризације за PP2 скуп података за 14 по сличности најбоље рангираних чворова (DC – централност по степену, BC – релациона централност, EVC – централност по својственом вектору, BetaC – бета централност са $\beta = -1$, CC – коефицијент кластеризације појединачних чворова)

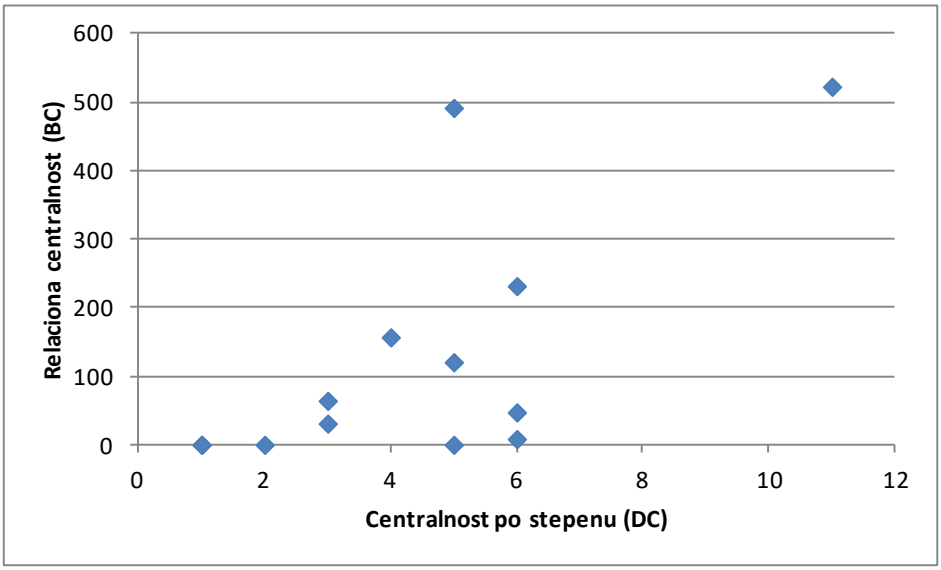
Скуп података	PP2					PP2-10				
Студент	DC	BC	EVC	BetaC	CC	DC	BC	EVC	BetaC	CC
bd130231	6	8,38	0,05	0,32	0,73	1	0	0,27	0,09	/
bi130168	4	156,88	0,03	0,23	0,17	1	0	0,00	0,16	/
jd130085	1	0	0,00	0,11	0	1	0	0,00	0,13	/
ks100543	5	0	0,04	0,33	1	2	0	0,47	0,26	1
ks110543	3	30,95	0,01	0,14	0,33	1	0	0,00	0,10	/
kv130675	3	27,86	0,06	0,15	0,33	1	0	0,27	0,09	/
mb130613	11	522,57	0,12	0,64	0,27	4	5	0,64	0,47	0,17
mv130683	6	47,28	0,06	0,38	0,73	2	0	0,47	0,27	1
nm130259	5	491,69	0,02	0,25	0,1	1	0	0,00	0,13	/
ns130447	2	0	0,01	0,17	1	1	0	0,00	0,11	/
nj130128	6	231,61	0,06	0,30	0,2	2	1	0,00	0,23	/
tn130673	3	64,15	0,02	0,22	0	1	0	0,00	0,10	/
ts130237	5	120,53	0,02	0,29	0,3	1	0	0,00	0,09	/
vv130186	1	0	0,00	0,15	0	1	0	0,00	0,16	/

Уобичајено, овакве чворове прати висока релациона централност и бета централност. Како релациона централност говори о броју најкраћих путева између свих осталих парова чворова на којима се посматрани чвор налази у односу на укупан број таквих најкраћих путева, јасно је да чворови са високом вредношћу оба наведена параметра представљају било изворе, било дистрибутере плагираног кода. Не може се са сигурношћу одредити коју од ове две улоге чвор има, али се емпиријски показује да чвор може бити дистрибутер, уколико постоји једна грана са израженом сличношћу ка чвору који онда потенцијално може бити извор плагијаризма, али све поменуто свакако указује на STAR случајеве колаборације.

За обе посматране мреже централност по степену и релациона централност значајно корелишу, као што се и види на Слици 6.3. и Слици 6.4. На обе слике хоризонтална оса приказује централност по степену, док вертикална оса приказује релациону централност. Одређени број тачака се на сликама преклапа, јер узимају исте вредности. За OS1 податке степен корелације износи 0,62, док за PP2 он износи 0,84, што представља статистички значајну корелацију на нивоу 0,01. У досадашњим студијама [142, 143] се показује да су ти степени корелације значајно нижи и да релациона централност у принципу не корелише са централношћу по степену код других типова социјалних мрежа.



Слика 6.3. 2D анализа резултата централности по степену и релационе централности за OS1 скуп података



Слика 6.4. 2D анализа резултата централности по степену и релационе централности за PP2 скуп података

Бета централност је у овој анализи примењена са фактором -1. Основна идеја употребе негативног пондера је, према Боначићевој теорији [125], истицање чворова који имају већу моћ, односно њихови суседи су у просеку мање повезани са другима. У контексту детекције плагијаризма, то одговара ситуацији у којој су суседи чвора преузели од њега задатак, али га нису много даље ширили са другима, што често одговара шемама колаборације у реалности (STAR случај). Стога је овој метрици дата предност у односу на класичну централност по својственом вектору. Бета централност не корелише добро ни у једном скупу података за читаву мрежу са централношћу по степену, али се показују високи степени корелације за

првих 26 приказаних чворова OS1 мреже и 14 приказаних чворова PP2 мреже, односно са повећавањем прага сличности. То такође наглашава претпостављену улогу ових чворова у мрежи.

На Слици 6.3. и Слици 6.4. се такође може приметити већи број чворова који има релативно ниску централност по степену и релациону централност. Са друге стране, примећује се да они имају релативно висок коефицијент кластеризације, што означава тежњу чвора ка стварању клике (GROUP случај колаборације). Карактеристични примери који могу да се уоче у Табели 6.2. су студенти sm100435 i tm100107. Ови студенти су међусобно повезани јаком везом, а такође су повезаним јаком везом и са чвором pp100300. У овој ситуацији се тешко може одредити извор плагијаризма, јер не постоји сасвим изражен централни чвор. Притом, применом прага сличности коефицијент кластеризације ових чворова расте, а централност по степену и релациона централност падају. На скупу података OS1-20, чворови sm100435 и tm100107 имају овај коефицијент једнак 1 и заједно са pp100300 чине клику. У PP2 мрежи се оваква ситуација може уочити за чворове ts130237, nj130128 и ns130047. Међутим, то се не види у Табели 6.3., јер је веза између ts130237 и ns130047 била на граници од 10%, па је елиминисана приликом филтрирања.

Оваква ситуација најчешће означава случај у коме студенти заједнички раде на програмском коду, а онда га појединачно предају уз модификације. Такође, у пракси се овакве ситуације јављају и када нека трећа особа пише програмски код за више студената, али га сама и не предаје, па стога недостаје изражен централни чвор. Такав код може после даље да буде дистрибуиран у мрежи, као што се види на примеру чвора pp100300 који има више улога у самој мрежи.

У проналажењу група студената описаном шемом колаборације може да помогне анализа клика. Ипак, ова анализе се не препоручује без претходног филтрирања мреже по прагу сличности. Анализа клика не узима у обзир тежину гране која спаја два чвора, већ само постојање гране. У складу са тим, у нефилтрираним мрежама где постоји велики број грана са малом сличношћу се може појавити велики број клика, чиме се онда ова анализа обесмишљава. У Табели 6.6. се види како се овај број значајно мења са величином мреже. Минимална величина анализираних клика је три, јер то једино има смисла уколико се жели анализирати колаборација групе студената.

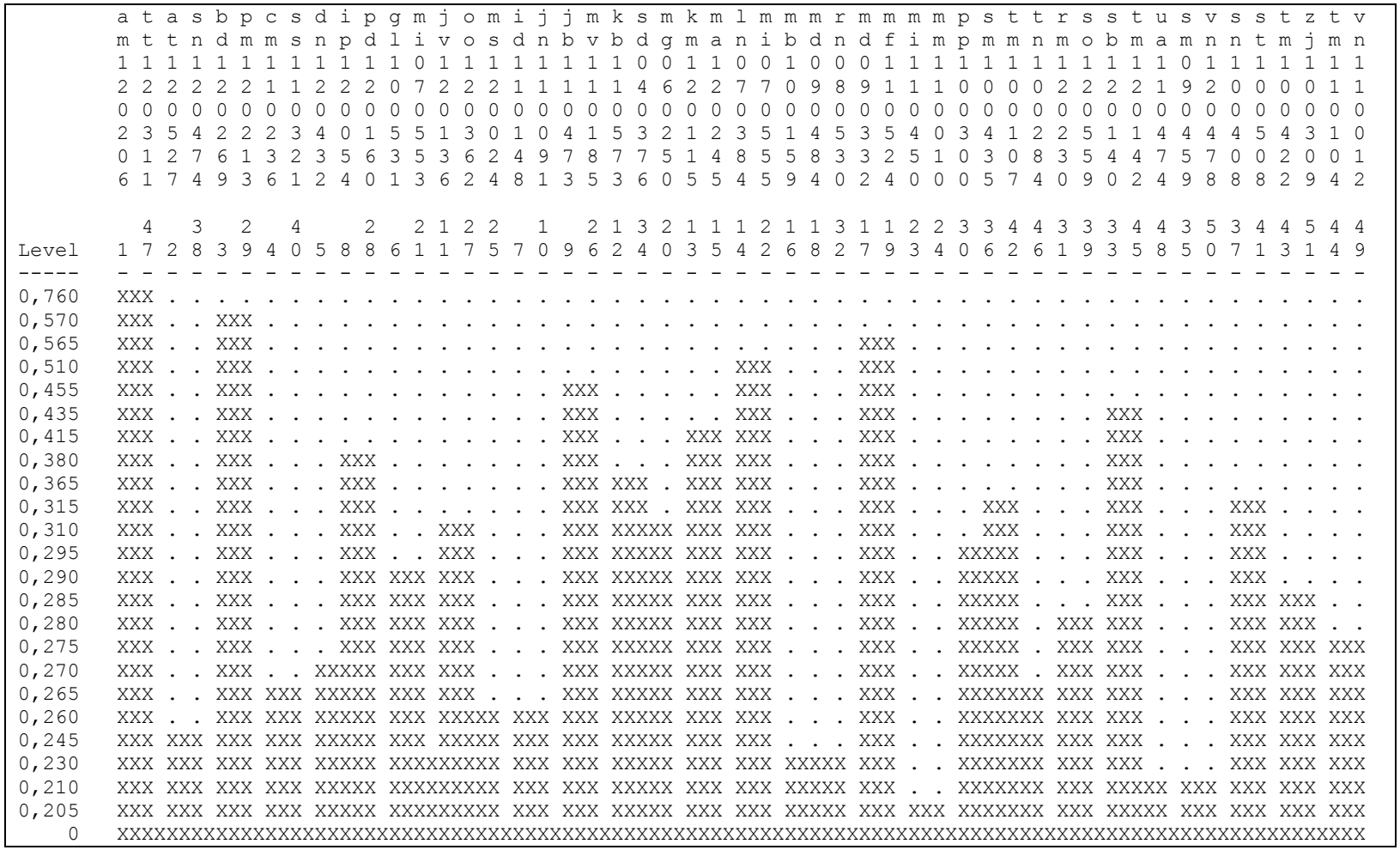
Табела 6.6. Анализа клика и група по Грановетеру за посматране скупове података (N – број клика или група по Грановетеру, MAX SIZE – величина највеће клике или групе)

Скуп података	OS1		OS1-10		OS1-20		PP2		PP2-10	
	N	MAX SIZE	N	MAX SIZE	N	MAX SIZE	N	MAX SIZE	N	MAX SIZE
Клике	46	6	25	5	3	3	56	6	1	3
Групе по Грановетеру	2	3	2	3	0	2	0	/	0	/

Са друге стране, за ове потребе се може искористити проналажење група (компоненти) по Грановетеру које је објашњено у Поглављу 6.2.3. Ова метода узима у обзир тежину грана. Међутим, она понекад може бити престрога и самим тим давати мањи број група, зато што сама, итеративно на основу тежина грана у мрежи одређује праг значаја за везу. Стога комбиновање филтрирања по прагу сличности и анализе клика може дати добре резултате.

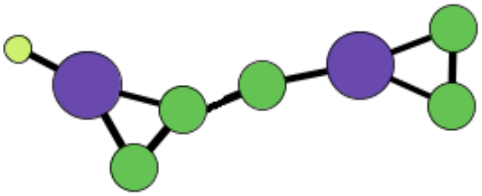
Као алтернатива се за кластеризацију може искористити анализа повезаних компоненти над тежинским мрежама. Као што је описано у Поглављу 6.2.3., оваква анализа итеративно пролази кроз мрежу полазећи од највеће тежине гране и постепено је смањујући. На основу тежине гране се онда врши дихотомизација графа и одређивање повезаних компоненти. Као резултат се добија одговарајућа подела која може да се прикаже у виду дендограма, као што је приказано на Слици 6.5. која приказује резултате овакве анализе коришћењем UCINET алата за OS1-20 скуп података. На хоризонталној оси се виде ознаке студената и редни бројеви њихових чворова у мрежи, а на вертикалној оси ниво сличности.

Анализом мреже је најтеже утврдити плагијаризам који је настао комбиновањем више постојећих извора или решења, чак и када су доступни (MIX случај). Такве чворове карактерише релативно ниска централност по степену, али релативно висока релациона централност. Притом, ниједна веза није јака, већ су те везе релативно слабе, јер су само делови програмског кода слични. У OS1 скупу података постоји тек неколико чворова који показују такве особине, али се тек за један чвор, cm090531, може делимично посумњати у овакав сценарио. Чак и након ручне инспекције кода то се не може са потпуном сигурношћу утврдити.



Слика 6.5. Одређивање повезаних компоненти у тежинској мрежи по нивоима сличности

Табела 6.7. Сумаризовани приказ карактеристичних шема колаборације и њихових особина

PAIR  Централност по степену једнака 1, релациона централност једнака 0	**STAR**  Висока централност по степену, релациона и бета централност
GROUP  Ниска централност по степену и релациона централност, висок коефицијент кластеризације	**MIX**  Ниска централност по степену и висока релациона централност

У Табели 6.7. је извршена сумаризација карактеристичних шема колаборације и њихових главних особина запажених у оквиру истраживања. Показано је да изоловани пар (PAIR) карактеришу централност по степену једнака 1 и релациона централност једнака 0, што је очигледан закључак.

Код групе студената са израженим централним чвором (случај STAR) се може уочити један чвор са изражено високом централношћу по степену, релационом и бета централношћу, док се код групе студената без израженог централног чвора (GROUP) запажају ниска централност по степену и релациона централност, али висок коефицијент кластеризације. Четврти случај, MIX, карактеришу ниска централност по степену и висока релациона централност.

7. ВИЗУЕЛИЗАЦИЈА РЕЗУЛТАТА

Немогућност јасног и прегледног приказа резултата детекције сличности у програмском коду је један од главних недостатака алата описаних у Поглављу 3. Текстуални приказ се показује као неефикасан и у њему се лако уочавају само флагрантни случајеви плагијаризма. Међутим, ниједан од поменутих алата не омогућава дубљу анализу веза и кластеризацију добијених резултата, већ се комплетна анализа препушта кориснику. Стога је потребно извршити визуелизацију резултата детекције, што се може урадити помоћу одговарајућих софтверских алата. Како би се процес визуелизације олакшао, потребно је извршити интеграцију алата за детекцију сличности у програмском коду са алатима за визуелизацију графова. То је постигнуто развојем одговарајућих додатака за увоз резултата детекције сличности у алате за визуелизацију. Резултати приказани у овом поглављу се налазе у ауторовом раду [144].

7.1. Приказ резултата у виду графа

Како се плагијаризам код универзитетских курсева може посматрати као друштвени феномен, логичан избор за представљање резултата детекције сличности на визуелан начин је у виду графа. Као што је објашњено у ранијим поглављима, граф као апстракција логички одговара проблему. Сваки студент и његов рад представљају један чвор неусмереног графа, док проценат сличности радова два студента представља тежину гране која их спаја. Алати за визуелизацију графова већ постоје и дозвољавају модификацију различитих аспеката

графичког приказа, од величине чворова, дебљине грана, до распореда чворова на екрану у зависности од изабраних параметара.

Од анализираних алата за детекцију сличности у програмском коду, приказ резултата у виду графа је омогућавао Sherlock [77], али се он већ дуже време не унапређује и није у широј употреби. Према тврдњама аутора, приказ резултата у виду графа је уведен и у алат FPDS, али сам алат није јавно доступан. Према расположивим информацијама и сликама из [59], чворови који представљају радове су распоређени кружно, а везе између чворова се приказују у зависности од задатог прага. Ипак, овакав фиксан приказ није најпогоднији у случајевима када постоји велики број чворова, односно студентских радова за поређење.

Од развијених додатака за Moss, помоћни алат Mossum [145] пружа могућности приказивања резултата у виду графа. Међутим, приказ се добија у виду низа статичних слика у PNG формату са којима није могућа никаква врста интеракције. JPlag нема могућности за визуелизацију резултата у виду графа.

7.2. Алати за визуелизацију графова

У овом поглављу су описани алати одабрани као најпогоднији за приказивање резултата детекције сличности и интеграцију са алатима JPlag и Moss. Постоји велики број алата за анализу и визуелизацију графова, а детаљнији преглед 30 најкоришћенијих алата се може видети у [146]. Приликом избора алата за визуелизацију, главни фактори који су узети у разматрање су били доступност, интерактивност и проширивост. Са становишта употребе у контексту детекције сличности, нарочито битна особина је била једноставна проширивост због интеграције са алатима за детекцију сличности. Разматрани су само бесплатни софтверски пакети.

7.2.1. Избор алата

Популарни алати за анализу социјалних и других мрежа попут UCINET-а и Pajek-а имају одличну подршку за анализу графова, али су релативно комплексни и акценат је мање на графичком приказу. UCINET има подршку за визуелизацију кроз алат NetDraw који користи UCINET формат за складиштење графова и резултата анализе, док је код Pajek-а визуелизација део пакета за анализу. Неколико бесплатних алата, као што су GraphViz, Otter и Walrus су такође разматрани, али су одбачени због сужених могућности за манипулацију

подацима и мењање графичког приказа. У том смислу, као најбоље решење су се показали NodeXL и Gephi.

NodeXL је развијен као додатак за Microsoft Excel, првенствено за визуелизацију и анализу друштвених мрежа помоћу графова са којима је омогућена одлична интеграција. Ипак, због једноставне проширивости и могућности за развој додатака (eng. *plugins*), изабран је алат Gephi за интеграцију са алатима за детекцију сличности, док су за NodeXL развијене скрипте за увоз података током евалуације алата. Сами алати NodeXL и Gephi су детаљније описани у Поглављу 6.3., а овде ће бити разматрани само битни аспекти са становишта визуелизације. У последњој верзији је дошло и до промене комерцијалног модела алата NodeXL, тако да се сада испоручује у верзијама *Basic* і *Pro*. Основна верзија је бесплатна, али са врло ограниченим могућностима, што додатно оправдава избор алата Gephi, као главног алата за визуелизацију резултата детекције сличности у програмском коду.

7.2.2. Могућности алата

Основна манипулација визуелним приказом графа у алату Gephi омогућава селекцију одређених чворова и грана, промену величине и дебљине чворова и грана, као и њиховог распореда. Чворовима се могу додељивати лабеле. Такође, алат омогућава филтрирање приказа на основу израчунатих метрика графа. На тај начин се, на пример, могу уклонити гране са тежином мањом од задатог прага и припадајући чворови.

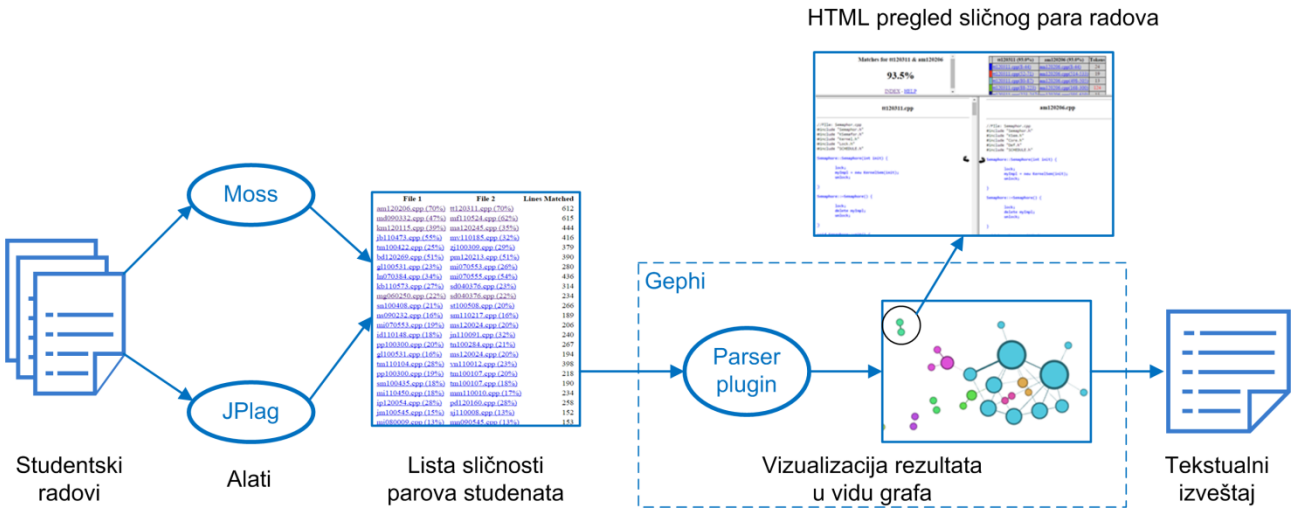
Поред основне манипулације приказом графа, кориснику је на располагању и одређен број предефинисаних начина за аутоматско распоређивање чворова. Поред уобичајених, као што су кружни, спирални или синусоидни распоред, посебно се истиче распоређивање чворова на екрану применом алгоритама *Force Atlas* и *Force Atlas 2*, базираних на алгоритму *Barnes-Hut* [147]. Код њих се чворови посматрају као тела која се међусобно одбијају и привлаче, а чворови се адекватно распоређују тако да природно рефлектују особине графа.

NodeXL има врло сличне могућности за манипулисање графом као алат Gephi. Одређене разлике постоје код алгоритама за распоређивање чворова и NodeXL је нешто богатији опцијама у том смислу, па су доступни и алгоритми који теже да прикажу граф на начин на који ће гране бити приближно исте дужине, са што мањим бројем пресека између њих, као што су *Fruchterman-Reingold*, *Harel-Koren Fast Multiscale,* и сл. На атрибуте графа и израчунате статистике се могу се примењивати разни динамички филтери и тиме кориговати његов приказ.

7.3. Интеграција алата

Интеграција одабраних алата извршена је у неколико корака. Како је за примарни алат за приказивање графова изабран Gephi, за њега су развијена четири додатка. Додаци су написани у програмском језику Java, на коме је написан и сам алат за визуелизацију, а за развој је коришћен основни код за додатке који се може преузети са сајта пројекта Gephi.

Два развијена додатка служе за парсирање резултата JPlag-а и Moss-а, на основу којих се формира граф и врши визуелизација. Трећи додатак омогућава једноставну манипулацију резултатима детекције сличности у оквиру самог графа. Помоћу њега је омогућена селекција два чвора и отварање одговарајуће HTML странице која приказује сличности између два рада које они представљају, чиме се кориснику омогућава ручна инспекција. Четврти додатак се користи за генерисање извештаја. Слика 7.1. илуструје читав ток процеса визуелизације резултата детекције сличности у програмском коду.



Слика 7.1. Ток процеса визуелизације резултата детекције сличности у програмском коду

7.3.1. Увоз JPlag и Moss резултата у Gephi

Опције за учитавање резултата рада алата JPlag и Moss су интегрисане у оквиру *Plugins* секције алата Gephi. Резултати рада ова два алата су смештени у оквиру одговарајућих HTML страница. Практично, било је потребно направити парсер који издваја неопходне информације са ових страница и смешта их у одговарајуће структуре података алата Gephi.

Приликом покретања развијеног додатка је омогућена селекција директоријума који садрже HTML странице са резултатима. Странице могу бити смештене локално, али се у случају алата Moss омогућава и достављање одговарајућег линка за преузимање резултата који су смештени на серверу. Након преузимања, странице се смештају локално и комплетно процесирање се слично врши и за JPlag и за Moss уз поштовање формата HTML страница које садрже резултате.

Са почетне HTML странице се у оба случаја коришћењем регуларних израза дохватају све неопходне информације, укључујући: називе свих чворова (студената), тежине грана (проценти сличности) и линкове ка HTML страницама које садрже оба предата рада, са означеним сличним деловима. Након тога, генерише се граф који се учитава у тренутни радни простор. Пре него што се контрола врати кориснику, зарад лакшег прегледа, примењују се следеће операције:

1) Интерно рангирање чворова на основу степена сваког чвора, ради истицања чворова са великим бројем веза. Величина сваког чвора мења се у зависности од степена чвора односно броја грана које су у вези са њим. На овај начин ће бити уочљивији они студенти који са већом вероватноћом представљају извор дељеног програмског кода.
2) Кластеризација на основу особине модуларности. Када се одреди кластер коме припада сваки од чворова, они добијају различите боје. Тако постају очигледније потенцијалне групе студената које међусобно размењују програмски код.
3) Лабеле у графу постају видљиве, тако да се на чворовима приказује назив студента (његово корисничко име под којим се врши обрада), а гране као лабелу узимају тежине, односно проценат сличности између радова које повезују. Ове лабеле постају видљиве само када је селектован неки чвор, тада се истичу његове везе са осталим студентима. Такође, дебљина гране се повећава у зависности од њене тежине, што је подразумевано за Gephi.
4) Примењује се алгоритам *Force Atlas 2* за распоред чворова на радној површини, тако да постају визуелно груписани студенти са међусобним везама. Овај корак је кључан за јасно приказивање и лако детектовање плагијаризма. Параметри коришћења поменутог алгоритма одабрани су након тестирања различитих опција и избора најпрегледнијег резултата.

7.3.2. Приказивање сличности два рада у оквиру алата Gephi

Да би се омогућила једноставнија ручна инспекција сличних кодова, омогућено је отварање одговарајућих страница са упоредним приказом које генеришу JPlag и Moss из алата Gephi. Након селекције два чвора из генерисаног графа, кориснику је дозвољено да приступи HTML страници, која приказује међусобну сличност радова које они репрезентују. Опција је додата у контекстни мени који се може добити десним кликом миша било где на радној површини.

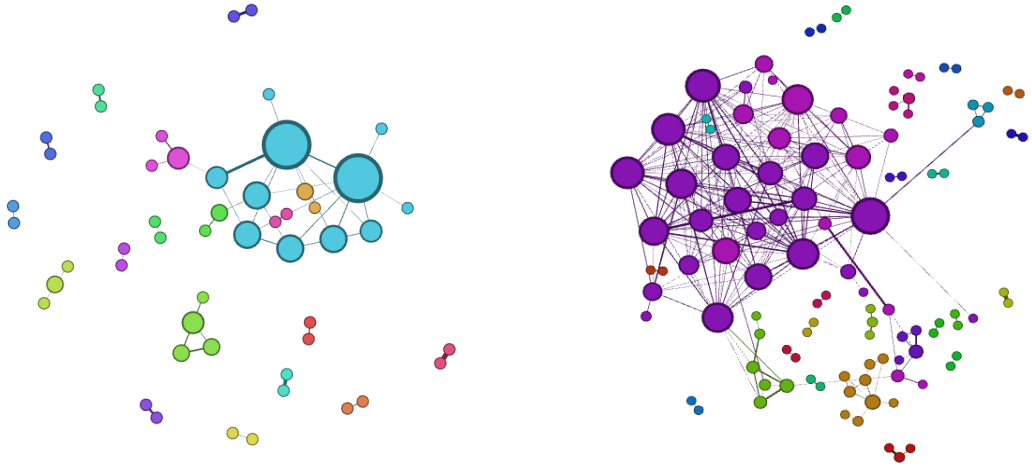
Имплементација ове операције је захтевала коришћење интерфејса за проширење, које нуди Gephi. На основу идентификатора чворова, тражи се грана која их спаја и из колекције њених атрибута се налази линк ка траженој страници. Линкови су претходно приликом парсирања обрађени тако да показују на датотеке на локалном диску. Странице се прегледају коришћењем подразумеваног интернет претраживача.

7.4. Приказ резултата

За тестирање интегрисаних алата коришћена је колекција радова студената са курса Оперативни системи 1 на Електротехничком факултету Универзитета у Београду. У колекцији се налазило 175 радова написаних на програмском језику C++ са просечно 1000 линија програмског кода. Радови из колекцију су пропуштани кроз системе JPlag и Moss и генерисане су HTML странице са резултатима.

Крајњи приказ резултата у виду графа увелико зависи од одабраних параметара алата за детекцију сличности и броја радова који се пореди. У том смислу, нису за приказ релевантни сви чворови, већ само они који са барем једним од преосталих имају довољно велики проценат сличности да постану сумњиви испитивачу. Ранија истраживања [28, 105] показују да проценат сличности од преко 50% готово извесно указује на почињен акт плагијаризма. Ниже вредности у распону од 20 до 50 процената се могу узимати у обзир уколико се сумња на делимични плагијаризам, уз детаљнију ручну инспекцију кода.

У сваком случају, емпиријски се показује да су густо повезани графови са великим бројем чворова често непрегледни. Једна таква ситуација је јасно уочљива на Слици 7.2. Са леве стране је приказан један ретко повезан граф са 51 чвором, док је са десне стране приказан један густо повезан граф са 107 чворова. Слична искуства су представљена у [148], где се сугерише да графови са више од 50 чворова брзо постају непрегледни са повећањем густине графа.



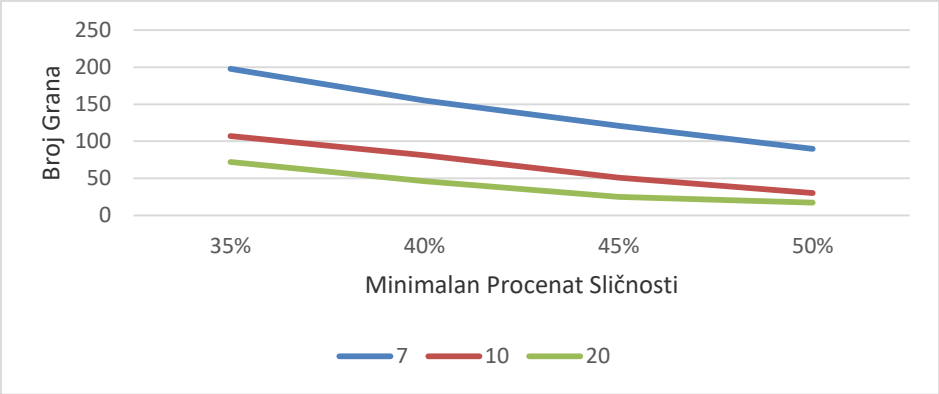
Слика 7.2. Приказ генерисаног графа са (а) 51 чвором, (б) 107 чворова

7.4.1. Утицај параметара алата за детекцију сличности на приказ графа

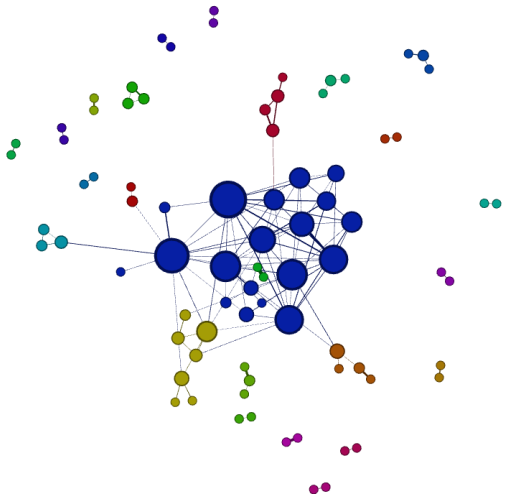
Код алата JPlag, параметри који могу променити број чворова генерисаног графа су праг сличности и минимална дужина низа токена за упаривање MML која дефинише рад коришћеног RKR-GST алгоритма. Како би се пронашле оптималне вредности за приказ графа, посматране су различите вредности за ова два параметра. За праг сличности коришћене су вредности од 35% до 50%, док је и минимална дужина низа токена подешавана на 20, 10 и 7. Мања вредност овог параметра омогућава већу прецизност, али и појаву већег броја лажно позитивних резултата. Посматран је, затим, утицај ових параметара на број чворова и грана у генерисаном графу.

Може се запазити убрзано опадање величине графа са порастом оба параметра. Када је праг сличности у питању, такво понашање је очигледно, јер се просто одбацују гране најмање сличности и чворови које они повезују. Што се минималне дужине низа токена тиче, смањивањем ове дужине алгоритам препознаје много више детаља, како стварних плагијата, тако и случајних сличности.

Стога би требало узети такву вредност да се из разматрања не елиминишу делимични плагијати, али не премалу, јер таква вредност продукује велики број чворова. За дати тест узорак, са Слике 7.3. се може утврдити да је за најбољи визуелни приказ погодно користити вредност 10. Како је оптимална величина графа око 70 чворова, са Слике 7.3. се види да за праг сличности треба узети 40%. То гарантује да ће на графу бити приказани само релевантни чворови, са великим процентом сличности, као на Слици 7.4.. Када је Moss у питању, једини параметар који се може варирати је праг сличности.



Слика 7.3. Зависност броја чворова од прага сличности и минималне дужине низа токена за упаривање



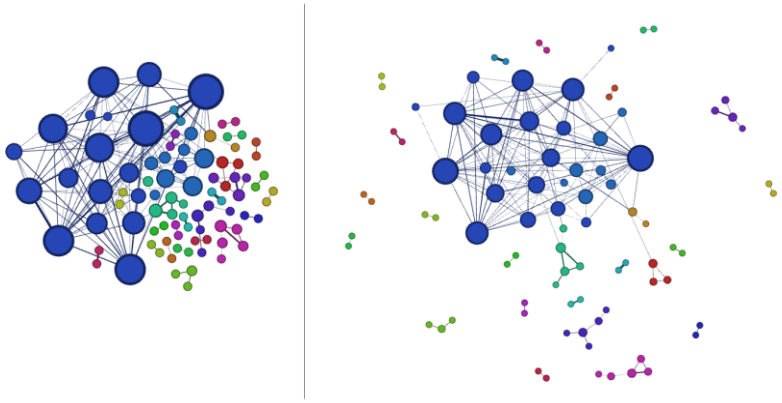
Слика 7.4. Граф генерисан као резултат алата JPlag са прагом сличности 40% и минималном дужином низа токена 10

7.4.2. Смернице за подешавање приказа

Начин презентовања резултата у виду графа је јако подложан променама у зависности од радова који се пореде. На коначни резултат утичу број радова, ниво знања студената, дефиниција задатка, алгоритми који се имплементирају, постојање шаблонског кода који је дат студентима и који подиже проценат сличности и сл. Резултујући граф може бити доста повезанији од оних који су до сада анализирани, а на масовним курсевима може бити потребе и за детаљном анализом више од 70 чворова. Стога ће у наставку бити предложени кораци који се могу применити над генерисаним графом, како би се добио што прегледнији приказ.

Уколико су чворови превише близу један другоме у резултујућем графу, а гране се преклапају, може се покушати са поновном применом алгоритма *Force Atlas 2*. Приликом иницијалног увоза резултата, овај алгоритам се позива тачно 1,5 секунди, што понекад није

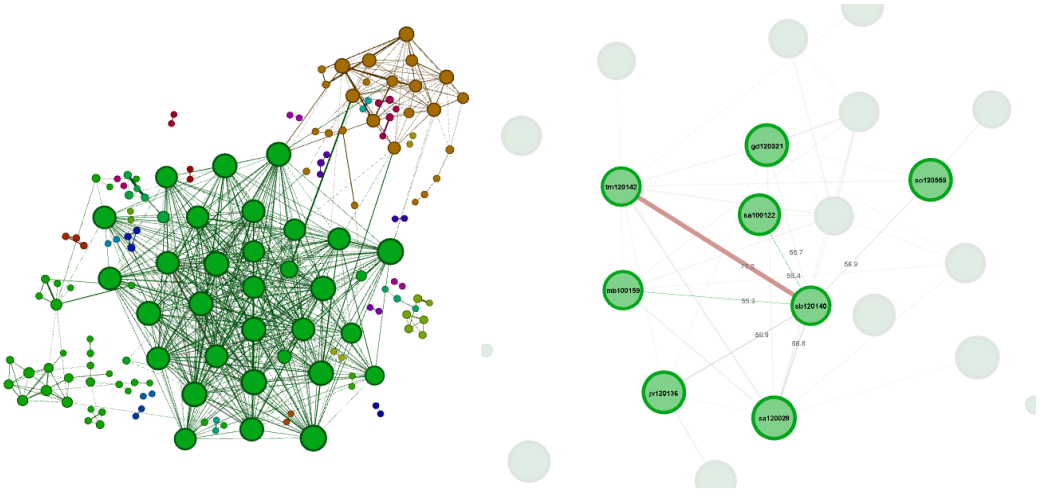
довољно, те је алгоритам потребно позвати поново са дужим временским интервалом. Тај временски интервал корисник може да контролише у оквиру алата Gephi. Ако ни такав граф не пружа задовољавајући приказ, може се покушати са опцијом за експанзију графа која ће једноставно раширити цео граф тако да буде прегледнији. Резултат примене ових корака се види на Слици 7.5.



Слика 7.5. Генерисани граф (а) пре и (б) после додатне примене Force Atlas 2 алгоритма за распоређивање

Уколико је граф непрегледан због великог броја грана, могу се користити филтери који ће ограничити приказ на само онај део графа који је од интереса кориснику. Чворови и гране се могу филтрирати по било ком свом атрибуту. На пример, могу се приказати само гране са тежином која припада одређеном опсегу. Такође, филтери се у алату Gephi могу комбиновати. Наредни филтер се примењује на приказ који је настао применом претходног. Стога, уколико желимо да уз филтрирање грана по тежини, додатно елиминишемо и чворове који сада постају сувишни, довољно је да истовремено филтрирамо по тежини гране и степену чвора.

Често ће се међу чворовима издвојити парови међусобно сличних радова у групацијама од само неколико њих, док ће са друге стране постојати велики кластер густо повезаних чворова. Приликом генерисања графа се одређују класе модуларности позивањем одговарајућих анализа и самим тим се одређује припадност сваког чвора једном од кластера. Та информација се може користити приликом филтрирања и на тај начин се може издвојити кластер који се жели анализирати. На филтрирани граф се може поново применити Force Atlas 2 алгоритам како би се добио оптималан приказ. Изглед графа пре и после примене поменутих филтера може се видети на Слици 7.6.



Слика 7.6. Упоредни приказ генерисаног графа (а) и његовог изгледа (б) након филтрирања по кластерима и тежини гране

Из резултујућег графа се могу трајно елиминисати чворови који нису од нарочитог интереса. То су често мали кластери чворова који се могу једноставно анализирати. Тада је погодно користити филтер *Partition* и изабрати одређени скуп чворова са задатим класама модуларности за издвајање одговарајућих чворова. По потреби је могуће применити додатне, већ поменуте филтере.

Једноставна претрага чворова и грана графа, сортирање по различитим критеријумима и сл. се може обавити у оквиру *Data Laboratory* секције програма. Жељена грана или чвор могу се изабрати за директан приказ у главној секцији за приказ графа.

Уколико *Force Atlas 2* алгоритам не да жељене резултате приказа, могуће је употребити неки од осталих алгоритама за распоређивање, а издваја се *Fruchterman Reingold*, алгоритам који подразумевано алат NodeXL користи за распоређивање. Алгоритам распоређује све чворове у концентричне кругове, уз тежњу да повезани чворови остану груписани и са што мањим бројем пресека између грана. Овај приказ може бити користан, уз комбинацију са неким од осталих приказаних корака за подешавање приказа.

7.4.3. Генерисање текстуалног извештаја

Приликом детекције плагијата, често је потребно генерисање извештаја који се шаљу различитим инстанцама (дисциплинске комисије и сл.) као доказ о почињеном акту плагијаризма. Стога је у оквиру алата Gephi развијен додатак који омогућава генерисање једноставног текстуалног извештаја. Додатак врши анализу графа методама које су доступне у оквиру алата Gephi и исписује основне детаље о његовој структури. За извештај се

дефинише проценат парова са највећом сличношћу који треба приказати и жељени број студената са највише њима сличних радова.

Генерисани извештај састоји се из три целине. У првом делу се приказују кластери студената који су настали као резултат испитивања модуларности графа. Приказује се број кластера на које је граф подељен. Затим следи испис сваке групе заједно са свим студентима који јој припадају. На овај начин може се на једноставан начин увидети подела студената са високим процентом сличности. Након тога, корисник може видети листу парова студената чији радови имају највећи проценат сличности, сортираних нерастуће по сличности.

Даље се у оквиру извештаја приказују подаци о свим паровима студената, чији радови показују сличност међу собом, али не са осталим радовима. Како је извор плагијаризма често једна особа и један рад, ова информација пружа добар увид о паровима на које треба обратити пажњу. На крају, могу се видети они чворови који у графу поседују највећи степен. Наведени су они студенти чији радови су показали сличност са великим бројем осталих студената, што може сугерисати да управо они представљају радове који су послужили као извор плагијаризма.

У тренутној верзији, алат Gephi не подржава све метрике које су неопходне да би се приликом генерисања извештаја презентовале шеме колаборације према методологији изложеној у Поглављу 6. То остаје као простор за додатно унапређење приликом развоја наредне верзије додатка за интеграцију алата за детекцију сличности са Gephi алатом за визуелизацију.

8.Закључак

У досадашњој историји човечанства, информације никада нису биле доступније него што је то случај данас. Са развојем електронских комуникација, нарочито Интернета, и дигитализацијом садржаја, дошло је до великих промена у начину коришћења информација и њихове обраде. Огромне количине садржаја су данас доступне кроз различите дигиталне репозиторијуме на Интернету, а његово дељење је значајно олакшано путем друштвених мрежа. Веома је једноставно преузети одређени садржај, обликовати га према сопственим потребама и представити као своје ауторско дело. Самим тим, долази до појаве плагијаризма, под којим се може сматрати свака ситуација у којој неко користи туђе ауторско дело без експлицитног навођења извора.

Појава плагијаризма јавља се у литерарним делима, образовању, науци, али и у индустрији и може се сматрати изузетно негативном појавом. Најчешћа мета плагијаризма је текст, па се поређењем текстова може утврдити да ли је једно дело настало плагирањем других. У томе велику помоћ пружају софтверски системи за детекцију сличности. Битно је нагласити да софтверски систем само одређује меру сличности између два документа, док финалну одлуку да ли нешто представља плагијат доноси човек.

У рачунарству, најчешће се плагира програмски код који сам по себи представља специфичну врсту текста, написаног на неком од формалних, програмских језика. Стога постоје одређене разлике у детекцији плагијаризма у програмском коду у односу на текстове написане на говорном језику. Те разлике су довеле до развоја софтверских система за

детекцију сличности у програмском коду који имају примене како у образовању будућих инжењера рачунарства, тако и у софтверској индустрији. У индустрији најчешће постоји потреба за поређењем мањег броја кодова већег обима, уобичајено у циљу утврђивања повреде ауторских права. У образовању се обично пореди већи број програмских кодова мањег обима који представљају домаће и пројектне задатке студената. Циљ је најчешће утврђивање плагијаризма, односно спречавање студената да на непоштен начин полажу испите и стичу академска звања.

Овај рад се управо бави проблемом детекције плагијаризма у програмском коду са претежним нагласком на употребу у академској средини. Главни циљ је био да се најпре размотре и евалуирају постојећи приступи и системи за детекцију сличности у програмском коду, а да се затим предложе унапређења ових система која би омогућила ефикаснију детекцију плагијаризма. Процес детекције плагијаризма, и поред делимичне аутоматизације, у великој мери укључује ангажовање наставника на финалној потврди, а сами системи за детекцију сличности, осим листе парова сличних радова и мере сличности, често не дају било какве додатне информације које би олакшале и убрзале процес потврђивања. Додатни циљеви овог рада су били да се шире сагледа ова појава у контексту образовања студената рачунарства, проучи њена преваленција и мере за њено превазилажење.

Први део рада је, пре свега, фокусиран на проблем плагијаризма у програмском коду, алгоритме за детекцију сличности и анализу постојећих алгоритама и система за детекцију сличности, како би се што боље сагледала тренутна ситуација на овом пољу и утврдили правци за могућа унапређења. У оквиру другог поглавља је из више углова разматран проблем плагијаризма, дата његова дефиниција и сагледане специфичности у вези са плагијаризмом у програмском коду. Плагијаризам је дефинисан као свака намерна или ненамерна употреба програмског кода који је написао неко други, а која није адекватно цитирана од стране онога ко предаје програмски код као свој рад. У оквиру овог поглавља су анализиране и најчешће лексичке и структуралне модификације које студенти чине да би прикрили учињени плагијаризам.

Међутим, иако сличност програмског кода често упућује на почињен плагијаризам, програмски код који пишу и предају студенти у академској средини може бити сличан и из других разлога. Ти разлози укључују сличан ниво експертизе, коришћење шаблонског кода са предавања или из литературе, пројектних образаца, детаљна спецификација задатка и сл.

Према томе, не мора свака сличност у програмском коду бити злонамерна и о томе свакако мора водити рачуна онај ко доноси финалну одлуку да ли нешто представља плагијаризам.

Ставови студената и наставника на тему плагијаризма су несумњиво од важности за боље разумевање и спречавање ове појаве. Истраживање спроведено на Електротехничком факултету Универзитета у Београду у оквиру овог рада је показало да студенти нису добро информисани на ову тему и да не разумеју баш најбоље која понашања су дозвољена, а која не. Иако се генерално слажу са дефиницијом плагијаризма и потребом за санкционисањем ове појаве, студенти се углавном не слажу око одговорности за учињено дело, поготову уколико она укључује праксе заједничког рада, слања програмског кода и сл. Испитивање је показало да је око четвртина испитаника склона оваквом понашању, док је око 7% признало да је једном током студија починило ово дело, што је у складу са истраживањима која су спроведена у свету. Боље информисање студената и прецизнија дефиниција плагијаризма у дисциплинским правилницима би свакако допринели смањењу ове непожељне појаве.

С обзиром да се и поред напора за смањивање појаве плагијаризма одређени број студената одлучује да преда туђ рад као своје ауторско дело, наставници користе различите системе за детекцију сличности у програмском коду. У оквиру трећег поглавља су детаљније размотрене технике за детекцију сличности у програмском коду, са посебним освртом на GST и *Winnowing* алгоритме за детекцију сличности у програмском коду који се користе у најпопуларнијим алатима за ову намену, JPlag и Moss.

У оквиру четвртог поглавља је извршена класификација 14 различитих система за детекцију сличности у програмском коду по њиховим најбитнијим карактеристикама. Најпре је идентификован низ карактеристика које један систем треба да поседује, као што су подржани програмски језици, проширљивост, коришћени алгоритми за детекцију сличности, презентација резултата и кориснички интерфејс, а затим је извршен преглед и квалитативно поређење најзначајнијих система из отворене литературе. Анализа је показала да су JPlag и Moss најкомплетнији системи за детекцију сличности у програмском коду у академском окружењу, док се интересантним концептима издвајају и алати SID, GPLAG, FPDS и SCSDS.

Након квалитативног поређења, извршена је и квантитативна, експериментална евалуација три изабрана система, JPlag, Moss и SPD. Анализа је обухватала евалуацију изабраних система коришћењем симулираног плагијаризма и реалних студентских кодова са неколико рачунарских предмета на Електротехничком факултету. Експеримент са

симулираним плагијаризмом је обухватио три различита програма написана на асемблерском језику и програмским језицима C и C++ над којима су вршене сукцесивне измене током осам сати. Продуковане су верзије програма након 1, 2, 4 и 8 сати рада и бележене све учињене измене, а циљ је био да се опонаша реална ситуација у којој студенти улажу одређено време у модификовање туђег програмског кода, како би заташкали почињени плагијаризам.

Резултати експеримента показују да се код краћих студентских задатака од неколико стотина линија кода за релативно кратко време од неколико сати могу учинити значајне структуралне измене, а сличност програмских кодова значајно пада већ после 4 сата рада. Међутим, у случају већих пројектних задатака од преко 1000 линија кода, као што је то случај на курсу из Оперативних система 1, та сличност остаје значајно велика чак и након 8 сати рада. Алати JPlag и Moss се показују знатно супериорнијим у односу на SPD код кога детектована сличност значајно пада већ након једног сата рада за сва три коришћена програмска кода.

Поређење изабраних алата коришћењем реалних студентских кодова је показало значајне сличности које могу бити сматране плагијаризмом у 5-10% свих студентских решења. Системи JPlag и Moss показују висок степен сагласности у смислу резултата детекције сличности. Спроведени експерименти генерално показују да само мањи број структурних измена, као што су измена контролних структура, уметање појединачних наредби и промена редоследа наредби унутар блока кода, значајно утиче на смањење сличности између кодова који се пореде. Ове измене нешто више утичу на смањење детектоване сличности код алата Moss, него код алата JPlag, што је углавном последица коришћеног алгоритма за детекцију сличности.

Евалуација описаних система за детекцију сличности је показала да је њихов највећи недостатак у контексту детекције плагијаризма превише уска фокусираност управо на фазу детекције сличности. Уобичајено, ови системи не поседују никакву логику за напреднији приказ и анализу резултата детекције сличности и имају неинтуитиван кориснички интерфејс. Такође, због временске сложености самих алгоритама за детекцију сличности, поставља се питање њихових перформанси у контексту поређења великог броја студентских радова на масовним програмерским курсевима. Уочени недостаци су били главна мотивација за три главна правца унапређења ових система, а то су паралелизација система за детекцију

сличности, визуелизација резултата у виду графа и анализа резултата методама за анализу социјалних мрежа, чији детаљи чине окосницу другог дела рада.

Паралелизација система за детекцију сличности је спроведена коришћењем хетерогеног рачунарског система са централним и графичким процесором над развијеном експерименталном, секвенцијалном верзијом базираном на алгоритму RKR-GST. Поређење секвенцијалне верзије са алатима JPlag и Moss је показало добру сагласност сва три алата. На централном процесору је извршена паралелизација на нивоу организације послова коришћењем *Pthreads* нити, док је на графичком процесору паралелизован RKR-GST алгоритам коришћењем CUDA технологије. Евалуација паралелних имплементација коришћењем два скупа тест примера показује убрзања до 6 пута на централном процесору и до 4 пута на графичком процесору у односу на секвенцијалну верзију.

Досадашња истраживања показују да је плагијаризам често друштвени феномен који може обухватати већи број учесника. Стога се резултати детекције сличности могу приказати у виду колаборативне, социјалне мреже (графа), а у анализи резултата показују се корисним поједине методе за анализу социјалних мрежа. У оквиру истраживања су идентификоване четири најчешће шеме колаборације, а затим је извршена њихова карактеризација различитим графовским метрикама. Показује се да у детекцији плагијаризма могу бити корисни параметри мреже као што су централност по степену, релациона централност, бета централност, као и степен кластеризације. Показано је и да централност по степену и релациона централност јако добро корелишу у оваквим мрежама. У кластеризацији оваквих мрежа се показује корисним одређивање слабо повезаних компоненти, анализа клика и група по Грановетеру. Потврда осмишљеног приступа је верификована на два реална скупа примера кроз које је показан и значај правилног одређивања прага сличности приликом анализе. За анализу мрежа је коришћен софтверски пакет UCINET.

Као трећи правац за унапређење, разматрана је визуелизација резултата. Приказана је интеграција система за детекцију сличности са алатима за визуелизацију графова, као што је Gephi, а затим су разматрани различити аспекти визуелног приказа мрежа насталих детекцијом сличности. Дискутован је утицај прага сличности на прегледност и читљивост резултујућег графа и разматрани различити начини за распоређивање чворова. Показује се да графови са преко 50 чворова брзо постају непрегледни са повећањем густине графа и да о томе треба водити рачуна код одређивања оптималног прага сличности за приказ.

Сумаризовано, у оквиру дисертације остварени су следећи научни доприноси:

- Преглед, анализа и класификација алгоритама за детекцију сличности у изворном програмском коду
- Преглед, анализа и класификација постојећих система за детекцију сличности у изворном програмском коду
- Класификација и анализа утицаја трансформација које студенти користе да би прикрили акт плагијаризма
- Евалуација изабраних система за детекцију сличности у изворном програмском коду коришћењем симулираног плагијаризма и базе реалних студентских радова
- Развој прототипа система за детекцију сличности у изворном програмском коду уз испитивање могућности за његову паралелизацију
- Паралелизација система за детекцију сличности у изворном програмском коду и евалуација њене ефективности
- Евалуација развијеног система у односу на постојеће системе у широком спектру карактеристика
- Предлог методологије за детекцију плагијаризма кроз колаборацију групе студената коришћењем техника за анализу социјалних мрежа
- Оптимизација параметара и поступка визуелизације резултата детекције плагијаризма коришћењем алата за приказ колаборације у виду графа
- Анализа ставова и пракси студената на тему плагијаризма у изворном програмском коду

У оквиру ове дисертације су предложена унапређења система за детекцију плагијаризма у програмском коду која значајно могу да утичу на побољшавање њихових перформанси и функционалности. Међутим, тиме је и даље покривен само један мањи домен целокупног проблема, а правци даљег истраживања и развоја су разноврсни, како са становишта система за детекцију сличности, тако и са становишта самог феномена плагијаризма и едукације.

Један значајан правац истраживања би се односио на даљу експерименталну анализу и карактеризацију алгоритама и система за детекцију сличности коришћењем различитих врста симулираног и реалног оптерећења. У оквиру овог рада је претежно извршена евалуација

система коришћењем програма написаних на програмским језицима C и C++ и једноставном асемблерском језику, али би свакако било од интереса сличну анализу направити за програмски код написан у другим популарним програмским језицима, као што су Java, Python, PHP и сл.

Исто тако би се могла додатно испитати скалабилност алгоритама и система у зависности од броја студената и величине поређених задатака. У том смислу би се могло експериментисати и са неком врстом предобраде којом би се из даље анализе уклонили програмски кодови са ниским степеном сличности. Додатно би се могли унапредити и тестови симулираног плагијаризма, како би се тестирала прецизност система за детекцију сличности у односу на различите модификације програмског кода.

Други правац развоја би водио ка имплементацији независног система за детекцију плагијаризма који би обухватио различите аспекте анализиране у оквиру овог рада. Тај систем би могао да укључи више алгоритама за детекцију сличности, по узору на систем SCSDS и прилагодљив *front-end* како би се лако додавала подршка за нове програмске језике.

Такав систем би због скалабилности и одржавања добрих перформанси користио паралелизацију, а била би омогућена визуелизација резултата у виду графа и њихова анализа методама за анализу социјалних мрежа. Функционалности би му биле проширене и корисним опцијама као што су уклањање шаблонског кода из поређења, поређење са историјом и сл.

Иако добијени резултати указују да се на централном процесору добијају нешто већа убрзања, аутор верује да GPU имплементација има већи потенцијал за даље усавршавање. Додатна убрзања би се могла постићи другачијом организацијом посла и коришћењем меморије за текстуре на GPU. Такође, у будућности се очекује даљи развој графичких процесора и њихова интеграција у заједнички меморијски простор са централним процесором. Тиме би се отклонила потреба за меморијским трансферима који представљају тренутно највећи режијски трошак и уско грло GPU имплементације и остварила још већа убрзања. Због другачијег начина функционисања, аутор претпоставља и да би имплементација *Winnowing* алгоритма за детекцију сличности на графичком процесору могла дати боље резултате, што свакако треба даље испитати.

Као подршка трећој и четвртој фази у детекцији плагијаризма, могле би се имплементирати различите метрике и методе за анализу социјалних мрежа. Те информације

би биле коришћене као допуна и подршка визуелном приказу, а омогућиле би и филтрирање приказа да се прикажу различите шеме колаборације. Визуелни приказ би се даље могао обогатити и другим контекстуалним информацијама о самим студентима. То могу бити подаци о пријатељству на социјалним мрежама, али исто тако и подаци из студенског досијеа, као што су просечна оцена, година студија, дужина студирања, статус финансирања, завршена средња школа, да ли су раније били сумњичени или кажњавани због плагијаризма и сл. Такође, требало би омогућити што једноставнији прелазак са приказа мреже на директан приказ сличних парова задатака.

У сваком случају, овакав систем би требало да у што могуће већој мери помогне наставнику и смањи количину ручног посла коју он мора да обави. У том смислу, могао би се развити експертски систем који би омогућио генерисање неке врсте „паметних“ извештаја. У ту сврху би се могли применити принципи машинског учења који би омогућили да се на основу задатих критеријума и семантичких информација изврши карактеризација детектованих колаборација унутар мреже.

Истраживање би се могло обавити и на тему стилова програмирања и елемената које студенти плагирају. Тиме би се још боље могло проучити понашање студената и уочити проблеми у усвајању знања који доводе до плагијаризма. Та сазнања би се могла искористити за унапређење наставе и превенцију плагијаризма, како би се смањила преваленција ове непожељне појаве.

С обзиром да су мреже настале детекцијом сличности заправо колаборационе мреже, интересантно би било упоредити их са правим колаборативним (коауторским) мрежама какве се користе у библиометрији и наукометрији за анализу научне продукције. Коауторство се у таквим мрежама сматра пожељном релацијом, док се код плагијаризма таква сарадња може сматрати злонамерном.

На крају, треба поменути могућност за даље истраживање ставова и пракси у вези плагијаризма у програмском коду. Истраживање спроведено међу студентима у оквиру овог рада је свакако допринело бољем разумевању овог феномена, али и отворило нова питања у троуглу између студената, наставника и индустрије. Свакако би, кроз нова истраживања, требало испитати погледе наставника, али и представника компанија на тему плагијаризма у програмском коду.

Литература

Сви извори које је аутор користио у току израде овог рада наведени су редом којим су референцирани у раду. За сваки наведени извор је приложена Интернет адреса, ако је била доступна у тренутку стварања овог списка. Формат је усклађен са IEEE упутством за навођење референци, датим у [149].

[1] R. Shackelford *et al.*, "Computing curricula 2005: The overview report," in *ACM SIGCSE Bulletin*, 2006, vol. 38, no. 1, pp. 456-457: ACM.

[2] M. Ahmadzadeh, E. Mahmoudabadi, and F. Khodadadi, "Pattern of plagiarism in novice students' generated programs: An experimental approach," *Journal of Information Technology Education,* vol. 10, 2011.

[3] M. Mišić, M. Lazić, and J. Protić, "A software tool that helps teachers in handling, processing and understanding the results of massive exams," in *Proceedings of the Fifth Balkan Conference in Informatics*, 2012, pp. 259-262: ACM.

[4] (2016, 08.09.). *ACM Code of Ethics and Professional Conduct*. Available: http://www.acm.org/about-acm/acm-code-of-ethics-and-professional-conduct

[5] (2016, 08.09.). *Princeton University Constitution of the Honor System*. Available: http://www.princeton.edu/honor/constitution/

[6] P. Brennecke, *Academic Integrity at MIT - A Handbook for Students*. 2016.

[7] J. Williams, "Plagiarism: Deterrence, Detection and Prevention," *The Handbook for Economics Lecturers,* pp. 1-19, 2005.

[8] "Definition of Plagiarize," ed: Merriam-Webster, 2016.

[9] (2016, 14.09.). *What Is Plagiarism?* Available: http://www.plagiarism.org/

[10] *Pravilnik o disciplinskoj odgovornosti studenata Univerziteta u Beogradu,* U. u. Beogradu, 2016.

[11] G. Steiner, *After Babel: Aspects of language and translation*. Oxford University Press, USA, 1998.

[12] P. Samuelson, "Self-plagiarism or fair use," *Communications of the ACM,* vol. 37, no. 8, pp. 21-25, 1994.

[13] G. Cosma and M. Joy, "Towards a definition of source-code plagiarism," *IEEE Transactions on Education,* vol. 51, no. 2, pp. 195-200, May 2008.

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

[15] J. P. Gibson, "Software reuse and plagiarism: a code of practice," in *ACM SIGCSE Bulletin*, 2009, vol. 41, no. 3, pp. 55-59: ACM.

[16] D. Chuda, P. Navrat, B. Kovacova, and P. Humay, "The Issue of (Software) Plagiarism: A Student View," *IEEE Transactions on Education,* vol. 55, no. 1, pp. 22-28, Feb 2012.

[17] M. Dick *et al.*, "Addressing student cheating: definitions and solutions," in *ACM SigCSE Bulletin*, 2002, vol. 35, no. 2, pp. 172-184: ACM.

[18] M. Joy and M. Luck, "Plagiarism in programming assignments," *IEEE Transactions on Education,* vol. 42, no. 2, pp. 129-133, May 1999.

[19] H.-L. Jian, F. E. Sandnes, Y.-P. Huang, L. Cai, and K. M. Y. Law, "On students' strategy-preferences for managing difficult course work," *IEEE Transactions on Education,* vol. 51, no. 2, pp. 157-165, May 2008.

[20] (2016). *Stack Overflow*. Available: http://stackoverflow.com/

[21] (2016). *GeeksforGeeks - A computer science portal for geeks*. Available: http://www.geeksforgeeks.org/

[22] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: improving the design of existing code. 1999," *ISBN: 0-201-48567-2*.

[23] H. M. Judi, S. Mohd Sallen, N. Hussin, and S. Idris, "The Use of Assignment Programming Activity Log to Study Novice Programmers' Behavior between Non-Plagiarized and Plagiarized Groups," *Information Technology Journal,* vol. 9, no. 1, pp. 98-106, 2010.

[24] Z. Đurić and D. Gašević, "A source code similarity system for plagiarism detection," *The Computer Journal,* p. bxs018, 2012.

[25] M. J. Wise, "YAP3: Improved detection of similarities in computer program and other texts," *ACM SIGCSE Bulletin,* vol. 28, no. 1, pp. 130-134, 1996.

[26] L. Prechelt, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with JPlag," *Journal of Universal Computer Science,* vol. 8, no. 11, pp. 1016-1038, 2002 2002.

[27] G. Cosma and M. Joy, "Source-code plagiarism: A UK academic perspective," 2006.

[28] M. Mišić, Ž. Šuštran, and J. Protić, "A Comparison of Software Tools for Plagiarism Detection in Programming Assignments," *International Journal of Engineering Education,* Article vol. 32, no. 2, pp. 738-748, 2016 2016.

[29] M. Joy, G. Cosma, J. Y.-K. Yau, and J. Sinclair, "Source code plagiarism—a student perspective," *IEEE Transactions on Education,* vol. 54, no. 1, pp. 125-132, 2011.

[30] M. Joy, J. Sinclair, R. Boyatt, J.-K. Yau, and G. Cosma, "Student perspectives on source-code plagiarism," *International Journal for Educational Integrity,* vol. 9, no. 1, 2013.

[31] M. Dixon, "Software Support for Lecturers Investigating Intra-corpal Source-code Plagiarism: Influence on Student Behaviour," *International Journal of Learning, Teaching and Innovation,* vol. 1, no. 1, 2014.

[32] F. Rosales, A. Garcia, S. Rodriguez, J. L. Pedraza, R. Mendez, and M. M. Nieto, "Detection of plagiarism in programming assignments," *IEEE Transactions on Education,* vol. 51, no. 2, pp. 174-183, May 2008.

[33] I. Baucal, M. Bojičić, and V. Radosavljević, "Akademsko nepoštenje u našem obrazovnom sistemu," Univerzitet u Beogradu, Filozofski fakultet2012.

[34] F. Culwin and T. Lancaster, "Visualising intra-corpal plagiarism," in *Information Visualisation, 2001. Proceedings. Fifth International Conference on*, 2001, pp. 289-296: IEEE.

[35] A. M. E. T. Ali, H. M. D. Abdulla, and V. Snasel, "Overview and Comparison of Plagiarism Detection Tools," in *DATESO*, 2011, pp. 161-172: Citeseer.

[36] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: A multilinguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering,* vol. 28, no. 7, pp. 654-670, Jul 2002.

[37] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering,* vol. 33, no. 9, pp. 577-591, 2007.

[38] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Science of Computer Programming,* vol. 74, no. 7, pp. 470-495, 2009.

[39] D. Rattan, R. Bhatia, and M. Singh, "Software clone detection: A systematic review," *Information and Software Technology,* vol. 55, no. 7, pp. 1165-1199, 2013.

[40] Y. Han *et al.*, "A New Detection Scheme of Software Copyright Infringement using Software Birthmark on Windows Systems," *Computer Science and Information Systems,* vol. 11, no. 3, pp. 1055-1069, Aug 2014.

[41] S. Stojanović, "Procena sličnosti procedura u binarnom kodu," Univerzitet u Beogradu-Elektrotehnički fakultet, 2015.

[42] S. Stojanović, Z. Radivojević, and M. Cvetanović, "Approach for estimating similarity between procedures in differently compiled binaries," *Information and Software Technology,* vol. 58, pp. 259-271, Feb 2015.

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

[44] X. Chen, B. Francia, M. Li, B. McKinnon, and A. Seker, "Shared information and program plagiarism detection," *IEEE Transactions on Information Theory,* vol. 50, no. 7, pp. 1545-1551, Jul 2004.

[45] T. Cormen, C. Leiserson, R. Rivest, and S. Clifford, *Introduction to Algorithms. 3rd Edn. Vol. 1*. Cambridge, MA: MIT Press, 2009.

[46] D. E. Knuth, J. Morris, James H, and V. R. Pratt, "Fast pattern matching in strings," *SIAM journal on computing,* vol. 6, no. 2, pp. 323-350, 1977.

[47] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM,* vol. 20, no. 10, pp. 762-772, 1977.

[48] D. E. Knuth, "Big omicron and big omega and big theta," *ACM Sigact News,* vol. 8, no. 2, pp. 18-24, 1976.

[49] S. Wu and U. Manber, "Fast text searching: allowing errors," *Communications of the ACM,* vol. 35, no. 10, pp. 83-91, 1992.

[50] A. Hume and D. Sunday, "Fast string searching," *Software: Practice and Experience,* vol. 21, no. 11, pp. 1221-1248, 1991.

[51] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM,* vol. 18, no. 6, pp. 333-340, 1975.

[52] B. Commentz-Walter, "A string matching algorithm fast on the average," in *International Colloquium on Automata, Languages, and Programming*, 1979, pp. 118-132: Springer.

[53] M. J. Wise, *Running karp-rabin matching and greedy string tiling*. Basser Department of Computer Science, University of Sydney, 1993.

[54] M. J. Wise, "Neweyes: a system for comparing biological sequences using the running Karp-Rabin Greedy String-Tiling algorithm," in *ISMB*, 1995, pp. 393-401.

[55] J. L. Donaldson, A.-M. Lancaster, and P. H. Sposato, "A plagiarism detection system," in *ACM SIGCSE Bulletin*, 1981, vol. 13, no. 1, pp. 21-25: ACM.

[56] M. H. Halstead, *Elements of software science*. Elsevier New York, 1977.

[57] K. L. Verco and M. J. Wise, "Software for detecting suspected plagiarism: Comparing structure and attribute-counting systems," *ACSE,* vol. 96, 1996.

[58] J. A. Faidhi and S. K. Robinson, "An empirical approach for detecting program similarity and plagiarism within a university programming environment," *Computers & Education,* vol. 11, no. 1, pp. 11-19, 1987.

[59] M. Mozgovoy, *Enhancing computer-aided plagiarism detection*. University Of Joensuu, 2007.

[60] S. Schleimer, D. S. Wilkerson, and A. Aiken, "Winnowing: local algorithms for document fingerprinting," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 2003, pp. 76-85: ACM.

[61] J.-W. Son, T.-G. Noh, H.-J. Song, and S.-B. Park, "An application for plagiarized source code detection based on a parse tree kernel," *Engineering Applications of Artificial Intelligence,* vol. 26, no. 8, pp. 1911-1918, Sep 2013.

[62] C. Liu, C. Chen, J. Han, and P. S. Yu, "GPLAG: detection of software plagiarism by program dependence graph analysis," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2006, pp. 872-881: ACM.

[63] T. Parr, *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.

[64] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems (TOPLAS),* vol. 9, no. 3, pp. 319-349, 1987.

[65] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development,* vol. 31, no. 2, pp. 249-260, 1987.

[66] J. Hage, P. Rademaker, and N. van Vugt, "Plagiarism detection for Java: a tool comparison," in *Computer Science Education Research Conference*, 2011, pp. 33-46: Open Universiteit, Heerlen.

[67] T. Lancaster and F. Culwin, "A comparison of source code plagiarism detection engines," *Computer Science Education,* vol. 14, no. 2, pp. 101-112, 2004.

[68] L. Prechelt, G. Malpohl, and M. Philippsen, "JPlag Finding plagiarisms among a set of programs," in "Technical Report 2000-1," Faculty of Informatics, University of Karlsruhe2000.

[69] (05.10.). *Sherlock: Plagiarism Detector*. Available: http://www.cs.usyd.edu.au/~scilect/sherlock/

[70] M. Joy, N. Griffiths, and R. Boyatt, "The boss online submission and assessment system," *Journal on Educational Resources in Computing (JERIC),* vol. 5, no. 3, p. 2, 2005.

[71] L. Mitić, "Komparativna analiza softverskih alata za detekciju sličnosti programskog koda," Msc, Univerzitet u Beogradu, Elektrotehnički fakultet, 2010.

[72] A. Ahtiainen, S. Surakka, and M. Rahikainen, "Plaggie: GNU-licensed source code plagiarism detection engine for Java exercises," in *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, 2006, pp. 141-142: ACM.

[73] D. Grune and M. Huntjens, "Het detecteren van kopieën bij informatica-practica," *Informatie,* vol. 31, no. 11, pp. 864-867, 1989.

[74] O. Ajmal, M. S. Missen, T. Hashmat, M. Moosa, and T. Ali, "EPlag: A two layer source code plagiarism detection system," in *Digital Information Management (ICDIM), 2013 Eighth International Conference on*, 2013, pp. 256-261: IEEE.

[75] A. Aiken. (20 February). *Measure of software similarity*. Available: http://theory.stanford.edu/~aiken/moss/

[76] (Accessed 20 February). *JPlag*. Available: http://jplag.ipd.kit.edu/

[77] (20 February). *Sherlock*. Available: https://www2.warwick.ac.uk/fac/sci/dcs/research/ias/software/sherlock/

[78] (Accessed 23 February). *Plaggie*. Available: http://www.cs.hut.fi/Software/Plaggie/

[79] (2016). *The software and text similarity tester SIM*. Available: http://dickgrune.com/Programs/similarity_tester/

[80] P. Bradford, M. Porciello, N. Balkon, and D. Backus, "The Blackboard learning system: The be all and end all in educational instruction?," *Journal of Educational Technology Systems,* vol. 35, no. 3, pp. 301-314, 2007.

[81] M. Dougiamas and P. Taylor, "Improving the effectiveness of tools for Internet based education," in *Teaching and Learning Forum*, 2000.

[82] S. Lonn and S. D. Teasley, "Saving time or innovating practice: Investigating perceptions and uses of Learning Management Systems," *Computers & Education,* vol. 53, no. 3, pp. 686-694, 2009.

[83] M. Machado and E. Tao, "Blackboard vs. moodle: Comparing user experience of learning management systems," in *2007 37th Annual Frontiers In Education Conference-Global Engineering: Knowledge Without Borders, Opportunities Without Passports*, 2007, pp. S4J-7-S4J-12: IEEE.

[84] D. Drašković, M. Mišić, and Ž. Stanisavljević, "Transition from traditional to LMS supported examining: A case study in computer engineering," *Computer Applications in Engineering Education,* 2016.

[85] (2016, 31.10.). *Programming Code Plagiarism Plugin*. Available: https://docs.moodle.org/31/en/Programming_Code_Plagiarism_Plugin

[86] M. Kaya and S. A. Özel, "Integrating an online compiler and a plagiarism detection tool into the Moodle distance education system for easy assessment of programming assignments," *Computer Applications in Engineering Education,* vol. 23, no. 3, pp. 363-373, 2015.

[87] V. Dagiene, B. Skupas, and E. Kurilovas, "Programming assignments in virtual learning environments: developments and opportunities for engineering education," *International Journal of Engineering Education,* vol. 30, no. 3, pp. 644-653, 2014.

[88] M. Mišić, A. Jović, and J. Protić, "Web servis za predaju i upoređivanje domaćih zadataka korišćenjem alata Moss," presented at the TREND 2016, Zlatibor, 2016.

[89] A. Bošnjaković, J. Protić, D. Bojić, and I. Tartalja, "Automating the Knowledge Assessment Workflow for Large Student Groups: A Development Experience," *International Journal of Engineering Education,* vol. 31, no. 4, pp. 1058-1070, 2015 2015.

[90] M. Mišić, L. Mitić, and J. Protić, "Softverska detekcija sličnosti programskog koda kao mera za otkrivanje plagijata na ispitima," presented at the TREND 2014, Kopaonik, 2014.

[91] D. A. Patterson, "Future of computer architecture," in *Berkeley EECS Annual Research Symposium (BEARS), College of Engineering, UC Berkeley, US*, 2006.

[92] B. Barney. (2016, 27.09.). *Introduction to Parallel Computing*. Available: https://computing.llnl.gov/tutorials/parallel_comp/

[93] A. Shan, "Heterogeneous processing: a strategy for augmenting moore's law," *Linux Journal,* vol. 2006, no. 142, p. 7, 2006.

[94] "CUDA C Programming Guide 7.5," ed: NVIDIA Corporation, 2016.

[95] M. J. Mišić, D. D. Dašić, and M. V. Tomašević, "An analysis of OpenACC programming model: Image processing algorithms as a case study," *Telfor Journal,* vol. 6, no. 1, pp. 53-58, 2014.

[96] E. Dedu, S. Vialle, and C. Timsit, "Comparison of OpenMP and classical multi-threading parallelization for regular and irregular algorithms," *Context,* vol. 2, p. 20, 2000.

[97] E. Ajkunić, H. Fatkić, E. Omerović, K. Talić, and N. Nosović, "A comparison of five parallel programming models for C++," in *MIPRO, 2012 Proceedings of the 35th International Convention*, 2012, pp. 1780-1784: IEEE.

[98] M. J. Mišić, Đ. M. Đurđević, and M. V. Tomašević, "Evolution and trends in GPU computing," in *MIPRO, 2012 Proceedings of the 35th International Convention*, 2012, pp. 289-294: IEEE.

[99] J. Fang, A. L. Varbanescu, and H. Sips, "A comprehensive performance comparison of CUDA and OpenCL," in *2011 International Conference on Parallel Processing*, 2011, pp. 216-225: IEEE.

[100] K. Karimi, N. G. Dickson, and F. Hamze, "A performance comparison of CUDA and OpenCL," *arXiv preprint arXiv:1005.2581,* 2010.

[101] *Standard for Information Technology Portable Operating System Interface (POSIX(R)) Base Specifications, Issue 7*, 2013.

[102] D. B. Kirk and W. H. Wen-mei, *Programming massively parallel processors: a hands-on approach*. Morgan Kafumann, 2012.

[103] M. Mišić, M. Živković, J. Protić, and M. Tomašević, "Detekcija sličnosti u programskom kodu korišćenjem GST algoritma," presented at the YUINFO 2016, Kopaonik, 2016.

[104] L. Kraus, *Rešeni zadaci iz programskog jezika C++*. Beograd: Akademska misao, 2011.

[105] K. W. Bowyer and L. O. Hall, "Experience using" MOSS" to detect cheating on programming assignments," in *Frontiers in Education Conference, 1999. FIE'99. 29th Annual*, 1999, vol. 3, pp. 13B3/18-13B3/22 vol. 3: IEEE.

[106] M. Mišić, Ž. Šuštran, and J. Protić, "Pregled i primena sistema za otkrivanje plagijata u programskim zadacima studenata," ed. Kopaonik: YUINFO 2015, 2015, pp. 473-478.

[107] M. Mišić, D. Nikolov, J. Protić, and M. Tomašević, "Paralelizacija GST algoritama za detekciju sličnosti u programskom kodu," in *Telecommunications Forum Telfor (TELFOR), 2016 24rd*, 2016, pp. 921-924.

[108] I. Foster, "Designing and building parallel programs," ed: Addison Wesley Publishing Company Reading, 1995.

[109] D. A. F. Alcantara, "Efficient hash tables on the gpu," University of California Davis, 2011.

[110] M. E. Newman, "The structure and function of complex networks," *SIAM review,* vol. 45, no. 2, pp. 167-256, 2003.

[111] M. Newman, *Networks: an introduction*. Oxford university press, 2010.

[112] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*. Cambridge university press, 1994.

[113] S. N. Dorogovtsev and J. F. Mendes, "Evolution of networks," *Advances in physics,* vol. 51, no. 4, pp. 1079-1187, 2002.

[114] A.-L. Barabasi and Z. N. Oltvai, "Network biology: understanding the cell's functional organization," *Nature reviews genetics,* vol. 5, no. 2, pp. 101-113, 2004.

[115] L. Danon *et al.*, "Networks and the epidemiology of infectious disease," *Interdisciplinary perspectives on infectious diseases,* vol. 2011, 2011.

[116] M. G. Bell and Y. Iida, *Transportation network analysis*. 1997.

[117] V. Latora and M. Marchiori, "Is the Boston subway a small-world network?," *Physica A: Statistical Mechanics and its Applications,* vol. 314, no. 1, pp. 109-113, 2002.

[118] A. Broido, "Internet topology: Connectivity of IP graphs," in *ITCom 2001: International Symposium on the Convergence of IT and Communications*, 2001, pp. 172-187: International Society for Optics and Photonics.

[119] Q. Chen, H. Chang, R. Govindan, and S. Jamin, "The origin of power laws in Internet topologies revisited," in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 2002, vol. 2, pp. 608-617: IEEE.

[120] M. Savić, M. Ivanović, M. Radovanović, Z. Ognjanović, A. Pejović, and T. Jakšić Krüger, "The structure and evolution of scientific collaboration in Serbian mathematical journals," *Scientometrics,* vol. 101, no. 3, pp. 1805-1830, December 2014.

[121] W. Glänzel and A. Schubert, "Analyzing scientific networks through co-authorship," in *Handbook of Quantitative Science and Technology Research*: Kluwer Academic Publishers, 2004, pp. 257-276.

[122] R. A. Hanneman and M. Riddle, "Introduction to social network methods," ed. Riverside, CA: University of California, Riverside, 2005.

[123] M. Tomašević, *Algoritmi i strukture podataka*. Beograd: Akademska misao, 2008.

[124] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks,* vol. 1, no. 3, pp. 215-239, 1978/01/01 1978.

[125] P. Bonacich, "Power and centrality: A family of measures," *American journal of sociology,* pp. 1170-1182, 1987.

[126] M. E. Newman, "The mathematics of networks," *The new palgrave encyclopedia of economics,* vol. 2, no. 2008, pp. 1-12, 2008.

[127] P. W. Holland and S. Leinhardt, "Transitivity in structural models of small groups," *Comparative Group Studies,* 1971.

[128] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature,* vol. 393, no. 6684, pp. 440-442, 1998.

[129] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Physical review E,* vol. 69, no. 2, p. 026113, 2004.

[130] M. S. Granovetter, "The strength of weak ties," *American journal of sociology,* pp. 1360-1380, 1973.

[131] M. Girvan and M. E. Newman, "Community structure in social and biological networks," *Proceedings of the national academy of sciences,* vol. 99, no. 12, pp. 7821-7826, 2002.

[132] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the national academy of sciences,* vol. 103, no. 23, pp. 8577-8582, 2006.

[133] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment,* vol. 2008, no. 10, p. P10008, 2008.

[134] N. Akhtar, "Social network analysis tools," in *Communication Systems and Network Technologies (CSNT), 2014 Fourth International Conference on*, 2014, pp. 388-392: IEEE.

[135] S. P. Borgatti, M. G. Everett, and L. C. Freeman, "Ucinet for Windows: Software for social network analysis," ed: Harvard, MA: Analytic Technologies, 2002.

[136] A. Mrvar and V. Batagelj, "Analysis and visualization of large networks with program package Pajek," *Complex Adaptive Systems Modeling,* vol. 4, no. 1, pp. 1-8, 2016.

[137] M. Bastian, S. Heymann, and M. Jacomy, "Gephi: an open source software for exploring and manipulating networks," *ICWSM,* vol. 8, pp. 361-362, 2009.

[138] D. Hansen, B. Shneiderman, and M. A. Smith, *Analyzing social media networks with NodeXL: Insights from a connected world*. Morgan Kaufmann, 2010.

[139] A. Zrnec and D. Lavbič, "Social network aided plagiarism detection," *British Journal of Educational Technology,* 2015.

[140] A. Zrnec and D. Lavbič, "The Role of Social Connections in Plagiarism Detection," in *International Workshop on Learning Technology for Education in Cloud*, 2015, pp. 54-63: Springer.

[141] L. Šubelj, Š. Furlan, and M. Bajec, "An expert system for detecting automobile insurance fraud using social network analysis," *Expert Systems with Applications,* vol. 38, no. 1, pp. 1039-1052, 2011.

[142] J. M. Bolland, "Sorting out centrality: An analysis of the performance of four centrality models in real and simulated networks," *Social networks,* vol. 10, no. 3, pp. 233-253, 1988.

[143] T. W. Valente, K. Coronges, C. Lakon, and E. Costenbader, "How correlated are network centrality measures?," *Connections (Toronto, Ont.),* vol. 28, no. 1, p. 16, 2008.

[144] M. Mišić, M. Milanović, and J. Protić, "Vizuelizacija rezultata detekcije plagijarizma u izvornom programskom kodu," *InfoM,* vol. 15, no. 57, pp. 11-18, 2016.

[145] (Accessed 10 February). *Mossum*. Available: https://github.com/hjalti/mossum

[146] (10 February). *Top 30 Social Network Analysis and Visualization Tools*. Available: http://www.kdnuggets.com/2015/06/top-30-social-network-analysis-visualization-tools.html/2

[147] J. Barnes and P. Hut, "A hierarchical O (N log N) force-calculation algorithm," *nature,* vol. 324, no. 6096, pp. 446-449, 1986.

[148] M. Ghoniem, J.-D. Fekete, and P. Castagliola, "On the readability of graphs using node-link and matrix-based representations: a controlled experiment and statistical analysis," *Information Visualization,* vol. 4, no. 2, pp. 114-135, 2005.

[149] "Preparation of Papers for IEEE Transactions and Journals," IEEE2013, Available: https://www.ieee.org/documents/transactions_journals.pdf.

[150] J. M. Olson and M. P. Zanna, "Attitudes and attitude change," *Annual review of psychology,* vol. 44, no. 1, pp. 117-154, 1993.

[151] J. A. Krosnick, C. M. Judd, B. Wittenbrink, D. Albarracín, B. T. Johnson, and M. P. Zanna, "The measurement of attitudes," *The handbook of attitudes,* vol. 21, p. 76, 2005.

[152] J. Pallant, *SPSS: priručnik za preživljavanje, prevod 3. izdanja*. Mikro knjiga, 2009.

[153] S. Fajgelj, *Psihometrija: metod i teorija psihološkog merenja*. Centar za primenjenu psihologiju, 2003.

[154] Z. Kovačić, *Multivarijaciona analiza*. Beograd: Univerzitet u Beogradu, Ekonomski fakultet, 1994.

[155] J. P. Stevens, *Applied multivariate statistics for the social sciences*. Routledge, 2012.

Списак скраћеница

ACM - *Association for Computing Machinery*

ALU - *Arithmetic Logic Unit*

API - *Application Programming Interface*

APT - *Abstract Parse Trees*

AST - *Abstract Syntax Trees*

BOSS - *BOSS Online Submission System*

CPU - *Central Processing Units*

CUDA - *Compute Unified Device Architecture*

ECTS - *European Credit Transfer and Accumulation System*

FCFS - *First Come First Served*

FPGA - *Field Programmable Gate Array*

GPGPU - *General-Purpose computing on Graphics Processing Units*

GPU - *Graphic Processing Units*

GST - *Greedy String Tiling*

HTML - *HyperText Markup Language*

IEEE - *Institute of Electrical and Electronics Engineers*

ILP - *Instruction Level Parallelism*

KMP - *Knuth-Morris-Pratt* алгоритам

LMS - *Learning Management Systems*

MML - *Minimum Match Length*

MOSS - *Measure Of Software Similarity*

MPI - *Message Passing Interface*

NLP - *Natural Language Processing*

OpenCL - *Open Computing Language*

OpenMP - *Open MultiProcessing*

OS1 - *Operativni sistemi 1*

PDG - *Program Dependence Graph*

PNG - *Portable Network Graphics*

POSIX - *Portable Operating System Interface [for uniX]*

PP1 - *Praktikum iz programiranja 1*

PP2 - *Praktikum iz programiranja 2*

PTK - *Parse Tree Kernel*

RKR-GST - *Running Karp Rabin Greedy String Tiling*

SID - *Software Integrity Diagnosis*

SIM - *software SIMilarity tester*

SIMD - *Single Instruction Multiple Data*

SM - *Streaming Multiprocessor*

SMT - *Simultaneous Multithreading*

SPD - *Sherlock Plagiarism Detector*

STL - *Standard Template Library*

YAP - *Yet Another Plagiarism detection system*

Списак слика

СПИСАК ТАБЕЛА

Биографија аутора

Марко Мишић, дипломирани инжењер електротехнике – мастер, рођен је 01.03.1984. у Смедереву, Република Србија од мајке Лидије и оца Југослава. Основну школу „Ђура Јакшић“ и гимназију „Бранко Радичевић“ је завршио у Ковину, оба пута као носилац Вукове дипломе и ученик генерације. Од почетка школовања је показивао склоности ка природним наукама и учествовао на такмичењима и семинарима младих истраживача.

Електротехнички факултет у Београду је уписао 2003. године. Све факултетске обавезе је завршио у року и дипломирао октобра 2007. године са просечном оценом 9,21 током студија и оценом 10 на дипломском испиту. Дипломске академске студије – мастер је уписао новембра 2007. године и завршио их априла 2010. године са просечном оценом 10 током студија и оценом 10 на мастер раду под насловом „Упоредна анализа паралелних алгоритама за сортирање података“. Докторске академске студије је уписао у децембру 2010. године.

Током студирања је четири године био ангажован као студент демонстратор на предметима из области програмирања и архитектуре рачунара. Од фебруара 2008. године је био хонорарно ангажован, а од октобра 2009. је запослен на Електротехничком факултету као сарадник у настави, а затим од септембра 2011. као асистент на Катедри за рачунарску технику и информатику. На Електротехничком факултету Марко Мишић учествује у извођењу наставе из већег броја предмета.

У летњим месецима 2009. године је обавио стручно усавршавање у компанији NVIDIA у Сједињеним Америчким Државама у трајању од 14 недеља, а током лета 2013. је учествовао на двомесечном програму „Summer of HPC“ на Универзитету Единбург, Велика Британија. Током 2008. години је био члан управног одбора Удружења студената електротехнике Европе и главни организатор сајма послова „JobFair 08 – Креирај своју будућност”. Марко Мишић говори енглески језик, а служи се француским и немачким.

Марко Мишић је аутор 33 научне публикације. Аутор је шест радова у научним часописима, од чега су два рада објављена у часописима са *impact* фактором. У зборницима радова међународних скупова је објавио осам научних радова, док је на скуповима националног значаја објављивао 19 пута.

Главне области интересовања су му паралелно и дистрибуирано програмирање са нагласком на програмирање графичких процесора као процесора опште намене. У последњих неколико година већу пажњу посвећује едукационим алатима и технологијама, библиографским анализама и методама за анализу социјалних мрежа. Нарочиту пажњу посвећује проблему детекције плагијаризма у изворном програмском коду.

A. Истраживање ставова и пракси студената на тему плагијаризма у програмском коду

Плагијаризам у програмском коду је сложен феномен који умногоме зависи од норми које га идентификују и одређују. Притом, ова појава је изразито штетна са становишта знања и вештина које будући инжењери рачунарства и информатике треба да поседују. Питање плагијаризма у програмском коду је нарочито осетљиво и због великог броја јавно доступних извора који се могу искористити за формирање плагијата и тенденције колективне израде програмског кода као последице колаборативног учења.

С обзиром да људски фактор пресудно одређује да ли ће он бити испољен или не, било је од значаја да се испита како студенти Електротехничког факултета Универзитета у Београду разумеју плагијаризам и какве ставове имају према њему. Са друге стране, значајно је било истражити колико су студенти склони да подлегну различитим видовима плагијаризма и различите праксе које се у том смислу јављају. Истраживање је искоришћено и у циљу испитивања ширих ставова студената Електротехничког факултета на тему академске честитости и академског интегритета.

A.1. Инструменти коришћени у истраживању

Да би се одговорило на претходно поменута питања, креиран је упитник експлоративног типа који се, поред општих питања, састојао од три креирана инструмента који користе Ликертове скале процене.

Креирани инструменти су:

1) скала ставова студената према плагијаризму и академској честитости,
2) скала учесталости пракси које се могу подвести под плагијаризам уопштено и плагијаризам у програмском коду,
3) скала процене плагијаризма у програмском коду.

Овакав тип упитника подразумева непостојање снажних претпоставки о феномену који се истражује, па су стога и сама питања нешто разноврснија. Оно што је била основна претпоставка у креирању инструмената, јесте најдоминантнија теорија социјалних ставова унутар психологије, која постулира три димензије става: афективну, когнитивну и бихејвиоралну [150, 151]. У складу са тим, уз мање модификације, три скале процене су обухватиле ставове према академској честитости, процењивање шта је плагијаризам у програмском коду и колико су студенти склони да прибегавају праксама плагијаризма.

Укупно су била постављена 74 питања са понуђеним одговорима, као и поље за слободан коментар на крају упитника. Скала ставова студената према плагијаризму и академској честитости је садржала 27 ставки. Скала учесталости пракси које се могу подвести под плагијаризам је садржала 30 ставки, док је скалу процене плагијаризма у програмском коду чинило 27 ставки.

Укупно 68 ставки је било засновано на Ликертовим скалама процене, а испитаници су давали одговоре у опсегу од 1 до 5, где оцена 1 представља најмање, а оцена 5 представља највеће слагање. У оквиру 6 ставки је коришћен приступ са присилним избором за процену одређених феномена или сличности конкретних програмских сегмената. Одређен број питања је био негативно формулисан.

На почетку упитника су постављена основна питања о испитанику, као што су пол, година студија, дужина студирања, просечна оцена, начин финансирања и одсек. Прва група питања је обухватала питања на тему дефиниције плагијаризма, информисаности студената са формалним документима који регулишу ово питање, утицаја феномена на образовање и науку, упознатости са праксама детекције од стране наставника, одговорности за почињено дело и дисциплинских казни за пријављене починиоце.

Друга скала је служила за процену пракси студената у смислу плагијаризма уопштено и плагијаризма у програмском коду. Питања су обухватала праксе које постоје приликом израде програмског кода у оквиру израде домаћих и пројектних задатака, као и праксе које се односе на понашање студената на испитима.

Трећа скала се односила на процене студената у којој мери одређена понашања приликом израде задатака представљају чињење плагијаризма, као и на процене у вези конкретних модификација програмског кода. У оквиру трећег инструмента задата су четири конкретна програмска сегмента који се састоје од парова кодова за које је било потребно да студенти процене у којој мери су слични. Они су засебно приказани у Табели А.2.

Анализа резултата је показала да је интерна конзистентност за све три скале на задовољавајућем нивоу (Кронбахова α≈0,6). Кронбахова алфа [152, 153] говори у којој мери различити испитаници слично одговарају на слична питања, а рачуна се по формули:

$$\alpha = \frac{n}{n-1}\left(\frac{Sx^2 - \sum Ss^2}{Sx^2}\right) \quad (A.1)$$

где је *n* број ставки у оквиру упитника, Sx^2 варијанса скора, а $\sum Ss^2$ сума варијанси ставки. Сматра се да је интерна конзистентност упитника на задовољавајућем нивоу уколико је Кронбахова α већа од 0,5. Нешто нижа Кронбахова алфа није изненађујућа, с обзиром на интенцију креатора инструмента да питања у оквиру упитника обухвате разноврсност феномена за сваку скалу понаособ, уместо да буду фокусирани на уско дефинисани конструкт.

Табела А.1. Питања постављена у оквиру истраживања

Основни подаци		
Питање	**Тип питања**	**Могуће вредности**
Пол	Више избора	М / Ж
Година студија	Више избора	1 до 6
Колико година укупно студирате?	Падајућа листа	1 до 15
Просек	Опсег	до 6.50, 6.5 - 7.5, 7.5 - 8.5, 8.5 - 9.5, преко 9.5
Начин финансирања	Више избора	Буџет, самофинансирајући
Одсек	Више избора	EF, IR, SI, OT, OG, OF, OE, OS

Ставови		
Питање	**Тип питања**	**Могуће вредности**
Следеће ставке испитују различите ставове у вези са плагијаризмом. Молимо Вас да за сваку ставку одговорите у којој се мери слажете са њом.	Ликерт	1 - 5
Плагијаризам програмског кода је свако коришћење туђег програмског кода без навођења аутора.		
Плагирање представља нарушавање принципа академске честитости.		
Неки аспекти плагијаризма нису озбиљно нарушавање принципа академске честитости.		
Плагирање туђег рада у науци је недопустиво.		
Плагирање туђег рада у образовању је недопустиво.		
Плагирање туђег рада у науци треба да води поништавању стечене дипломе.		
Плагирање туђег рада у образовању треба да води поништавању стечене дипломе.		
Студенти приликом уписа на факултет треба да буду упознати шта је плагијаризам и на које начине се кажњава.		
Високошколске институције треба да развијају своје механизме за детектовање плагијаризма.		
Преписивање на испиту је врста плагијаризма.		
Плагијаризам нарушава смисао научног рада.		
Плагијаризам нарушава кредибилитет образовног процеса.		
Упознат сам са правилницима о дисциплинској одговорности студената на мом факултету.		
Наставници користе алате за детекцију плагијаризма у програмском коду.		
Подржавам коришћење алата за детекцију плагијаризма од стране наставника.		
Колики је претпостављени удео домаћег задатка у односу на коначну оцену на предмету да би се студент одлучио да почини плагијаризам?	Скала процената	5%, 10%, 20%, 30%, 40%, 50% и више
Колика сличност програмског кода у процентима треба да произведе дисциплинску одговорност студената?	Скала процената	10%, 20%, 30%, 40%, 50% и више
Ко је одговоран за плагијаризам у програмском коду?	Ликерт	1 - 5
Особа која користи туђ програмски код без цитирања извора		
Наставници који се не развијају процедуре за спречавање плагијаризма		
Непостојање јасних дефиниција и правила шта је плагијаризам		
Особа која даје свој програмски код		
Следеће тврдње представљају последице које студент може да сноси за плагирање туђег рада. Молимо Вас да за сваку ставку одговорите у којој се мери слажете са њом.	Ликерт	1 - 5
Студент треба да буде удаљен са факултета.		
Студент треба да буде привремено удаљен са факултета.		
Студент треба да буде пријављен дисциплинској комисији.		
Студенту треба да буду одузети сви поени са задатка на коме је почињен плагијаризам.		
Студенту треба да буду одузет део поена са задатка на коме је почињен плагијаризам.		
Студента треба упозорити на недолично понашање.		

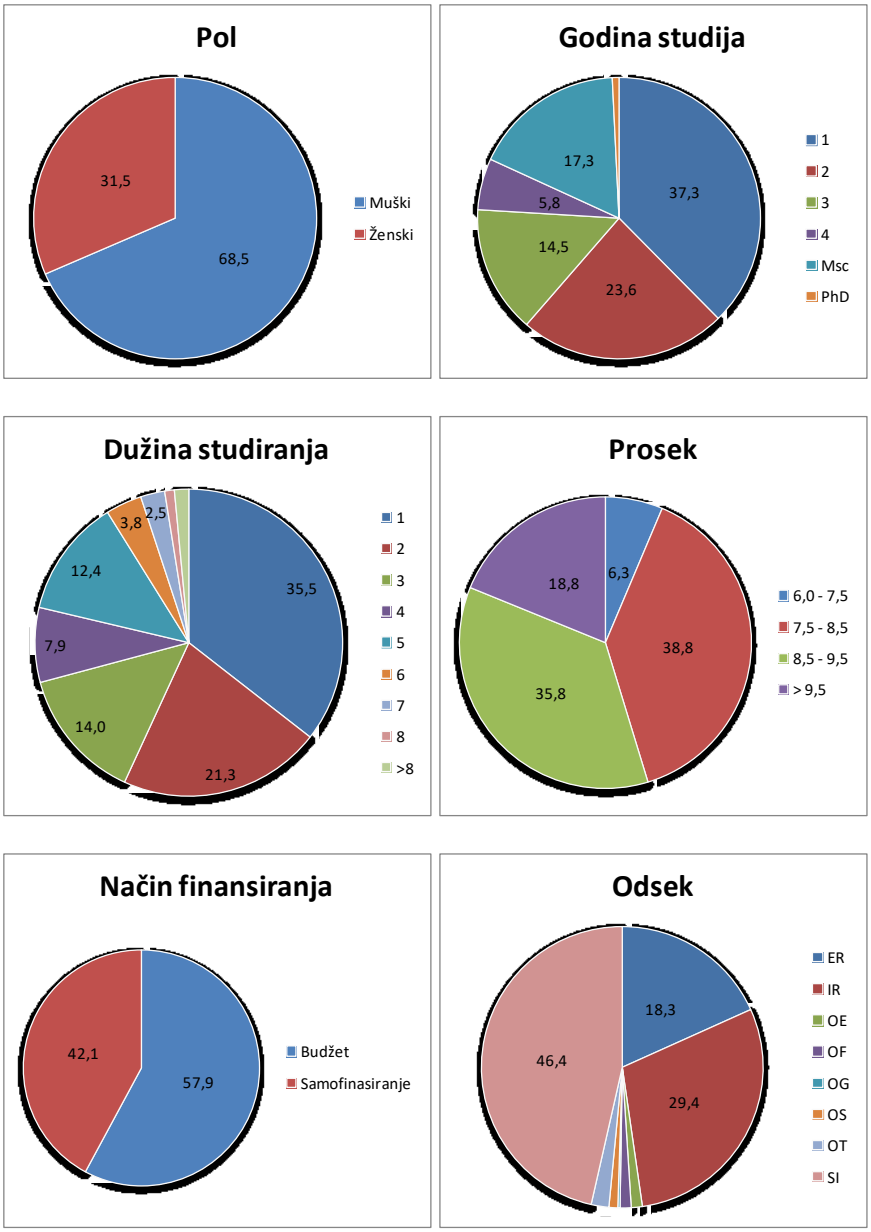
Праксе		
Питање	**Тип питања**	**Могуће вредности**
Коришћење алата за детекцију плагијаризма утиче на моје понашање приликом израде и предаје домаћег задатка.	Ликерт	1 - 5
Следеће ставке испитују различите праксе које студенти користе приликом писања програмског кода. Молимо вас да за сваку од тврдњи процените колико сте Ви у њој често учествовали.	Ликерт	1 - 5
Слао сам свој домаћи задатак другим студентима.		
Користио сам и предавао комплетно туђ задатак као свој.		
Користио сам или предавао делове туђег задатак као свој рад.		
Дискутујем о дизајну програмског решења са другим колегама.		
Показујем свој програмски код приликом дискусије о решењу задатка са другим колегама.		
Шаљем свој програмски код другима приликом дискусије о решењу задатка са колегама.		
Радим на решавању домаћих задатака заједно са другим колегама.		
Користим делове туђег рада са интернета или из других часописа и приказујем као свој.		
Следеће ставке испитују различите праксе нарушавања академске честитости. Молимо вас да за сваку од тврдњи процените колико сте Ви у њој често учествовали, а колико је она према Вашем мишљењу генерално распрострањена међу студентима нашег факултета.	Ликерт	1 - 5
Платио сам колеги да ми реши домаћи задатак.		
Колега ми је платио да му урадим домаћи задатак.		
Видео сам или чуо да је неко платио колеги да му реши домаћи задатак.		
Урадио сам задатак на испиту за другог студента.		
Колега је урадио задатак за мене на испиту.		
Видео сам или чуо да је неко свом колеги урадио задатак на испиту.		
Полагао сам испит уместо колеге.		
Колега је излазио на испит уместо мене.		
Видео сам или чуо да је неко полагао испит уместо колеге.		
Полагао сам испит коришћењем “бубица”.		
Видео сам или чуо да је колега полагао испит коришћењем “бубица”.		
Процене		
Питање	**Тип питања**	**Могуће вредности**
Да ли следећа понашања представљају плагијаризам?	Ликерт	1 - 5
Копирање комплетног програмског кода представља плагијаризам.		
Копирање блока (неколико линија) програмског кода представља плагијаризам.		
Копирање једне линије програмског кода представља плагијаризам.		
Копирање и измена дела или комплетног програмског кода представља плагијаризам.		
Копирање коментара у програмском коду представља плагијаризам.		
Копирање структуре програмског решења представља плагијаризам.		
Копирање корисничког интерфејса програма представља плагијаризам.		
Коришћење туђих скупова података за тестирање програма представља плагијаризам.		
Добровољно писање програмског кода другом студенту представља плагијаризам.		
Два или више студената раде групно на задатку који треба да се ради индивидуално и предају веома сличне програмске кодове.		

Писање програмског кода уз новчану или другу надокнаду другом студенту представља плагијаризам.	Ликерт	1 - 5
Поновно коришћење дела кода који је већ предат у оквиру претходног решеног домаћег задатка представља плагијаризам.		
Коришћење програмског кода из књиге или других званичних материјала (слајдова, материјала са вежби) представља плагијаризам.		
Коришћење програмског кода са Интернета или незваничних материјала (скрипти, задатака решених од стране студената) представља плагијаризам.		
У којој мери сматрате да следеће измене програмског кода могу прикрити плагијаризам?	Ликерт	1 - 5
Замена имена идентификатора (променљивих, константи, типова, имена функција)		
Замена редоследа блокова кода		
Замена редоследа појединачних линија унутар блока		
Додавање или уклањање коментара		
Додавање или уклањање форматирања (идентација, бланко знаци, нови ред)		
Замена петље еквивалентном		
Замена позива функције телом функције		
Замена позива функције алтернативном, сличном функцијом		
Измена исписа програма		

А.2. Испитивани узорак

У спроведеном истраживању је учествовало 394 студената са свих година и нивоа студија, углавном са Одсека за рачунарску технику и информатику (IR), Одсека за софверско инжењерство (SI) и заједничке прве године Електротехничких одсека (ER). Око две трећине испитаника су чинили мушкарци, а око трећину жене, што је у складу са статистикама уписа на Електротехнички факултет у претходних неколико година. Највећи проценат испитаника долази са прве године студија, али су сразмерно заступљене и остале године студија. Нешто већи број испитаника долази са мастер студија, јер је у оквиру курса Социолошки и професионални аспекти рачунарства одржано предавање на тему плагијаризма у програмском коду, након кога је студентима прослеђен упитник. То је повећало интересовање за само истраживање.

Испитивање је спроведено коришћењем електронске анкете. Тамо где је било могуће, испитаници су попуњавали упитник на почетку термина лабораторијских вежби, махом у оквиру предмета Практикум из програмирања 2 на првој години Одсека IR и Одсека SI. До осталих испитаника је упитник дистрибуиран путем листи електронске поште на предметима различитих година студија којима је аутор истраживања имао приступ. Карактеристике испитиваног узорка се могу видети на Слици А.1.



Слика А.1. Карактеристике испитиваног узорка

У погледу дужине студирања, заступљен је и значајан број студената који студирају дуже од предвиђеног трајања студија. У узорку су присутни испитаници различитих просечних оцена студирања и начина финансирања. Испитаници углавном студирају Одсек IR и Одсек SI, као и заједничку прву годину, што и јесте била циљна група овог истраживања. Студенти на овим одсецима имају већи број програмерских предмета и самим тим је плагијаризам у програмском коду значајнихе заступљен. Учешће осталих испитаника је мање од 5 процената и махом је последица предмета које су бирали на одсецима IR и SI или присуства на заједничким студентским листама електронске поште до којих је упитник прослеђен.

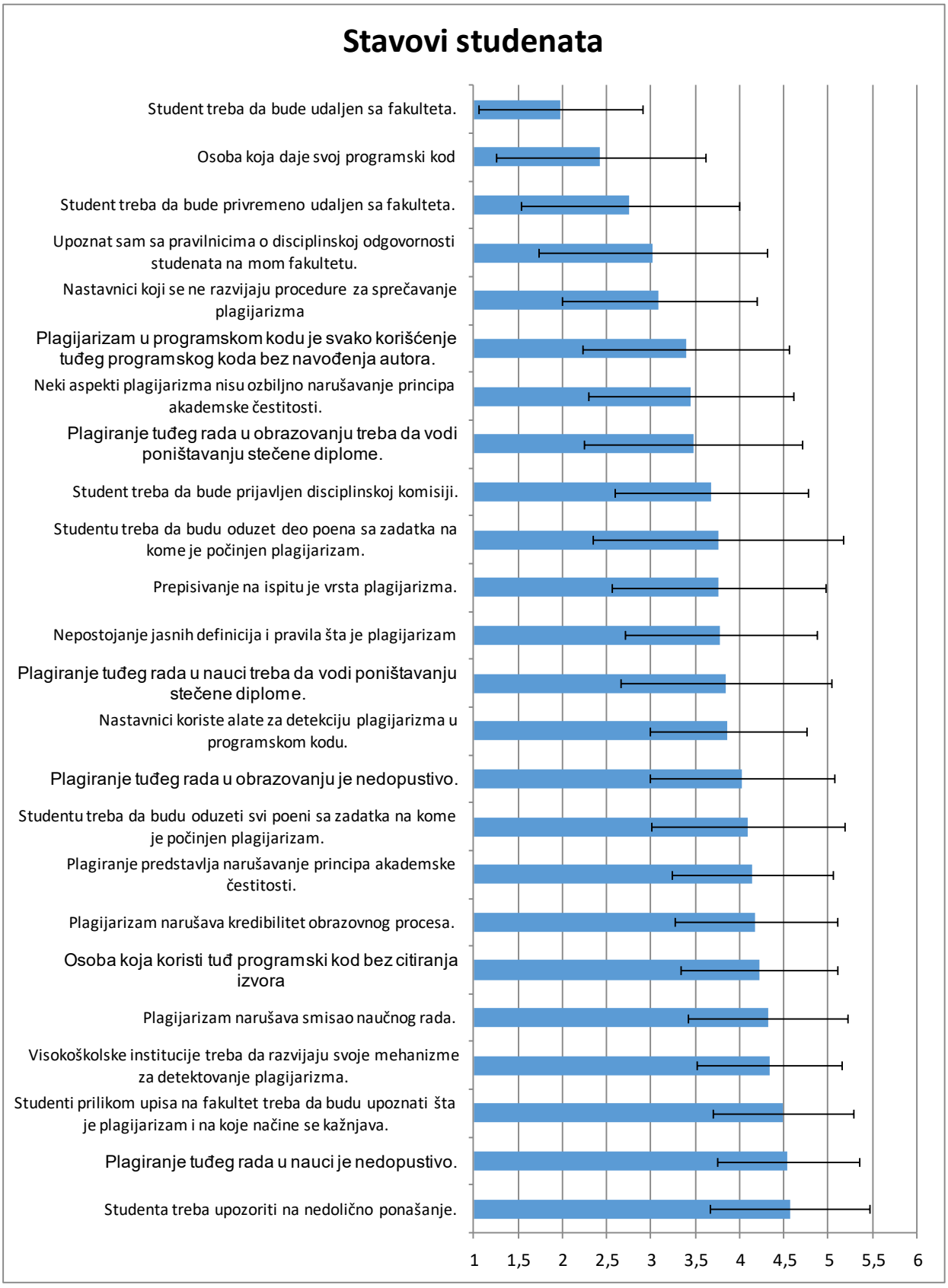
А.3. Резултати истраживања

У овом поглављу су приказани резултати спроведеног истраживања о ставовима и праксама студената на тему плагијаризма у програмском коду. Резултати су обрађени софтверским пакетом SPSS, а графикони направљени у пакету Excel. Најпре су дате аритметичке средине и стандардне девијације за сва питања, а затим су издвојени и најинтересантнији резултати. На крају је приказана факторска анализа.

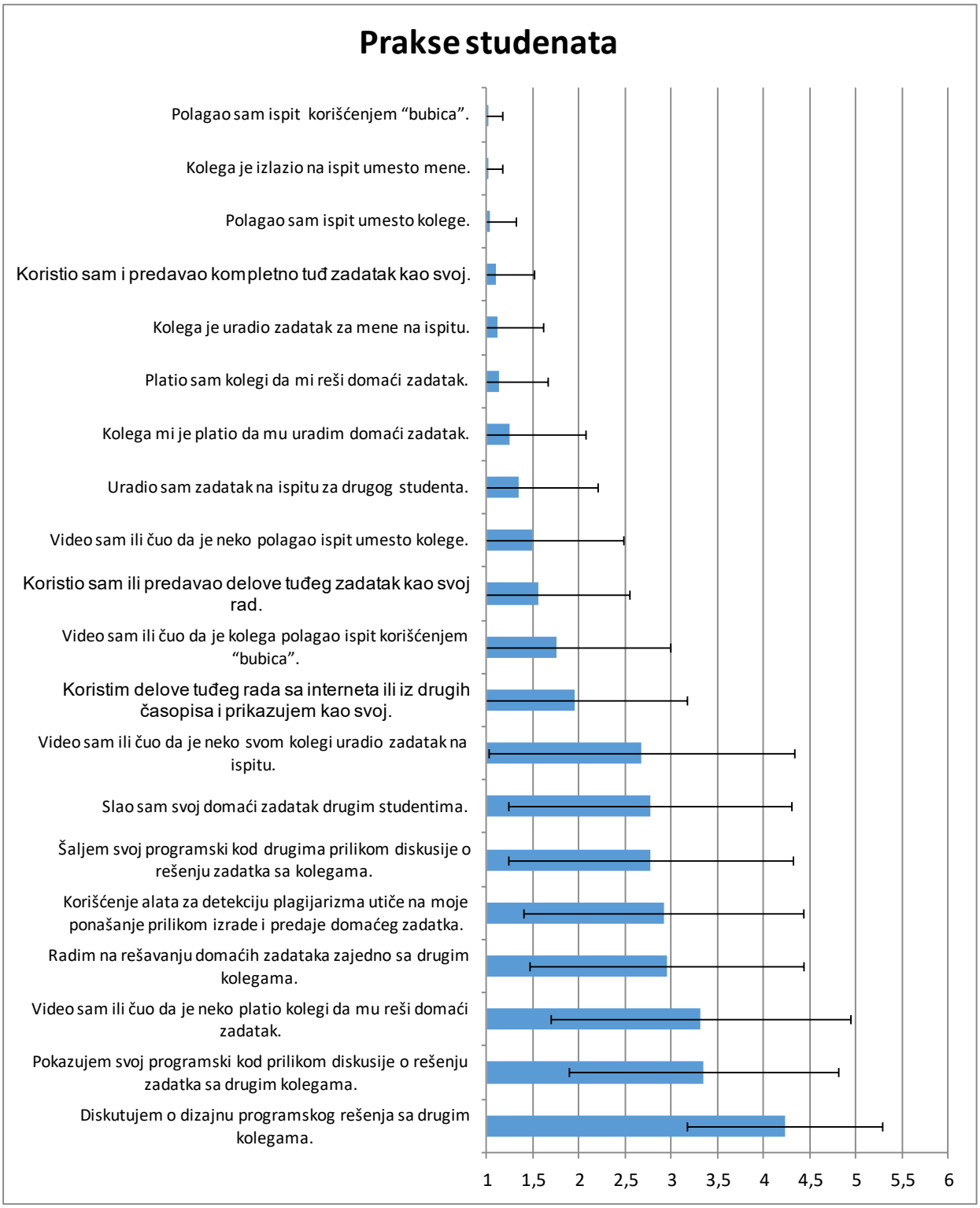
А.3.1. Ставови студената

Слика А.2. приказује аритметичке средине и стандардне девијације одговора на питања о ставовима студената на тему плагијаризма у програмском коду. Резултати анкете показују да се нешто више од половине испитаника слаже са дефиницијом плагијаризма у програмском коду која говори да је то свако коришћење туђег програмског кода без навођења аутора. Међутим, око четвртине испитаника није сасвим сигурно, док се око петина испитаника не слаже. Већина се ипак слаже да наставна установа треба да развија механизме за детекцију ове појаве и да студенти треба да буду упознати шта представља плагијаризам и на који начин се кажњава. Притом, око 60% студената није упознато са правилима у вези плагијаризма на Електротехничком факултету која су дефинисана у оквиру Правилника о дисциплинској одговорности студената [10]. Ипак, мора се приметити да овај правилник не дефинише прецизно појам плагијаризма, иако експлицитно наводи да није дозвољено подношење делимичних или потпуних плагијата у оквиру предиспитних и испитних обавеза.

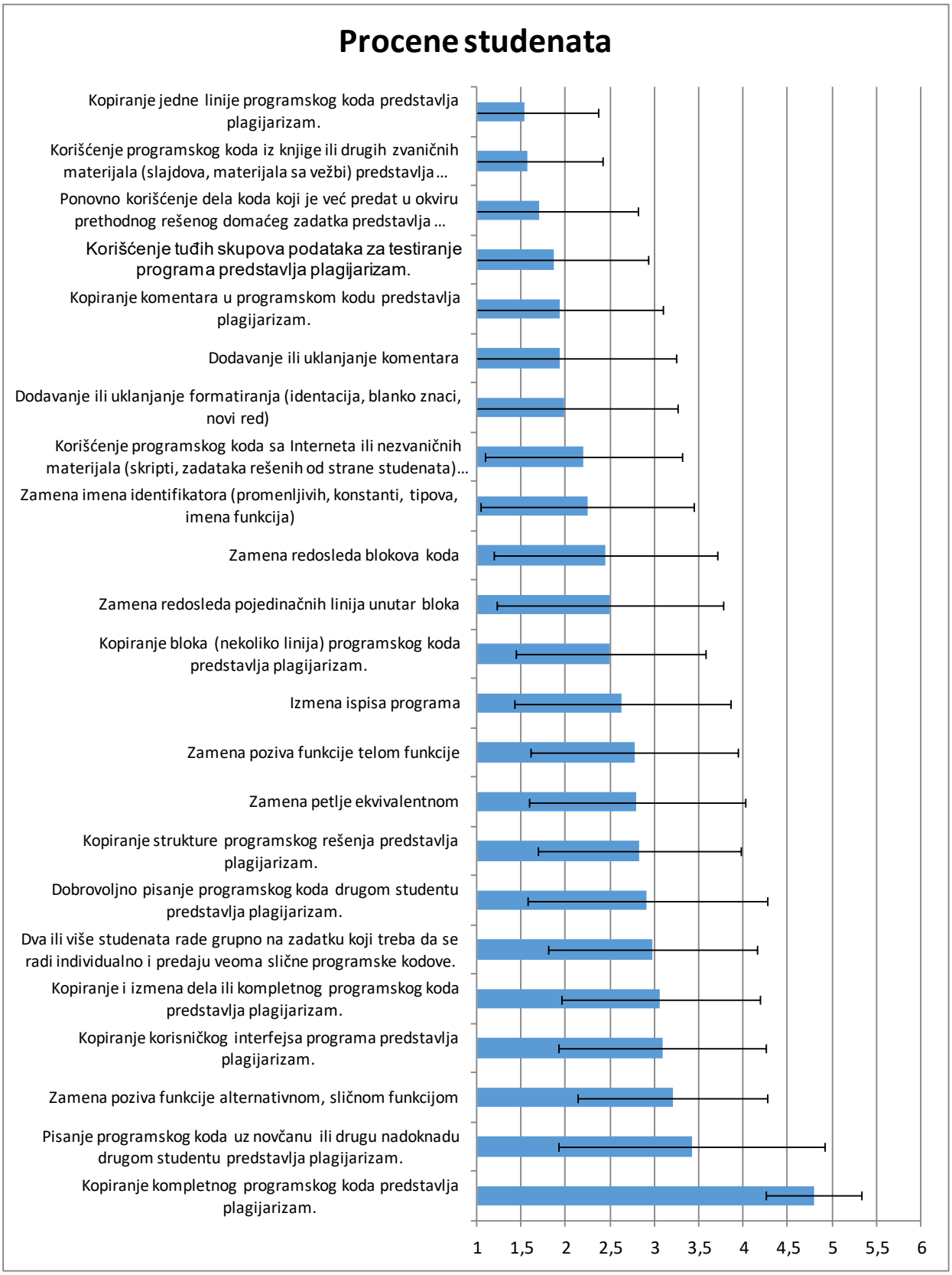
Велики број студената се слаже да плагијаризам представља нарушавање принципа академске честитости. Међутим, доста је интересантно приметити да постоје блажи погледи у односу на плагијаризам у образовању у односу на плагијаризам у науци. Исто се може приметити и у вези разлике између кажњавања ове појаве у образовању и науци, где већи број испитаника сматра да треба поништити стечену докторску диплому, него диплому са основних студија. Такође, стандардне девијације резултата на поменута питања су веће од просека, што говори о већој поларизацији испитаника.



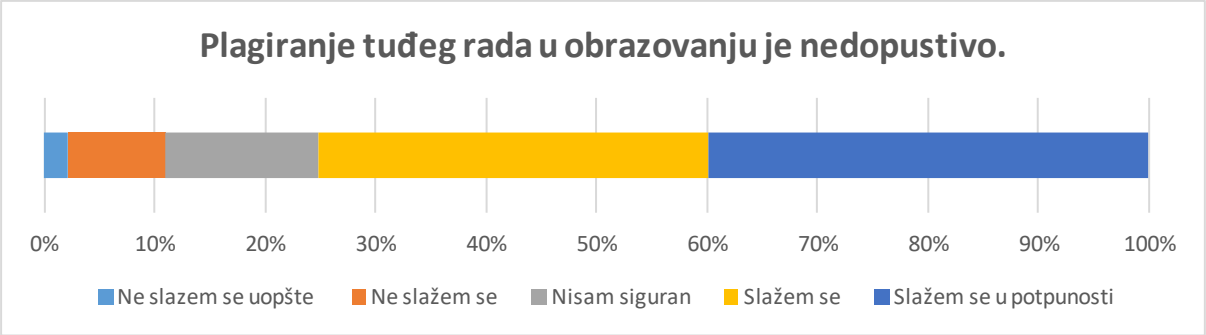
Слика А.2. Аритметичка средина и стандардна девијација за одговоре испитаника у вези ставова о плагијаризму и академској честитости



Слика А.3. Аритметичка средина и стандардна девијација за одговоре испитаника у вези пракси плагијаризма



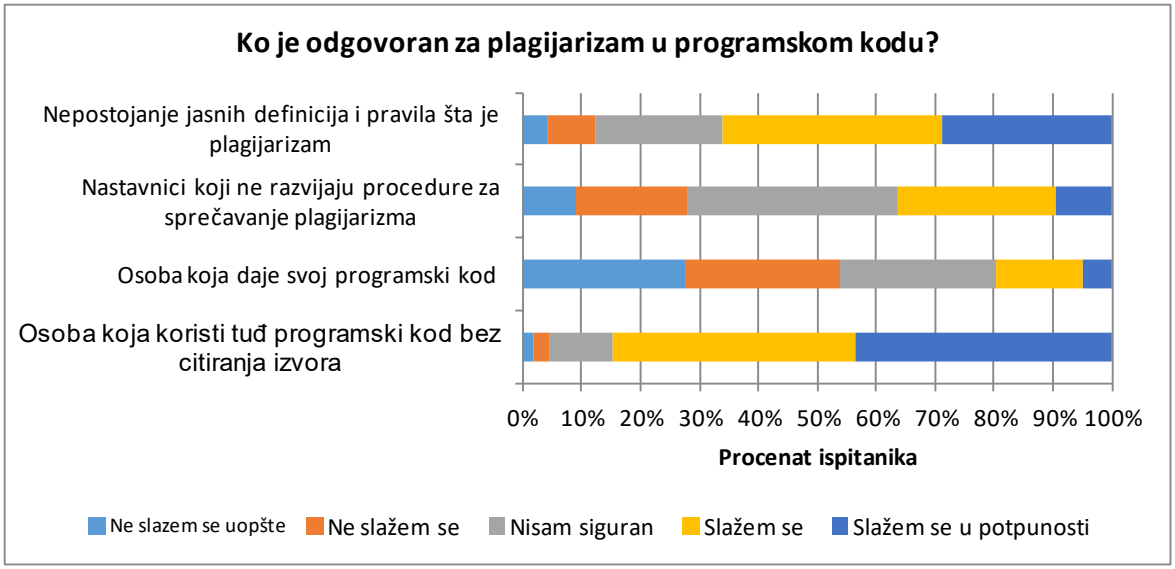
Слика А.4. Аритметичка средина и стандардна девијација за одговоре испитаника у вези процена плагијаризма



Слика А.5. Структура одговора на питање да ли је плагирање туђег рада у образовању допустиво

Уколико се погледа структура одговора испитаника на питање да ли је плагирање туђег рада у образовању допустиво (Слика А.5), долази се до запажања да сваки десети испитаник сматра да је плагијаризам допустив. Ако се у збир укључе и они који нису сигурни, око четвртина испитаника има став који би од стране велике већине наставника био окарактерисан као дискутабилан.

Истраживање говори да су испитаници у просеку склонији плагијаризму уколико је удео задатка у финалној оцени већи од 30%. Такав став има око 60% испитаника, што говори да су овом појавом потенцијално угроженији предмети у оквиру којих се задају обимнији пројектни задаци који захтевају шира и темељнија знања и вештине.



Слика А.6. Поређење одговора испитаника у вези одговорности за учињено дело

Такође, студенти се размимоилазе у ставовима на тему одговорности за почињено дело, што се примећује на Слици А.6. Док највећи број испитаника сматра да је за плагијаризам свакако одговорна особа која користи туђ програмски код без цитирања аутора,

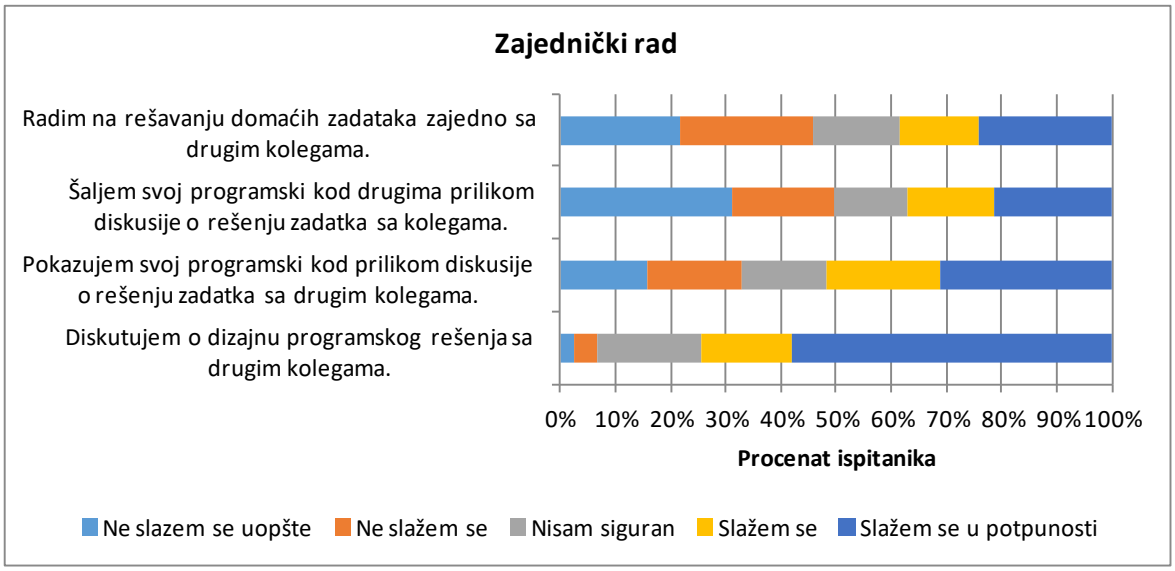
тек петина испитаника сматра да је одговорна и особа која је програмски код проследила. Давање програмског кода другима већина студената очигледно сматра као позитивну ствар и помоћ другима, а не као кршење академских правила и процедура. Може се сматрати да је ово у складу са културолошким цртама нашег поднебља које свакако форсира заједништво и помоћ другима. Такође, преко 60 процената студената сматра да је непостојање јасних правила и дефиниција шта је плагијаризам одговорно за распрострањеност ове појаве. Око 30 процената испитаника сматра да су одговорни и наставници који не развијају процедуре за спречавање ове појаве, мада преко 35 процената људи није сигурно да ли је то одговорност наставника или не.

Студенти у просеку сматрају да тек сличности од преко 50% треба да произведу дисциплинске мере према потенцијалним плагијаторима. Притом, студенти су склонији блажим санкцијама, као што су упозоравање студента на недолично понашање и делимично или комплетно одузимање бодова са спорне активности. Наведене санкције су најчешће у домену наставника. Међутим, дисциплински правилници најчешће прописују пријављивање студената одговарајућем дисциплинском органу факултета, са чиме се слаже тек око 50 процената испитаника. Дисциплинска комисија Електротехничког факултета у оваквим случајевима може да изрекне и казне привременог или трајног удаљавања студената са факултета са чиме се слаже тек мали број студената.

А.3.2. Праксе студената

Испитивање пракси студената је имало за циљ боље разумевање начина на који се студенти понашају приликом израде домаћих и пројектних задатака, како би им се лакше указало на то шта су дозвољена понашања, а шта не. Слика А.3. приказује аритметичке средине и стандардне девијације одговора на ова питања.

Запажа се да највећи број студената у извесној мери сарађује приликом израде домаћих и пројектних задатака, што се види на Слици А.7. Највећи број њих сарађује у виду дискусије о дизајну програмског решења са другим колегама, што се може сматрати позитивном праксом [14]. Међутим, не тако мали број испитаника учествује и у другим праксама које се најчешће не сматрају коректним. Преко 50 процената студената показује свој код другим колегама, а око 40 процената га прослеђује другима. Исто толики проценат испитаника наводи да заједнички ради на решавању домаћих задатака са другима, чак и када предметна правила захтевају да то буде самосталан рад.



Слика А.7. Праксе студената у вези заједничког рада

Анализа такође показује несклад између пракси слања и примања програмског кода или домаћих задатака. Око 40 процената студената је потврдило да је слало домаћи задатак другим студентима, али исто тако преко 90% студената категорички тврди да никада није предавало туђ домаћи задатак. Тек незнатан број признаје да је предавало комплетно туђ домаћи као свој рад, али зато око 7% признаје да је користило и предавало делове туђег домаћег задатка као своје, што се поклапа са ранијим истраживањима о преваленцији плагијаризма [28, 90] у опсегу од 5 до 10% укупног броја студената на неком предмету. Око четвртина испитаника користи туђе резултате са интернета или из литературе и приказује их као своје. Истраживање у Словачкој [16] је показало да је око 30% студената починило неки облик плагијаризма бар једном током студирања, тако да су добијени резултати у сагласности и са трендовима уоченим у другим истраживањима из отворене литературе.

Истраживање је укључивало и одређене обрасце понашања на испитима који би се могли подвести под категорију тешког нарушавања принципа академске честитости. Таква понашања су полагање испита уместо колеге, полагање испита путем бубица, преписивање задатака на испиту, плаћање трећем лицу за израду домаћег задатка и сл. Питања у оквиру ових ставки су била двојако формулисана. У оквиру једног скупа питања су испитиване личне праксе у смислу да ли је студент у некој од њих учествовао, док је у оквиру другог скупа испитивана преваленција појаве, а студенти су упитани да ли су видели или чули да је неко други вршио такву праксу.

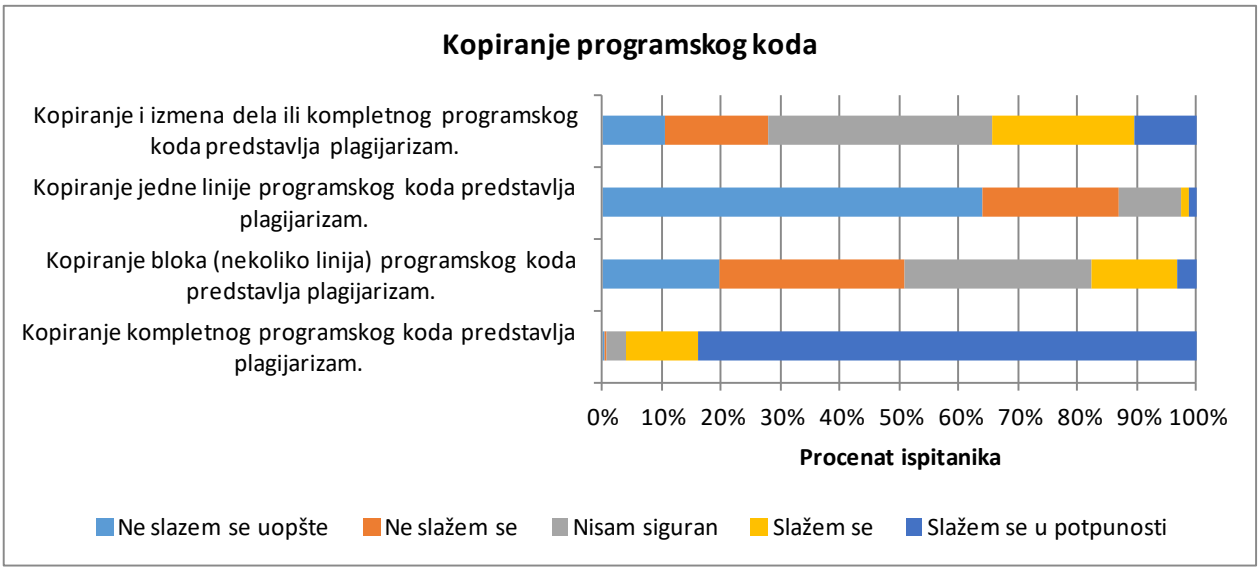
Резултати истраживања показују несагласност између ова два скупа питања. Много мањи број студената признаје да је сам учествовао у некој пракси, него да је видео или чуо да се пракса спроводи. То говори да су због социјалне пожељности одговора и повољне слике о себи студенти много мање спремни да признају сами чињење неког дела, што онда не значи да се нека пракса не спроводи. На пример, око 5% студената признаје да је урадило задатак за колегу на испиту, а тек 1% признаје да је неко за њих урадио задатак на испиту. Да контраст буде још већи, око 35 процената испитаника тврди да је видело или чуло да је неко свом колеги урадио задатак на испиту.

Слични резултати се добијају и у вези праксе плаћања колеги да уради домаћи или пројектни задатак. Иако тек мали број признаје да је тако нешто затражио од колеге (1 проценат) или да је добио такву понуду (око 4,5 процената), скоро 50% студената тврди да су му познати случајеви такве праксе. Знатно мање варијације постоје код пракси полагања испита уместо колеге или полагања испита путем бубица, где су случајеви такве праксе скоро безначајно заступљени. Такође, мање од 10 процената студената у оба случаја је видело или чуло за такву праксу. Мањи проценти се могу тумачити тиме што је овакву праксу знатно теже спровести, а Факултет је у прошлости донео како оштрије превентивне мере, тако и оштрије дисциплинске мере за два поменута случаја.

На крају, треба истаћи доста велике стандардне девијације за групу питања која се тичу лакших облика плагијаризма као што су слање и коришћење домаћих задатака, преписивање на испиту, показивање кода и сл. Из тога се може се извући закључак да је један број студената доста добро упознат са праксама и појавом плагијаризма у њиховом окружењу, док друге то уопште не интересује. Претпоставка је да они студенти који су морални и поштују правила се много не обазиру и не информишу о непоштеним праксама које други чине у свом окружењу.

А.3.3. Процене плагијаризма

Трећи део истраживања је обухватао процене студената да ли неко понашање или чињење представља плагијаризам или не. Реализован је кроз три групе питања. Прва група питања је обухватала процену да ли неко понашање представља плагијаризам или не, друга група питања се односила на модификације програмског кода које могу да се користе за прикривање плагијаризма, док је трећа група била везана за конкрете програмске сегменте.

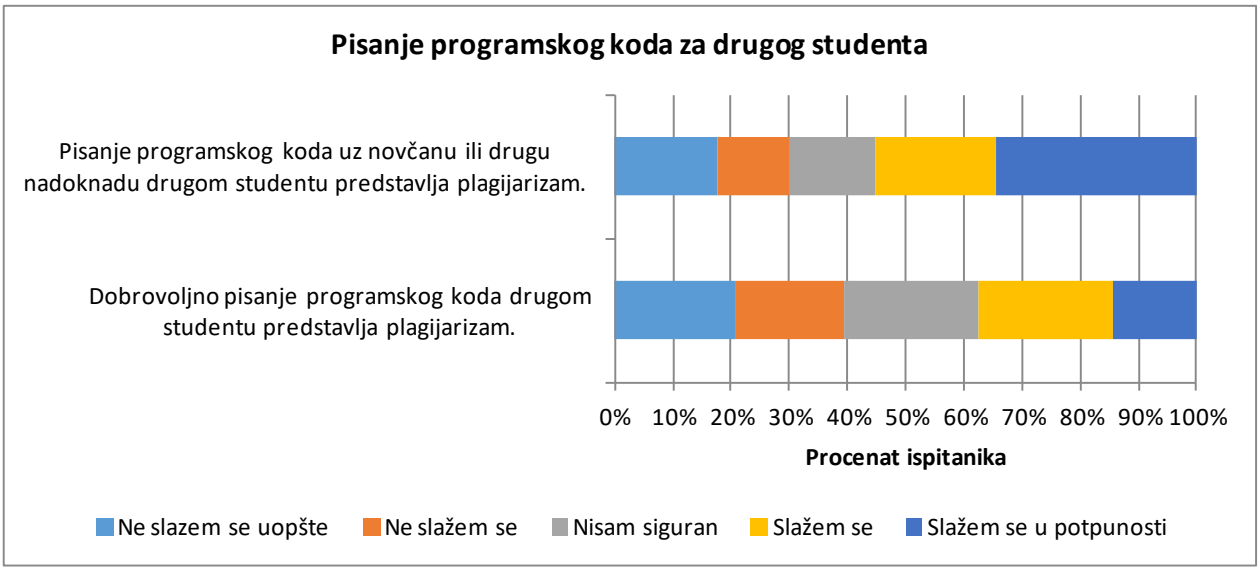


Слика А.8. Процене студената у вези копирања програмског кода

Студенти исправно препознају да копирање и предавање комплетног кода представља плагијаризам, али се у много мањој мери слажу да плагијаризам представља копирање једне линије или блока кода (Слика А.8.). Такође, око четвртине студената сматра да копирање и измена програмског кода не представљају плагијаризам, што се може сматрати забрињавајућим.

Када се гледају други аспекти у вези са програмским кодом, студенти имају још мекше погледе. Копирање коментара, скупова података за тестирање, па чак и структуре програмског решења и корисничког интерфејса за већину студената не представља плагијаризам. Док би се за прве две ствари и већина наставника сложила, копирање структуре програмског решења и дизајна корисничког интерфејса може представљати врсту интелектуалне крађе, поготову на курсевима који значајније покривају ове програмерске аспекте.

Интересантно је и различито гледање на израду програмског кода за другог колегу, као што се може видети на Слици А.9. Око 55% студената се слаже да писање кода за другог колегу уз новчану или другу надокнаду представља плагијаризам. Међутим, погледи на такву праксу се разликују уколико се то чини без накнаде, када тај број пада на 37%, уз још виши проценат оних који нису сигурни да ли то предствља плагијаризам или не. Слично важи и за групни рад на задатку који треба да се ради самостално, где тек 35% њих исправно процењује да то представаља плагијаризам.



Слика А.9. Процене студената у вези израде програмског кода за другог студента

Преко 90% студената добро процењује да коришћење званичних материјала као што су књиге, слајдови са предавања и сл. не представља плагијаризам. Међутим, проблематично је гледиште у вези незваничних материјала и скрипти са Интернета, где око 60% студената такође не сматра да коришћење таквих материјала представља нарушавање принципа академске честитости. Иако понекад корисни, овакви материјали често представљају извор нетачаних информација, па наставници често не гледају благонаклоно на њихову употребу.

Друга група питања у оквиру трећег дела се односила на различите измене које студенти могу да изврше у програмском коду да би прикрили плагијаризам. Студенти углавном исправно препознају да једноставније лексичке и структуралне измене не могу прикрити плагијаризам, попут замене имена идентификатора, редоследа блокова у коду, додавања или уклањања коментара и сл. Слично важи и за озбиљније измене као што су замена петљи еквивалентном, библиотечких позива алтернативним и сл., али уз нешто већи број оних који мисле да се тако плагијаризам може сакрити.

На крају истраживања о проценама плагијаризма, студентима су дата четири програмска сегмента (Табела А.2). Сваки сегмент се састојао од пара кодова. Три пара кодова су тако изабрана да представљају плагијаризам, а један програмски код у пару је настао трансформацијом из другог. У Сегменту А су учињене само лексичке модификације, док је над Сегментом С извршена озбиљнија трансформација кода. Сегмент D не представља плагијаризам, што се, и поред одређених структурних сличности, могло лако уочити због различите обраде која се унутар кодова у оквиру њега врши.

Табела А.2. Задати програмски сегменти за процену сличности

Сегмент А

```
    putchar('\n');
    //*******************************
    //stampanje koordinata matrice
    for (i = 0; i < dim; i++)
    {
      for (j = 0; j < dim; j++)
      {
        if (i != j)
          printf("[%d,%d] ", i, j);
        else printf("****");
      }
      putchar('\n');
    }
    //*******************************
    //ispitivanje da li je trougaona matrica
    bot = 0;
    top = 0;
    for (i = 0; i < dim; i++)
    for (j = 0; j < dim; j++)
    {
      if (i<dim && mat[i][j] == 0)bot++;
      if (i>dim && mat[i][j] == 0)top++;
    }
```

```
  printf("\n");
  for(i=0;i<m;i++){ //*ISPIS INFO MATRICE*//
    for(j=0;j<m;j++)
    {
      if(i!=j)printf("[%d,%d] ",i,j);
      else printf("XXXXX ");

    }
    printf("\n");
  }

  //***ISPITIVANJE DA LI JE TROUGAONA ***///
  dole=gore=0;

  for(i=0;i<m;i++)
    for(j=0;j<m;j++)
    {
      if(j>i && wut[i][j]==0)dole++;
      if(j<i && wut[i][j]==0)gore++;

    }
```

Сегмент В

```
int code(char c)
{
  if (islower(c)) return c - 'a';
  if (isupper(c)) return c - 'A' + 26;
  if (isdigit(c)) return c - '0' + 52;
  if (c == ' ') return 62;
  if (c == '.') return 63;
  return -1;
}
```

```
int vrednost(char c)
{
    if (c >= 'a' && c <= 'z') return c - 'a';
    if (c >= 'A' && c <= 'Z') return c - 'A' + 26;
    if (c >= '0' && c <= '9') return c - '0' + 52;
    if (c == ' ') return 62;
    if (c == '.') return 63;
    return -1;
}
```

Сегмент С

```
void obrada (char *ul_pomeraj, Elem *prvi){
  Elem *trenutni=prvi;
  int pomeraj=atol(ul_pomeraj);

  while(trenutni) {
    trenutni->sadrzaj->vreme_poj+=pomeraj;
    trenutni->sadrzaj->vreme_ukl+=pomeraj;
    trenutni=trenutni->sledeci;
  }
}
```

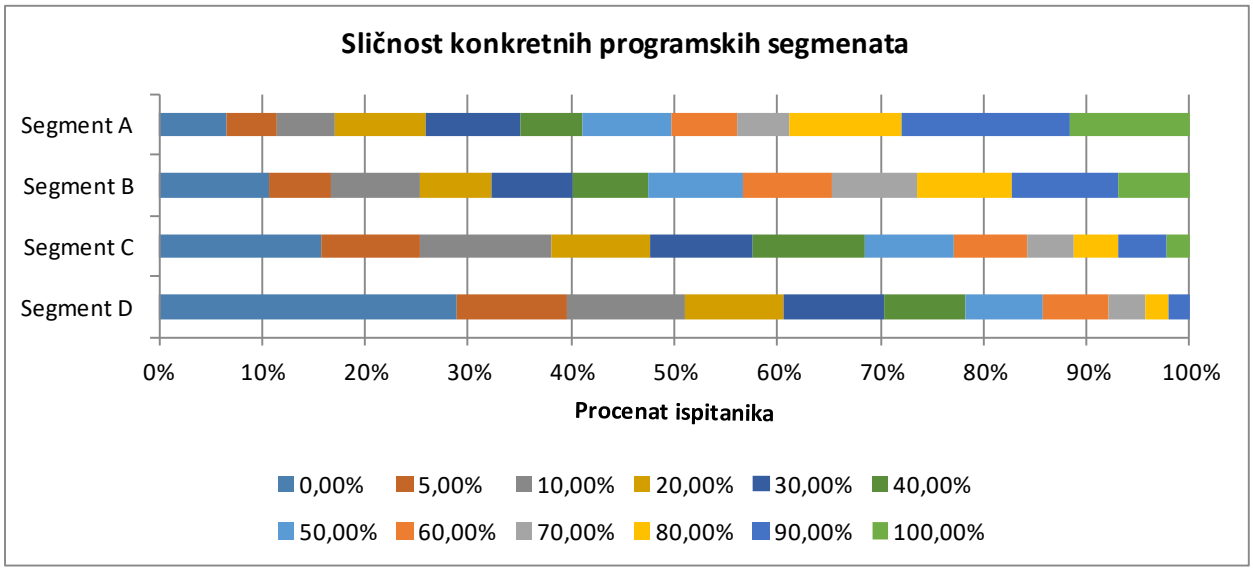
```
void titles ( Subtitle_list *list, int disp)
{
  Subtitle_list *current;

  for(current = list; current != NULL;
current=current->next)
  {
    current->info->t_appear+=disp;
    current->info->t_remove+=disp;
  }
}
```

Сегмент D

```
for (int i = 0; i < cols; i++) {
  int temp = rows, k = 0;
  for (int j = 0; j < rows; j++) {
    if (ind[j] == 1) temp--;
    else mat[k++][i] = mat[j][i];
  }
rows = temp;
```

```
s = 0;
for (int i = 0; i < row; i++) {
   for (int j = 0; j < col; i++)
      if (ind[i+j] == 1)
        s += mat[i][j];
}
```

Слика А.10. приказује резултате процене студената. Испитаници су требали да одреде колико сличним сматрају парове кодова у оквиру сегмента. Иако се проценти сличности не поклапају са оним што би дао алат за детекцију сличности, уочава се да резултати доста добро одсликавају релативни поредак, у смислу сличности, који је предвиђен приликом њиховог писања. Студенти исправно уочавају да је пар програма у оквиру Сегмента А најсличнији, док за Сегмент D сматрају да је најмање сличан или да уопште није сличан.



Слика А.10. Процене плагијаризма за конкретне програмске кодове

А.3.4. Факторска анализа

Како би се додатно расветлила природа мерених конструката и покушала сумаризација добијених резултата, извршена је факторска анализа над добијеним резултатима за сваку скалу. Факторска анализа представља технику мултиваријантне статистичке анализе која има за циљ идентификацију и разумевање латентне димензије, односно заједничких карактеристика више посматраних варијабли [154]. Једну варијаблу у овом контексту представљају одговори на једно питање у оквиру спроведеног истраживања. У контексту факторске анализе, једно питање из упитника се обично назива ставка (енг. *item*).

Ова анализа се примарно се користи за редукцију и сумаризацију података, како би се, уколико то подаци дозвољавају, већи број реалних варијабли на основу пронађене заједничке димензије превео у мањи број латентних димензија које се називају фактори и који стоје у основи података. Екстракција фактора се врши у складу са принципом максимизовања процента објашњене варијансе мерених варијабли (у овом случају ставки са три инструмента), при чему у коначном факторском решењу, различите варијабле различито корелирају са фактором. За интерпретацију фактора је важно које варијабле највише корелирају са екстрахованим фактором, односно којим варијаблама је сваки појединачни фактор највише засићен.

Кључна ствар у вези факторске анализе је да разичите варијабле могу у различитој мери бити повезане (корелисане) са датим фактором, односно давати му различита засићења. Засићења појединачних ставки у великој мери утичу на интерпретацију фактора у складу са одређеним теоријским конструктом или знањем из праксе и других истраживања. Приликом екстракције фактора из низа варијабли могуће је користити различите начине ротације: ортогоналне у којој су фактори неповезани (под правим углом) и ротиране (која дозвољава корелације између фактора). У овом истраживању, коришћене су три ортогоналне факторске анализе, понаособ за сваки инструмент, а потом су добијени фактори корелисани коришћењем Пирсоновог коефицијента корелације који је коришћен и за обичне корелације појединачних варијабли. Све три факторске анализе у оквиру овог истраживања су користиле ортогоналну (*Varimax*) ротацију која даје међусобно независне и некорелисане факторе, а фактори су одређени методом главних компоненти [153, 155].

Табела А.3. Факторска матрица структуре скале за процену ставова према плагијаризму

Плагијаризам у програмском коду је свако коришћење туђег програмског кода без навођења аутора.	,388
Плагирање представља нарушавање принципа академске честитости.	,598
Неки аспекти плагијаризма нису озбиљно нарушавање принципа академске честитости.	-,322
Плагирање туђег рада у науци је недопустиво.	,470
Плагирање туђег рада у образовању је недопустиво.	,602
Плагирање туђег рада у науци треба да води поништавању стечене дипломе.	,528
Плагирање туђег рада у образовању треба да води поништавању стечене дипломе.	,617
Студенти приликом уписа на факултет треба да буду упознати шта је плагијаризам и на које начине се кажњава.	,518
Високошколске институције треба да развијају своје механизме за детектовање плагијаризма.	,602
Преписивање на испиту је врста плагијаризма.	,486
Плагијаризам нарушава смисао научног рада.	,600
Плагијаризам нарушава кредибилитет образовног процеса.	,670
Упознат сам са правилницима о дисциплинској одговорности студената на мом факултету.	,133
Наставници користе алате за детекцију плагијаризма у програмском коду.	,016
Колики је минимални удео домаћег задатка у односу на коначну оцену на предмету да би се студент одлучио да почини плагијаризам?	-,107
Колика сличност програмског кода у процентима треба да произведе дисциплинску одговорност студената?	-,412
Особа која користи туђ програмски код без цитирања извора	,474
Особа која даје свој програмски код	,240
Наставници који се не развијају процедуре за спречавање плагијаризма	,359
Непостојање јасних дефиниција и правила шта је плагијаризам	,025
Студент треба да буде удаљен са факултета.	,544
Студент треба да буде привремено удаљен са факултета.	,604
Студент треба да буде пријављен дисциплинској комисији.	,623
Студенту треба да буду одузети сви поени са задатка на коме је почињен плагијаризам.	,461
Студенту треба да буду одузет део поена са задатка на коме је почињен плагијаризам.	,014
Студента треба упозорити на недолично понашање.	,202
Коришћење алата за детекцију плагијаризма утиче на моје понашање приликом израде и предаје домаћег задатка.	-,004

Факторска матрица структуре сакле за процену ставова према плагијаризму у програмском коду је дата у Табели А.3. Све ставке скале за процену ставова према плагијаризму групишу се на првом фактору који објашњава 20% варијансе. Највеће засићење фактором имају оне ставке који наглашавају аспекте попут моралности, честитости и важности поштовања ових начела током образовног процеса и научног рада, као и снажно кажњавање оних који нарушавају ове принципе.

Табела А.4. Факторска матрица структуре скале за процену учесталости пракси које се могу подвести под плагијаризам.

Ставка / Фактор	1	2
Слао сам свој домаћи задатак другим студентима.	,762	-,236
Користио сам и предавао комплетно туђ задатак као свој.	,427	,453
Користио сам или предавао делове туђег задатак као свој рад.	,644	,160
Дискутујем о дизајну програмског решења са другим колегама.	,517	-,398
Показујем свој програмски код приликом дискусије о решењу задатка са другим колегама.	,734	-,288
Шаљем свој програмски код другима приликом дискусије о решењу задатка са колегама.	,742	-,264
Радим на решавању домаћих задатака заједно са другим колегама.	,675	-,224
Користим делове туђег рада са интернета или из других часописа и приказујем као свој.	,506	-,048
Платио сам колеги да ми реши домаћи задатак.	,286	,229
Колега ми је платио да му урадим домаћи задатак.	,366	,015
Видео сам или чуо да је неко платио колеги да му реши домаћи задатак.	,641	-,221
Урадио сам задатак на испиту за другог студента.	,547	,267
Колега је урадио задатак за мене на испиту.	,462	,379
Видео сам или чуо да је неко свом колеги урадио задатак на испиту.	,672	-,132
Полагао сам испит уместо колеге.	,303	,713
Колега је излазио на испит уместо мене.	,155	,800
Видео сам или чуо да је неко полагао испит уместо колеге.	,354	,070
Полагао сам испит коришћењем “бубица”.	,186	,787
Видео сам или чуо да је колега полагао испит коришћењем “бубица”.	,474	-,002

Ставке са скале за процену учесталости пракси које се могу подвести под плагијаризам детектују два независна фактора. Први фактор објашњава 18% варијансе док други фактор објашњава 12% варијансе. Факторска матрица структуре скале за процену учесталости пракси које се могу подвести под плагијаризам се може видети у Табели А.4.

Може се приметити да детектовани фактори имају различита засићења на различитим ставкама. Први фактор се може окарактерисати, на основу ставки на којима је у највећој мери засићен, као „лаки плагијаризам“, односно као склоност да се плагијаризам схвати као помагање другом студенту у виду давања одређених одговора током самих испита или приликом израде домаћих задатака, заједнички рад и сл.

Други фактор се може назвати „тешким плагијаризмом“, јер њега у највећој мери засићују ставке који говоре о грубом кршењу процедура и непостојању било каквог улагања

труда у учење приликом изласка на испит. Он обухвата ставке које говоре о полагању испита уз помоћ „бубица“, коришћењу туђег идентитета приликом полагања испита за другог колегу, или случајевима када неко други то ради за студента. Тешки плагијаризам се не може објаснити ни оценама, ни буџетским статусом студента ($p>0,05$), а једину разлику прави дужина студирања и година студија. Не постоји статистичка значајност разлика између аритметичких средина (*t-test*). Тешком плагијаризму су склонији старији студенти, на вишем студијском нивоу који дуже студирају ($p=0,2;\ n<0,01$). Ипак, треба имати у виду да је проценат студента који подлежу тешком плагијаризму веома мали унутар узорка и да је то евентуални разлог зашто је тешко идентификовати факторе који на њега утичу.

Табела А.5. Факторска матрица структуре скале за процену шта представља плагијаризам у програмском коду

Ставка / Фактор	1	2	3
Копирање комплетног програмског кода представља плагијаризам.	,020	,348	-,098
Копирање блока (неколико линија) програмског кода представља плагијаризам.	,010	,549	-,131
Копирање једне линије програмског кода представља плагијаризам.	,006	,542	-,102
Копирање и измена дела или комплетног програмског кода представља плагијаризам.	-,096	,524	-,222
Копирање коментара у програмском коду представља плагијаризам.	,133	,314	-,294
Копирање структуре програмског решења представља плагијаризам.	,002	,635	-,071
Копирање корисничког интерфејса програма представља плагијаризам.	,011	,593	-,019
Коришћење туђих скупова података за тестирање програма представља плагијаризам.	,105	,491	-,077
Добровољно писање програмског кода другом студенту представља плагијаризам.	-,027	,423	-,364
Писање програмског кода уз новчану или другу надокнаду другом студенту представља плагијаризам.	,016	,344	-,359
Два или више студената раде групно на задатку који треба да се ради индивидуално и предају веома сличне програмске кодове.	-,054	,506	-,307
Поновно коришћење дела кода који је већ предат у оквиру претходног решеног домаћег задатка представља плагијаризам.	,067	,268	-,135
Коришћење програмског кода из књиге или других званичних материјала (слајдова, материјала са вежби) представља плагијаризам.	,143	,509	-,094
Коришћење програмског кода са Интернета или незваничних материјала (скрипти, задатака решених од стране студената) представља плагијаризам.	,015	,622	-,189
Замена имена идентификатора (променљивих, константи, типова, имена функција)	,686	-,074	,076
Замена редоследа блокова кода	,827	-,025	,006
Замена редоследа појединачних линија унутар блока	,833	,014	,044
Додавање или уклањање коментара	,822	,008	,068
Додавање или уклањање форматирања (идентација, бланко знаци, нови ред)	,836	,017	,042
Замена петље еквивалентном	,777	-,040	-,026
Замена позива функције телом функције	,763	,016	-,024
Замена позива функције алтернативном, сличном функцијом	,550	-,034	-,093
Измена исписа програма	,733	-,036	-,052
Сегмент А	-,068	,460	,496
Сегмент В	-,059	,498	,627
Сегмент С	,043	,450	,720
Сегмент D	,027	,365	,635

Факторском анализом треће скале за процену плагијаризма у програмском коду су добијена три независна фактора. Одговарајућа факторска матрица и засићења појединачних ставки у оквиру скале су дати у Табели А.5. Први фактор објашњава 18 процената варијансе, док преостала два објашњавају по 9 процената варијансе. Очигледно, први фактор се односи на модификације програмског кода које студенти чине да би прикрили плагијаризам. Други фактор се односи на процену одређених пракси, као што су коришћење неких аспеката туђег рада или различитих материјала при изради програмских задатака. Трећи фактор углавном описује праксе заједничког рада или рада за другу особу, а карактеришу га негативна засићења на тим ставкама, што упућује да су испитаници остељивији и мање толерантни на појаву плагијаризма и ове праксе оцењују изразито негативно. То показује и корелација фактора, дата у Табели А.6., где трећи фактор процена значајно корелише са ставовима на тему плагијаризма.

Табела А.6. Корелације фактора.

	Ставови	**Лаки плагијаризам**	**Тешки плагијаризам**	**Процене 1**	**Процене 2**	**Процене 3**
Ставови	1	-,017	-,091	-,054	,116*	,260**
Лаки плагијаризам	-,017	1	,000	-,009	,046	-,118*
Тешки плагијаризам	-,091	,000	1	-,035	,073	-,093
Процене 1	-,054	-,009	-,035	1	,000	,000
Процене 2	,116*	,046	,073	,000	1	,000
Процене 3	,260**	-,118*	-,093	,000	,000	1

Посматрањем корелација појединачних ставки из упитника које припадају различитим скалама, могу се приметити одређена поклапања. На пример, одговори студената чије праксе подразумевају коришћење туђег рада са Интернета или из часописа и књига корелишу са одговорима на питање да ли коришћење програмског кода са Интернета или незваничних материјала представља плагијаризам. Такође, пракса групног решавања програмских задатака негативно корелира са одговорима у вези дефиниције плагијаризма, али и са упознатошћу студената са начином његовог кажњавања и уопште коришћења алата за његову детекцију од стране наставника.

B. Изјава о ауторству

Име и презиме аутора: **Марко Мишић**

Број индекса: **2010/5027**

Изјављујем

да је докторска дисертација под насловом

Унапређења система за детекцију плагијаризма у изворном програмском коду

- резултат сопственог истраживачког рада;
- да дисертација у целини ни у деловима није била предложена за стицање друге дипломе према студијским програмима других високошколских установа;
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио/ла интелектуалну својину других лица.

Потпис аутора

У Београду, 02.02.2017. _____

C. Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора: **Марко Мишић**

Број индекса: **2010/5027**

Студијски програм: **Електротехника и рачунарство**

Наслов рада: **Унапређења система за детекцију плагијаризма у изворном програмском коду**

Ментори: **проф. др Јелица Протић и проф. др Мило Томашевић**

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла ради похрањена у **Дигиталном репозиторијуму Универзитета у Београду.**
Дозвољавам да се објаве моји лични подаци везани за добијање академског назива доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.
Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис аутора

У Београду, 02.02.2017. _____

D. Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

Унапређења система за детекцију плагијаризма у изворном програмском коду

која је моје ауторско дело.

Дисертацију са свим прилозима предао/ла сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигиталном репозиторијуму Универзитета у Београду и доступну у отвореном приступу могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство (CC BY)

2. Ауторство – некомерцијално (CC BY-NC)

3. Ауторство – некомерцијално – без прерада (CC BY-NC-ND)

4. Ауторство – некомерцијално – делити под истим условима (CC BY-NC-SA)

5. Ауторство – без прерада (CC BY-ND)

6. Ауторство – делити под истим условима (CC BY-SA)

(Молимо да заокружите само једну од шест понуђених лиценци.

Кратак опис лиценци је саставни део ове изјаве).

Потпис аутора

У Београду, 02.02.2017.

1. **Ауторство**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце, чак и у комерцијалне сврхе. Ово је најслободнија од свих лиценци.

2. **Ауторство – некомерцијално**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела.

3. **Ауторство – некомерцијално – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца не дозвољава комерцијалну употребу дела. У односу на све остале лиценце, овом лиценцом се ограничава највећи обим права коришћења дела.

4. **Ауторство – некомерцијално – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца не дозвољава комерцијалну употребу дела и прерада.

5. **Ауторство – без прерада**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, без промена, преобликовања или употребе дела у свом делу, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце. Ова лиценца дозвољава комерцијалну употребу дела.

6. **Ауторство – делити под истим условима**. Дозвољавате умножавање, дистрибуцију и јавно саопштавање дела, и прераде, ако се наведе име аутора на начин одређен од стране аутора или даваоца лиценце и ако се прерада дистрибуира под истом или сличном лиценцом. Ова лиценца дозвољава комерцијалну употребу дела и прерада. Слична је софтверским лиценцама, односно лиценцама отвореног кода.