УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Саша Д. Стојановић

ПРОЦЕНА СЛИЧНОСТИ ПРОЦЕДУРА У БИНАРНОМ КОДУ

докторска дисертација

Београд, 2015

UNIVERSITY OF BELGRADE

SCHOOL OF ELECTRICAL ENGINEERING

Saša D. Stojanović

ESTIMATION OF PROCEDURE SIMILARITY IN BINARY CODE

Doctoral Dissertation

Belgrade, 2015

Ментор

др Вељко Милутиновић, редовни професор
Електротехнички факултет Универзитета у Београду

Чланови комисије

др Вељко Милутиновић, редовни професор
Електротехнички факултет Универзитета у Београду

др Захарије Радивојевић, доцент
Електротехнички факултет Универзитета у Београду

др Ненад Митић, ванредни професор
Математички факултет Универзитета у Београду

др Драган Бојић, доцент
Електротехнички факултет Универзитета у Београду

др Милош Цветановић, доцент
Електротехнички факултет Универзитета у Београду

Датум одбране:

Изјава захвалности

Желео бих да изразим неизмерну захвалност свима који су ми помогли да продубим разумевање проблема и истражим решења приказана у овој дисертацији, свима који су ми помогли својим несебичним саветима и коментарима, као и свима који су ми пружили подршку и мотивацију да овај рад завршим.

Професору Вељку Милутиновићу, ментору, који ме је увео у свет истраживања и који ми је пренео своја драгоцена искуства како у истраживачком раду, тако и у свакодневном животу. Драгим пријатељима и колегама, Милошу Цветановићу и Захарију Радивојевићу, коменторима, са којима сам кренуо у истраживање приказано у овој дисертацији, са којима сам делио како лепе и узбудљиве, тако и оне мање лепе и мање занимљиве тренутке током рада, и чије су искуство и савети били од непроцењиве вредности. Захваљујем се и свим члановима комисије на корисним и конструктивним коментарима који су помогли да овај рад буде још бољи.

Захваљујем се и свим члановима катедре за рачунарску технику и информатику који су имали разумевања током рада на дисертацији онда када нисам постизао све своје обавезе и који су несебично уступили своје ресурсе за потребе истраживања. Захваљујем се Драгану Миладиновићу и Милошу Тошићу, администраторима у лабораторији катедре за рачунарску технику и информатику, који су помогли да се за потребе израчунавања у оквиру истраживања искористе сви расположиви ресурси РТИ лабораторија.

Ову дисертацију посвећујем родитељима, Ружи и Бобану, сестри Драгани и двема ујаковим принцезама, Владани и Маји, који су моја највећа подршка и неисцрпни извор мотивације.

Сви поменути доприноси су овој дисертацији дали један нови, посебан смисао и помогли да стекнем једно ново искуство које је за мене од непроцењивог значаја. Захваљујем се свима.

Наслов

Процена сличности процедура у бинарном коду.

Резиме

У овом раду предложен је приступ за убрзавање откривања употребе софтверске библиотеке која је доступна у облику изворног кода и која је бесправно уграђена у бинарни код. Претпоставка је да су из бинарног кода одстрањене све додатне информације које би могле помоћи приликом откривања употребе софтверске библиотеке, што је чест случај у наменским уређајима. Стога је у раду коришћена ARM архитектура која је једна од најзаступљенијих у наменским уређајима.

Приступ је заснован на софтверским метрикама и састоји се од две фазе. У првој фази рачуна се сличност између бинарног облика тражене процедуре (процедура из библиотеке) и сваке од процедура бинарног кода за који се сумња да користи библиотеку. У другој фази се рангирају процедуре из посматраног бинарног кода у складу са израчунатом сличношћу према траженој процедури. На крају се издваја првих N процедура и даље анализира ручно, при чему експерт бира вредност N у складу са расположивим временом и важношћу откривања неовлашћене употребе библиотеке.

Одређивање сличности између две бинарне процедуре почиње прикупљањем вредности софтверских метрика. У раду је предложено укупно 19 различитих софтверских метрика. Вредности софтверских метрика се

пореде једним од три предложена компаратора, чиме се добијају парцијалне мере сличности упоређених процедура. Парцијалне мере сличности се потом трансформишу како би се покушале добити вредности које верније представљају сличност. Трансформисане парцијалне мере сличности се даље комбинују помоћу једне од 7 предложених формула како би се израчунала једна вредност која представља сличност упоређених процедура.

Поред описаног приступа предложено је и пет техника које треба да побољшају резултате коригујући примећене аномалије. Део техника одстрањује инструкције и процедуре које при поређењу уносе шум и доводе до погрешних закључака. Други део техника прикупља додатне информације које су изостављене једноставним метрикама које су предложене у основном приступу.

Евалуација је показала да је применом предложеног приступа и техника могуће постићи одзив од 53% у случају тестова базираних на STAMP окружењу и 56% у случају тестова базираних на BusyBox пакету. Наведени резултати су за случај када се као успех посматра проналазак тражене процедуре на првој позицији. Одзив који постиже предложени приступ је скоро дупло већи одзив од другог најбољег тестираног алата, који постиже свега 27% уколико се као успех посматра проналазак процедуре на првој позицији. У погледу F и F_2 мера, предложени приступ заједно са техникама постиже знатно веће вредности од свих других тестираних алата.

Кључне речи

Сличност кода, плагијаризам, софтверски клонови, нарушавање лиценцних права, рангирање процедура, софтверске метрике, поређење бинарног кода

Научна област

Електротехника и рачунарство

Ужа научна област

Рачунарска техника и информатика

УДК

621.3:004.8

Title

Estimation of procedure similarity in binary code

Abstract

This work proposes an approach to accelerate the discovery of a case when a software library, available in the source code, is used in a binary code without an appropriate permission. It is assumed that the binary code does not contain any additional information that could help in detecting the use of the software library, which is often the case in embedded devices. This study uses ARM architecture which is one of the most commonly found architectures in embedded devices.

Approach is based on software metrics and consists of two phases. The first phase calculates similarity between the searched binary procedure originating from the library, and each of the procedures found in the binary code that is suspected to use the library. In the second phase of the proposed approach, the procedures from the binary code are ranked in accordance with the similarity to the searched procedure. After sorting, expert conducts manual analysis of the top N procedures, where N is selected in accordance with the available time of expert and the importance of detecting that the library is used.

In order to determine similarity between the two binary procedures, values of the proposed software metrics are collected. The paper proposes a total of 19 different software metrics. The values of software metrics are compared with one out of the three proposed comparators, thus obtaining partial similarity measures

between compared procedures. Partial similarity measures are then transformed in such a way that more closely represent the similarity. Furthermore, they are combined using one of the 7 proposed formulas in order to calculate a single value representing the similarity between procedures.

In addition to the described approach, a set of 5 techniques is proposed with the aim to improve results by correcting the observed anomalies. One set of these techniques removes instructions and procedures that introduce noise and lead to misleading conclusions. The other set of techniques collects additional information that is omitted by using simple metrics proposed in the basic approach.

The evaluation shows that using the proposed approach and techniques, a recall of 53% in the case of tests from STAMP environment and 56% in the case of tests of BusyBox package can be achieved if the goal is to find the searched procedures in the first position. The proposed approach achieves almost double recall when compared to the second-best tested tool, which achieves a recall of only 27% observing the first position only. With respect to the F and F_2 measures, the proposed approach, along with techniques, greatly outperforms all other tested tools.

Keywords

Code Similarity, Plagiarism, Software Clone, License Violation, Procedure Ranking, Software metrics, Binary Code Comparison

Scientific field

Electrical and Computer Engineering

Scientific subfield

Computer Engineering and Information Theory

UDC

621.3:004.8

Садржај

1. Увод

Енорман развој софтверске индустрије и борба за опстанак на тржишту довели су до појаве великог броја различитих пословних модела. Читав низ пословних модела заснива се на доступности изворног кода. Доступност изворног кода не значи нужно и да је употреба тог кода потпуно бесплатна, већ се много чешће код нуди под условом да се прихвати одговарајућа лиценца која дефинише права и обавезе приликом употребе тог кода. Један облик лиценцирања софтвера је и двоструко лиценцирање, где се за потребе развоја изворни код нуди бесплатно, док се за потребе продукције очекује одговарајућа накнада за коришћење кода [1]. Овакав тип лиценцирања је посебно погодан за софтверске библиотеке јер доступност изворног кода библиотеке олакшава употребу библиотеке на различитим архитектурама и омогућава анализирање могућности библиотеке и пре куповине исте.

Притисак који тржиште врши на софтверске компаније неретко доводи до ситуације у којој запослени, свесно или несвесно, употребе изворни код софтверске библиотеке на начин који није дозвољен лиценцом под којом је код издат. Штете које на тај начин настају су веома велике [2], па је откривање употребе софтверске библиотеке од изузетно великог значаја.

Откривање употребе је закомпликовано чињеницом да је финални производ који користи софтверску библиотеку често софтвер у бинарном облику који је добијен употребом преводиоца који није познат, као и опција преводиоца које нису познате особи која ради проверу употребе софтверске библиотеке. Додатна комплексност долази због непознавања кода који користи библиотеку, па самим тим нису познате ни адресе на којима се налазе делови библиотеке и које се у оквиру кода користе. Веома често су из бинарног кода одстрањене и све информације, попут назива потпрограма, које би могле помоћи при упаривању делова бинарног кода са изворним кодом софтверске библиотеке, што је практично увек случај код уграђених уређаја са фабричким софтвером. У таквим околностима откривање захтева експертско знање и представља комплексан и дуготрајан посао јер је за сваку процедуру из изворног кода библиотеке потребно анализирати једну по једну процедуру из посматраног бинарног кода како би се пронашле употребљене процедуре из библиотеке.

Убрзање процеса откривања употебе софтверске библиотеке у неком бинарном коду постиже се употребом алата који имплементира неки од приступа за поређење кода. Алатима се најчешће проналазе процедуре које потичу из библиотеке и које су уграђене у посматрани бинарни код за који се сумња да користи библиотеку. Да би се алати могли применити, најчешће је потребно обезбедити да и библиотека и код који потенцијално користи библиотеку буду у истом облику. Процес враћања бинарног кода, који потенцијално користи библиотеку, у изворни код је веома комплексан посао и захтева софистициране алате [3]. Такви алати су ретки и њихов резултат

није обавезно исти полазни изворни код чијим је превођењем добијен посматрани бинарни код, већ семантички еквивалентан облик полазног изворног кода. Друга могућност је превођење изворног кода библиотеке у бинарни облик. Међутим, ни у овом случају поређење није тривијално јер се пореде два бинарна кода која су настала у два, потенцијално потпуно различита процеса превођења, што може да резултује изменама које и за експерта представљају озбиљан изазов. Разлог лежи у чињеницама да различити алати различито преводе исту језичку конструкцију, да користе различите алгоритме за алокацију регистара, да за исту намену користе различите инструкције, као и да користе различите алгоритме за оптимизацију генерисаног кода.

Од алата се очекује да резултат приликом потраге за процедуром из библиотеке унутар бинарног кода посматраног извршног програма буде проналазак тачно једне процедуре за коју је установљено да потиче од изворног кода тражене процедуре. При томе је претпостављено да само једна процедура у посматраном бинарном коду заиста потиче од изворног кода тражене процедуре. За све остале процедуре се очекује да буде установљено да не потичу од изворног кода тражене процедуре. Да би се очекивани резултат добио неопходно је да процес поређења не буде осетљив на разлике које настају у процесу превођења, а да при томе примети све разлике међу процедурама које се пореде. Два наведена захтева су контрадикторна. Смањењем осетљивости на разлике повећава се вероватноћа да се пронађе процедура која потиче од изворног кода тажене процедуре, али исто тако се повећава и вероватноћа погрешног закључка за процедуре које не потичу од

изворног кода тражене процедуре. Са друге стране, повећањем осетљивости на разлике међу процедурама које треба упоредити умањује се и могућност проналажења одговарајуће процедуре у случајевима када процеса превођења направи велику разлику између процедура.

У овом раду предложен је приступ за процену сличности процедура у бинарном коду предвиђен за употребу у ситуацијама када је потребно наћи сличности између бинарних кодова који се претежно разликују, као што је случај када се пореде два кода преведена са два различита алата, са потенцијално различитим коришћеним опцијама [4]. Због великих разлика које процеси превођења уносе у процедуре у бинарним кодовима, циљ предложеног приступа је убрзање потраге за процедурама из библиотеке. Убрзање се постиже смањењем броја процедура које експерт треба да пореди тако што се за сваку од процедура из библиотеке одабере неколико најсличнијих процедура из бинарног кода, а затим се одабране процедуре анализирају од најсличније до најмање сличне.

Процена сличности у предложеном приступу се базира на употреби софтверских метрика које су конструисане за примену над асемблерском репрезентацијом бинарног кода. Метрике су конструисане са циљем да се занемаре детаљи који се мењају у зависности од процеса превођења, а да се задрже они који су претежно инваријантни. Сличност упоређених процедура се процењује на основу сваке од коришћених метрика чиме се добијају парцијалне мере сличности. Затим се добијене парцијалне мере сличности

комбинују на један од предложених начина у јединствену меру сличности упоређених процедура.

Процењена мера сличности се не користи као апсолутна мера сличности између процедура, већ се користи за рангирање процедура из бинарног кода програма за који се сумња да користи библиотеку. Процедуре се рангирају у складу са сличношћу према бинарном облику тражене процедуре за којом се трага у циљу откривања употребе библиотеке. Како би се побољшали резултати које приступ постиже, предложено је и 5 техника које се примењују у различитим фазама приказаног приступа [5]. Једна група предложених техника усмерена је на елиминисање одређених варијација које преводиоци уносе у току превођења кода, док друга група има за циљ да ублажи негативне последице поређења процедура на високом нивоу апстракције.

Поред увода који је дат у овој глави, преостали део рада подељен је на следећи начин. У другој глави дат је преглед области у којима се појављује поређење кода и укратко су описани најинтересантнији алати и приступи у тим областима. У трећој глави дефинисан је проблем који се посматра у овом раду заједно са претпоставкама и ограничењима. На крају треће главе дата су и истраживачка питања на која дисертација треба да одговори. У четвртој глави описано је окружење за евалуацију које се састоји од описа тестова који се користе и мера којима ће се оцењивати успех предложеног приступа и осталих коришћених алата и приступа. У петој глави приказана је евалуација одабраних постојећих алата и приступа. У шестој глави описан је предложени

приступ, а затим и евалуиран. Након евалуације предложеног приступа предложене су и додатне технике које су такође евалуиране. На крају шесте главе дати су одговори на истраживачка питања. Седма глава садржи закључак, док су референце приказане у осмој глави.

2. Преглед литературе

У овој глави ће најпре бити дат преглед литературе у областима у којима се користи поређење кода. Потом ће свака област бити описана и дат приказ карактеристичних приступа и алата. У оквиру описа алата и приступа биће дати и описи приступа који ће бити имплементирани за потребе поређења са предложеним приступом. Приступи се морају имплементирати јер одговарајући алат или не постоји или није погодан за директну примену на посматрани проблем.

2.1 Поређење кода и примена

Технике поређења софтверског кода постоје скоро од развоја првих софтверских апликација и предмет су изучавања многих истраживача широм света како тада, тако и дан данас [6, 7, 8, 9, 10]. Потреба за поређењем кода се појављује у различитим областима у којима постоје различити циљеви па се и сами алгоритми разликују. Тако циљ поређења може да буде откривање истих или сличних делова кода, али исто тако и откривање разлика у упоређеним кодовима. Такође, од алата може да се очекује да за два дела кода процењује да ли се ради о истим кодовима или не, док неки алати само процењују ниво сличности упоређених кодова без процењивања да ли је

нађена мера сличности довољна да би се упоређене процедуре прогласиле истим или не.

Постојећи приступи се значајно разликују по питању задатка који се пред њих поставља, што најчешће зависи од области примене. У неким областима кодови који се пореде су претежно исти, па се исти и слични делови упарују како би се открили разлике између њих. Са друге стране, у неким областима се очекује да се кодови који се пореде значајније разликују, али да и даље имају неку врсту сличности, па се од алата очекује да пронађе делове кода који су по неким критеријумима слични.

Друга битна разлика из угла овог рада јесте врста резултата који алат даје. У прву групу алата спадају они код којих је резултат одлука да ли две поређене процедуре потичу од истог изворног кода или не, у зависности од осетљивости, дешава се да се за већи број процедура из извршног програма констатује да потичу од изворног кода тражене процедуре. Такви приступи помажу тако што одређени број процедура елиминишу из разматрања. Међутим, експерту који спроводи потрагу се даје већи број процедура за које се тврди да потичу од изворног кода тражене процедуре и при томе му се не сугерише редослед по којем треба да провери да ли нека од сугерисаних процедура заиста потиче од изворног кода тражене процедуре или не. Такође, не постоји гаранција да процедура која заиста потиче од изворног кода тражене процедуре неће бити одбачена.

Друга група приступа процењује сличност процедура из извршног програма са траженом процедуром. Мера сличности може да се користи како

би се проценило да ли процедуре потичу од истог изворног кода или не, али исто тако пружа могућност уређивања скупа процедура из програма од најсличније до најмање сличне са траженом процедуром. Две основне предности у односу на прву групу приступа су могућност контроле осетљивости на измене и скраћивање времена које експерт утроши. Уређивање скупа процедура из програма по опадајућој вредности мере сличности са траженом процедуром може да вишеструко смањи време претраге. Поред смањења времена претраге, експерту који врши потрагу се омогућава да контролише осетљивост на измене у коду одабиром величине скупа најсличнијих процедура које ће да провери. Рангирање претраживаних објеката је поступак добро познат у системима за дохватање информација (eng. *Information Retrieval Systems*) [11]. Основна разлика рангирања у овом раду у односу на системе за дохватање информација јесте што међу претраживаним процедурама постоји тачно једна процедура која одговара траженој процедури, док у системима за дохватање информација може да се нађе и већи број докумената који одговарају терминима који су коришћени у претрази.

Међу првим проблемима који су захтевали поређење кода је појава сличних или идентичних делова кода у оквиру изворног кода неког софтверског производа. Разлога за појаву поновљених делова има више, али су најчешћи мултиплицирање делова кода у циљу привидног убрзања развоја софтвера, као и коришћење шаблона за решавање стандардних проблема. Понављањем делова кода пропагирају се и грешке које су евентуално начињене и отежава се каснија измена кода, што може да

значајно утиче на повећање цене одржавања софтвера. Стога је за потребе одржавања софтвера веома важно пронаћи идентичне или веома сличне делове кода и по могућности модификовати код тако да се избегну вишеструка појављивања делова. Овакви делови кода су названи софтверски клонови и зачета је читава област која се бави приступима и алатима за откривање софтверских клонова за потребе одржавања софтвера. Област је изузетно популарна и порастом комплексности и величине софтверских кодова, област све више добија на значају [12].

Други сличан проблем је појава плагијаризма. Предмет плагијаризма може да буде текст на неком говорном језику, али исто тако и неки софтверски код. Као и у случају одржавања софтвера, циљ је наћи исте или сличне делове кода у облику софтверских клонова. Разлика у односу на одржавање софтвера је у томе што се при откривању плагијаризма пореде две различите верзије програмског кода, оригинална верзија и верзија настала као плагијат оригиналне верзије. Стога су и разлози за настајање разлика другачије природе. Плагијат кода настаје са циљем да обавља исту функционалност, али да се од оригиналног кода што више разликује како би се отежало препознавање сличности са оригиналном верзијом кода. Оваква врста плагијаризма се веома често јавља у академској средини међу студентским радовима, па је зато и честа тема истраживања. Приступи и алати који се у области откривања плагијаризма користе су веома слични као у области откривања софтверских клонова и у наставку ће бити разматрани заједно.

Поређење кода такође лежи у основи приступа из области откривања нарушених лиценцних права у софтверској индустрији, којој припада и овај рад. Проблеми нарушавања лиценцних права могу да настану на различите начине, почев од употребе изворног кода јавно доступног за некомерцијалну употребу до копирања бинарног кода прочитаног из неког уређаја. За разлику од откривања софтверских клонова и плагијаризма, када је био доступан изворни код, у већини случајева при откривању нарушавања лиценцних права на располагању је само бинарни облик кода који нарушава лиценцна права, па стога није могућа директна примена алата намењених за откривање софтверских клонова у изворном коду. Веома брз развој софтверске индустрије доводи до све чешћих нарушавања лиценцних права што софтверској индустрији наноси све већу штету, па је за очекивати даљи развој ове области.

За разлику од претходне три области, у којима је поређење кода централни део свих приступа, постоје и друге области у којима се појављује поређење кода на бинарном нивоу као што су: анализа рањивости софтвера, откривање злонамерног кода и компакција кода. У области анализе рањивости софтвера поређење кода се користи како би се уочиле разлике између две верзије истог софтвера када то није могуће урадити на други начин. Даље се анализирају само промене које су настале од претходе верзије. Откривање злонамерног кода такође користи поређење кода како би се открило да ли неки код личи на неки познати злонамерни код. Поређење кода се појављује и у области компакције кода, када је потребно смањити величину кода. Потреба за компакцијом кода се најчешће појављује код

уграђених уређаја где су ресурси веома често ограничени. Табела 2.1 приказује је преглед четири области и примере приступа који су применљиви на проблем посматран у овом раду.

Табела 2.1 Преглед области и примери алата који пореде код на бинарном нивоу (CD - откривање софтверских клонова, LV - откривање нарушавања лиценцних права, MD - откривање злонамерног кода и VA - анализа рањивости софтвера). Три осенчена поља у колонама MD и VA представљају карактеристике алата које су делимично мотивисале увођење две од пет техника за побољшање резултата предложеног приступа.

Акроним	**CD**	**LV**	**MD**	**VA**
Алат	ACD [13]	Zip (BAT) [14]	Opcode [15]	Dullien [16]
Област	Откривање клонова	Нарушавање лиценцних права	Откривање злонамерног кода	Анализа рањивости
Користи токене	да (упарује токене)	да (упарује токене)	да (броји токене)	не
Користи метрике	не	не	да	да
Користи графове	не	не	не	да
Ниво анализе унутар процедуре	Блокови променљиве величине	Блокови променљиве величине	Блокови фиксне величине	Базични блокови
Ниво анализе између процедура	не	да (цео програм)	не	да (граф позива)

Акроним	CD	LV1	MD	VA
Превођење	Исти преводилац	Различити преводиоци	Различити преводиоци	Исти преводилац
Извор разлика	Програмер	Преводилац	Програмер + преводилац	Програмер
Ниво различитости	Претежно исти	Претежно различити	Претежно различити	Претежно исти
Локалност разлика	Претежно локализовано	Разбацано	Локализовано	Локализовано
Мотивација	Потрага за сличном процедуром	Потрага за еквивалентном процедуром	Потрага за кодом који личи на злонамеран	Потрага за закрпама
Циљ	Наћи сличности	Наћи сличности	Наћи сличности	Наћи разлике

2.2 Софтверски клонови и плагијаризам

Слични делови кода се у софтверу појављују из више разлога. Два истакнута разлога у случају одржавања софтвера су коришћење образаца при решавању сличних проблема и копирање кода уз евентуалне модификације копираних делова [12]. Иако постоје снажни разлози против копирања делова кода због компликовања и поскупљивања процеса одржавања софтвера, постоје и одређене предности, попут бржег писања кода, бржег развоја нових функционалности, избегавање позива и повратка из процедура и заобилажења недостатака коришћеног језика. Све поменуте предности доводе до повећања заступљености копирања у току развоја

софтвера. Копирани делови се у неким случајевима задрже у оригиналном облику, док се у другим појављују у незнатно измењеном облику ради обезбеђивања нове функционалности. Накнадна измена неког од делова кода који су копирани услед отклањања уочених недостатака најчешће захтева исту или сличну измену у свим начињеним копијама, било да су копије задржане у оригиналном облику или да су измењене. Обзиром да прављење копија кода по правилу није документовано, неопходни су софистицирани алати који ће помоћи да се пронађу слични делови кода како би се ублажили или у потпуности отклонили негативни ефекти направљених копија.

У случају плагијаризма, копирани делови кода настају услед неетичког понашања ученика и студената. По правилу, плагијатор покушава да измени код како би прикрио трагове копирања. Најједноставније измене настају изменама коментара и другачијим форматирањем кода. Нешто сложеније измене настају као резултат систематичног преименовања променљивих и процедура. Озбиљније измене настају као последица промене редоследа инструкција, као и додавања нових или брисања непотребних делова кода. Најозбиљније измене настају када се делови кода замене функционално еквивалентним деловима кода, али се у таквим ситуацијама доводи у питање у којој мери нови код представља плагијат оригиналног кода.

Копије кода, које се појављују у изворном коду, назване су софтверским клоновима и у складу са обликом у којем је копија задржана, класификоване су у један од 4 типа софтверских клонова [7]. Копија изворног

кода која је идентична оригиналном коду, уз евентуалну измену белих знакова и коментара, представља софтверски клон типа 1. Синтаксно идентична копија кода настала евентуалним преименовањима назива променљивих, типова, поља структура и класа, као и процедура представља софтверски клон типа 2. Даљим изменама копије кода у смислу измене постојећих наредби, додавања нових и брисања старих, добијају се софтверски клонови типа 3. Уколико су ипак начињене измене у већем обиму, тако да се не може препознати синтаксна сличност два упоређена кода, али се може показати семантичка идентичност упоређених кодова, тада упоређени кодови спадају у групу софтверских клонова типа 4.

У пракси се појављују сва четири типа софтверских клонова, али се као резултат копирања и евентуалног модификовања најчешће појављују прва три типа софтверских клонова. Тип 4 најчешће није резултат копирања већ резултат решавања истог проблема на више различитих начина и знатно је комплекснији за препознавање. Стога су за област откривања софтверских клонова за потребе одржавања софтвера од посебног значаја прва три типа софтверских клонова.

Већина постојећих алата се ограничава на препознавање неког од прва три типа софтверских клонова, што у великој мери зависи од нивоа на којем алат пореди код [8] . Као што је приказано у Табела 2.2, према начину на који раде поређење кода, алати се могу у грубо поделити у зависности од тога да ли су базирани на поређењу текста, поређењу лексичких елемената или токена, поређењу синтаксних стабала, поређењу графова који се на основу

зависности у програму могу конструисати или поређењу софтверских метрика. Сви алати могу да препознају софтверске клонове типа 1. Алати који раде поређење на нивоу текста најчешће могу да препознају комплетан скуп или само подскуп клонова типа 3, док ретки међу њима имају могућност да препознају одређени подскуп софтверских клонова типа 2 и 4. Алати који поређење раде на нивоу токена по правилу могу да открију и софтверске клонове типа 2, док неки имају могућност откривања и одређеног подскупа клонова типа 3. Софтверске клонове прва три типа или неки њихов подскуп могу да открију скоро сви алати базирани на поређењу синтаксних стабала, док алати базирани на поређењу софтверских метрика и графова зависности у програму могу да открију и неке од софтверских клонова типа 4.

Алгоритми коришћени у постојећим алатима и приступима за откривање софтверских клонова најчешће раде у неколико фаза [10]. На почетку се ради претпроцесирање изворног кода. Претпроцесирани изворни код се трансформише кроз процесе екстракције и нормализације у облик који је погоднији за поређење. Резултат трансформације се доводи на улаз алгоритма за поређење који процењује сличност упоређених делова кода и упарује их у складу са процењеном сличношћу. Резултати поређења се корелишу са изворним кодом који је био на улазу процеса поређења након чега се ради филтрирање и приказ резултата поређења.

Претпроцесирањем улазног кода се из изворног кода одстрањују делови који не утичу на сличност и не мењају значење кода, као и делови који би могли да произведу велики број лажних софтверских клонова.

Следећи корак претпроцесирања је раздвајање изворног кода на делове који немају преклапања и који представљају највеће делове који као такви могу бити упоређени са осталим деловима у циљу откривања софтверских клонова. Добијени делови кода се даље деле на потенцијално мање јединице кода над којима се ради упаривање, што зависи од технике коришћене за поређење кода.

Табела 2.2 Преглед области откривања софтверских клонова и плагијаризма у зависности од начина рада.

	Текст	Токени	Метрике	AST	PDG
Пример алата	Duploc [17], SimCad [18]	Dup [19], CCFinder [20]	CLAN [21], Davey [22]	cpdetector [23], Deckard [24]	GPlag [25], Duplix [26]
Ниво језика	сви	претежно виши	претежно виши	виши	виши
Зависност од језика	нема	висока	висока	висока	висока
Проширивост	Ништа или лексер	Лексер	Лексер или парсер	Парсер	Парсер
Типови клонова које открива	Тип 1,3	Тип 1,2	Тип 1,2,3,4	Тип 1,2,3	Тип 1,2,3,4
Комплексност рачунања	Средња	Мала	Средња	Средња	Велика

Осим код алата који раде на текстуалном нивоу, резултати претпроцесирања се даље трансформишу у одговарајућу интерну репрезентацију која ће бити коришћена за потребе поређења. Процес трансформације се састоји од екстракције и нормализације. Сама екстракција представља процес добијања интерне репрезентације од изворног кода, и у зависности од приступа може да се базира на издвајању токена, парсирању изворног кода и анализи графова контроле тока и графова зависности по подацима. У зависности од приступа, у оквиру трансформације изворног кода у интерну репрезентацију користе се различите технике за нормализацију, које имају за циљ да сваки део кода доведу у одговарајући предефинисани облик кода који је семантички еквивалентан полазном. Стандардне технике нормализације обухватају уклањање сувишних белих знакова, уклањање коментара, нормализација идентификатора и преправљање кода у циљу довођења кодова које треба упоредити на исти предефинисани облик (нпр. у наредби if морају да се користе витичасте заграде). Већина техника за нормализацију помаже откривању већег скупа софтверских клонова типа 1 који су настали као резултат различитог форматирања изворног кода, док технике попут нормализације идентификатора могу да помогну у откривању одређених софтверских клонова типа 2.

Интерна репрезентација, добијена у процесу трансформације, користи се да се неким од приступа за поређење кода препознају софтверски клонови, након чега се препознати клонови филтрирају у циљу смањења броја погрешно препознатих клонова. У зависности од конкретног приступа,

филтрирање се ради мануелно или аутоматизовано. Аутоматизовано филтрирање се базира на употреби неке од хеуристика и појављује се у две варијанте. У првој варијанти се ради сортирање препознатих клонова од оних који највероватније јесу клонови до оних са најмањом вероватноћом, док се у другој варијанти доноси финална одлука да ли два упоређена дела кода заиста представљају софтверски клон или не.

2.2.1 Алати базирани на поређењу текста

Копиране делове кода који остану у неизмењеном облику могуће је пронаћи директним поређењем изворног кода, при чему се изворни код посматра као обичан текст који садржи идентичне делове. Предност оваквог приступа је што алати за поређење не зависе од језика на којем је код написан, што развој алата чини једноставнијим и омогућава примену једног алата на већи број различитих језика. Ипак, основни недостатак оваквих приступа јесте што су веома осетљиви чак и на једноставне измене изворног кода.

Одређивање сличности делова изворног кода на нивоу текста подразумева проналажење идентичних секвенци знакова у упоређеним деловима кода. Упаривање исувише кратких секвенци знакова може да произведе велики број софтверских клонова услед упаривања кључних речи и других језичких елемената који се често појављују у програмима. Стога већина приступа дефинише минималну дужину секвенци које могу да представљају софтверске клонове. Тако алат NiCAD[27] сугерише да софтверски клон не може да буде краћи од 6 линија изворног кода.

Најједноставнији, али и временски најзахтевнији приступ је поређење свих могућих секвенци. У случају поређења већих делова изворног кода број секвенци може бити изузетно велики, што уз квадратну сложеност може да захтева неприхватљиво пуно времена. Због тога већина приступа прибегава техникама које смањују број секвенци које учествују у поређењу тако што елиминишу секвенце за које је веома мало вероватно да могу представљати софтверски клон и секвенце које се у деловима могу наћи у другим секвенцама које се могу упарити. Један начин елиминације секвенци је бирање само оних секвенци чији почетак и крај задовољавају одређене унапред дефинисане критеријуме. Обзиром да алатима у овој групи није познат језик на којем код написан, за границе се бирају знакови који задовољавају критеријуме који не зависе од језика. Примери таквих критеријума су знак за прелазак у нови ред, почетно слово речи или оператори. Неки хибридни приступи ипак укључују и спецификацију језика тако што дозвољавају да упаривање може почети само од неких карактеристичних елемената језика. Један такав приступ је од Manber [28], где се предлаже да упаривање може почети само од кључних речи језика који се посматра. Иако поменути приступ користи информације о језику, поређење ипак ради на нивоу текста. Још једна техника за смањење броја секвенци које је потребно упоредити је приказана у једном од првих приступа који ради поређење на нивоу текста [29, 30], који предлаже да се над свим могућим секвенцама одабране фиксне дужине израчунају хеш вредности и да се касније пореде само секвенце које имају исте хеш вредности, чиме се смањује број потребних поређења. У таквом случају

поређење се ради на нивоу секвенци фиксне дужине, док софтверски клон може да почне као и да се заврши на било ком знаку у коду.

Једна од негативних страна приступа базираних на поређењу текста је ограничена применљивост без претходне обраде услед превелике осетљивости чак и на једноставне измене, попут оних које се сусрећу у клоновима типа 1. Примери таквих измена су различито форматирање копираног кода, као и додавање, измена или брисање коментара. Против наведених измена алати из ове групе се боре скупом техника за нормализацију кода које примењују у току трансформације кода. Две технике које се често користе и у другим приступима су усмерене ка нормализацији коментара и нормализацији белих знакова. Нормализација коментара најчешће подразумева потпуно одстрањивање коментара. Ипак, неки приступи предлажу да се и коментари пореде, јер се због скраћивања времена програмирања коментари често задрже у непромењеном облику иако се копирани део кода измени у циљу обезбеђивања нових функционалности. Нормализација белих знакова подразумева свођење свих белих знакова на појављивања белих знакова по унапред дефинисаним правилима. У зависности од приступа који се користи за поређење, бели знакови могу да се у потпуности одстране или да се сва вишеструка узастопна појављивања белих знакова сведу на једнострука појављивања. У алгоритмима у којима прелом кода по линијама није коришћен у процесу поређења, знакови који означавају крај реда се могу обрисати у потпуности.

Други облици борбе против софтверских клонова који нису типа 1 често захтевају од алата познавање језика и представљају хибридне приступе. Један такав алат је NICAD који представља компромис између комплекснијих приступа отпорних на веће измене у кодовима које је потребно упоредити и једноставнијих приступа поређења кода у текстуалном облику. Поменути алат у процесу трансформације користи 3 технике: форматирање кода по одређеном кодном стандарду, ограничену нормализацију идентификатора и одбацивање делова кода који не утичу директно на функционалност и самим тим нису од интереса за поређење. Код се форматира тако што се свака наредба запише у више линија по унапред дефинисаним правилима, да би се упаривањем читавих линија у форматираном коду заобишле разлике које су се појављивале у неким деловима наредбе и које би чиниле да су оригинални редови са посматраном наредбом различити иако је већина наредбе идентична. Нормализација идентификатора подразумева замену свих идентификатора једним универзалним идентификатором. Оваква измена може да доведе до промене функционалности посматраних кодова и као таква, може да доведе до појаве погрешно упарених делова кода и лажних софтверских клонова. Стога приступ предлаже ограничену нормализацију идентификатора, тако да се само одређена појављивања идентификатора замене универзалним идентификатором. Тиме је начињен компромис који смањује могућност појаве лажних софтверских клонова, али исто тако смањује отпорност на измене. Овакав приступ је оправдан чињеницом да се у формирању софтверских клонова у процесу развоја софтвера најчешће не раде

систематична преименовања променљивих и функција, што се често дешава у случају плагијаризма. Трећа предложена техника је одбацивање делова кода попут декларација променљивих, што не утиче директно на функционалност посматраног кода.

Пронађене софтверске клонове је потребно корелисати са изворним кодом како би се нађени софтверски клонови приказали кориснику. Већина алата софтверске клонове приказује у облику парова секвенци кода или евентуално група секвенци кода које представљају софтверске клонове. Групе се појављују уколико је неки део кода копиран више пута. Неки од алата за приказ користе и графичку репрезентацију тако што клонове приказују визуелно као скуп тачака на 2Д графику [31, 32]. На свакој од оса су тачке које представљају минималне делове кода који се директно пореде. Делови кода су узети тако да се не преклапају и поређани тако да се њиховим спајањем по редоследу појављивања на оси добије секвенца кода у којој се траже клонови. Свака од оса представља једну од две секвенце кода које учествују у поређењу и у којима се траже софтверски клонови. Уколико се при поређењу минималних делова кода који представљају две тачке на различитим осама утврди да су исти, на графику се у пресеку црта тачка. У супротном, у пресеку нема тачке. На описани начин, софтверски клонови се могу препознати као дијагоналне линије на слици, што омогућава аутоматизовано препознавање. Уколико се услов поређења мало релаксира или се дозволе кратки прекиди и/или скокови на линијама, могуће је препознати и софтверске клонове типа 3 [33].

Постоје и приступи базирани на тексту који су намењени откривању концептуалних клонова високог нивоа. Један такав приступ је Marcus i Maletic [34] који користи чињеницу да идентификатори и коментари често носе корисне информације о значењу програма. На текст изворног кода примењују латентно семантичко индексирање (енг. *Latent semantic indexing - LSI*). Циљ је у упоређеним секвенцама кода открити речи са истим или сличним значењем и на основу нађених речи проценити сличност упоређених секвенци кода.

Применљивост алата базираних на поређењу текста на проблем посматран у овом раду је ограничене услед непостојања изворног кода програма за који се сумња да користи библиотеку. Алати би могли бити примењени на асемблерску репрезентацију бинарног кода програма и библиотеке, при чему би се морала осмислити нормализација коришћених регистара и меморијских адреса. Чак и тако прилагођени, алати не би могли да се боре ни са неким једноставнијим изменама попут промене редоследа инструкција и супституцијом функционално еквивалентним инструкција.

2.2.2 Алати базирани на поређењу токена

Алати који пореде токене користе лексичку анализу, као што то раде и компајлери, да улазни текст трансформишу у низ токена. Добијене секвенце токена се касније пореде како би се међу њима пронашле краће секвенце токена које се понављају у различитим деловима кода. Поређење кода на нивоу токена их чини отпорним на једноставније измене попут различитог форматирања кода, различите употребе бланко знакова, а поједностављује се

и откривање клонова у којима је дошло до преименовања променљивих и функција. Ипак, за разлику од поређења кода на нивоу текста, токенизација кода захтева делимично познавање језика, па је за сваки језик потребно прилагодити алат. Обзиром да се упаривање ради на нивоу узоркованих токена, за потребе прилагођавања довољно је имплементирати нови лексер.

Отпорност ових алата на измене зависи од мере у којој се поређење секвенци токена релаксира. Директним поређењем секвенци токена уз захтев да сви токени буду идентични, постиже се отпорност само на клонове типа 1. Отпорност на неке од клонова типа 2 постиже се релаксирањем поређења токена који одговарају идентификаторима и литералним константама. Потпуно релаксирање у смислу изједначавања свих токена који представљају идентификатор и исто тако свих токена који представљају литералну константу чини алате потпуно отпорним на преименовање. Ипак, на описани начин се изједначавају сви идентификатори, па се може десити да два кода који употребљавају чак и различит број идентификатора, буду проглашени истим што представља лажни софтверски клон.

Отпорност на софтверске клонове типа 3 постиже се релаксирањем услова да целе упоређене секвенце морају бити потпуно идентичне. Уколико се омогући да у једној од секвенци недостаје неки од токена, постиже се отпорност на уметнуте и обрисане исказе. Ако се дозволи да се у упареној секвенци могу појавити и различити токени, постиже се отпорност на измену појединих исказа. Ипак, повећањем отпорности на измене повећава

се и могућност погрешног упаривања делова кода који имају потпуно различиту функционалност али имају сличну структуру.

Један од првих приступа базираних на токенима приказан је у алату Дуп [19]. Приступ предлаже токенизацију линија изворног кода у параметризоване и непараметризоване токене. Параметризовани токени су идентификатори и литералне константе, док су остали токени непараметризовани. У циљу убрзања приступ предлаже рачунање вредности хеш функције за сваку линију тако што се сви идентификатори и литералне константе замене једним симболом. Потом се иза израчунате вредности редом надовежу хеш вредности свих идентификатора и литералних константи по редоследу појављивања у посматраној линији, чиме се добија садржај који се касније користи за поређење. Пореде се само линије које имају исту хеш вредност, док се за хеш вредности индетификатора и литералних константи тражи 1-1 пресликавање да би две линије биле проглашене истим. Иако пореди појединачне линије, приступ сугерише увођење минималне дужине секвенце која ће бити проглашена софтверским клоном. За тражење секвенци које се упарују приступ користи алгоритме за упаривање подстрингова базиране на суфиксним стаблима.

Приступ описан у Дуп алату може да открије клонове типа 2 који се добијају преименовањем идентификатора и променом литералних константи, а без промене редоследа, што је познато као параметризован клон типа 2. Увођењем динамичког програмирања [35] и откривања најдуже дељене подсеквенце која се састоји од упарених линија, тако да се упарене

линије поређане у истом редоследу у упоређеним деловима кода са евентуално уметнутим линијама, постиже се отпорност и на неке клонове типа 3. Број измена које су потребне да би се од једног упоређеног кода добио други представља удаљеност уређивања (едитовања) и користи се за одређивање мере сличности упоређених делова кода. Динамичко програмирање је касније унапређено у оквиру алата CCFinder [20] унутар којег се у процесу нормализације примењују технике уређивања кода у складу са одређеним кодним стандардом како би се избегле ситне разлике попут различите употребе витичастих заграда у програмском језику C.

Због поређења секвенци алати Dup и CCFinder нису отпорни на замену редоследа инструкција. Како би превазишао тај проблем, CP-Miner [36, 37] користи технику за откривање подсеквенци [38]. Након парсирања, откривају се све честе подсеквенце и тако се долази до базичних секвенци које представљају копиране делове кода. Потом се примењују различите технике укључујући и преименовање идентификатора како би се одстранили лажни клонови, након чека се приступа формирању дужих секвенци које представљају клонове и поново понавља елиминација лажних софтверских клонова.

У ову групу алата спадају и неки од алата за откривање плагијаризма. Два најпознатија алата за откривање плагијаризма су Moss [39] и JPlag [40], оба базирана на токенима. Moss користи методу рачунања отисака и приказује најдуже секвенце поклапања отисака. JPlag директно упарује секвенце токена покушавајући наћи најдужа поклапања. Алгоритам се

понавља све док је могуће наћи још неку секвенцу довољне дужине која се поклапа. Обзиром да редослед упарених подсеквенци у упоређеним секвенцама токена не мора бити исти, постиже се отпорност на замену редоследа инструкција.

Применљивост алата базираних на токенима на проблем посматран у овом раду је нешто већа у односу на алате базиране на тексту услед могућности параметризације токена који би представљали операнде инструкција у асемблерском коду. Такође, релаксирање поређења секвенци у смислу дозвољавања уметнутих и обрисаних инструкција додатно доприноси применљивости на посматрани проблем. Ипак, уколико преводилац измени редослед инструкција у већем обиму, секвенце постају исувише различите да би биле препознате овим алатима. Такође, алати нису отпорни ни на неке друге оптимизације попут оптимизација петљи и уграђивања процедура на месту позива, када могу да прикажу велики број лажних софтверских клонова услед копирања делова кода које је извео сам преводилац.

2.2.3 Алати базирани на апстрактним синтаксним стаблима

Алати који користе апстрактна синтаксна стабла (енг. *Abstract Syntax Tree - ACT*) крећу од претпоставке да се улазни код парсира парсером за одговарајући језик који генерише АСТ. Приликом поређења алати у овој категорији користе неку од техника како би нашли иста подстабла, након чега приказују делове изворног кода који одговарају упареним подстаблима као софтверске клонове. Добра страна оваквих приступа јесте што се у АСТ

задрже практично све информације о посматраним изворним кодовима, па је могуће применити разне софистициране технике како би се открили клонови. Ипак, како би се један овакав алат прилагодио неком другом језику, неопходно је направити читав парсер за тај језик. Такође, за разлику од претходне две групе техника, изворни кодови који учествују у поређењу треба да буду синтаксно исправни како би се уопште могло приступити поређењу.

Један од првих алата који је увео поређење засновано на AST је CloneDR који је предложен у Baxter et al. [41]. Алат на почетку парсира улазни изворни код како би добио AST, а затим примењује три алгоритма како би открио софтверске клонове. У првом алгоритму открива подстабла која се поклапају. Како би се убрзала претрага, за подстабла се рачуна хеш функција и пореде се само она стабла која имају исту хеш вредност. Хеш функција је одабрана тако да задржи оне делове који у клоновима морају бити идентични, док занемарује детаље на којима се клонови могу разликовати. Други алгоритам тражи секвенце упарених подстабала што одговара тражењу секвенци наредби у програмима, док трећи алгоритам покушава да пронађе и оне клонове који нису потпуно идентични комбинујући претходно нађена упаривања секвенци подстабала.

Један приступ који је нешто раније предложио откривање разлика између два кода засновано на варијанти стабла парсирања предложио је Yang [42]. Приступ упарује стабла користећи технику динамичког програмирања, чиме се повећава отпорност на уметнуте и обрисане делове кода. За

постизање веће отпорности на измене неки приступи примењују и технике откривања скривеног знања. У складу с тим Wahler [43] открива егзактне и параметризоване клонове тако што АСТ конвертује у XML репрезентацију на коју примењује технике откривања скривеног знања како би открио клонове [44]. Корак даље иде алат предложен од Evans and Fraser [45], који поред параметризовања чворова стабла који представљају идентификаторе и литералне константе уводи параметризовање произвољног чвора у стаблу. На тај начин омогућава се упаривање клонова типа 3 у којима долази до разлика у појединим инструкцијама. Иако се на описани начин омогућава проналазак већег броја приближних клонова, мана приступа је што расте временска сложеност претраге јер простор претраге постаје знатно већи.

Иако је у АСТ задржана већина информација, идентификатори су занемарени како се не би десило да копирани делови кода са измењеним називима променљивих и процедура не буду препознати као софтверски клонови. У потпуности се занемарује и ток података што може довести до лажних клонова који се састоје од истих наредби између којих се подаци прослеђују на различит начин. Приступ је такође осетљив и на замену редоследа инструкција, па чак и на замену контролних структура програма. Све ове недостатке могу у одређеној мери да превазиђу алати засновани на графовима зависности у програму (енг. *Program Dependence Graph - PDG*).

Применљивост алата базираних на AST на проблем посматран у овом раду је ограничена услед непостојања изворног кода. Да би се приступи применили неопходно је имплементирати нови парсер који ће од

асемблерске репрезентације бинарног кода направити AST. Међутим, и у том случају за очекивати је веће стабло јер се неке инструкције из виших програмских језика преводе већим бројем машинских инструкција. Поред величине, за очекивати је да се генерисана стабла разликују у већој мери услед измена које преводиоци уносе. Повећање величина стабала значајно увећава време поређења, док велики број разлика у стаблу доводи у питање применљивост тренутно примењиваних алгоритама за откривање сличности међу оваквим стаблима.

2.2.4 Алати базирани на графовима зависности у програму

Алати засновани на графовима зависности у програму (PDG) подижу интерну репрезентацију програма за један степеник више задржавајући информације о контроли тока и/или току података који у програму постоје. Самим тим је задржана и семантика програма па је софистицираним алгоритмима могуће веома прецизно препознавање клонова. Када од програма направе PDG, алати у овој групи користе неки од алгоритама за откривање изоморфизма између подграфова како би се нашли слични делови. На крају, делови изворних кодова који одговарају упареним подграфовима приказују се као нађени клонови.

Алати у овој групи су отпорни на многе измене попут уметања и брисања инструкција, замене редоследа инструкција, преплитање кода, као и на неконтинуалност у коду. Мане су им велика сложеност алгоритама за откривање изоморфизма и слаба скалабилност. Стога алати прибегавају различитим техникама и хеуристикама које убрзавају рад.

Komondoor and Horwitz [46, 47] су предложили приступ и развили алат PDG-DUP који користи PDG и тражи изоморфне графове уз помоћ технике парчања програма(енг. *program slicing*). Техника проналази скуп свих инструкција које на одређеној позицији могу утицати на вредност посматране промењиве. Аутори алата су такође предложили приступ за груписање нађених софтверских клонова. При томе се чува семантика оригиналног кода како би се обезбедила аутоматизована екстракција процедура за потребе рефакторизације кода.

Откривање сличних подграфова максималне дужине могуће је и итеративним алгоритмом који предлаже Krinke [48]. PDG који се користи у овом приступу додаје ознаке на чворове и гране. Да би се чворови упарили, неопходно је да имају исте ознаке. За упаривање грана неопходно је да се упаре чворови, као и да гране имају исту ознаку. Упаривање графова је релаксирано тако да се сличним подграфовима сматрају они графови за које важи да се за сваку путању у једном графу може наћи одговарајућа путања у другом графу. Стога се за проналажење максималног подграфа који може да се упари користи приступ који упарује све путање без петље, а које крећу од једног почетног чвора. У првом кораку упарују се чворови који су једну ивицу далеко од почетног чвора. Даље се покушава упаривање наредних чворова који су једну ивицу даље од претходно упарених чворова, све у складу са претходним описом. Описани поступак се наставља све док је могуће даље упаривање.

У ову групу алата спада и GPLAG намењен откривању плагијаризма [25]. Аутори су се ограничили на пет измена у изворном коду које обухватају измену формата и преименовање идентификатора, промену редоследа инструкција, промену структура за контролу тока и уметање кода. Ниједна од наведених измена не нарушава изоморфизам подграфова који одговарају еквивалентним кодовима у складу са претходним ограничењима. Ипак, пошто се може десити да се појави и нека друга измена осим наведених пет, аутори предлажу релаксирано поређење графова, тако да се бројеви чворова у графовима могу разликовати до одређене границе, а да при томе та два графа буду проглашена сличним.

Да би се алати засновани на PDG применили на проблем посматран у овом раду неопходно је имплементирати нови парсер који ће од бинарног кода или његове асемблерске репрезентације конструисати граф зависности. За очекивати је да алат буде знатно отпорнији на добар део измена које преводиоци уносе самим тим што се у графу задржава само суштински поредак инструкција у складу са стварним зависностима између њих, што не сме да се промени у процесу превођења. Ипак, постоје и оптимизације попут супституције једне инструкције функционално еквивалентном и одмотавања петљи, које би захтевале адаптацију постојећих алгоритама за поређење PDG како би се открили сличности. Додатна мана алата заснованих на PDG је и временска сложеност коју имају алгоритми за поређење графова, посебно у случају када се у тим графовима очекује појава мањих разлика на које процес поређења треба да буде отпоран.

2.2.5 Алати базирани на софтверским метрикама

Алати базирани на метрикама прикупљају различите вредности софтверских мера процедура и уместо директног поређења кода, пореде вредности метрика. У току токенизације и/или парсирања програма издвајају се вредности мера које могу да буду број линија, број позива функција, број грана у графу контроле тока, као и многе друге мере. Мере могу да се прикупљају на нивоу различитих јединица, али су то најчешће класе, функције или делови функција. Добијене вредности метрике се затим пореде како би се утврдило да ли су упоређене јединице кода заиста клонови.

Mayrand [49] предлаже рачунање вредности неколико мера на нивоу функционалних јединица програма. Функционалне јединице које имају сличне вредности метрике се проглашавају за софтверске клонове. Делимилно сличне јединице се не детектују. Како би израчунао вредности метрика, приступ користи IRL (енг. *Intermediate Representation Language*). Метрике су рачунате на основу назива, распореда, израза и контроле тока функције. Због рачунања метрика на нивоу функције и копирања унутар тела те функције не могу бити откривена. Такође, копирање мањих делова тела једне функције у другу такође не може бити откривено из истог разлога. Ове мане су веома озбиљне у погледу откривања клонова обзиром да до копирања често долази унутар функција, а ређе читавих функција. Сличне методе и метрике се предлажу и у Patenaude [50], где се гледа број позива из функције, број наредби у функцији, McCabe цикломатска сличност, као и бројеви променљивих.

Два приступа за откривање софтверских клонова предлаже и Kontogiannis [51] од којих је један базиран на софтверским метрикама, док други користи динамичко програмирање. Приступ базиран на метрикама користи модификоване верзије пет познатих метрика како би упоредио делове кода ограничене са begin-end. Делови кода су слични и представљају клонове уколико су вредности следећих пет метрика приближно исте: број позиваних функција, однос броја улазних и излазних променљивих према броју позиваних функција, McCabe цикломатска комплексност, модификована Albrecht метрика (тежинска суме бројева одређених појава у коду) и модификована Henry-Kafura метрика за квалитет тока информација. Приступ формира AST, затим рачуна вредности метрика и придружује их одговарајућим чворовима у AST. Потом прави табелу са деловима кода и придруженим метрикама коју сортира по вредностима метрика и коју даље користи како би одабрао делове кода који се могу упарити. Други приступ полази од претпоставке да се се унутар кода често ради исецање делова кода и премештање на другу позицију. Стога се два дела кода између begin-end сматрају клоновима уколико је један могуће добити од другог малим бројем трансформација типа уметања, брисања и измене линије.

Софтверске метрике се користе и за откривање клонираних веб страна, као и клонирних делова на web странама. Један такав приступ предлаже се у Di Ducca et al. [52]. Метрике се користе и у приступу за полуаутоматизовано откривање клонираних функција у скриптовима [53, 54]. На почетку се аутоматизовано бирају функције које представљају

потенцијалне клонове, а затим се визуелном инспекцијом утврђује да ли се заиста ради о клоновима или не. Сличан приступ примењен је и у овом раду.

Примена алата базираних на софтверским метрикама захтева најмање имплементацију лексера, а у неким случајевима и парсера за бинарни код или његову асемблерску репрезентацију. Све у зависности од метрика које се прикупљају, од алата се може очекивати и да спроведе одређену анализу кода како би препознао неке конструкције попут петљи, што је у вишим програмским језицима доступно већ након парсирања. Алати овог типа су погодни за приману на посматрани проблем самим тим што имају могућност да избором одговарајућих метрика занемаре многе промене настале у процесу превођења, а да при томе задрже информације неопходне за поређење кода. Ипак, због занемаривања дела информација о упоређеним кодовима постоји опасност да се упаре и делови кода који не потичу од истог изворног кода, што може да представља њихов озбиљан недостатак.

2.3 Нарушавање лиценцних права

Иако је нарушавање лиценцних права (енг. *license violation*) област којом се бави овај рад, веома су ретки алати и приступи у овој области. Приступ који се предлаже у Hemel et al. [14] користи три технике како би се борио против потенцијалних разлика између кодова које треба да упореди. Прва техника најпре евидентира све литералне низове знакова који се појављују у траженом коду, а потом покушава да пронађе исте литерале у коду за који се сумња да крши лиценцна права. Друга техника полази од претпоставке да ће алгоритам за компресију података успети да постигне

већи степен компресије при компримовању два кода заједно уколико у тим кодовима постоје исте секвенце инструкција. Као мера успешности се узима однос величине архиве која садржи два кода и збира величина архива од којих свака садржи само један од два упоређена кода. Што је однос мањи, очекује се да је већа сличност између упоређених кодова. Трећа техника се базира на рачунању бинарних разлика (енг. *binary deltas*) између кодова. Што је разлика мања, то је већа вероватноћа да посматрани код крши лиценцна права.

Алат BAT (Binary Analysis Tool) заснован на овом приступу користи се за откривање да ли посматрани код крши лиценцна права неког од кодова из репозиторијума. Иако приступ предлаже три технике, у оквиру алата имплементирана је само прва техника. Разлог томе је што је алат првенствено коришћен за фабричке кодове уграђене у рутере, у оквиру којих се појављивао одређени број литералних низова знакова. За потребе поређења у овом раду од таквог алата се не очекују резултати обзиром да у неким од кодаова уопште нема стринг литерала. Зато је за потребе поређења имплементирана друга техника која за откривање сличности између два кода користи алгоритам компримовања података.

2.4 Анализа рањивости софтвера

Рањивост софтвера може да има веома велике последице у виду великих финансијских и материјалних губитака које злонамерни појединци и организације могу да нанесу другима. Стога је анализа рањивости софтвера (енг. *vulnerability analysis*) од изузетне важности, посебно за системе од којих

зависе људски животи и материјална и финансијска добра. Један облик анализе спроводи се када су већ познате рањивости и недостаци једне верзије софтвера и жели се утврдити да ли су те рањивости елиминисане у новој верзији. Још битније, циљ анализе је и да се провери да ли су се можда у новој верзији кода појавиле неке нове рањивости, што треба тражити у новим и измењеним деловима кода. У циљу убрзања анализе нове верзије најбоље је наћи све разлике између старе и нове верзије, а затим анализирати само нађене разлике. Један такав приступ предложио је Dullien [16]. Приступ је описан у наставку и имплементиран је за потребе поређења.

Приступ полази од два бинарна кода који се састоје од процедура и покушава упарити процедуре које су скоро идентичне. Први предуслов за упаривање процедуре је да су идентичне по задатим критеријумима (тзв. селекторима). Као селекторе приступ користи број базичних блокова, број ивица у графу контроле тока и број позиваних потпрограма. Други предуслов за упаривање процедура јесте да у посматраним скуповима не постоји ниједна друга процедура која је по коришћеним критеријумима идентична упоређеним процедурама. У циљу смањења рестриктивности другог предуслова у свим корацима алгоритма се посматрају подскупови процедура из упоређених бинарних кодова. При формирању првих упарења подскупови се формирају издвајањем процедура које задовољавају неко својство. Нека својства која се користе су број улазних и излазних грана процедура у графу позива, исти називи, референце ка истим стринговима и бројеви појављивања појединих инструкција. Када даље упаривање на основу наведених својстава није могуће, нови подскупови се формирају од функција

које се позивају из упарених процедура. Такође, подскупови се формирају и од функција које позивају упарене функције. Поступак се понавља за сваке две упарене процедуре све до тренутка када није могуће направити даље упаривање. Унутар упарених процедура се ради даље упаривање базичних блокова и инструкција.

2.5 Откривање злонамерног кода

Веома важно место у данашњем рачунарству заузима и област откривања злонамерног кода (енг. *malware detection*). Извори настанка таквог кода су разноврсни и најчешће мотивисани лошим намерама да се са рачунара корисника добију одређене информације, као и да се тај рачунар искористи за даље ширење злонамерног кода. Стога је од огромног значаја брзо и поуздано откривање злонамерног кода, па је велики број рачунара опремљен неким софтверским системом који стално ради на откривању злонамерног кода и тако штити рачунар и све податке које корисник држи у рачунару.

За откривање злонамерног кода користе се различити приступи. Најчешће приступи користе потписе познатих злонамерних кодова како би препознали њихову појаву унутар неког програма. У потписима се често користе и информације о позиваним системским позивима, што омогућава да се корисник упозори да неки софтвер испољава потенцијално малициозно понашање и пре него што је откривена нова верзија злонамерног кода. Ипак, од интереса за домен који се посматра су решења која се приликом поређења кода не ослањају на називе процедура, као ни на називе системских позива. У

складу с тим, у наставку је описан приступ који предлаже Santos [15]. Поменути приступ је имплементиран за потребе поређења.

Приступ за откривање злонамерног кода пореди секвенце операционих кодова. Посматрају се секвенце фиксне дужине и узима у обзир да различити операциони кодови имају различиту релевантност приликом поређења. Приступ најпре утврђује релевантност операционих кодова тако што спроведе анализу великог броја злонамерних и регуларних кодова како би се за сваки операциони код нашли бројеви појављивања у злонамерном и регуларном коду. Користећи нађене бројеве појављивања и меру узајамних информација (енг. *mutual information*) одреди се релевантност операционог кода. Сматра се да инструкције које се подједнако појављују у оба кода имају мању релевантност за откривање мутираних варијанти малициозног кода, те им се стога приликом поређења даје мањи значај. На тај начин се очекује занемаривање шума који настаје међу инструкцијама по основу којих се малициозни и регуларни кодови не разликују битно. Након рачунања релевантности операционих кодова, пребројавају се и све секвенце операционих кодова одабране дужине унутар упоређених процедура и рачунају њихове фреквенције појављивања. Тако се за сваку процедуру формира један вектор чије су компоненте фреквенције појављивања секвенци које се у тој процедури појављују. Свака од компонената вектора се множи производом релевантности свих операционих кодова који се појављују у посматраној секвенци како би се елиминисао шум који уносе ирелевантни кодови. На крају се применом косинусне сличности на векторе који описују упоређене процедуре рачуна сличност између процедура.

Приступ такође сугерише да се комбинују резултати добијени за више различитих дужина посматраних секвенци, при чему су испробане само секвенце дужине један и два, као и њихова комбинација.

Описани приступ није директно применљив на проблем посматран у овом раду. Разлог је непостојање две различите групе софтвера које су неопходне за одређивање релевантности операционих кодова. Стога је за потребе поређења са предложеним приступом имплементиран описани приступ без одређивања релевантности операционих кодова.

2.6 Компакција кода

Компакција кода (енг. *code compaction*) је област која се бави смањивањем величине кода [55, 56]. Примену је нашла у индустрији, када је ради постизања уштеда на рачун смањења количине потребне меморије потребно смањити величину програма. Смањивање простора могуће је и у току превођења програма, али исто тако и након завршеног превођења. Када се смањивање кода ради након завршеног превођења неопходно је поредити делове кода како би се нашли исти или довољно слични делови који могу да се замене само једним кодом приближно исте величине као што је величина једног од полазних кодова. Иако се у оквиру приступа ради поређење кода на бинарном нивоу, поређење не може бити отпорно на велике разлике између кодова јер компакција кода мора бити без губитака. Сличан приступ приказан је у оквиру алата за откривање нарушености лиценцних права, при чему је коришћена компресија података. С обзиром да ће поменути приступ

бити коришћен за потребе поређења са предложеним приступом, компакција кода даље неће бити разматрана засебно.

3. Дефиниција проблема

Неовлашћена употреба софтверске библиотеке чији изворни код је јавно доступан и чија употреба у комерцијалне сврхе није бесплатна, у савременом свету представља озбиљан проблем. Откривање употребе софтверске библиотеке може да буде веома комплексан проблем, што ће бити показано евалуацијом постојећих алата и приступа. Комплексност лежи у чињеници да значајне измене које преводилац унесе од алата захтевају флексибилност и отпорност на велике разлике између кодова које пореде, што са друге стране доводи до упаривања различитих процедура. Због тога се у овом раду посматра потпроблем који се односи на убрзање проналажења употребљених процедура. У складу с тим, сви тестови који се у раду користе задовољавају услов да садрже бинарни облик процедуре који је добијен превођењем изворног кода тражене процедуре.

Превођењем изворног кода неке софтверске библиотеке могуће је добити велики број различитих, функционално еквивалентних облика бинарног кода. Разлике међу бинарним кодовима могу да буду резултат различитог процеса превођења као и последица прилагођавања изворног кода библиотеке у циљу отклањања недостатака постојећег кода или додавања нових функционалности. Прилагођавањем изворног кода се добија

нови облик кода који није идентичан посматраном оригиналном коду софтверске библиотеке. Стога се доводи у питање до које границе је могуће мењати код а да се измењена верзија и даље посматра као копија оригиналног кода софтверске библиотеке. Имајући у виду претходно питање и чињеницу да се библиотеке веома често користе без икаквих измена, овај рад ће разматрати само случај када је посматрани бинарни облик библиотеке добијен превођењем оригиналног изворног кода без икаквих измена.

Преведени код је веома често доступан само у бинарном облику из којег су одстрањени сви или скоро сви записи који би могли помоћи при упаривању делова кода са изворним кодом тражене софтверске библиотеке. Посебан случај су наменски уређаји у које је по правилу учитан машински код који даље није потребно преправљати у циљу извршавања учитаног кода и који не садржи никакве додатне информације попут табеле глобалних симбола, табеле релокација и других структура које су често саставни део извршних програма. Чак и када су одређене информације о називима променљивих и процедура доступне, систематизованом променом назива могуће је прикрити очигледне трагове који би водили откривању употребе библиотеке. Стога ће се у раду разматрати најкомплекснији случај када у коду уопште не постоје никакве додатне информације о називима симбола. Како се таква ситуација најчешће дешава у случају наменских уређаја, за потребе евалуације предложеног приступа користиће се ARM архитектура која представља основу већине савремених наменских уређаја [57].

Под претпоставком да се из уређаја може прочитати машински код, у циљу откривања употребе софтверске библиотеке неопходно је упарити делове бинарног кода са изворним кодом библиотеке. Обзиром да су кодови које треба упоредити доступни у различитим облицима и да практично сви доступни алати захтевају да делови кода који ће се упоредити буду у истом облику, директна примена неког од постојећих алата није могућа. У циљу примене постојећих алата неопходно је извршити трансформацију кодова који се пореде у облик који алат захтева, што може да буде један од облика у којима се налази један од кодова који се пореде или неки трећи облик различит од оба облика у којима су доступни кодови.

Прва могућност је да се трансформисањем бинарног кода у изворни омогући примена алата за откривање софтверских клонова који пореде изворни код, што представља начин рада већине алата у поменутој групацији. Таква трансформација се назива декомпајлирање и представља веома комплексан задатак чији резултат рада није јединствен. Обзиром да називи симбола коришћени у оригиналном изворном коду нису познати у процесу декомпајлирања, декомпајлирани код практично не може бити идентичан оригиналном изворном коду од којег је добијен декомпајлирани бинарни код. Као резултат декомпајлирања се добија облик кода који је функционално еквивалентан и који представља скуп софтверских клонова оригиналног изворног кода. Разлике могу да буду само на нивоу назива језичких елемената, али исто тако могу бити резултат трансформација које је компајлер применио током превођења програма, као и оних које је декомпајлер применио у процесу синтезе изворног кода.

Друга могућност је превођење изворног кода библиотеке у бинарни код и употреба неког од алата који раде поређење бинарног кода. Као и у првом случају, добијени бинарни код по правилу није идентичан (поредећи бајт по бајт) деловима бинарног кода за који се испитује да ли користи посматрану библиотеку. Код са којим се библиотека повезује утиче на адресу на коју ће библиотека бити смештена. Различите адресе које се на одговарајућим местима појављују у кодовима који се пореде отежавају поређење. Ипак, систематским преправљањима адреса, што представља поступак сличан преименовању променљивих и потпрограма, могуће је значајно олакшати процес поређања, а у неким случајевим направити и потпуно идентичне кодове. Много озбиљније разлике уносе сами компајлери и алгоритми које користе приликом превођења, као и оптимизације које се од компајлера захтевају приликом превођења, те стога ни на овај начин није могуће елиминисати разлике које настају у процесу превођења.

Трећа могућност је превођење кодова које је потребно упоредити у неки трећи облик, погоднији за поређење. Један такав пример је ACD [13], који кодове преведе на асемблер као један вид нормализације кода, а затим упореди добијене асемблерске кодове. Слично томе, алат предложен у Juričić [58] користи овај приступ како би упоредио кодове написане на различитим програмским језицима.

У процесу евалуације користиће се алати од којих сваки на улазу очекује неки облик кода, потенцијално различит од облика погодних осталим алатима. Једини начин да се поређење изведе јесте да се

припремљени тестови преведу у одговарајући облик за сваки од алата који ће се користити. У случају алата који користе изворни код користиће се декомпајлиран облик посматраног програма. Иако је тражена процедура доступна у облику изворног кода, користиће се декомпајлирани облик процедуре која је претходно преведена у бинарни облик, чиме се смањује негативан утицај процеса декомпајлирања на поређење. За остале алате биће коришћени или бинарни облици програма и тражене процедуре или њихови еквиваленти кода на асемблеру добијени дисасемблирањем бинарних облика, што је случај са предложеним приступом.

У циљу убрзања откривања употребе софтверске библиотеке у овом раду ће бити предложен нови приступ за процену сличности процедура доступних у бинарном облику. Основна идеја је да се уз увођење нових софтверских метрика намењених за примену на бинарном коду, предложи алгоритам за процену сличности процедура доступних у бинарном облику. Проценом сличности процедуре која потиче из библиотеке са сваком од процедура из бинарног кода убрзало би се откривање употребе библиотеке тако што би се прво анализирале процедуре које су најсличније траженој процедури.

У складу са описаним проблемом и ограничењима, задатак овог истраживања је да одговори на следећа питања:

1. Да ли додавање нових метрика доприноси повећању одзива предложеног приступа у односу на употребу метрика које су инспирисане одговарајућим метрикама намењеним вишим програмским језицима?

2. Да ли одзив који постиже предложени приступ зависи од компајлера, нивоа оптимизације и контекста у којем је преведена тражена процедура?

3. Да ли предложени приступ постиже већи одзив од постојећих алата?

4. Да ли је појединачним додавањем нових техника за ублажавање негативних последица поређења на основу предложених софтверских метрика могуће добити већи одзив од одзива који постиже основни алгоритам предложеног приступа?

5. Да ли је додавањем свих техника заједно могуће добити синергистички ефекат ?

6. Да ли је уз употребу предложених техника могуће постићи већи одзив од одзива постојећих алата и приступа који су коришћени за упоређивање са предложеним приступом.

4. Окружење за евалуацију

У овој глави је описано окружење за евалуацију. У Секцији 4.1 су описани тестови и поступак припреме тестова за евалуацију, а у Секцији 4.2 мере које се користе за оцену успеха упоређених алата и приступа.

4.1 Опис тест података

Евалуација предложеног приступа, као и компаративна анализа постојећих и реконструисаних алата је спроведено коришћењем две групе тест узорака. Прва група тест узорака се базира на STAMP тестном окружењу [59] и представља синтетичке тестове који се користе за прелиминарну анализу резултата предложеног решења, као и поређење са постојећим и реконструисаним алатима. Друга група тест узорака се базира на BusyBox [60] софтверском пакету и представља реалан случај употребе софтверске библиотеке на којем се верификују резултати предложеног приступа.

За тест узорке у првој групи користе се процедуре следећих пет програма из STAMP тестног окружења: Bayes, Genome, Intruder, SSCA2 и Vacation. У Табела 4.1 је за сваки од наведених програма наведен број процедура од којих се програм састоји. Да би процедура била кандидат за потрагу, неопходно је да та процедура буде део програма који се претражује.

Табела 4.1 Број процедура у сваком од пет коришћених програма из STAMP окружења, као и број дељених процедура између свака два програма.

	Број процедура	Број дељених процедура између програма				
		Bayes	Genome	Intruder	Ssca2	Vacation
Bayes	280	280	214	213	214	214
Benome	239	214	239	226	226	227
Intruder	257	213	226	257	226	226
Ssca2	245	214	226	226	245	226
Vacation	297	214	227	226	226	297

Друга група тест узорака се формира од процедура Networking дела BusyBox софтверског пакета. Цео пакет BusyBox није коришћен јер су доступне верзије неких преводилаца имале ограничење по питању величине излазног програма. Разлог због којег је коришћен баш овај део пакета јесте употреба MatrixSSL библиотеке[61] која је издата под двоструком лиценцом, што је и био један од основних мотива овог рада.

Сви програми из оба теста (пет из првог и један из другог) преведени су са следећих пет преводилаца за ARM архитектуру: Keil (v4.60.0), IAR (v6.50.2), CodeSourcery (v4.7.2), CrossWorks (v2.3.0), and SysProgs (v4.6.3). Приликом превођења, сваки програм је преведен сваким од наведених компајлера са најнижим и највишим нивоом оптимизације. Програми су преведени и са оптимизацијама за величину и брзину у случајевима када преводилац подржава те оптимизације и када се излази превођења разликују

од осталих излаза истог преводиоца. Како би се стекао први утисак о разликама које преводиоци уносе, у Табела 4.2 су приказане величине програма у бајтовима, као и број линија које би се добиле ако се код дисасемблирао, за сваки од програма и компајлера са две различите опције превођења, без оптимизација (O0) и са највишим нивоом оптимизација (O3).

Табела 4.2 Величине програма након превођења са сваким од алата и два различита нивоа оптимизације, изражено у бајтовима и броју линија.

		bayes		genome		intruder		ssca2		vacation	
		Вел.	Лин.	Вел.	Лин.	Вел.	Лин.	Вел.	Лин.	Вел.	Лин.
Code Sourcery	O0	113308	28734	98612	23362	99972	24051	143192	36402	103732	25650
	O3	83184	17772	66448	12516	66812	12792	92444	20737	70376	14115
Cross Works	O0	49276	17609	29708	10657	37180	13444	35948	11861	38956	14194
	O3	36172	12631	22812	8179	25964	9390	26716	9222	27292	9931
IAR	O0	15624	10005	6156	4245	9600	6189	10544	5904	12656	7883
	O3	13436	8450	5228	3618	7886	5165	9216	5140	10640	6720
Keil	O0	31056	11517	20232	7738	25896	9836	31984	10659	26248	9973
	O3	24596	8908	16144	5914	19972	7370	22372	7678	20088	7415
SysProgs	O0	123592	33492	109464	28215	110664	28819	156120	42226	114380	30430
	O3	88484	20945	76240	16838	76488	17046	101124	25017	79156	18098

Да би превођење уопште било могуће, изворни кодови програма су прилагођени. Први проблем при превођењу је био недостатак одговарјућих процедура из стандардних библиотека. Стога су за потребе превођења направљене празне имплементације недостајућих функција. Други проблем

се односи на употребу нестандардних језичких конструкција, што онемогућава превођење неким од коришћених преводилаца. Стога је код преправљен тако да га је могуће превести свим преводиоцима. Приликом модификовања кода тежило се задржавању оригиналних функционалности програма, осим у случајевима када би то захтевало улагање нерационално великог труда. Описани приступ модификовања кода је прихватљив за посматрани рад јер се преведени програми неће извршавати. Након описаних прилагођавања исти изворни код се преводи са свим поменутим преводиоцима и опцијама.

Преведени код је потом дисасемблиран IDA алатом који представља индустријски стандард у домену реверзног инжењеринга [62]. Резултат је облик кода који може да се доведе на улаз парсера експерименталног алата развијеног за потребе тестирања предложеног приступа. Уз дисасемблиране процедуре су задржани и називи уколико су исти постојали у извршном програму. За оне преводиоце који су одстрањивали називе процедура, називи су прикупљени из MAP датотека које преводиоци на захтев генеришу. Задржани називи нису коришћени ни за потребе предложеног приступа ни за потребе постојећих алата и приступа већ само за потребе контроле да ли је тестирани алат нашао тражену процедуру. Извршни програми су декомпајлирани истим алатом како би се омогућила употреба постојећих алата који раде над изворним кодом. Обзиром да неки алати захтевају изворни код који је могуће превести, а да то није било могуће увек, декомпајлиране верзије су ручно дорађене.

На основу дисасемблираних и декомпајлираних верзија програма формирају се тест узорци. Узорци се бирају на случајан начин. Циљ је да ниво поузданости буде 99%, а да интервал поузданости буде 5%. У складу са тестом Колмогорова и Смирнова [63], да би се постигао постављени циљ, неопходно је да тест узорак садржи најмање 1063 појединачна теста. Због алгоритма који је коришћен за формирање тест скупа у којем се сваки тест може наћи подједнако вероватно, циљни број тестова је постављан на нешто већу вредност и у већини случајева износи 1500 тестова по узорку.

Тестови који су коришћени за евалуацију предложеног приступа и постојећих алата и приступа састоје се од једне процедуре која се тражи и скупа процедура из бинарног програма у којем се процедура тражи. Програм из којег се бира тражена процедура и програм у којем се она тражи бирају се насумично, при чему се може десити да се исти програм изабере оба пута. Програм се идентификује називом проблема чији изворни код је преведен, као и називима преводиоца и опције преводиоца коришћеним приликом превођења. Бинарни облик тражене процедуре се бира из једног од одабраних програма тако да у другом програму постоји процедура која потиче од изворног кода тражене процедуре. У случају тестова базираних на BusyBox пакету намеће се и додатни услов да тражена процедура потиче из библиотеке. За одређене тестова може се захтевати и да оба изабрана програма потичу од истог изворног кода, као и да су преведени истим преводиоцем и/или истим нивоом оптимизације коришћених преводилаца, све у зависности од хипотезе која се тестира.

Предложени приступ у неким случајевима захтева постојање претходног знања. За ту намену из преведених програма издвајају се узорци са по две произвољно одабране процедуре. Процедуре се бирају тако да у пола парова буду процедуре које не потичу од истог изворних кодова, док у преосталој половини парова важи обрнуто. Употреба оваквих узорака биће објашњена приликом описа компонената предложеног приступа које захтевају претходно знање.

4.2 Мере успеха

Очекивани резултат приликом потраге за процедуром из библиотеке унутар посматраног извршног програма је проналазак тачно једне процедуре за коју је установљено да потиче од изворног кода тражене процедуре. За све остале процедуре се очекује да буде установљено да не потичу од изворног кода тражене процедуре. Да би се очекивани резултат добио неопходно је да процес поређења не буде осетљив на разлике које настају у процесу превођења, а да при томе примети све разлике међу процедурама које се пореде. Два наведена захтева су контрадикторна. Смањењем осетљивости на разлике повећава се вероватноћа да се пронађе процедура која потиче од изворног кода тажене процедуре, али исто тако се повећава и вероватноћа погрешног закључка за процедуре које не потичу од изворног кода тражене процедуре. Са друге стране, повећањем осетљивости на разлике међу упоређеним процедурама умањује се и могућност проналажења одговарајуће процедуре у случајевима када процес превођења направи велику разлику између упоређених процедура.

Важност проналаска тражене процедуре условљава подешавање осетљивости алата тако да буде отпоран и на веће разлике међу упоређеним кодовима, па чак и по цену да се уложи мало више времена у мануелно поређење процедура. Последица је да се у пракси од алата ипак очекује да за једну тражену процедуру врати и већи број процедура за које сумња да потичу од изворног кода тражене процедуре. У идеалном случају, алат би за сваку тражену процедуру вратио по једну процедуру и то баш ону која потиче од исте процедуре у изворном коду. Међутим, у реалном случају се дешава да за неке тражене процедуре уопште не буде нађена процедура која потиче од исте процедуре у изворном коду, било да алат не нађе ниједну довољно сличну процедуру или да међу сличним процедурама ниједна не потиче од изворног кода тражене процедуре. Стога однос броја нађених и броја тражених процедура представља прву меру која описује постигнуће алата. Ова мера је позната као одзив (енг. *recall*) и често се рачуна у процентима. У описаном случају, када алат за једну тражену процедуру врати већи број сумњивих процедура, одзив као мера успеха није довољан јер не описује додатни напор који експерт треба да уложи да би међу сумњивим процедурама пронашао баш ону која потиче од изворног кода тражене процедуре или утврдио да таква процедура не постоји међу онима које је алат означио сумњивим. Друга мера, која надопуњује прву, представља однос броја нађених процедура и укупног броја прегледаних процедура. Та мера се у теорији назива прецизност (енг. *precision*), често се изражава у процентима и говори који део утрошеног експертског времена је оправдано утрошен на поређење процедура које потичу од исте процедуре у изворном коду. Важно

је напоменути и да алати вишеструко скраћују време које експерт уложи у претрагу, али такав податак не представља мерило успеха алата, већ само значај који алат има приликом потраге за траженом процедуром.

Поред описаних мера, у литератури се користи и такозвана F мера. Ова мера комбинује одзив и прецизност у једну јединствену меру према следећој формули:

$$F_\beta = \frac{(1+\beta^2) * preciznost * odziv}{\beta^2 * preciznost + odziv}$$

У наведеној формули β представља коефицијент који омогућава подешавање утицаја који на F меру имају одзив и прецизност. Вредност овог коефицијента може да буде било који ненегативан реалан број. Вредност један даје подједнак значај прецизности и одзиву. Вредности веће од један истичу одзив (за $\beta \to +\infty$ вредност F мере тежи вредности одзива), док вредности мање од један стављају акценат на прецизности (за $\beta = 0$ вредност F мере је иста као вредност прецизности). Најчешће вредности које се користе су 0.5, 1.0 и 2.0. Обзиром да одзив у овом раду има благу предност, поред стандардне F мере, код које је вредност коефицијента 1.0, користиће се и F_2 мера.

Као успех може да се посматра случај када је само најсличнија процедура баш она која се тражи. Међутим, због важности проналаска тражене процедуре и спремности експерта да уложи мало више времена и труда, успехом се може сматрати и случај када се тражена процедура нађе међу ***n*** најсличнијих процедура, где је ***n*** број најсличнијих процедура које

експерт жели да анализира. Због тога ће све мере бити приказане у зависности од броја ***n***, који ће се у наставку рада помињати као број посматраних позиција. Приликом рачунања одзива, процедура се рачуна као нађена уколико је међу ***n*** најсличнијих процедура. Ситуација је нешто компликованија у случају рачунања прецизности. Уколико је процедура нађена на позицији ***k***, где је $k \leq n$, тада се рачуна да је од ***k*** прегледаних процедура пронађена једна. У супротном, уколико је $k > n$, тада се рачуна да је ***n*** процедура анализирано без потребе, и да није пронађена процедура која потиче од изворног кода тражене процедуре. У случају алата који не могу да сортирају процедуре по сличности, већ само за више од једне процедуре констатују да су сличне са траженом процедуром, приликом рачунања прецизности увек се рачуна да је прегледано ***n*** процедура.

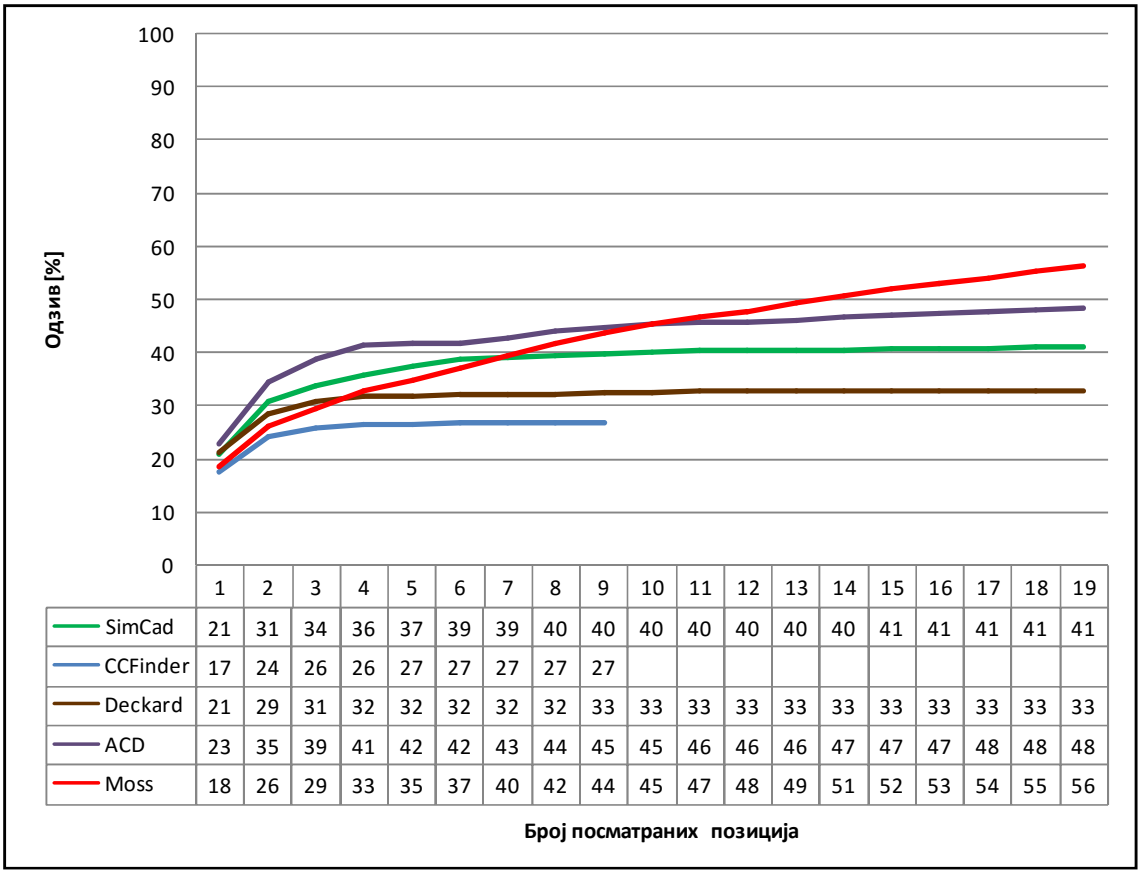
Приликом сортирања процедура из бинарног кода се може догодити да процедура која потиче од изворног кода тражене процедуре има исту вредност мере сличности са траженом процедуром као и неке друге процедуре које су различите од тражене процедуре. Нека су све те процедуре алгоритмом сортирања рангиране почев од позиције ***k*** до позиције ***p***, укључујући обе границе. У том случају се одговарајућа процедура може наћи на било којој позицији од ***k*** до ***p*** са подједнаком вероватноћом и та вероватноћа износи $1/(p-k+1)$. Како би се урачунао ефекат проналажења већег броја процедура са истом сличношћу према траженој процедури, уместо да се рачуна да је једна процедура нађена на некој од позиција у интервалу [***k***..***p***], рачуна се као да је на свакој од позиција нађено $1/(p-k+1)$ процедура. Коначан ефекат описаног приступа је да се у случају када се све

процедуре са истом сличношћу према траженој процедури рангирају међу најсличнијих ***n*** процедура (случај када је ***p*** ≤ ***n***), рачуна да је нађена процедура. Аналогно важи и у случају када су све процедуре са истом сличношћу рангиране после најсличнијих ***n*** процедура (случај када је ***k*** > ***n***), када се рачуна да није нађена тражена процедура. У случају када се део процедура са истом сличношћу рангира међу најсличнијих ***n***, а други део после (случај када је ***k*** ≤ ***n*** и ***p*** > ***n***), рачуна се да је нађено ***(n-k+1)/(p-k+1)*** процедура. Слично важи и приликом рачунања прецизности, с тим да се рачуна да је прегледано ***min(p,n)*** процедура.

5. Евалуација постојећих алата и приступа

Обим разлика које приликом превођења настају најбоље се може истаћи кроз приказ резултата које постижу алати из исте и сродних области. У евалуацији учествују доступни, као и реконструисани алати који су описани у другој глави. Алати се тестирају тест узорком базираним на STAMP тестном окружењу. Тестови су припремљени у складу са описом датим у Глави 4, при чему се програми бирају потпуно произвољно.

Резултати тестирања постојећих алата приказани су на Слика 5.1. Са слике се види да за мање вредности броја посматраних позиција најбољи резултат постиже алат ACD. За већи број посматраних позиција у благу предност прелази алат Moss. Највећи одзив постигнут је када се посматра само прва позиција и износи свега 23%. Другим речима, посматрајући само прву позицију, очекује се да се нађе мање од четвртине тражених процедура. Повећањем броја позиција које се посматрају расте и одзив. Међутим, чак и приликом посматрања првих 19 позиција, највећи одзив је нешто већи од 50%. Примећује се и да преостала три алата већ након релативно малог броја посматраних позиција улазе у засићење и даље повећавање броја позиција за њих нема смисла.

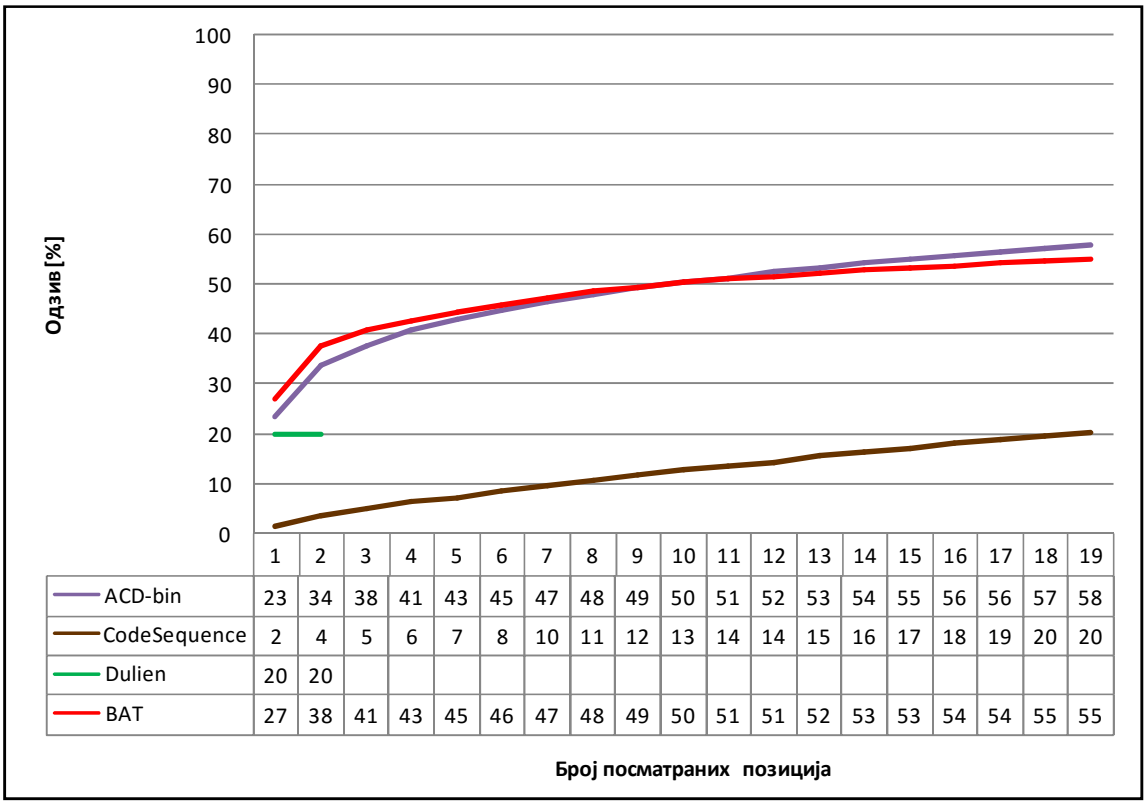


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	21	31	34	36	37	39	39	40	40	40	40	40	40	40	41	41	41	41	41
CCFinder	17	24	26	26	27	27	27	27	27										
Deckard	21	29	31	32	32	32	32	32	33	33	33	33	33	33	33	33	33	33	33
ACD	23	35	39	41	42	42	43	44	45	45	46	46	46	47	47	47	48	48	48
Moss	18	26	29	33	35	37	40	42	44	45	47	48	49	51	52	53	54	55	56

Слика 5.1 Одзиви постојећих алата у зависности од броја посматраних позиција. Алати прихватају изворни код добијен декомпајлирањем, осим Moss алата који може да прихвати и асемблер одређених архитектура.

На Слика 5.2 приказан је одзив алата које је аутор дисертације реконструисао на основу описа датих у одговарајућим радовима. Из приложеног се примећује да се највећи одзив за мали број позиција постиже употребом алата BAT, док већ за број позиција већи од 11 ситуација се незнатно мења и незнатно већи одзив постиже ACD. Максимална вредност одзива посматрајући само прву позицију износи 27%, што је више у односу на постојеће алате, али и даље оставља пуно простора за евентуално побољшање. Треба приметити и да је приступ ACD коришћен у оба поређења.

За прво поређење је коришћен постојећи алат који на улазу прихвата изворни код. Обзиром да поменути алат поређење ради на нивоу бинарног кода, за друго поређење је реконструисан алгоритам поређења и директно примењен на поређене бинарне кодове.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ACD-bin	23	34	38	41	43	45	47	48	49	50	51	52	53	54	55	56	56	57	58
CodeSequence	2	4	5	6	7	8	10	11	12	13	14	14	15	16	17	18	19	20	20
Dulien	20	20																	
BAT	27	38	41	43	45	46	47	48	49	50	51	51	52	53	53	54	54	55	55

Слика 5.2 Одзиви алата који представљају реконструкције постојећих приступа у зависности од броја посматраних позиција. Алати прихватају бинарни код или његову асемблерску репрезентацију. Приказани ACD алат представља реконструкцију алгоритма оригиналног алата, с тим да реконструисани алат на улазу прихвата бинарну репрезентацију програма.

6. Предложени приступ и евалуација

Две процедуре у бинарном облику које су различитим процесима превођења настале од истог изворног кода могу да се разликују до те мере да ни вешто око експерта на први поглед не може утврдити да се ради о истој процедури. Да би се препознала сличност неопходно је занемарити разлике које су приликом превођења настале. Другим речима, неопходно је препознати карактеристике кода које у одређеној мери могу да идентификују процедуру и на које процес превођења нема утицаја или има мали и ограничен утицај. Такве карактеристике се издвајају као вредности софтверских метрика упоређених процедура и у наставку процеса поређења репрезентују упоређене процедуре.

Циљ предложеног приступа је да прилагоди постојеће и конструише нове софтверске метрике са циљем да се занемаре разлике настале у процесу превођења, а да се и даље задржи довољно информација како би вредности софтверских метрика могле довољно верно репрезентовати процедуре у процесу поређења. Занемаривањем делова кода се ипак долази у ситуацију да на основу појединачних вредности метрика и потпуно различите процедуре личе једна на другу. Зато је неопходно пронаћи начин да се на основу вредности већег броја метрика донесе закључак о сличности

процедура. Један начин коришћен у овом приступу, јесте да се употребом неке од формула које приступ предлаже израчуна тотална мера сличности упоређених процедура, при чему термин "тотална" означава да је вредност израчуната на основу већег броја софтверских метрика. Тотална мера сличности се користи како би се процедуре поређале од најсличније до најмање сличне и не представља апсолутну меру сличности на основу које би се поуздано могло закључити да ли упоређене процедуре потичу од истог изворног кода или не.

Поред метрика и поступка за одређивање тоталне мере сличности приступ предлаже увођење 5 додатних техника. Технике имају за циљ да ублаже негативне ефекте који настају као резултат губитка информација. Ублажавање се постиже прикупљањем додатних информација које нису директно обухваћене предложеним софтверским метрикама, као и додатним процесирањем резултата на основу већ постојећих техника.

Остатак главе организован је у пет поглавља. У прва два поглавља дати су опис и евалуација основног приступа. У трећем поглављу су описани неки од недостатака предложеног приступа и уведено је пет техника намењених за повећање одзива, док је у четвртом поглављу приказана спроведена евалуација предложених приступа. Резултати евалуације основног приступа и предложених техника искоришћени су да би се у петом поглављу дали одговори на постављена истраживачка питања.

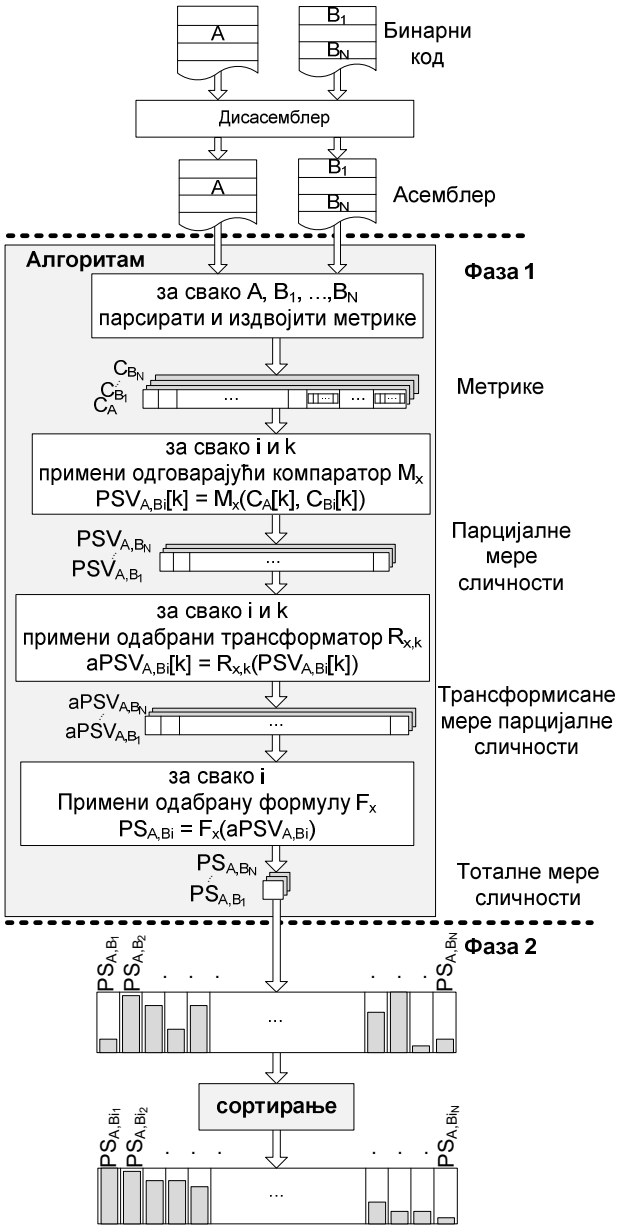
6.1 Основни приступ

Основни приступ, без предложених техника приказан је на Слика 6.1. Алгоритам на улазу очекује једну процедуру која се тражи (означену као *A*) и скуп процедура међу којима се тражи процедура *A*. Процедуре у скупу су означене са B_i, где *i* иде од *1* до *N*, а *N* је број процедура у посматраном скупу. Процедуре у скупу најчешће потичу из једног извршног програма. Тренутна верзија експерименталног алата очекује да су и тражена процедура и процедуре из скупа дисасемблиране. Овај захтев се може избећи конструкцијом додатног парсера извршног програм и анализом извршног програма у циљу утврђивања граница потпрограма, што у мери у којој то ради коришћени алат представља релативно једноставан задатак.

На почетку прве фазе за сваку од процедура прикупљају се вредности метрика $C_i=<c_{i,1};c_{i,2};...;c_{i,M}>$, где је *i* ознака процедуре за коју се метрике прикупљају. Број метрика је *M* и њихове вредности су означене као $c_{i,j}$, где *j* означава метрику. Вредност метрике може да буде не само обичан број, већ и већа структура података, што је описано у Секцији 6.1.1.

У другом кораку прве фазе добијене вредности метрика процедуре *A* се пореде са вредностима метрика сваке од процедура B_i примењујући одговарајуће компараторе M_X (компаратори су описани у Секцији 6.1.3.). Компаратор прихвата вредности исте метрике упоређених процедура *A* и B_i. Резултат компаратора над вредностима метрике *k* процедура *A* и B_i, је парцијална мера сличности тих процедура у складу са коришћеном метриком и означена је са $PSV_{A,Bi}[k]$. Скуп свих парцијалних мера сличности за

процедуре A и B_i означен је са $PSV_{A,Bi}$. Као резултат другог корака прве фазе добијени су вектори парцијалних мера сличности процедуре A са процедурама B_i у ознаци $PSV_{A,Bi}$, где $i \in [1,N]$.



Слика 6.1 Структура основног дела предложеног приступа. Прва фаза рачуна парцијалне мере сличности, трансформише их, и на основу трансформисаних вредности рачуна тоталну меру сличности. Друга фаза издваја *n* процедура најсличнијих траженој процедури.

У трећем кораку прве фазе на сваку од добијених парцијалних мера сличности примењује се одабрани трансформатор R_X (трансформатори су описани у Секцији 6.1.3.). За сваку парцијалну меру сличности постоји посебно истрениран трансформатор. Сваки од трансформатора истрениран метриком *k* на улазу прихвата парцијалну меру сличности две процедуре израчунату на основу одговарајуће метрике *k*, означену као $PSV_{A,Bi}[k]$. Као резултат се добија прилагођена мера парцијалне сличности по основу метрике *k* означена као $aPSV_{A,Bi}[k]$.

У четвртом, последњем кораку прве фазе рачуна се тотална мера сличности. За рачунање се користи одабрана формула *Fx* (формуле су описане у Секцији 6.1.4.) која се примењује на прилагођени вектор парцијалних мера сличности две процедуре $aPSV_{A,Bi}$. Резултат је низ мера тоталне сличности процедуре *A* са сваком од процедура B_i у ознаци $PS_{A,Bi}$, где се *i* креће у опсегу *[1..N]*.

Друга фаза алгоритма се састоји од издвајања *n* процедура најсличнијих траженој процедури. Вредност *n* се одређује у зависности од расположивог експертског времена и важности откривања употребе библиотеке. Мање вредности скраћују време потребно за мануелно поређење процедура, али је и вероватноћа проналаска тражених процедура мања (смањује се број процедура које ће бити пронађене). Повећањем вредности повећава се и вероватноћа да се нађе већи број тражених процедура, што захтева више времена за мануелно поређење због већег броја процедура.

6.1.1 Метрике

Софтверска метрика представља пребројиво својство програмског кода, и за потребе овог рада прикупља се на нивоу процедура. Својства процедура која се пребројавају могу да буду различите појаве у програмском коду попут појаве неких инструкција или појаве неких операнада. Пребројавањем једне појаве унутар процедуре добија се скаларна метрика чија вредност може да се запише у облику једног целог ненегативног броја. Уколико се ипак пребројава већи број различитих појава истог типа, тада се добија векторска метрика. Векторска метрика се репрезентује скупом уређених парова, где се сваки пар састоји од ознаке појаве и броја појављивања те појаве.

Обзиром да се вредност метрике добија пребројавањем одређених својстава и занемаривањем осталих, може се рећи да метрика представља апстракцију процедуре, и да је на тај начин поређење процедура апстраховано. Циљ апстраховања процедура метрикама јесте да се унутар изостављених делова нађе што већи број измена, да варијације услед различитих процеса превођења буду што мање у делу процедуре који се пребројава, као и да пребројане вредности у довољној мери могу да диференцирају посматране процедуре. Обзиром да свака од метрика само пребројава појаве, потпуно занемарујући семантику програма, практично ни једна метрика не задовољава сва три циља. Ни за једну метрику се чак не може рећи ни да задовољава у потпуности било који од три наведена циља. Уместо тога, свака од метрика тежи постављеним циљевима, а комбиновање

метрика треба да ублажи последице несавршености сваке од предложених метрика.

Други разлог зашто је практично немогуће направити идеалну метрику јесу измене које преводиоци уносе. Чак и да се уради детаљна анализа алгоритама и оптимизација које користе преводиоци, не може се очекивати да се неће појавити нови преводилац. Такође, не може се очекивати ни да наредне верзије постојећих преводилаца неће променити неки од постојећих алгоритама или додати неки нови алгоритам за превођење и оптимизовање кода. Стога у овом раду неће бити исцрпно анализирани сви коришћени алгоритми, нити све постојеће оптимизације преводилаца, већ ће за сваку од метрика кроз примере бити дата мотивација, као и кратак осврт на њене слабе тачке.

Метрике које се предлажу у овом приступу су делом инспирисане метрикама коришћеним у вишим програмским језицима, док је други део метрика осмишљен наменски за примену над асемблерским кодом. Одабране су метрике које су једноставне за мерење и чије имплементације имају малу временску сложеност. Стога све предложене метрике занемарују редослед инструкција, као и зависности између њих, чиме се занемарују многе промене које настану у процесу превођења. У исто време се губи и значајна количина информација које идентификују процедуре.

Преглед предложених метрика је дат у Табела 6.1. Метрике су класификоване према типу вредности, врсти мере коју користе и типу тока на који се односе. Према типу вредности, метрика може да буде скаларна или

Табела 6.1 Преглед предложених метрика.

Опис метрике	Акроним	Тип вредности	Тип мере	Тип тока
Број инструкција	AIN	S	A	-
Број скокова	ABN	S	A	C
Број позива	ACN	S	A	C
Број петљи	APN	S	A	C
Број аритметичких инструкција	AAN	S	A	D
Број логичких инструкција	ALN	S	A	D
Број инструкција за пренос података	ATN	S	A	D
Фреквенција скокова	ABF	S	N	C
Фреквенција позива	ACF	S	N	C
Фреквенција петљи	APF	S	N	C
Фреквенција аритметичких инструкција	AAF	S	N	D
Фреквенција логичких инструкција	ALF	S	N	D
Фреквенција инструкција за пренос података	ATF	S	N	D
Бројеви појављивања за сваку инструкцију засебно	EIN	V	A	-
Фреквенције појављивања за сваку инструкцију засебно	EIF	V	N	-
Бројеви доскока за сваку одредишну адресу засебно	EBN	V	A	C
Фреквенције доскока за сваку одредишну адресу засебно	EBF	V	N	C
Бројеви позива за сваку позивану процедуру засебно	ECN	V	A	C
Фреквенције позива за сваку позивану процедуру засебно	ECF	V	N	C

векторска. Према врсти мере метрика може да буде апсолутна или нормализована. Према типу тока на који се односи, метрика може да буде усмерена ка току контроле, току података или ниједан од претходна два. Предложене метрике и мотивација за увођење описани су у наставку, а на Слика 6.2 приказани су примери мерења вредности предложених метрика.

Најједноставнија метрика, потекла из виших програмских језика, јесте бројање линија програмског кода (*AIN* у Табела 6.1). У контексту овог рада броје се машинске наредбе од којих се састоји посматрана процедура. Добра страна ове метрике је што за процедуре које потичу од истог изворног кода даје висок ниво парцијалне мере сличности у случајевима када се процедуре разликују због различитог редоследа инструкција, односно када се неке инструкције замене функционално еквивалентним инструкцијама. Ова метрика даје висок ниво сличности и када се број инструкција релативно мало промени, што се дешава у различитим случајевима. Примери таквих случајева су употреба различитих алгоритама за алокацију ресурса и генерисање различитог кода за приступ параметрима који се прослеђују процедури приликом позива. Такође добра страна је што за процедуре које потичу од различитих изворних кодова и које се значајно разликују по величини даје низак ниво сличности. Лоша страна ове метрике је што за процедуре које потичу од различитих изворних кодова и које имају приближно исти број инструкција погрешно даје високу вредност парцијалне мере сличности. Такође лоша страна ове парцијалне мере сличности показује се у случајевима када при поређењу процедура које

Изворни код	Неоптимизовано	a) Ремапирање и промена редоследа		b) Одмотавање петљи		c) Супституција инструкција	
		Клонови типа 2 и 3		Клон типа 4		Клон типа 3	

`int n;`
`void sumProd()`
`{`
`int sum=0;`
`int prod =1;`
`for (int i=1;`
`i<=n; i++)`
`{`
`prod=prod*2;`
`sum=sum + i;`
 	
`foo(sum,prod);`
`}`
`}`
	`push {lr}`
	`mov r1, 1`
	`xor r0, r0, r0`
	`mov r2, 1`
	`loop_0x321:`
	`mul r1, r1, 2`
	`add r2, r2, r0`
	`bl sub_0x01234`
	`add r0, r0, 1`
	`ldr r3,=var_0x2`
	`ldr r3, [r3]`
	`cmp r0, r3`
	`blt loop_0x321`
	`pop {pc}`
		`push {lr}`
		`mov r1, 1`
		`xor r0, r0, r0`
		`mov r2, 1`
		`loop_0x333:`
		`add r2, r2, r0`
		`mul r1, r1, 2`
		`bl sub_0x555`
		`add r0, r0, 1`
		`ldr r3,=var_0x4`
		`ldr r3, [r3]`
		`cmp r0, r3`
		`blt loop_0x333`
		`pop {pc}`
				`push {lr}`
				`mov r1, 1`
				`xor r0, r0, r0`
				`mov r2, 1`
				`loop:`
				`mul r1, r1, 2`
				`add r2, r2, r0`
				`bl sub_0x01234`
				`add r0, r0, 1`
				`ldr r3, =var_0x2`
				`ldr r3, [r3]`
				`cmp r0, r3`
				`bge endOfLoop`
				`mul r1, r1, 2`
				`add r2, r2, r0`
				`bl sub_0x01234`
				`add r0, r0, 1`
				`ldr r3, =var_0x2`
				`ldr r3, [r3]`
				`cmp r0, r3`
				`blt loop`
				`endOfLoop:`
				`pop {pc}`
						`push {lr}`
						`mov r1, 1`
						`xor r0, r0, r0`
						`mov r2, 1`
						`loop_0x321:`
						`lsl r1, r1, 1`
						`add r2, r2, r0`
						`bl sub_0x01234`
						`add r0, r0, 1`
						`ldr r3,=var_0x2`
						`ldr r3, [r3]`
						`cmp r0, r3`
						`blt loop_0x321`
						`pop {pc}`	

PSV акроним	**Метрика**	**Метрика и PSV вредност**		**Метрика и PSV вредност**		**Метрика и PSV вредност**	
SAIN	13	13	1	21	0.62	13	1
SABN	1	1	1	2	0.5	1	1
SACN	1	1	1	2	0.5	1	1
SAPN	1	1	1	1	1	1	1
SAAN	4	4	1	8	0.5	4	1
SALN	1	1	1	1	1	1	1
SATN	6	6	1	8	0.75	6	1
SABF	1/13	1/13	1	2/21	0.81	1/13	1
SACF	1/13	1/13	1	2/21	0.81	1/13	1
SAPF	1/13	1/13	1	1/21	0.62	1/13	1
SAAF	4/13	4/13	1	8/21	0.81	4/13	1
SALF	1/13	1/13	1	1/21	0.62	1/13	1
SATF	6/13	6/13	1	8/21	0.83	6/13	1
LEIN	{push:1,mov:2, xor:1,mul:1,add:2, bl:1,ldr:2,cmp:1, b:1,pop:1}	{push:1,mov:2, xor:1,mul:1,add:2, bl:1,ldr:2,cmp:1, b:1,pop:1}	1	{push:1,mov:2, xor:1,mul:2,add:4, bl:2,ldr:4,cmp:2, b:2,pop:1}	0.70	{push:1,mov:2, xor:1,lsl:1,add:2, bl:1,ldr:2,cmp:1, b:1,pop:1}	0.81
REIN			1		0.70		1
LEIF	{push:1/13, mov:2/13,xor:1/13, mul:1/13,add:2/13, bl:1/13,ldr:2/13, cmp:1/13,b:1/13, pop:1/13}	{push:1/13, mov:2/13,xor:1/13, mul:1/13,add:2/13, bl:1/13,ldr:2/13, cmp:1/13,b:1/13, pop:1/13}	1	{push:1/21, mov:2/21,xor:1/21, mul:2/21,add:4/21, bl:2/21,ldr:4/21, cmp:2/21,b:2/21, pop:1/21}	0.73	{push:1/13, mov:2/13,xor:1/13, lsl:1/13,add:2/13, bl:1/13,ldr:2/13, cmp:1/13,b:1/13, pop:1/13}	0.81
REIF			1		0.73		1
REBN	{1}	{1}	1	{2}	0.5	{1}	1
REBF	{1/13}	{1/13}	1	{2/21}	0.81	{1/13}	1
RECN	{1}	{1}	1	{2}	0.5	{1}	1
RECF	{1/13}	{1/13}	1	{2/21}	0.81	{1/13}	1

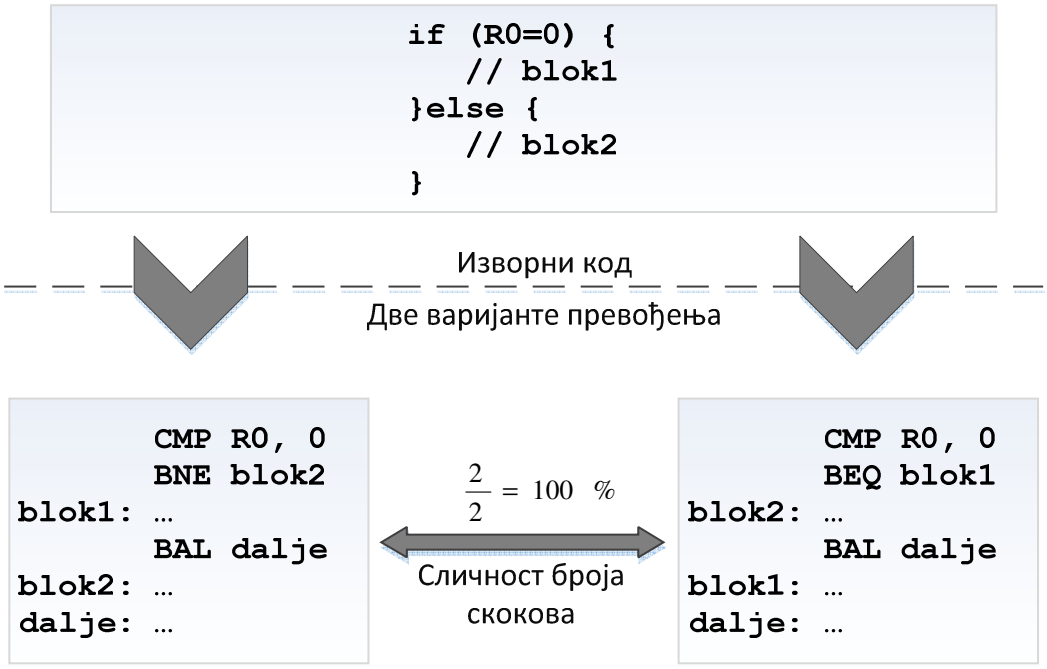
Слика 6.2 Примери рачунања метрика и парцијалних мера сличности на примерима различито преведене процедуре

потичу од истог изворног кода дође до веће разлике у броју инструкција, као што је у оптимизацијама одмотавања петље и уграђивања процедура на месту позива. Иако има и мане, у зависности од начина комбиновања парцијалних сличности, ова метрика може да умањи сличност процедура које се по величини значајно разликују од тражене чиме их помера ка крају листе. Ипак, може се десити да исто уради и са процедурама које потичу од истог изворног кода уколико је једна од упоређених процедура значајно измењена у процесу оптимизације.

Друга метрика, такође инспирисана метрикама за више програмске језике, пребројава инструкције скока за које се очекује да остану у приближно истом броју (*ABN* у Табела 6.1). Машинске инструкције скока настају од наредби за контролу тока програма и услова који се користе у тим наредбама. Неретко се дешава да одређене наредбе буду другачије преведене, али да то не утиче на број скокова. Пример такве наредбе дат је на Слика 6.3. У датом примеру ова метрика резултује у максималној парцијалној сличности упоређених процедура. Ипак, иста оптимизација изведена над *if* наредбом код које недостаје једна од условних грана може да резултује у промењеном броју скокова. Уколико недостаје блок 1, тада се у примеру на слици у првом случају може изоставити наредба *BAL dalje*. Аналогно важи и за други случај када је блок 2 празан.

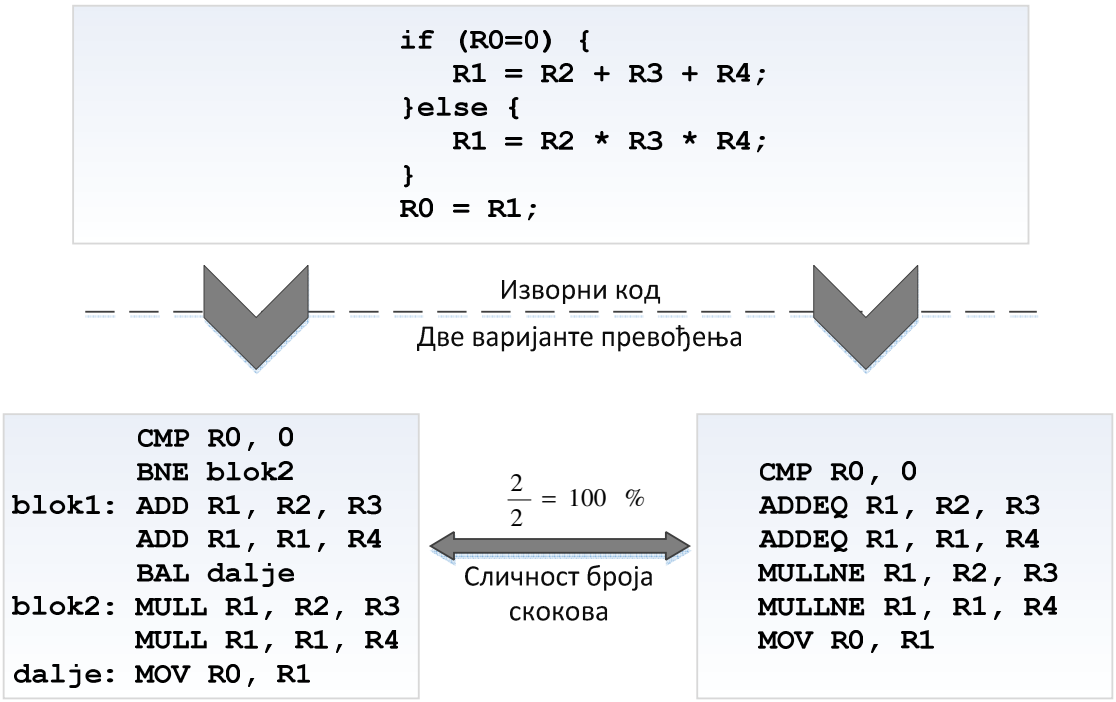
Други проблем који може да настане је код архитектура које омогућавају условно извршавање инструкција различитих од инструкција за

контролу тока програма. Тада преводилац има могућност избора да ли наредбу *if* преводи користећи условне скокове или користи условно извршавање наредби, као што је приказано у примеру на Слика 6.4. Унутар блока може да буде само једна наредба, али исто тако може да се појави и већи број наредби. Ублажавање или чак потпуна елиминација описаног проблема постиже се тако што се условно извршаване инструкције броје и као инструкције скока. Свака инструкција са постфиксом за условно извршавање којој не претходи инструкција са истим постфиксом се рачуна као једна инструкција условног скока. Тако је постигнуто да се низ инструкција са истим постфиксом рачуна као блок који се условно прескаче једном инструкцијом скока. Могућа је и комбинација стандардних условних



Слика 6.3 Пример различитог превођења *if* наредбе. Друга инструкција има другачији услов под којиме се извршава, па су и делови кода из *then* и *else* грана заменили места.

скокова и условно извршаваних инструкција, што може да поремети добијену вредност посматране метрике. Случај је делимично разрешен тако што се инструкција условног скока која се појави у секвенци инструкција са истим условним суфиксом ипак рачуна као додатна инструкција скока иако има исти суфикс.



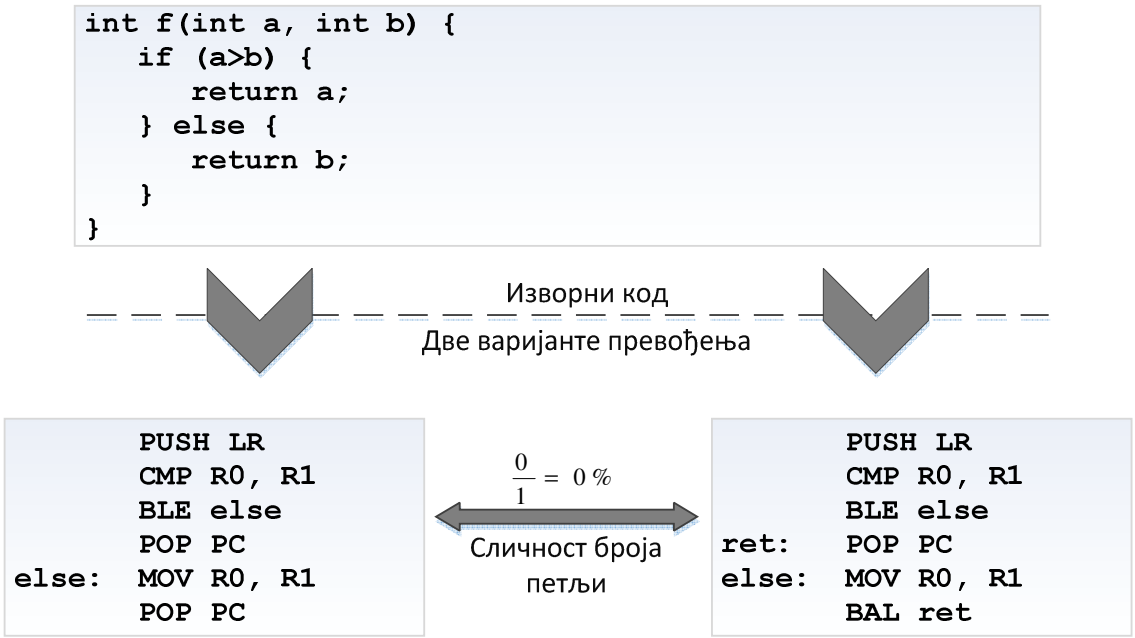
Слика 6.4 Пример различитог превођења *if* наредбе у случају *ARM* архитектуре код које све инструкције могу да се извршавају условно. Са леве стране је приказан класичан код са скоковима, док је са десне стране приказан код у којем нема скокова, већ се наредбе из *then* и *else* грана извршавају под одговарајућим условима.

Трећа метрика, такође инспирисана метрикама намењеним вишим програмским језицима, пребројава позиве потпрограма (*ACN* у Табела 6.1).

Ова метрика је отпорна на велики број оптимизација, попут оних које различито распоређују инструкције, мењају инструкције еквивалентним, различитим редоследом рачунају делове израза и слично. Мана ове метрике је што број позива може да се промени када дође до уграђивања одређених процедура на месту позива, као и у случају када дође до одмотавања петљи и прераспоређивања инструкција по итерацијама петље. До промене броја позива може да дође и када се оптимизује рачунање истог подизраза, уколико се у подизразу позива нека процедура. Такође, лоша страна метрике је што процедуре са малим бројем позива других процедура није могуће значајније разликовати јер се дешава да процедуре имају исти мали број позива, што даје високу меру парцијалне сличности иако упоређене процедуре потичу од два различита дела изворног кода и позивају различите процедуре. Штавише, метрика не прави разлику између процедура које велики број пута позивају једну процедуру и оних које мањи број пута позивају више различитих процедура, а које у збиру имају исти број позива. Описане предности и мане сугеришу да метрика у великом броју случајева може да помогне тако што ће да смањи вредност парцијалне мере сличности међу процедурама које имају значајно различит број позива других процедура и тако их дискриминисати у односу на процедуре са истим и сличним бројем позива.

Четврта метрика, такође инспирисана метрикама намењеним вишим програмским језицима, пребројава петље (*APN* у Табела 6.1). Обзиром да анализа кода за потребе откривања петљи може да буде сложена, а да је један од циљева да се метрике прикупљају једноставном обрадом текста са

малом временском сложеношћу, у овом приступу је коришћена апроксимација броја петљи. Уместо анализе кода, сваки скок чија је одредишна адреса мања од адресе инструкције скока урачунат је и као једна петља. Апроксимација није потпуно коректна уколико у процесу оптимизације дође до таквог размештања кода тако да се на неки блок враћа уназад, али да се након извршавања тог блока не долази до инструкције за скок уназад. Описана аномалија приказана је на Слика 6.5. Приказана процедура је на два места имала наредбу *return*. Уместо да се оба пута генерише исти код који служи за повратак из процедуре, преводилац је први пут генерисао код, док је за другу инструкцију изгенерисао скок уназад.



Слика 6.5 Пример аномалије апроксимативног пребројавања петљи. Иако изворни код не садржи ниједну петљу, у десном примеру превођења појавио се скок уназад који се апроксимативним бројањем рачуна као једна петља.

Иако се очекује да број петљи остане исти, поред аномалија које могу да се појаве услед апроксимативног пребројавања петљи, ова метрика не даје добре резултате у случају неколико оптимизација. Примери таквих оптимизација су спајање више петљи у једну, или раздвајање петље у више мањих петљи. Број петљи може да се промени и у случају када дође до уграђивања тела процедуре на месту позива, али је тај случај мање вероватан с обзиром да се уграђивање на место позива обично ради само за веома мале потпрограме у којима се не појављују петље.

Пета метрика пребројава аритметичке инструкције (*AAN* у Табела 6.1). Мотивација за увођење ове метрике је претпоставка да је за рачунање израза потребан исти или сличан број аритметичких операција, док редослед операција може да се промени. Ипак, постоје и друге оптимизације, попут елиминације вишеструког рачунања истих подизраза, одмотавања петљи, дељења и спајања петљи, као и уграђивања процедура на месту позива, које могу да промене број аритметичких инструкција. Такође, може се десити да се за извођење неких аритметичких операција користе неке друге операције које нису категорисане првенствено као аритметичке. Класичан пример су операције множења и дељења целих бројева степеном броја *2*. Множење целих бројева представљених у другом комплементу се изводи померањем садржаја броја улево за онолико места колики је експонент, дописујући на крај нуле. Дељење се ради аналогно, померањем удесно, дописујући знак броја. У ARM архитектури постоји и могућност да се померање изведе над једним од операнада у аритметичким инструкцијама и инструкцијама за

пренос података, па се приликом пребројавања и померања операнда тумаче као једна додатна аритметичка инструкција.

Шеста метрика је слична претходној, али за разлику од ње броји појављивања инструкција за рад са битовима (*ALN* у Табела 6.1). Ту су укључене логичке инструкције и инструкције за обично и цикличко померање података. Броје се и оне инструкције које су већ раније урачунате као аритметичке инструкције. Мотивација за предлагање ове метрике и њене мане су слични као у случају аритметичких инструкција.

Седма метрика обухвата инструкције за пренос података (*ATN* у Табела 6.1). Метрика је инспирисана чињеницом да се подаци који се налазе у променљивим обрађују на исти начин, па се очекује да релативна разлика у броју инструкција за пренос података неће бити велика. Ипак, може се десити да након измена које преводиоци унесу две различите процедуре буду проглашене сличнијим него две процедуре које потичу од истог изворног кода, све у зависности од начина на који преводиоци користе аутоматске променљиве, генеришу код за приступ локалним и глобалним променљивим, као и алгоритама које користе за алокацију ресурса за променљиве. Још једна аномалија при мерењу вредности ове метрике се јавља у случају ARM архитектуре због инструкција за дохватање из меморије и смештање у меморију већег броја података. Како би се превазишле разлике када се у једном процесу превођења појави једна инструкција за вишеструки пренос, док се у другом појави већи број инструкција за појединачни пренос,

инструкција за вишеструки пренос се броји онолико пута колико се података преноси.

Осма метрика пребројава појављивања за сваку инструкцију која се појављује у посматраној процедури (*EIN* у Табела 6.1). Вредност метрике се памти у вектору који садржи парове назива инструкција и бројева појављивања. У вектору се за сваки мнемоник памти колико пута се појавио. При томе се памте само основни мнемоници, без префикса и суфика. Како би се ублажио ефекат супституције једне инструкције другом, у случају када се наиђе на неку инструкцију која може да замени неку другу инструкцију, тада се рачуна да се појавила и та друга инструкција. Пример пребројавања инструкција је дат на Слика 6.2. Добра страна метрике је што је отпорна на оптимизације које раде просто премештање кода или замену једне инструкције еквивалентном. Такође, за разлику од претходних метрика, и за релативно мале процедуре ова метрика може да има велики број различитих вредности, па је за очекивати да добро диференцира упоређене процедуре. Ипак, нема гаранције да се неће десити да две процедуре које обављају потпуно различиту функционалност користе исте инструкције и да по овој метрици буду проглашене потпуно сличним. Такође, постоје и оптимизације које могу да промене скуп инструкција који се користи, као што су трансформације петљи и уграђивање тела процедуре на место позива.

Девета метрика броји доскоке на појединачне адресе (*EBN* у Табела 6.1). Вредност метрике се памти у низу уређених парова адреса и бројева. Сваки пар садржи адресу на коју се доскаче и број доскока на ту адресу.

Метрика је предложена као алтернатива метрици која броји укупан број скокова, с тим да се осим укупног броја скокова задрже и додатне информације о тим скоковима, попут адреса доскока. Предности и мане су сличне као и у случају метрике која броји укупан број скокова, с тим што се од ове метрике очекује да мало боље диференцира процедуре, али исто тако да буде мање отпорна на оптимизације контролних структура.

Десета метрика пребројава позиве појединачних процедура (*ECN* у Табела 6.1). Вредност метрика се памти као низ уређених парова адреса процедура и бројева позива. Сваки пар садржи адресу једне позиване процедуре и број позива те процедуре. Ова метрика је инспирисана чињеницом да уколико се оптимизацијом не уклони или не дода неки део кода, број позива процедура се не мења. Метрика није отпорна на уграђивање процедура на месту позива јер на тај начин нестаје позив, па се и мера парцијалне сличности смањује. Метрика такође није отпорна на оптимизације које као резултат дају нове делове кода или одбацују постојеће делове, попут одмотавања петљи и елиминације мртвог кода.

Све поменуте метрике су базиране на апсолутним бројевима појављивања одговарајућих појава. Као што је међу манама метрика наведено, ниједна од тих метрика није била отпорна на неке од оптимизација попут одмотавања петљи. Како би се делимично ублажио ефекат репликације кода, у случајевима када део кода који се реплицира представља већи део процедуре, за сваку од метрика осим прве предлаже се рачунање и нормализоване метрике. За скаларне метрике нормализација се ради

дељењем вредности ненормализоване метрике бројем инструкција посматране процедуре. За векторске метрике поступак је сличан, с тим да је бројач у свим елементима вектора потребно поделити бројем инструкција посматране процедуре. На описани начин се добијају метрике приказане у Табела 6.1, чији се акроними завршавају словом F.

6.1.2 Компаратори

Компаратор је формула у којој учествују вредности неке од метрика две упоређене процедуре. Као резултат, компаратор даје једну вредност која представља парцијалну меру сличности у складу са посматраном метриком. Резултат сваког од компаратора је у опсегу *[0,1]*, чиме се омогућава да парцијалне сличности по свим метрикама буду равноправно посматране. При томе, вредност 0 означава минималну сличност упоређених процедура, док вредност 1 означава максималну сличности. За поређење скаларних метрика у о овом раду се предлаже један компаратор, а за векторске метрике се предлажу два компаратора.

Компаратор за скаларне метрике приказан је у Једнакости (1). Парцијална сличност по овом компаратору се рачуна тако што се од две процедуре узму вредности скаларне метрике за коју се рачуна парцијална мера сличности и потом мања вредност подели већом. У случају да су обе вредности једнаке нули, усвојено је да вредност парцијалне сличности буде *0.5*. Ова вредност је изабрана како би се избегла ситуација да две процедуре буду проглашене потпуно сличним зато што ниједна нема одређену особину,

а да при томе не дође до значајног смањења посматране парцијалне мере сличности.

$$M_{sc}(c_{A,k}, c_{B_i,k}) = \frac{\min(c_{A,k}, c_{B_i,k})}{\max(c_{A,k}, c_{B_i,k})} \quad (1)$$

$$m = |c_{A,k}|;\ n = |c_{B_i,k}|;\ u = |U|$$

$$M_{lvc}(c_{A,k}, c_{B_i,k}) = \frac{\sum_{j=1}^{u} \frac{\min(c_{A,k}[j], c_{B_i,k}[j])}{\max(c_{A,k}[j], c_{B_i,k}[j])}}{u} \quad (2)$$

$$M_{rvc}(c_{A,k}, c_{B_i,k}) = \frac{\sum_{j=1}^{\min(m,n)} \frac{\min\left(\text{rank}(c_{A,k}, j), \text{rank}(c_{B_i,k}, j)\right)}{\max\left(\text{rank}(c_{A,k}, j), \text{rank}(c_{B_i,k}, j)\right)}}{\max(m, n)} \quad (3)$$

Други компаратор, приказан у Једнакости (2), намењен је за примену над векторским метрикама које у себи имају ознаке које једнозначно идентификују појаве у процедури. То својство задовољава само осма метрика која пребројава појављивања сваке од инструкција која се појављује у процедури. Преостале две векторске метрике нису погодне јер се као ознаке користе адресе. Разлог томе је чињеница да исти део кода у два различита процеса превођења може да буде смештен на две различите адресе, па адреса на којој је код смештен не може да идентификује посматрани код. Први корак је одређивање скупа различитих ознака које се појављују у векторима метрика упоређених процедура. Потом се за сваку ознаку из формираног скупа израчуна однос мањег и већег броја појављивања ознаке у упоређеним процедурама, а затим се израчуна аритметичка средина израчунатих односа. Том приликом није могуће дељење нулом јер се посматрају само ознаке које

су се појавиле бар једном у бар једној од упоређених процедура, па већи број појављивања инструкције не може бити 0.

Табела 6.2 Преглед *PSV* елемената који се добијају применом компаратора на предложене метрике (S - компаратор за скаларне метрике, L - компаратор за векторске метрике заснован на ознакама, R - компаратор за векторске метрике заснован на рангирању вредности).

	Скаларне	**Векторске**	
	M_{sm}	**M_{lvm}**	**M_{rvm}**
Апсолутне	SAIN, SABN, SACN, SAPN, SAAN, SALN, SATN	LEIN	REIN,REBN,RECN
Нормализоване	SABF, SACF, SAPF, SAAF, SALF, SATF	LEIF	REIF, REBF, RECF

Трећи компаратор је такође намењен за примену над векторским метрикама и може се применити и у случајевима када ознаке не представљају јединствене идентификаторе појава. Компаратор је приказан у Једнакости (3). Као први корак поређења, вектори метрика упоређених процедура се уређују у нерастућем поретку бројева појављивања. У следећем кораку, на крај краћег вектора се додају елементи са бројем појављивања једнаким нули, док се дужине упоређених вектора не изједначе. Потом се рачунају односи бројева појављивања који се налазе на истој позицији у векторима и то тако да се мања вредност дели већом, након чега се рачуна

аритметичка средина израчунатих односа. Овакав компаратор је могуће применити на све *3* врсте векторских метрика.

Као резултат примене компаратора добијају се парцијалне мере сличности чији акроними су приказани у Табела 6.2. За скаларне метрике акроним је добијен додавањем слова S на почетак акронима метрике. За векторске метрике на почетак акронима је додато слово L у случају када се користи други компаратор, док се додаје слово R у случају употребе трећег компаратора.

6.1.3 Трансформатори

Вредности парцијалне мере сличности у одређеној мери говоре о сличности упоређених процедура. Анализом вредности парцијалних мера сличности на посебном скупу података за учење могуће је уочити да се приликом поређења процедура које потичу од истог изворног кода неке вредности јављају чешће, а неке ређе. На основу такве анализе могуће је трансформисати вредности парцијалних мера сличности како би се вредностима које се чешће појављују дао већи значај.

Трансформатори се базирају на претходном знању које се добија из емпиријских дистрибуција вредности парцијалних мера сличности процедура. За потребе дефинисања трансформатора уводе се функције $P_i(x)$ и $N_i(x)$. Прва представља емпиријску дистрибуцију вероватноће за i-ти елемент вектора *PSV* који одговара пару процедура које потичу од истог изворног кода. Аналогно важи и за другу функцију, с тим да парови процедура потичу од различитих изворних кодова.

У овом раду предложено је шест трансформатора који су приказани Једнакостима (4)-(9). Први трансформатор, приказан у Једнакости (4), представља случај када се парцијална мера сличности уопште не трансформише, већ се необрађена користи у остатку алгоритма. Други трансформатор, приказан у Једнакости (5), мења вредност парцијалне мере сличности вероватноћом појављивања те вредности код процедура које потичу од истог изворног кода. Трећи трансформатор користи експоненцијалну функцију приказану у Једнакости (6). Функција је одабрана тако да се повећањем односа функција $P_i(x)$ и $N_i(x)$ повећава и вредност функције, али тако да вредност трансформатора никад не пређе преко један. Смањењем односа до нуле, долази се до минималне вредности функције која износи нула.

Четврти трансформатор, приказан у Једнакости (7) рачуна вероватноћу да се за дату вредност парцијалне мере сличности ради о процедурама које потичу од истог изворног кода. Пети и шести трансформатор су настали модификацијом четвртог. Пети трансформатор, приказан у Једнакости (8), је параметризован и користи параметар T који је у опсегу $(0,0.5]$. Овај трансформатор даје исте вредности као четврти трансформатор у случају када је оригинална вредност парцијалне мере сличности мања од вредности параметра T или већа од вредности $1-T$. У супротном, када је оригинална вредност између T и $1-T$, трансформатор враћа вредност 0.5. На тај начин се одређене вредности које нису ни довољно велике да сугеришу висок ниво сличности, ни довољно мале да сугеришу

различитост функција постављају на средишњу вредност и тиме се смањује њихов утицај на касније рачунање тоталне мере сличности.

Шести трансформатор је веома сличан петом и приказан је у Једнакости (9). Од петог трансформатора се разликује по томе што за вредности мање од *T* враћа вредност нула, док за вредности веће од *1-T* враћа вредност један. На тај начин се покушава додатно умањити утицај разлика које настају у процесу превођења, тако што се за процедуре које се по посматраној метрици разликују мање од задатог параметра *T* сматра да процедуре са вероватноћом *1* по посматраној метрици потичу од истог изворног кода, док се за сличности које су мање од *T* сматра да процедуре потичу од различитог кода.

$$R_{1,i}(x) = x \tag{4}$$

$$R_{2,i}(x) = P_i(x) \tag{5}$$

$$R_{3,i}(x) = 1 - 2^{-\frac{P_i(x)}{N_i(x)}} \tag{6}$$

$$R_{4,i}(x) = \frac{P_i(x)}{P_i(x) + N_i(x)} \tag{7}$$

$$R_{5,i}(x) = \begin{cases} R_{4,i}(x) & R_{4,i}(x) < T \\ 0.5 & T \le R_{4,i}(x) \le 1 - T \\ R_{4,i}(x) & R_{4,i}(x) > 1 - T \end{cases} \tag{8}$$

$$R_{6,i}(x) = \begin{cases} 0 & R_{4,i}(x) < T \\ 0.5 & T \le R_{4,i}(x) \le 1 - T \\ 1 & R_{4,i}(x) > 1 - T \end{cases} \tag{9}$$

6.1.4 Формуле

Након рачунања адаптираних вредности парцијалне мере сличности долази се до последњег корака прве фазе у којем се рачунају тоталне мере сличности упоређених процедура. За рачунање се користи формула која на

улазу прихвата два вектора са адаптираним вредностима парцијалних мера сличности и као резултат враћа један број у опсегу *[0,1]*. Формула може да буде било која функција која пресликава домен $[0,1]^n$ на интервал *[0,1]*. У овом раду испитано је неколико функција у својству формула предложеног приступа и те функције су приказане Једнакостима (10)-(16).

$$S \subseteq \{1, \dots, |PSV_{A,B_i}|\}$$

$$F_1(x) = \frac{\sum_{i \in S} R_{k,i}(PSV_{A,B_i}[i])}{n} \tag{10}$$

$$F_2(x) = \sqrt[n]{\prod_{i \in S} R_{k,i}(PSV_{A,B_i}[i])} \tag{11}$$

$$F_3(x) = \frac{n}{\sum_{i \in S} \frac{1}{R_{k,i}(PSV_{A,B_i}[i])}} \tag{12}$$

$$F_4(x) = \prod_{i \in S} R_{k,i}(PSV_{A,B_i}[i]) \tag{13}$$

$$F_5(x) = \sum_{i \in S} W_i \cdot R_{k,i}(PSV_{A,B_i}[i]) \tag{14}$$

$$F_6(x) = \prod_{i \in S} \left(R_{k,i}(PSV_{A,B_i}[i]) + W_i\right) \tag{15}$$

$$F_7(x) = \frac{1}{1 + e^{-\sum_{i \in S} W_i \cdot R_{k,i}(PSV_{A,B_i}[i])}} \tag{16}$$

Прве три формуле, описане Једнакостима (10)-(12), су аритметичка, геометријска и хармонијска средина. Четврта формула је обичан производ адаптираних вредности парцијалних мера сличности и приказана је у Једнакости (13). Наредне три формуле, описане једнакостима (14)-(16), су тежинска сума, производ и логистичка функција. Тежине се рачунају употребом регресионе анализе како би се минимизовала грешка на

примерима парова процедура за које је познато да ли потичу од истог изворног кода или не.

Поред описаних формула, могуће је применити и технике машинског учења и класификације. У овом раду су испитане машине са векторима подршке (енг. *Support Vector Machines - SVM*) [64], Наивни Бајес (енг. *Naive Bayes - NB*)[65] и к најближих суседа (енг. *k nearest neighbor - k-NN*)[66]. Све поменуте технике захтевају претходно знање како би се припремиле за примену на откривање сличних процедура, и временски су знатно захтевније у односу на претходне формуле које не захтевају тренирање на претходном знању.

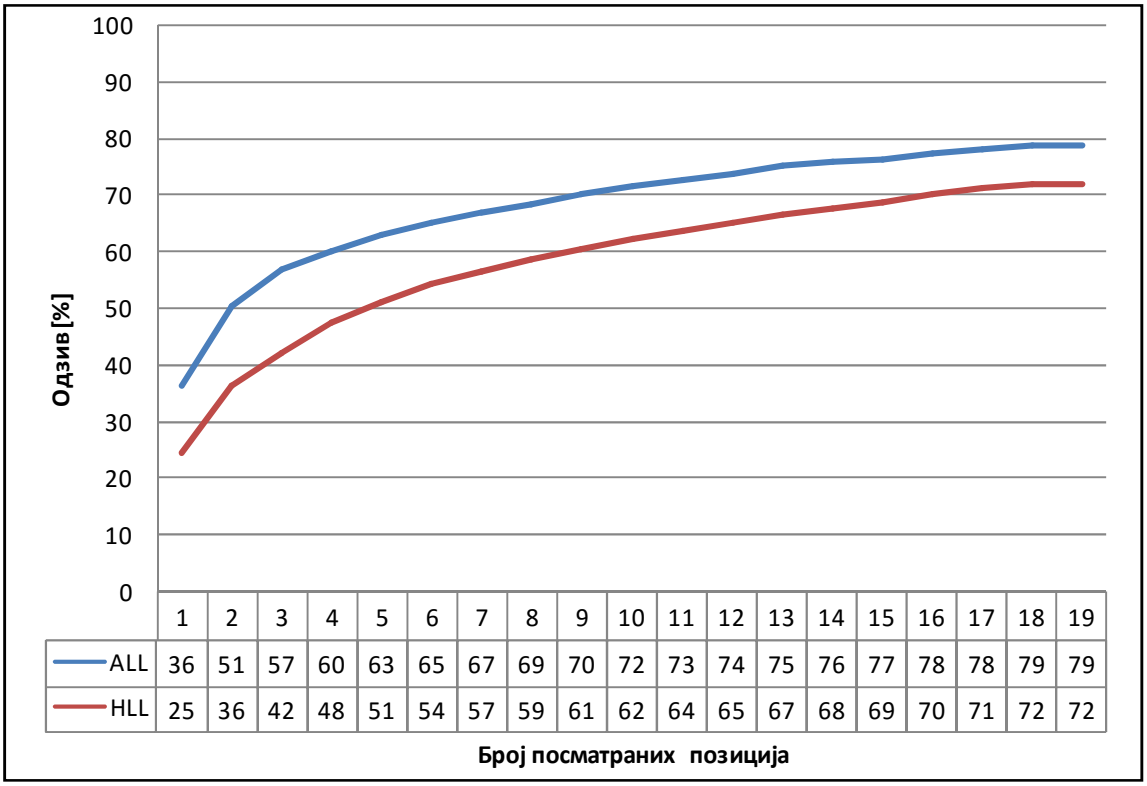
6.2 Евалуација основног приступа

У овој секцији је приказана евалуација претходно предложеног приступа. Евалуација је спроведена у окружењу које је описано у глави 4, а све у складу са ограничењима и циљевима датим у глави 3. Основни циљ евалуације је да одговори на прва три истраживачка питања постављена у глави 3. Евалуација се састоји од четири експеримента који су описани у наредне четири секције.

6.2.1 Експеримент 1.1

Циљ првог експеримента је да одговори на прво истраживачко питање. На питање да ли нове метрике доприносе повећању одзива предложеног приступа је одговорено мерењем доприноса који имају новопредложене технике. Мерење је урађено прво над контролном групом која је сачињена од комбинације само оних *PSV* елементи који потичу од

софтверских метрика намењених вишим програмским језицима (ознака *HLL* на Слика 6.6), а затим је урађено мерење и са експерименталном групом која укључује комбинације са свим *PSV* елементима (ознака *ALL* на Слика 6.6).



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
ALL	36	51	57	60	63	65	67	69	70	72	73	74	75	76	77	78	78	79	79
HLL	25	36	42	48	51	54	57	59	61	62	64	65	67	68	69	70	71	72	72

Слика 6.6 Одзив конфигурација *PSV* елемената контролне и експерименталне групе у зависности од броја посматраних позиција (*HLL* - највећи одзив постигнут конфигурацијама *PSV* елемената заснованих на метрикама адаптираним из виших програмских језика; *ALL* - највећи одзив постигнут конфигурацијама *PSV* елемената заснованих на адаптираним и новим метрикама).

Експеримент је спроведен над 3300 тестова. Тестови су описани у глави 4 и састоје се од једне процедуре која се тражи и скупа процедура једног бинарног програма у којем се тражи употреба библиотеке. Процедура

и програм у којем се процедура тражи су преведени независно, и при томе је сваки пут коришћен произвољан преводилац, произвољан ниво оптимизације и произвољан контекст превођења за тражену процедуру. Затим су све процедуре дисасемблиране, након чега је тражена процедура упоређена са сваком од процедура из посматраног бинарног програма. Поређење је рађено у свим конфигурацијама за контролну и експерименталну групу, након чега је израчуната сличност помоћу свих комбинација предложених трансформатора и формула.

Одзив предложеног приступа користећи контролну и експерименталну групу приказани су на Слика 6.6. Одзив је приказан у зависности од броја посматраних позиција, при чему се тај број креће у опсегу *[1,19]*. Црвена линија представља резултате који су добијени са конфигурацијама из контролне групе, док плава линија представља резултате добијене конфигурацијама из експерименталне групе. За сваку позицију приказани су резултати најбоље конфигурације, најбољег трансформатора и најбоље формуле, за сваку од две групе посебно. Највећи апсолутни раст одзива је ако се као успех посматра проналазак у прве две и прве три позиције и тада износи 15%. Ипак, највећи релативни пораст одзива је ако се као успех посматра проналазак на првој позицији и тада је релативни пораст одзива 44%.

6.2.2 Експеримент 1.2

Циљ другог експеримента је да одговори на друго истраживачко питање које разматра утицај преводиоца, нивоа оптимизације и окружења у

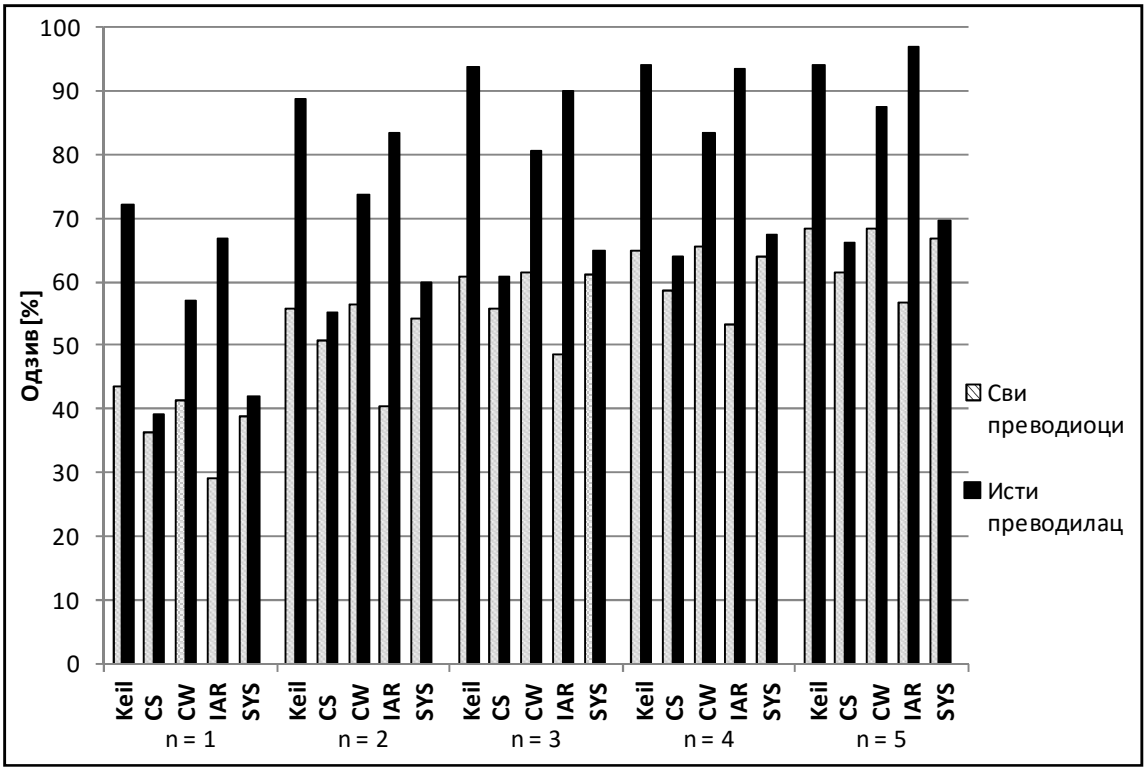
којем се преводи тражена процедура на одзив који предложени приступ постиже. На питање се одговара тако што се приликом акумулације резултата праве посебне суме по сваком од три наведена критеријума. Тако се добија одзив који предложени приступ постиже у зависности од тога који преводилац је коришћен за превођење тражене процедуре, који ниво оптимизације је коришћен и у којем окружењу је преведена тражена процедура.

Експеримент је спроведен над 3300 тестова. Тестови су у складу са описом датим у глави 4 и састоје се од једне процедуре која се тражи и скупа процедура једног бинарног програма у којем се тражи употреба библиотеке. Процедура и програм у којем се процедура тражи преведени су независно, и при томе је сваки пут коришћен произвољан преводилац, произвољан ниво оптимизација и произвољан контекст превођења за тражену процедуру. Затим су све процедуре дисасемблиране, након чега је тражена процедура упоређена са сваком од процедура из посматраног бинарног програма. Поређење је рађено у свим конфигурацијама, при чему је сличност израчуната помоћу свих комбинација предложених трансформатора и формула. За сваку позицију је узета најбоља постигнута вредност.

У циљу одговарања на прво потпитање неопходно је испитати утицај преводиоца који се користи за превођење тражене процедуре на одзив предложеног приступа. Стога су резултати груписани у пет група, тако да свака група одговара једном од коришћених преводилаца. У оквиру сваке групе резултати су подељени у две подгрупе. У првој подгрупи су резултати

за тестове у којима је бинарни програм у којем се тражи процедура преведен произвољним преводиоцем, при чему се може десити да то буде исти преводилац којим је преведена тражена процедура. Ова група је на Слика 6.7 приказана светлим стубићима. Осим IAR преводиоца, одзиви свих осталих преводилаца се мењају на сличан начин и крећу се од 36% у случају CodeSourcery преводиоца и посматрања само прве позиције, до 68% у случају CrossWorks преводиоца и посматрања првих пет позиција. У просеку, резултати за IAR преводилац су 27% мањи ако се посматра само прва позиција, ондносно 16% мањи ако се посматра првих пет позиција. У другој подгрупи су резултати за тестове у којима је бинарни програм добијен помоћу истог преводиоца као и тражена процедура. Ова подгрупа је на Слика 6.7 приказана тамним стубићима. Најбољи резултати у овој подгрупи се добијају за Kail преводилац и крећу се у опсегу од 72% ако се посматра само прва позиција, до 94% ако се посматра првих пет позиција. Резултати који се добијају за IAR преводилац су незнатно лошији осим када се посматра првих пет позиција, када IAR постиже највећу вредност одзива. Резултати за остала три преводиоца се мењају на сличан начин и крећу се у опсегу од 39% ако се посматра само прва позиција, до 87% ако се посматра првих пет позиција. Приказани резултати такође показују да уколико се приликом превођења тражене процедуре користи исти преводилац којим је преведен посматрани бинарни програм, одзив расте од 1.07 до 2.28 пута ако се посматра само прва позиција, односно од 1.03 до 1.71 ако се посматра првих пет позиција.

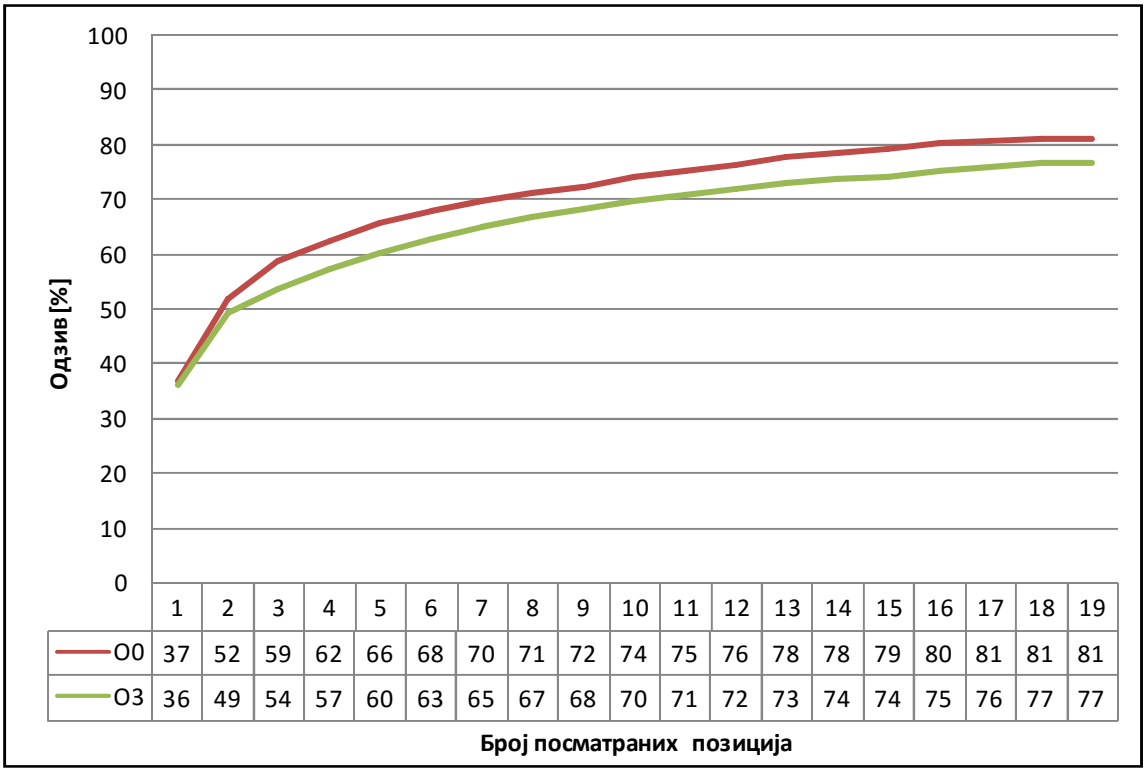
У циљу одговарања на друго потпитање неопходно је испитати утицај нивоа оптимизације који је коришћен приликом превођења тражене



Слика 6.7 Одзив предложеног приступа у зависности од коришћеног преводиоца у функцији броја посматраних позиција (Сви преводиоци - резултати у случају када је бинарни програм у којем се тражи процедура преведен произвољним преводиоцем; Исти преводилац - резултати у случају када су тражена процедура и бинарни програм преведени истим преводиоцем).

процедуре на одзив који постиже предложени приступ. Стога су резултати груписани у две групе, као што је приказано на Слика 6.8. Прва група резултата добијена је од тестова у којима је при превођењу тражене процедуре коришћен O0 ниво оптимизације, док друга група резултата одговара тестовима у којима је за исту намену коришћен O3 ниво оптимизације. Незнатно бољи одзив постиже се при употреби O0

оптимизације и износи 37% ако се посматра само прва позиција, односно 81% ако се посматра првих 19 позиција. Употребом нижег нивоа оптимизације постигнут је 1.03 пута бољи одзив ако се посматра првих 19 позиција, док је највеће релативно повећање одзива 1.1 пута ако се посматра првих пет позиција.

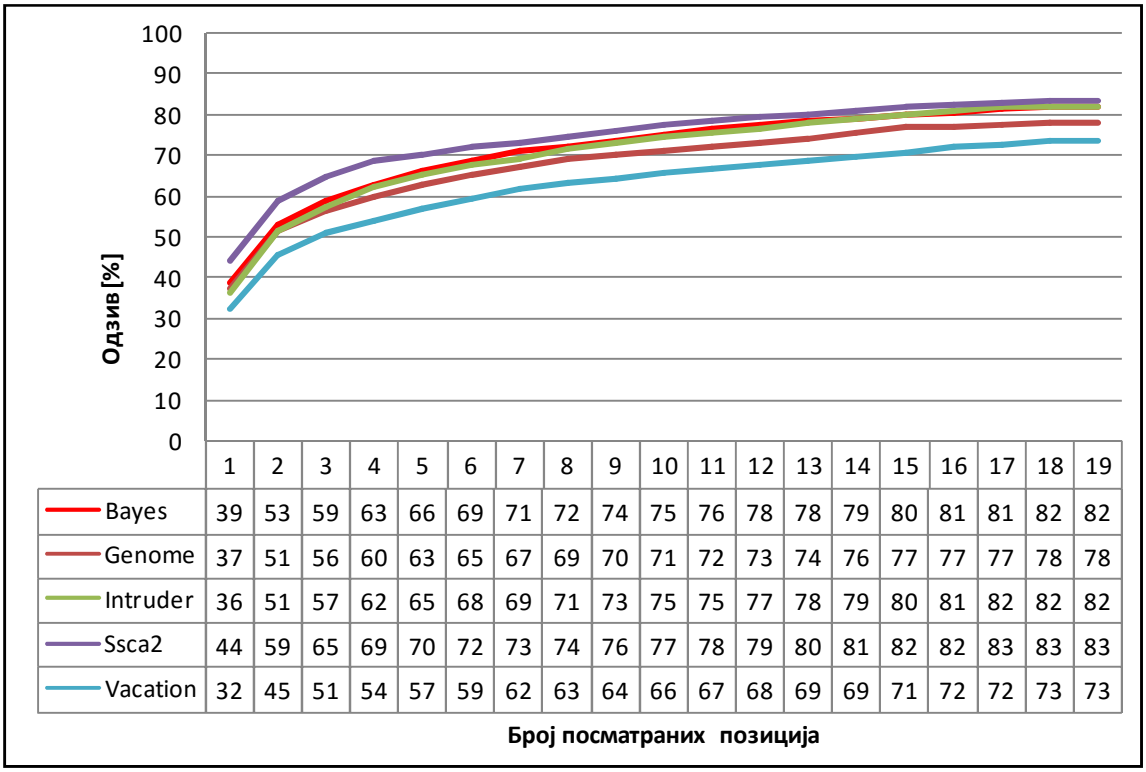


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
O0	37	52	59	62	66	68	70	71	72	74	75	76	78	78	79	80	81	81	81
O3	36	49	54	57	60	63	65	67	68	70	71	72	73	74	74	75	76	77	77

Слика 6.8 Одзив предложеног приступа у зависности од броја посматраних позиција, груписан по нивоима оптимизације који су коришћени при превођењу тражене процедуре.

У циљу одговарања на треће потпитање резултати су груписани у зависности од контекста у којем је преведена тражена процедура. Процедуре су превођене као део једног од следећих пет програма: Bayes, Genome, Intruder, Ssca2 и Vacation. Из резултата приказаних на Слика 6.9 се види да се

у случају посматрања прве позиције одзив предложеног приступа креће од 32% за Vacation до 44% за Ssca2. У случају посматрања првих *19* позиција одзив се креће од 73% за Vacation до 83% за Ssca2.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Bayes	39	53	59	63	66	69	71	72	74	75	76	78	78	79	80	81	81	82	82
Genome	37	51	56	60	63	65	67	69	70	71	72	73	74	76	77	77	77	78	78
Intruder	36	51	57	62	65	68	69	71	73	75	75	77	78	79	80	81	82	82	82
Ssca2	44	59	65	69	70	72	73	74	76	77	78	79	80	81	82	82	83	83	83
Vacation	32	45	51	54	57	59	62	63	64	66	67	68	69	69	71	72	72	73	73

Слика 6.9 Одзив предложеног приступа у зависности од броја посматраних позиција, груписан по програмима у оквиру којих је преведена тражена процедура.

6.2.3 Експеримент 1.3

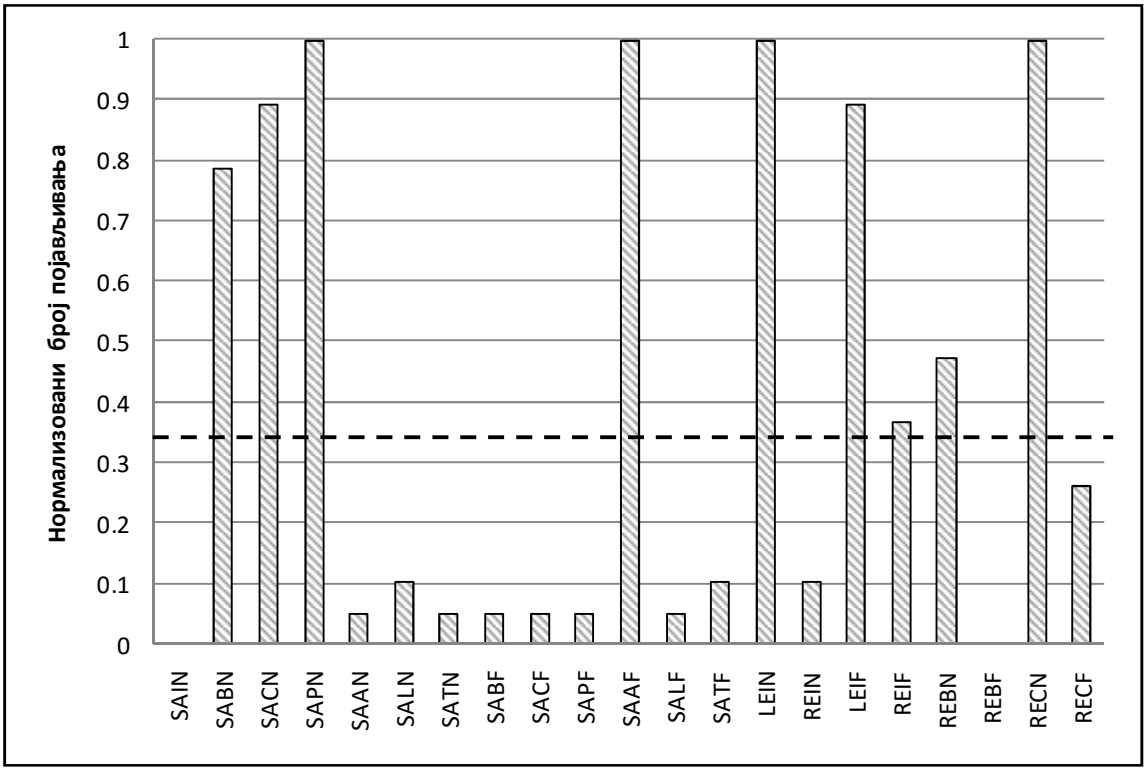
Циљ овог експеримента је да одговори на треће истраживачко питање које испитује да ли предложени приступ постиже боље резултате од постојећих решења. Пре поређења је неопходно одабрати конфигурацију предложеног приступа која ће бити коришћена. Потом је неопходно дефинисати поставке постојећих алата које ће бити коришћене за потребе

поређења. На крају, сви алати ће на улазу добити исти тест узорак, након чега ће резултати бити упоређени.

За потребе избора конфигурације предложеног приступа спроведен је тест који укључује све комбинације *PSV* елемената, трансформатора и формула, аналогно првом тесту. Први корак је избор *PSV* елемената. У другом кораку одабрани су трансформатори и формуле.

При избору *PSV* елемената посматране су конфигурације које постижу највећи одзиви када се посматра првих *n* процедура за вредности $n \in [1,19]$. При томе су за свако *n* коришћени трансформатор и формула са којима се добија најбољи одзив. У добијених *19* конфигурација пребројано је колико пута се користи сваки од *PSV* елемената. Нормализовани бројеви појављивања *PSV* елемената приказани су на Слика 6.10. За даљу употребу одабрани су елементи који се користе у више од трећине посматраних конфигурација, што је на Слика 6.10 назначено испрекиданом линијом. Одабрано је укупно девет *PSV* елемената. Три елемента су израчуната на основу метрика које су инспирисане метрикама намењеним вишим програмским језицима (метрике које пребројавају скокове, позиве и петље (*SABN*, *SACN* и *SAPN*)). Преосталих шест елемената је израчунато на основу предложених нових метрика употребом предложених компаратора и то су: фреквенција аритметичких инструкција (*SAAF*), означено поређени вектори који садрже бројеве и фреквенције појављивања за сваку од инструкција (*LEIN* и *LEIF*), рангирано поређени вектор који садржи фреквенције појављивања сваке од инструкција (*REIF*), рангирано поређени вектори који

садрже бројеве доскока на адресе које се појављују у инструкцијама скока (*REBN*) и рангирано поређени вектори који садрже бројеве позива позиваних потпрограма (*RECN*).

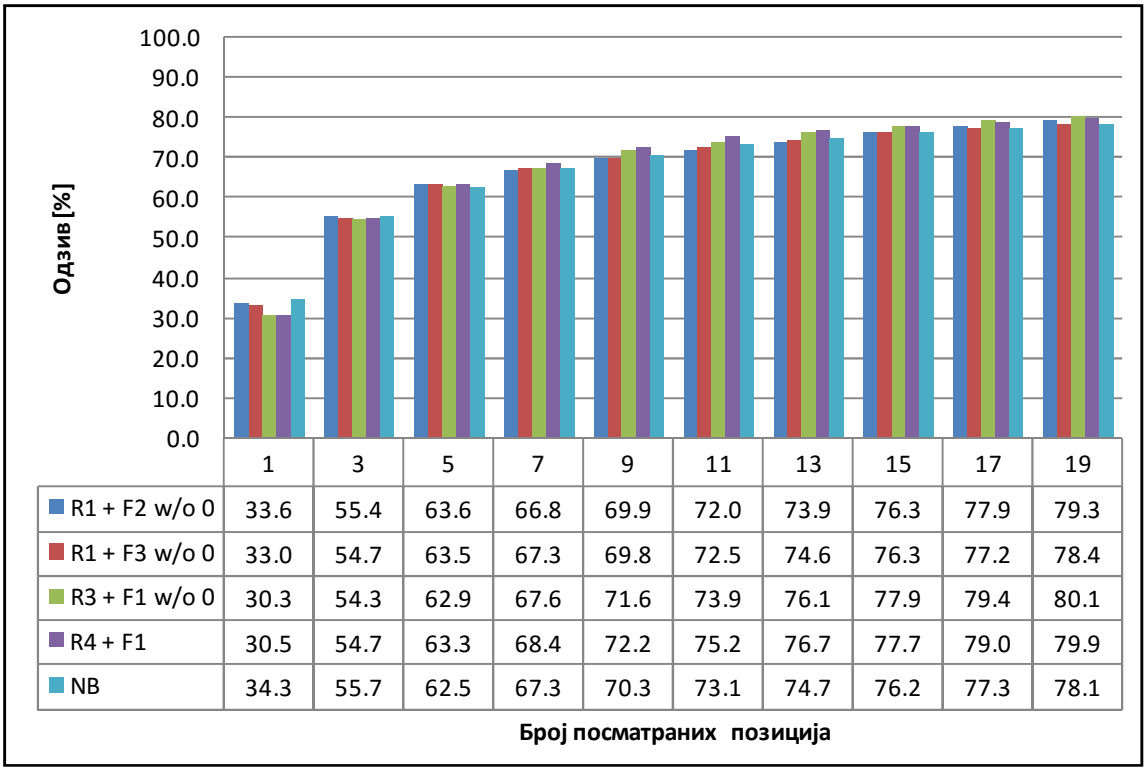


Слика 6.10 Нормализовани бројеви појављивања појединих *PSV* елемената у конфигурацијама које дају највећи одзив за сваки од бројева посматраних позиција.

Резултати коришћени за избор *PSV* елемената су такође коришћени и за избор филтера и формуле. Међу свим предложеним формулама најбољи одзив постижу аритметичка, геометријска и хармонијска средина (F_1, F_2 и F_3). Аритметичка средина у комбинацији са трансформатором R_4 постиже најбоље резултате ако се примени на све одабране елементе *PSV* или у комбинацији са трансформатором R_3 ако се примени на ненула вредности

одабраних *PSV* елемената. Геометријска и хармонијска средина постижу најбољи одзив у комбинацији без трансформатора (трансформатор који не мења вредност означен са R_1) ако се примене на ненула вредности одабраних *PSV* елемената. Поред предложених формула, испитана је могућност коришћења следеће три технике машинског учења: машине са векторима подршке, Наивни Бајес и к најближих суседа. На Слика 6.11 приказани су добијени одзиви за пет најбољих комбинација формула и трансформатора укључујући и поменуте три технике машинског учења. Одзиви су приказани у зависности од броја посматраних позиција *n*, при чему су за n узете непарне вредности из опсега *[1,19]*. Наивни Бајес, геометријска и хармонијска средина постижу највећи одзив за мали број посматраних позиција, док при посматрању више од шест позиција благу предност има аритметичка средина потпомогнута трансформаторима. Због трансформатора које користе, комбинације трансформатора и формула са аритметичком средином захтевају претходно знање. И технике машинског учења такође захтевају претходно знање, што се односи на приказану технику Наивни Бајес. Са друге стране, комбинације са хармонијском и геометријском средином не користе трансформатор па не захтевају претходно знање и тренирање. Због добрих резултата и непостојања потребе за претходним знањем, за поређење је одабрана геометријска средина.

Други део подешавања тиче се подскупа постојећих алата који ће бити коришћени за поређење. У овој фази, за поређење ће бити коришћени само постојећи алати који у неком облику могу бити примењени на решавање посматраног проблема и сви су из домена откривања софтверских клонова и



	1	3	5	7	9	11	13	15	17	19
R1 + F2 w/o 0	33.6	55.4	63.6	66.8	69.9	72.0	73.9	76.3	77.9	79.3
R1 + F3 w/o 0	33.0	54.7	63.5	67.3	69.8	72.5	74.6	76.3	77.2	78.4
R3 + F1 w/o 0	30.3	54.3	62.9	67.6	71.6	73.9	76.1	77.9	79.4	80.1
R4 + F1	30.5	54.7	63.3	68.4	72.2	75.2	76.7	77.7	79.0	79.9
NB	34.3	55.7	62.5	67.3	70.3	73.1	74.7	76.2	77.3	78.1

Слика 6.11 Одзиви најбољих пет комбинација филтера и формула укључујући и предложене три технике машинског учења, све у зависности од броја посматраних позиција (*R_1* + *F_2* w/o 0 - геометријска средина без трансформатора примењена на ненула *PSV* елементе, *R_1* + *F_3* w/o 0 - хармонијска средина без филтера примењена на ненула *PSV* елементе, *R_3* + *F_1* w/o 0 - аритметичка средина са експоненцијалним трансформатором примењена на ненула *PSV* елементе, *R_4* + *F_1* - аритметичка средина са трансформатором заснованим на нивоу поверења када се примени на све одабране *PSV* елементе, *NB* - Наивни Бајес примењен на све одабране *PSV* елементе).

откривања плагијаризма у студентским радовима. Преглед и најважније карактеристике ових алата дати су у Табела 6.3. Сви алати осим *Moss* алата на

улазу захтевају изворни код па је бинарне програме потребно декомпајлирати. *Moss* алат има могућност да на улазу прихвати и асемблерски код. Ипак алат не подржава *ARM* архитектуру, па је алат коришћен у режиму за *Mips* архитектуру која има одређених сличности са *ARM* архитектуром, али се ипак разликује од ње. Додатно, *Deckard* и *ACD* очекују изворни код који може да се преведе, па је зато излаз декомпајлера морао да се ручно доради како би ови алати могли бити примењени.

Табела 6.3 Преглед постојећих алата који се користе у експерименту.

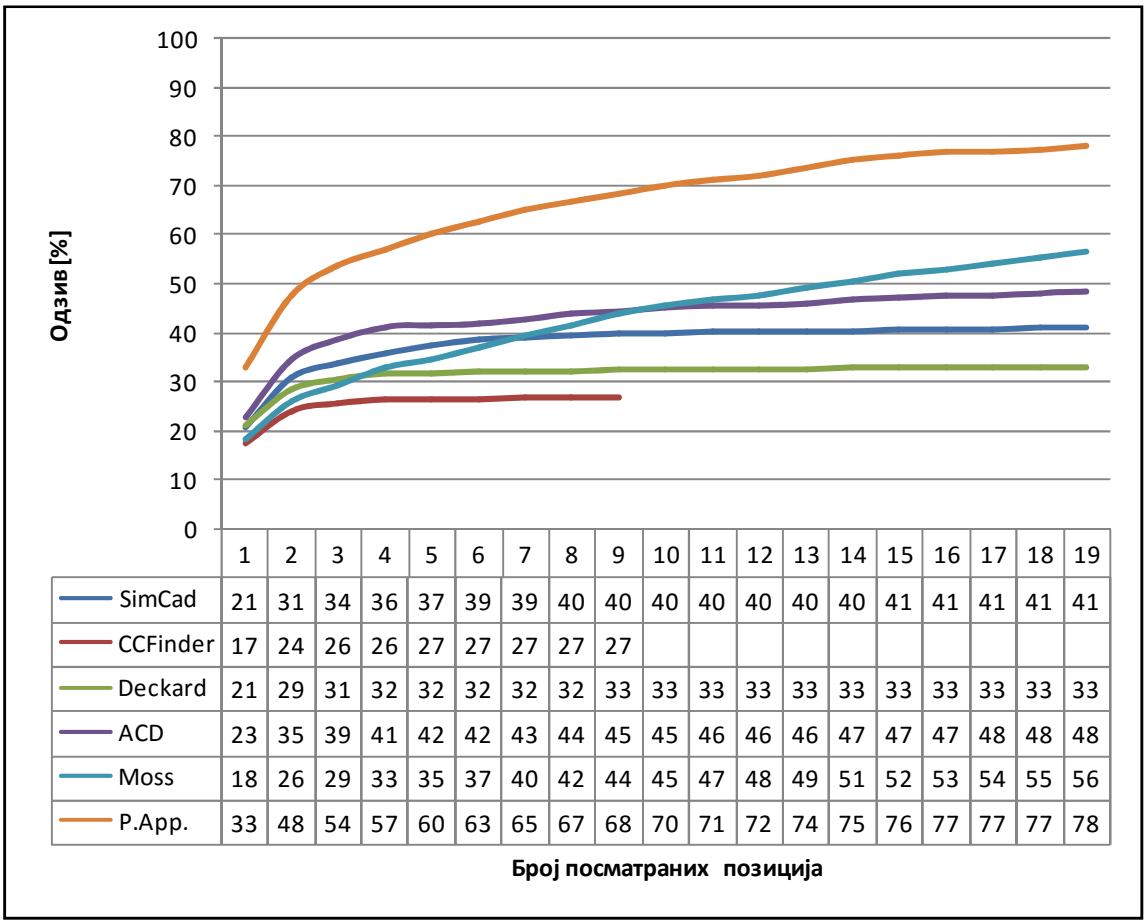
	SimCad	**CCFinder**	**Deckard**	**ACD**	**Moss**
Подржани језици	C, C#, Java, Py	C/C++, C#, Cobol, Java, VB, Text	C, Java, Php	C/C++	C/C++, C#, Cobol, Java, VB, MIPS, Text…
Језик у експерименту	C	C	C	C	ASM
Ниво поређења	блок, процедура	датотека	датотека	датотека	датотека
Техника откривања клонова	базиран на тексту	базиран на токенима	базиран на AST	базиран на тексту (ASM добијен од C)	базиран на тексту
Типови клонова	1, 2 и 3	1, 2 и 3	1, 2 и 3	1, 2 и 3	1, 2 и 3

Неки од алата не раде поређење кода на нивоу процедура већ користе друге границе. Како би се резултати приказали униформно, декомпајлирани код је морао бити издељен на велики број датотека тако да свака датотека садржи код тачно једне процедуре. Пошто алати на излазу не дају сличност упоређених кодова, за рангирање процедура из бинарног програма који се претражује коришћени су бројеви упарених линија или токена које алати враћају.

Након конфигурисања предложеног решења и постојећих алата спроведен је експеримент над 1503 насумично узорковања теста. За сваки тест се насумично одаберу два бинарна програма. Из првог програма се насумично одабере једна од процедура, таква да у другом бинарном програму постоји процедура која потиче од изворног кода процедуре одабране из првог програма. Процедуре су дисасемблиране за потребе тестирања предложеног приступа и *Moss* алата, док се за остале алате процедуре декомпајлирају и деле у засебне датотеке.

Добијени резултати у облику одзива постојећих алата и предложеног приступа у зависности од броја позиција које се посматрају приказани су на Слика 6.12. Са слике се види да у урађеном експерименту предложени приступ постиже боље резултате од свих постојећих алата са којима је поређење рађено, и то за све вредности броја посматраних позиција. Конкретно, ако се посматра само прва позиција, предложени приступ постиже 1.44 пута бољи одзив у односу на најбољи од тестираних постојећих алата. Међу осталим резултатима се примећује да за приказане вредности

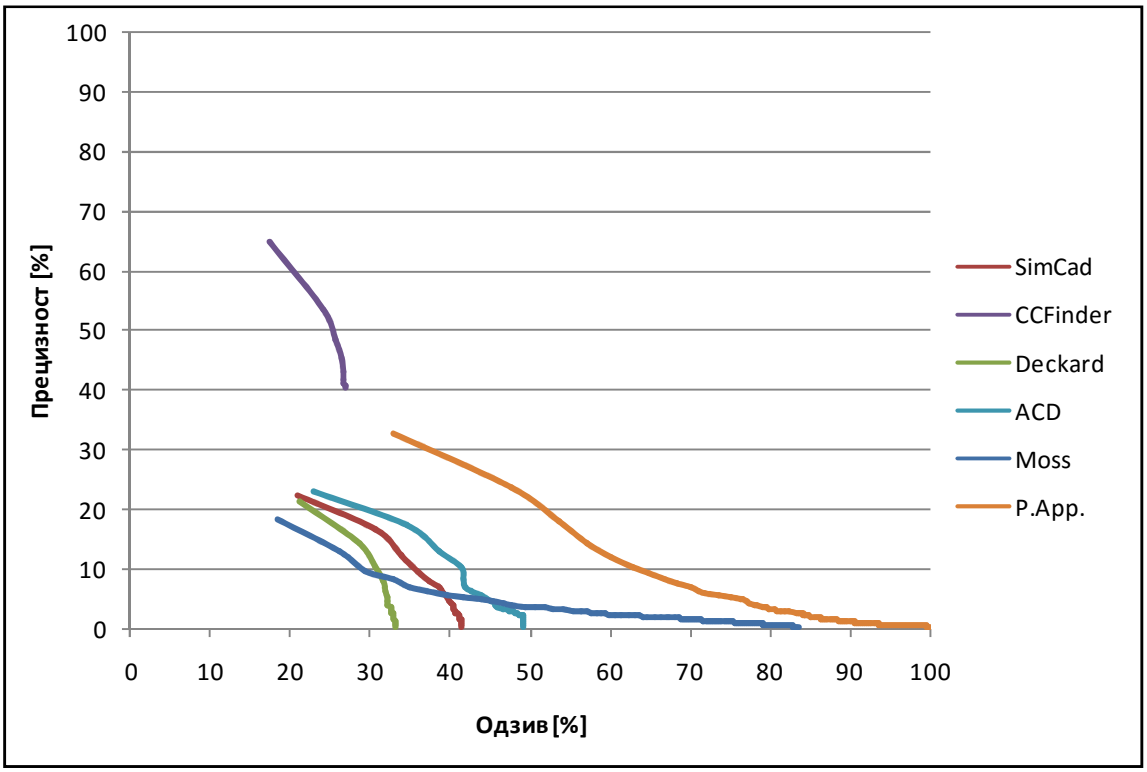
броја посматраних позиција, алати *ACD* и *Moss* имају монотоно растуће функције, док одзив осталих алата улази у засићење чак и за мање од првих *5* посматраних позиција. Такође треба приметити да предложени приступ рангира процедуру која потиче од изворног кода тражене процедуре међу првих *19* позиција у приближно 78% случајева, док у преосталих 22% случајева одговарајућа процедура буде рангирана после *19*. позиције.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	21	31	34	36	37	39	39	40	40	40	40	40	40	40	41	41	41	41	41
CCFinder	17	24	26	26	27	27	27	27	27										
Deckard	21	29	31	32	32	32	32	32	33	33	33	33	33	33	33	33	33	33	33
ACD	23	35	39	41	42	42	43	44	45	45	46	46	46	47	47	47	48	48	48
Moss	18	26	29	33	35	37	40	42	44	45	47	48	49	51	52	53	54	55	56
P.App.	33	48	54	57	60	63	65	67	68	70	71	72	74	75	76	77	77	77	78

Слика 6.12 Одзив предложеног приступа и постојећих алата у функцији броја посматраних позиција, случај *STAMP* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

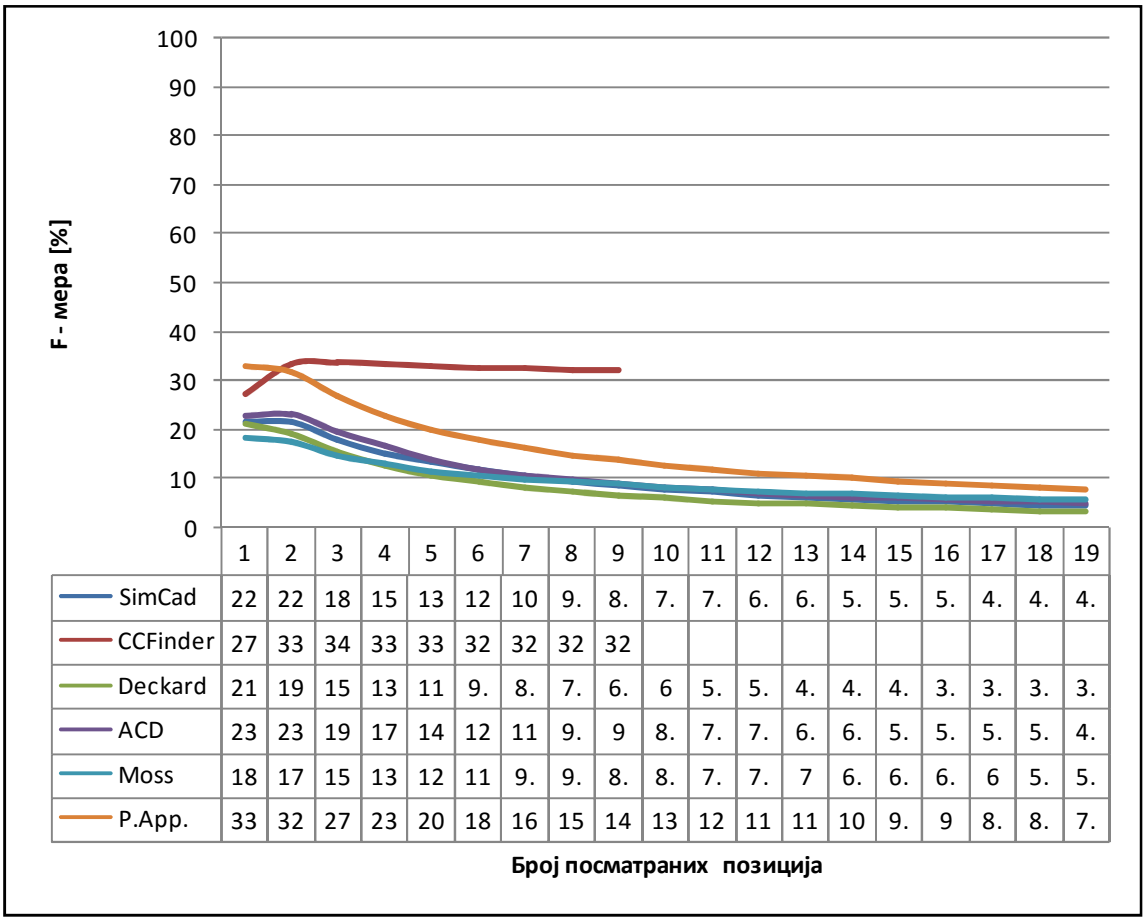
Друга мера кроз коју је предложени приступ евалуиран је прецизност. Криве које показују како се прецизност предложеног приступа и постојећих алата мења са променом одзива приказане су на Слика 6.13. Са слике се види да највећу прецизност може да постигне *CCFinder* док је предложени приступ други најбољи алат у погледу прецизности. Ипак, прецизност коју постиже *CCFinder* се веома брзо смањује чак и за релативно мали пораст одзива. Чак је и највећа вредност одзива који постиже *CCFinder* мања од најмање вредности одзива који постиже предложени приступ. Стога се са приложене слике може закључити да предложени приступ може бити веома значајан додатак



Слика 6.13 Прецизност коју остварују предложени приступ и постојећи алати у зависности од постигнутог одзива, случај *STAMP* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

постојећим алатима који постижу велику прецизност, али мали одзив када се примене у посматраном домену.

Трећа мера кроз коју ће бити посматран предложени приступ је *F* мера, приказана на Слика 6.14. На слици се примећује да предложени приступ има највећу вредност *F* мере уколико се посматра само прва позиција. Међутим,

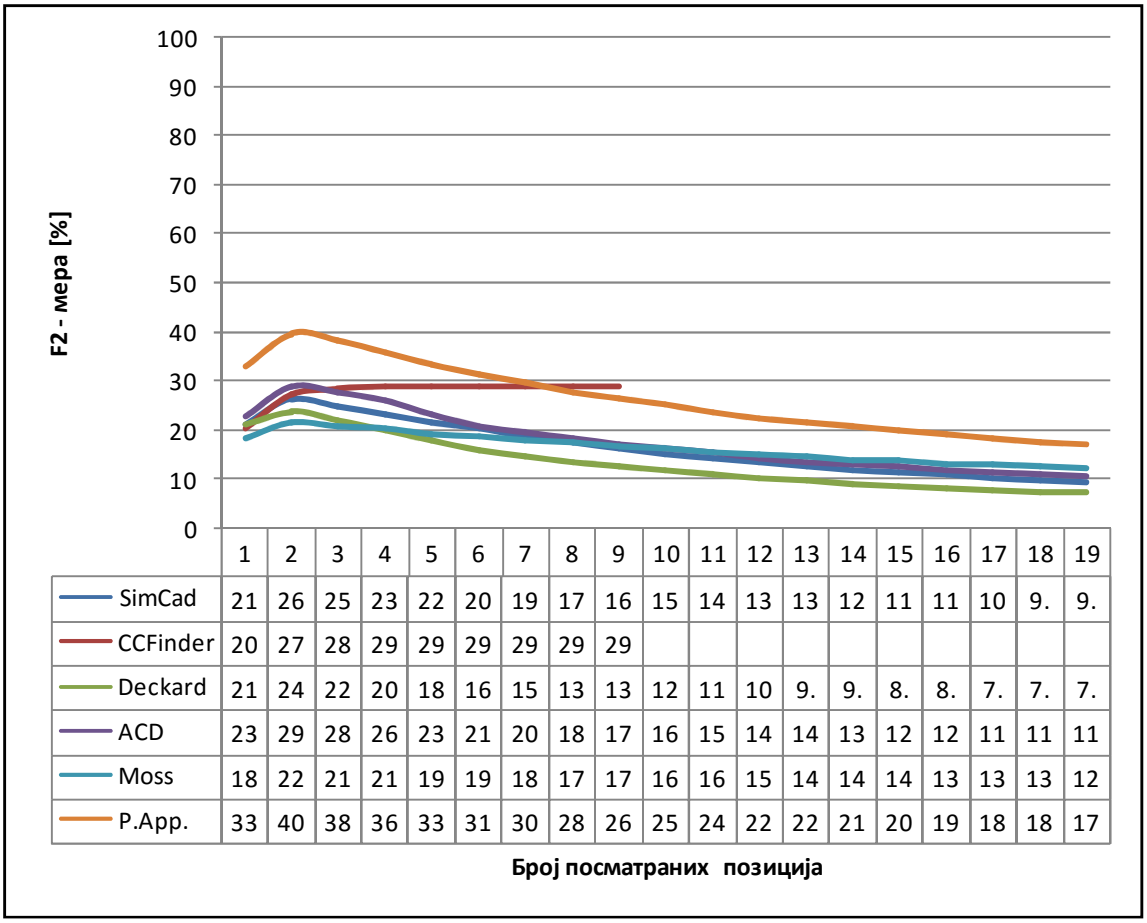


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	22	22	18	15	13	12	10	9.	8.	7.	7.	6.	6.	5.	5.	5.	4.	4.	4.
CCFinder	27	33	34	33	33	32	32	32	32										
Deckard	21	19	15	13	11	9.	8.	7.	6.	6	5.	5.	4.	4.	4.	3.	3.	3.	3.
ACD	23	23	19	17	14	12	11	9.	9	8.	7.	7.	6.	6.	5.	5.	5.	5.	4.
Moss	18	17	15	13	12	11	9.	9.	8.	8.	7.	7.	7	6.	6.	6.	6	5.	5.
P.App.	33	32	27	23	20	18	16	15	14	13	12	11	11	10	9.	9	8.	8.	7.

Слика 6.14 *F* мера предложеног приступа и постојећих алата у зависности од броја посматраних позиција, случај *STAMP* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

већ приликом посматрања бар прве две позиције, највећу вредност *F* мере даје *CCFinder*, док предложени приступ даје другу најбољу вредност. Уједно, *CCFinder* постиже и највећу вредност *F* мере у случају када се посматрају прве три позиције и она износи 33.6%. Одатле би се могао извести закључак да у случају када је подједнако важно пронаћи процедуру колико и да се време експерта ефикасно искористи, најбоље је користити *CCFinder* и посматрати прве три позиције. Приближно исти ефекат, уз смањење *F* мере 1.02 пута, би се такође постигао употребом предложеног приступа уз посматрање само прве позиције, с тим да би се тако одзив повећао 1.28 пута.

У циљу давања већег значаја одзиву, на Слика 6.15 је приказана F_2 мера. Из приложеног се примећује да предложени приступ даје највећу вредност ове мере ако се посматра не више од првих седам позиција. Уколико се посматра више од првих седам позиција, тада незнатно бољи резултат постиже *CCFinder*. Апсолутно највећу вредност постиже предложени приступ уколико се посматрају прве две позиције и тада износи 39.6%. У складу с тим, у приказаном експерименту се може закључити да би најбољи ефекат по питању постизања што већег одзива и што ефикаснијег утрошка експертског времена, у случају када се ипак истиче важност проналаска тражене процедуре, дала употреба предложеног приступа уз посматрање прве две позиције.

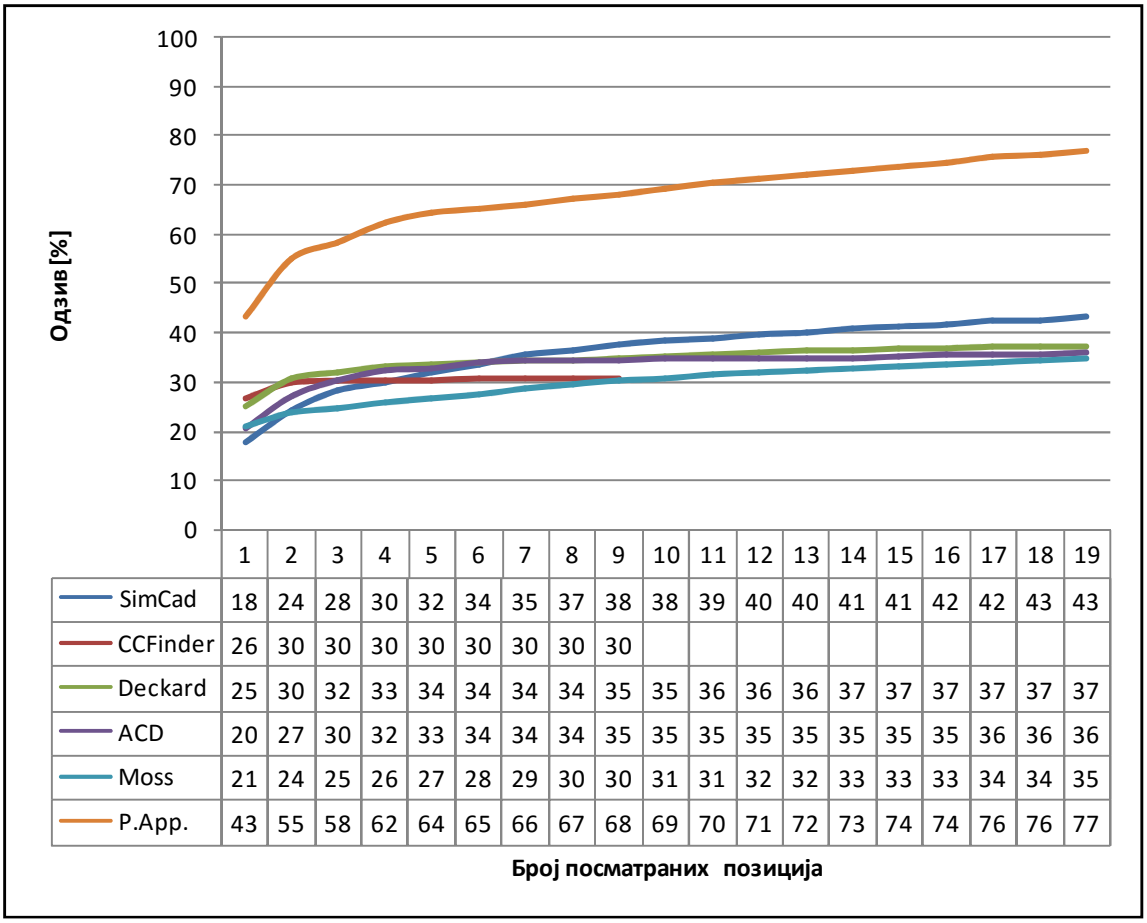


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	21	26	25	23	22	20	19	17	16	15	14	13	13	12	11	11	10	9.	9.
CCFinder	20	27	28	29	29	29	29	29	29										
Deckard	21	24	22	20	18	16	15	13	13	12	11	10	9.	9.	8.	8.	7.	7.	7.
ACD	23	29	28	26	23	21	20	18	17	16	15	14	14	13	12	12	11	11	11
Moss	18	22	21	21	19	19	18	17	17	16	16	15	14	14	14	13	13	13	12
P.App.	33	40	38	36	33	31	30	28	26	25	24	22	22	21	20	19	18	18	17

Слика 6.15 F_2 мера предложеног приступа и постојећих алата у зависности од броја посматраних позиција, случај *STAMP* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

6.2.4 Експеримент 1.4

Циљ четвртог експеримента је да понови поређење предложеног приступа и одабраних постојећих алата у реалном случају употребе библиотеке издате под двоструком лиценцом. Поређење је урађено аналогно поређењу изведеном у трећем експерименту, с разликом што се користе улазни подаци засновани на софтверском пакету *BusyBox*. Софтверски пакет и библиотека која се тражи су описани у Глави 4 (окружење за евалуацију). За спровођење експеримента направљен је узорак који садржи 800 тестова.

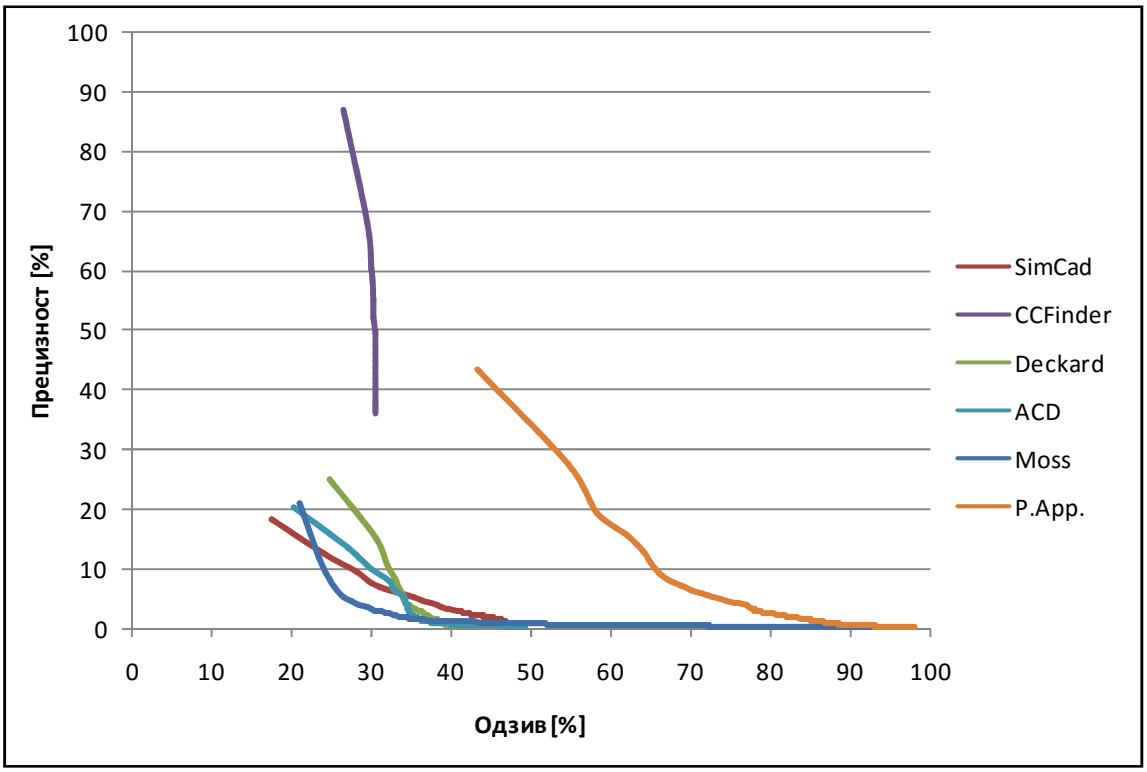


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	18	24	28	30	32	34	35	37	38	38	39	40	40	41	41	42	42	43	43
CCFinder	26	30	30	30	30	30	30	30	30										
Deckard	25	30	32	33	34	34	34	34	35	35	36	36	36	37	37	37	37	37	37
ACD	20	27	30	32	33	34	34	34	35	35	35	35	35	35	35	35	36	36	36
Moss	21	24	25	26	27	28	29	30	30	31	31	32	32	33	33	33	34	34	35
P.App.	43	55	58	62	64	65	66	67	68	69	70	71	72	73	74	74	76	76	77

Слика 6.16 Одзив предложеног приступа и постојећих алата у функцији броја посматраних позиција, случај *BusyBox* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

Одзив предложеног приступа и упоређених постојећих алата у зависности од броја посматраних позиција приказани су на Слика 6.16. Са слике се види да предложени приступ и у овом случају постиже најбољи одзив од свих тестираних алата за сваки број посматраних позиција. Одзив предложеног приступа је значајно порастао у односу на случај теста заснованог на *STAMP* ако се посматра само прва позиција и износи 43%, док је одзив остао скоро исти када се посматра првих 19 позиција. У исто време, одзиви осталих алата нису значајније промењени, осим што је код неких

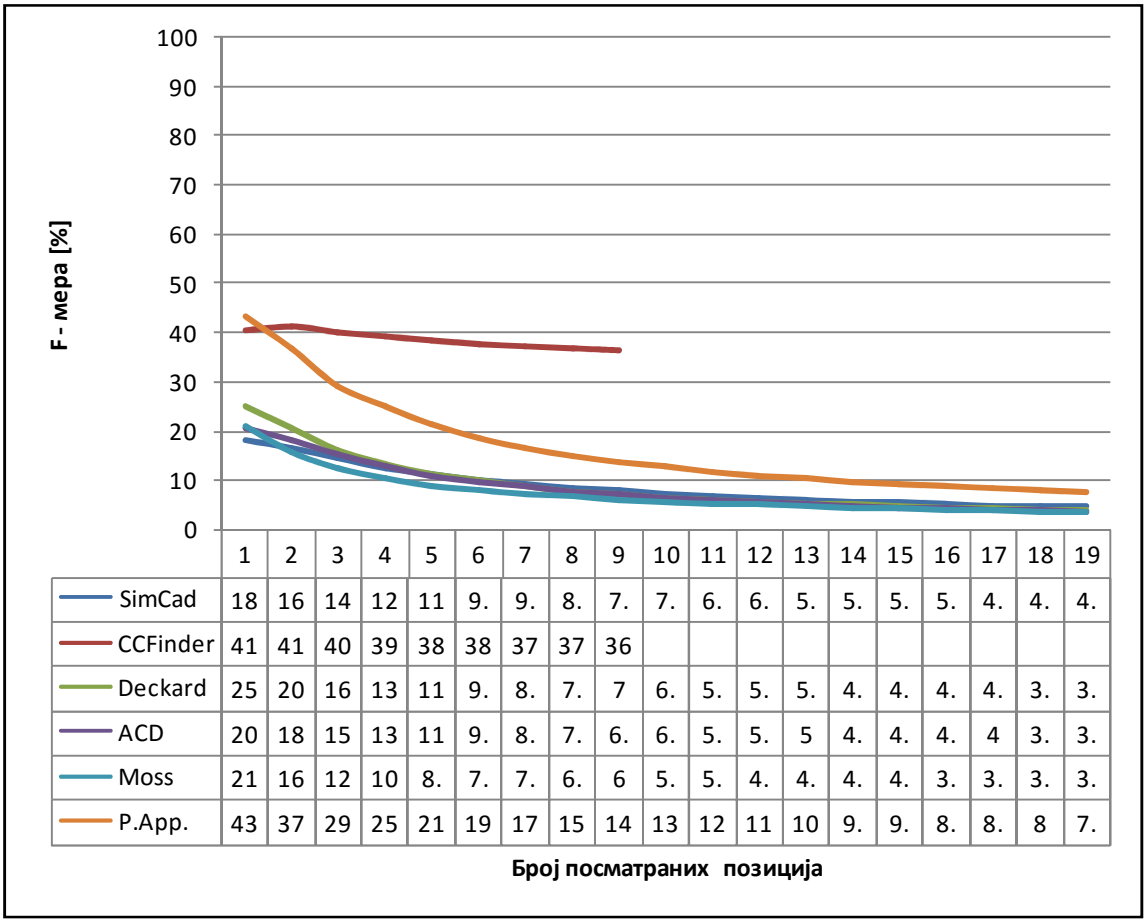
алата попут *Moss* пораст одзива са порастом броја позиција знатно мањи у односу на пораст у случају теста заснованог на *STAMP*. Пораст одзива када се посматра само прва позиција се може тражити у чињеници да је просечна величина процедуре у *BusyBox* пакету неколико пута већа у односу на *STAMP* окружење, што омогућава да се појави већи број различитих вредности метрика, па се и процедуре могу боље диференцирати.



Слика 6.17 Прецизност предложеног приступа и постојећих решења у зависности од одзива, случај *BusyBox* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ)

На Слика 6.17 приказана је прецизност коју постижу предложени приступ и постојећи алати у зависности од постигнутог одзива. У поређењу са истим графиком за случај тестова базираних на *STAMP* окружењу,

примећује се да су облици и релативни односи кривих веома слични. Основна разлика је што уз нешто већи одзив алати постижу и већу прецизност. Једини алат чији резултати су незнатно лошији је *Moss*.

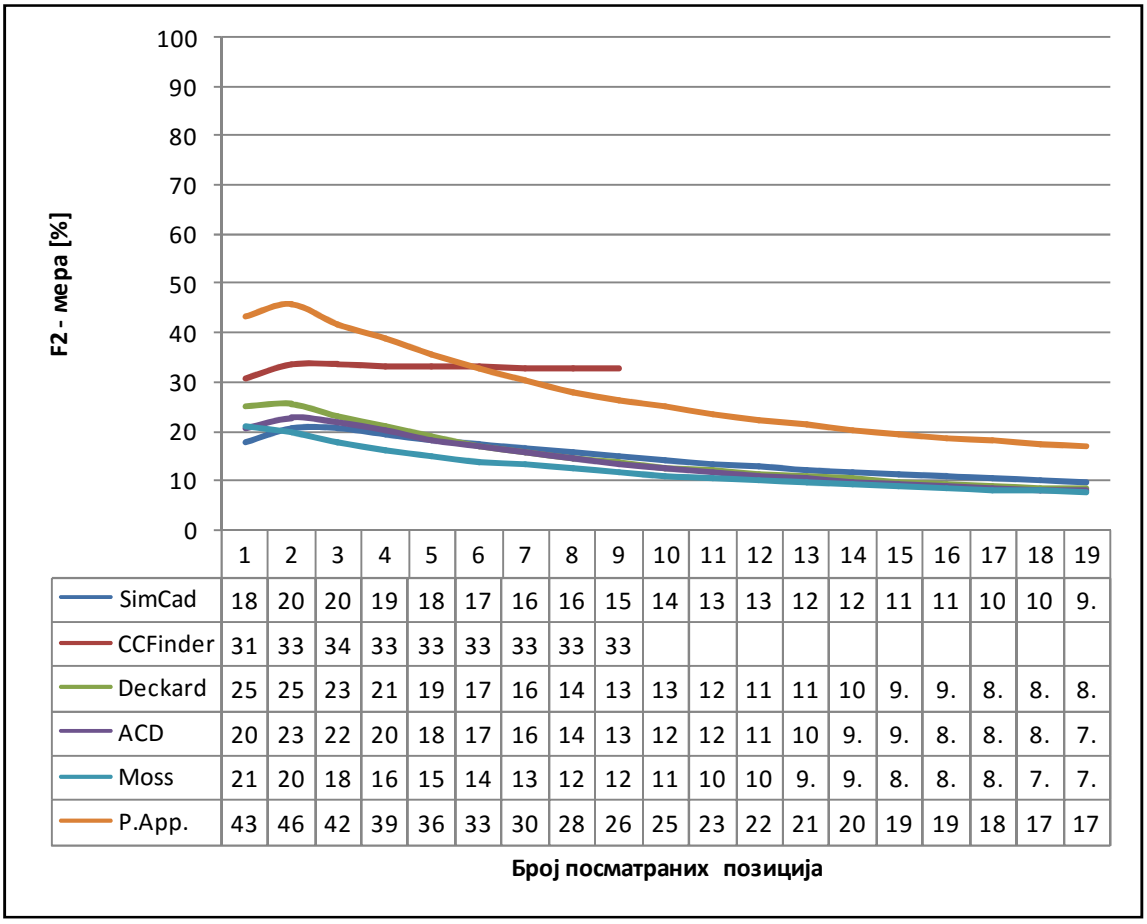


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	18	16	14	12	11	9.	9.	8.	7.	7.	6.	6.	5.	5.	5.	5.	4.	4.	4.
CCFinder	41	41	40	39	38	38	37	37	36										
Deckard	25	20	16	13	11	9.	8.	7.	7	6.	5.	5.	5.	4.	4.	4.	4.	3.	3.
ACD	20	18	15	13	11	9.	8.	7.	6.	6.	5.	5.	5	4.	4.	4.	4	3.	3.
Moss	21	16	12	10	8.	7.	7.	6.	6	5.	5.	4.	4.	4.	4.	3.	3.	3.	3.
P.App.	43	37	29	25	21	19	17	15	14	13	12	11	10	9.	9.	8.	8.	8	7.

Слика 6.18 *F* мера предложеног приступа и постојећих алата у зависности од броја посматраних позиција, случај *BusyBox* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

На Слика 6.18 приказане су вредности *F* мере које постижу предложени приступ и постојећи алати за тестове базиране на *BusyBox* пакету. Обзиром на нешто већи одзив и прецизност, како предложеног приступа, тако и постојећих алата, вредности *F* мере су нешто веће у односу

на вредности у трећем тесту, када је коришћен узорак базиран на *STAMP* окружењу. Ипак, релативни однос предложеног приступа и другог најбољег од постојећих алата је задржан. Гледајући само прву позицију, највећу вредност *F* мере постиже предложени приступ. У осталим случајевима највећу вредност постиже *CCFinder*.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
SimCad	18	20	20	19	18	17	16	16	15	14	13	13	12	12	11	11	10	10	9.
CCFinder	31	33	34	33	33	33	33	33	33										
Deckard	25	25	23	21	19	17	16	14	13	13	12	11	11	10	9.	9.	8.	8.	8.
ACD	20	23	22	20	18	17	16	14	13	12	12	11	10	9.	9.	8.	8.	8.	7.
Moss	21	20	18	16	15	14	13	12	12	11	10	10	9.	9.	8.	8.	8.	7.	7.
P.App.	43	46	42	39	36	33	30	28	26	25	23	22	21	20	19	19	18	17	17

Слика 6.19 F_2 мера предложеног приступа и постојећих алата у зависности од броја посматраних позиција, случај *BusyBox* (*SimCad*, *CCFinder*, *Deckard*, *ACD*, *Moss* - постојећи алати, *P.App.* - предложени приступ).

Слично понашање предложени приступ и постојећи алати показују и када се посматра F_2 мера која је приказана на Слика 6.19. Највеће вредности и

даље постиже предложени приступ уколико се посматра до првих шест позиција. Приликом посматрања већег броја позиција, највећу вредност F_2 мере постиже *CCFinder*.

6.3 Технике за побољшање резултата

Иако претходно приказани и евалуирани предложени приступ постиже већи одзив од свих тестираних постојећих алата, неопходно је анализирати и слабости предложеног приступа у циљу евентуалног побољшања резултата које постиже. Иако је поређењем процедура на основу малог дела њихових карактеристика знатно смањен утицај преводиоца на процес прорeђења, такав начин поређења има и недостатака. Ипак, занемаривање великог броја карактеристика приликом поређења доводи и до тога да се неретко две процедуре које потичу од различитих изворних кодова прогласе истима. У прилог погрешном закључивању иде и чињеница да ни одабране карактеристике не успевају увек да занемаре све разлике које приликом поређења настану.

Једна могућа интерпретација теорије информација на предложени приступ и процес рачунања тоталне сличности процедура јесте да инструкције одговарају симболима, процедуре порукама, а да софтверске метрике представљају поруке компримоване са губицима. Циљ компресије је да задржи информације које веродостојно представљају изворни код процедуре независно од процеса превођења. Ипак, описани процес компресије може да испусти одређене битне информације, али исто тако и да задржи одређене информације које воде погрешном закључку. Стога

резултати предложеног приступа који се заснива на информацијама задржаним у метрикама могу бити побољшани тако што ће се повећати количина информација које имају позитиван утицај на рангирање процедура и истовремено смањити количина информација које имају негативан утицај на рангирање процедура. Повећање и смањење количине информација може бити на нивоу порука или на нивоу симбола, што доноси четири класе могућих побољшања предложеног приступа. Поменуте четири класе могућих побољшања су приказане на Слика 6.20. Свака од класа може да побољша резултате тако што ће повећати ранг процедуре која потиче од изворног кода тражене процедуре или тако што ће смањити ранг процедура које не потичу од изворног кода тражене процедуре.

Слика 6.20 Четири класе могућих побољшања и њихови очекивани ефекти.

Смањење количине информација које негативно утичу на рангирање може бити постигнуто одбацивањем одређених симбола и порука. Елиминација симбола у контексту поређења процедура може бити посматрана као изостављање одређених инструкција (доњи леви угао на

Слика 6.20). Инструкције које могу бити изостављене су оне које се често појављују и као такве носе малу количину информација, а које се често појављују у деловима кода који су различито преведени. Примери су инструкције за приступ аргументима и локалним променљивим процедуре који се налазе на стеку и у регистрима, у зависности од процеса превођења (инструкције које раде са стеком и инструкције за пренос података). Елиминација порука у контексту поређења процедура може да се посматра као изостављање целих процедура (доњи десни угао на Слика 6.20). Могу се изоставити оне процедуре које имају велику вредност тоталне сличности са траженом процедуром захваљујући метрикама које носе малу количину информација, али се од ње значајно разликују по метрикама које носе значајну количину информација. Пример су процедуре које имају сличан број петљи, позива процедура, и сличне вредности осталих скаларних метрика, али се од тражене процедуре значајно разликују по метрици која пребројава поједине инструкције (*EIN*).

Повећање количине информација које могу позитивно утицати на процес рангирања може да се изведе груписањем симбола и порука у складу са зависностима које између њих постоје. Груписање симбола у контексту поређења процедура може да се посматра као нова метрика (горњи леви угао на Слика 6.20). Информације које могу бити додатно екстраховане су зависности између појединих инструкција унутар процедуре. Један такав пример су секвенце узастопних инструкција различите дужине, чиме се делимично издвајају и информације о редоследу инструкција које нису постојале ни у једној од предложених метрика. Груписање порука у

контексту поређења процедура такође може да се посматра као увођење нове метрике која се добија од вредности постојећих метрика већег броја процедура (горњи десни угао на Слика 6.20). Информације које се на тај начин могу екстраховати су односи међу процедурама унутар једног програма у циљу побољшања поређења процедура на основу њиховог окружења. Примери су кратке процедуре које преводиоци у току оптимизације могу да угради на место позива.

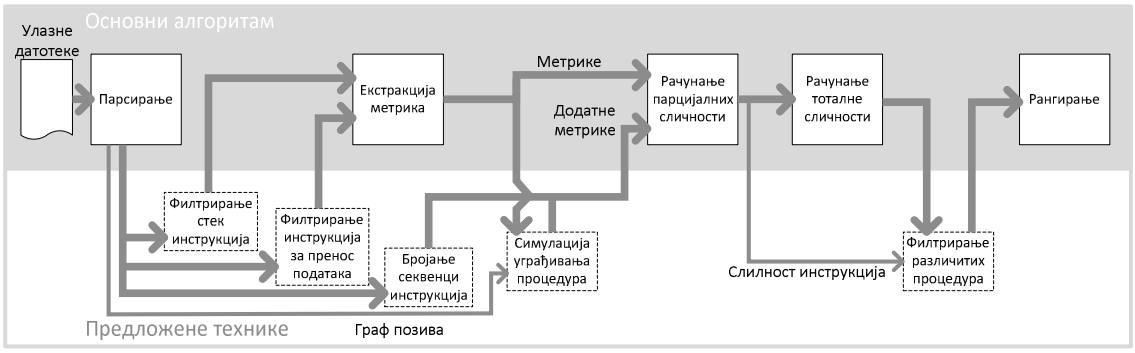
6.3.1 Основни алгоритам предвиђен за евалуацију техника

Основни алгоритам се састоји од одређених корака предложеног приступ и представља основу на коју ће бити надограђене технике. Од корака који постоје у предложеном приступу укључени су парсирање, одређивање вредности метрика, одређивање парцијалних мера сличности, рачунање тоталне мере сличности и рангирање. Важно је напоменути да ниједан од задржаних корака предложеног приступа не захтева претходно знање.

Након парсирања улазних фајлова одређује се број петљи, број инструкција за пренос података, број позива сваког од позиваних потпрограма и бројеви појављивања за сваку од инструкција која се појављује у посматраној процедури. Вредности метрика се даље користе за рачунање парцијалних мера сличности. При томе се за бројеве позива користи компаратор базиран на рангирању, док се за поређење бројева појављивања појединих инструкција користи компаратор заснован на ознакама. На добијене вредности парцијалних мера сличности се као

формула за рачунање тоталне мере сличности примењује аритметичка средина. На основу тоталних мера сличности тражене процедуре и процедура из бинарног програма ради се рангирање процедура из програма.

Основни алгоритам је проширен са пет техника које имају за циљ повећање одзива и прецизности основног алгоритма (Слика 6.21). Прве две технике су примењене непосредно након парсирања. Преостале три технике примењене су у каснијим фазама основног алгоритма, и то редом, у току прикупљања метрика, након прикупљања метрика и након рачунања парцијалних мера сличности а пре рангирања. Прве две технике филтрирају инструкције које раде са стеком и инструкције за трансфер података. Трећа техника задржава додатне парцијалне информације о редоследу инструкција (пребројава секвенце инструкција одређене дужине). Четврта техника симулира уграђивање процедура на месту позива, за шта је неопходан граф позива који се формира у току парсирања бинарног програма. Пета техника



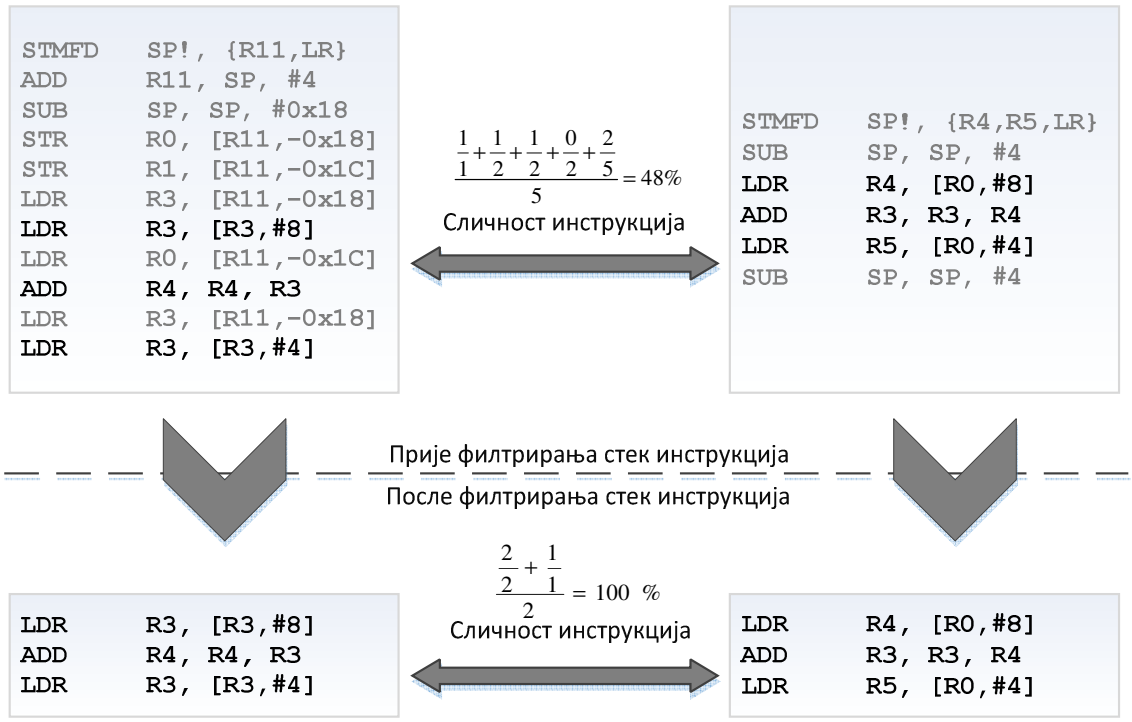
Слика 6.21 Основни алгоритам преузет из предложеног приступа и проширен предложеним техникама.

одбацује из даљег разматрања процедуре које се од тражене процедуре значајно разликују по инструкцијама које се у њима користе. Наведене технике су објашњене у наставку уз адекватне примере који илуструју очекиване ефекте предложених техника.

6.3.2 Филтрирање инструкција за рад са стеком

Прва техника је инспирисана различитим начинима на које преводиоци генеришу код за приступ аргументима и локалним променљивим. Елиминацијом инструкција које раде са стеком, прва техника смањује негативан утицај који на поређење процедура имају различити начини рада са аргументима и локалним променљивим. Један пример разлика које настају услед различитих начина употребе стека приказан је на Слика 6.22. На левој страни је приказан код који је на почетку сачувао вредности аргумената на стеку (*STR* инструкције у четвртој и петој линији). Када те вредности касније затребају, генерише се код који их дохвати са стека (*LDR* инструкције у шестој, осмој и десетој лини дохватају вредности аргумената које су сачуване у четвртој и петој линији). На десној страни је код који не чува вредности аргумената, већ користи садржаје регистара у којима су добијени аргументи. Овакав приступ може бити и резултат оптимизације кода приказаног са леве стране, пошто се примети да се вредности аргумената у регистрима *R0* и *R1* не мењају. Иако су оба асемблерска кода настала превођењем истог изворног кода, парцијална сличност у складу са метриком која пребројава појављивања инструкција износи свега 48%. Након што се примени прва техника и из даљег

разматрања одстране све инструкције које раде са стеком, поменута парцијална мера се значајно увећа. Приказани примери представљају идеалан случај за примену прве технике јер су у оба резултујућа кода остале исте инструкције. Генерално, то не мора да важи, али се очекује да занемаривањем инструкција које раде са стеком смањи број разлика међу упоређеним процедурама, као што је случај при нормализацији кода [67]

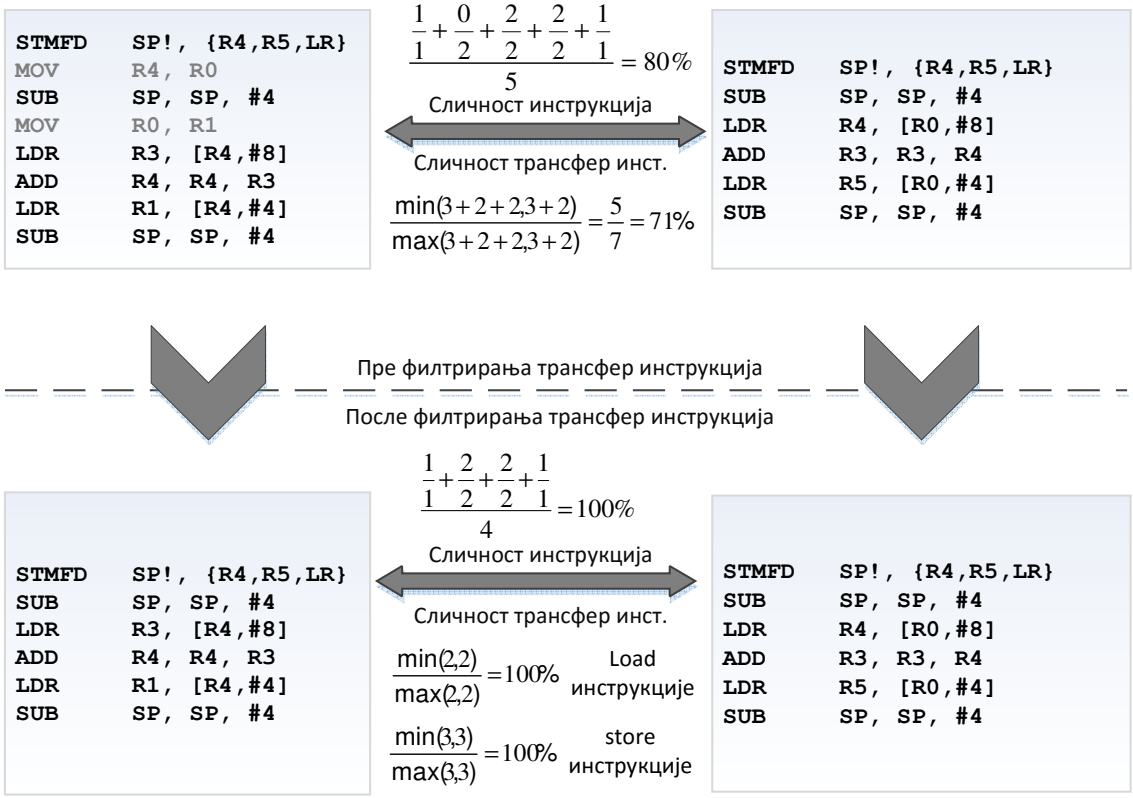


Слика 6.22 Пример технике занемаривања инструкција за рад са стеком (прва техника).

6.3.3 Филтрирање инструкција за пренос података

Друга техника је инспирисана разликама које настају као последица различитих алгоритама алокације регистара. Елиминацијом инструкција за пренос података између регистара смањују се и разлике које настају као последица непотребних трансфера података насталих због неоптималне

алокације регистара. На левој страни Слика 6.23 приказан је код у којем се појављују сувишне инструкције услед субоптималног алгоритма алокације регистара [68]. На десној страни Слика 6.23 је приказан код у којем је уместо регистра *R1* коришћен регистар *R5*, па није било потребно ослобађати регистар *R1*, чиме је остао слободан и *R0* регистар. Стога је било могуће у регистру *R0* задржати оригиналну вредност и користити је без пребацивања у регистар *R4*, чиме је и регистар *R4* остао слободан за другу намену.



Слика 6.23 Пример примене технике занемаривања инструкција за пренос података између регистара (друга техника).

Пре примене друге технике вредност парцијалне мере сличности рачуната на основу метрике која пребројава инструкције износила је 80%, док је вредност парцијалне мере сличности рачуната на основу броја

инструкција за пренос података износила 71%. Након примене друге технике, пребројаване су преостале *LDR* и *STR* инструкције. При томе су инструкције за вишеструко читање података из меморије и упис података у меморију рачунате онолико пута колико се података чита/уписује. Примена друге технике довела је до повећања вредности поменутих парцијалних мера сличности. У општем случају повећање парцијалних мера сличности не мора бити као у приказаном примеру, али се очекује да техника допринесе занемаривању разлика међу процедурама насталих неоптималном употребом регистара. Техника може помоћи и у случају када се без потребе прави копија аргумената у помоћним регистрима.

6.3.4 Сличност секвенци операционих кодова

Трећа техника допуњује недостатке метрика предложеног приступа. Наиме, ниједна од предложених метрика није задржала информације о редоследу инструкција. Такав приступ може да доведе до ситуације да се две потпуно различите процедуре које се састоје од истих инструкција, другачије поређаних, прогласе идентичним. На Слика 6.24 приказани су примери процедура које су веома сличне ако се посматрају инструкције које се у њима појављују, а које потичу од два различита изворна кода. Посматрајући парцијалну меру сличности рачунату на основу метрике која пребројава појављивања инструкција добија се релативно висока вредност од 62%. Трећа техника уводи нову меру која у посматраном примеру даје знатно мању вредност парцијалне сличности, свега 20%, што резултује у нижем рангу процедура које не потичу од изворног кода тражене процедуре. Нова мера се састоји од бројева појављивања секвенци сукцесивних инструкција

одређене дужине. Дужина може да буде било који природан број. Веће вредности задржавају више информација, али уједно смањују отпорност на одређене измене које настају у процесу превођења кода. У примеру на Слика 6.24 одабрана је дужина 2. Важно је напоменути да се броје све могуће секвенце сукцесивних инструкција одабране дужине (сукцесивне секвенце се преклапају, као што се може видети на Слика 6.24). Одговарајућа парцијална мера сличности, названа сличност секвенци операционих кодова, се добија применом компаратора заснованог на ознакама, при чему се ознака састоји од инструкција из секвенце у неизмењеном редоследу. Сличан приступ виђен је у приступу који је предложен у Santos et al. [15], с разликом што се за поређење вредности метрике користи косинусна сличност. Такође треба приметити да је сличност секвенци операционих кодова за секвенце дужине један исто што и сличност на основу метрике која пребројава појављивања појединих инструкција.

Слика 6.24 Пример примене технике бројања секвенци операционих кодова (трећа техника).

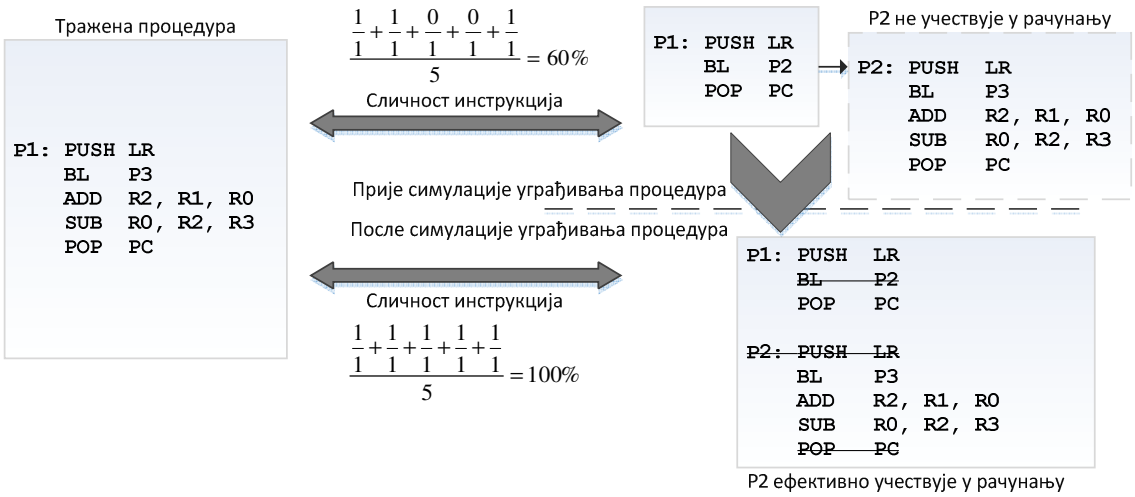
6.3.5 Симулација уграђивања процедура на месту позива

Четврта техника се односи на разлике које настају услед употребе различитих хеуристика за одлучивање о уграђивању процедура на места

позива [69]. Техника је предвиђена за случај када један преводилац генерише код у којем је нека процедура уграђена на местима позива те процедуре, док други преводилац остави позиве. Процедура која се угради на места позива најчешће више није присутна у посматраном бинарном програму јер је њено тело већ уграђено уместо свих позива те процедуре. Четврта техника симулира процедуре које се уграђују на месту позива тако што агрегира вредности метрика позивајуће и позиваних процедура. Метрике се агрегирају тако што се на вредности метрика позивајуће процедуре додају вредности метрика свих процедура које су позиване, при чему метрике сваке процедуре дода онолико пута колико пута се та процедура позива из посматране процедуре. Поред додавања метрика позиваних процедура, метрике позивајуће процедуре се коригују за сваки ранији позив јер више не постоји потреба ни за позивом потпрограма ни за повратком из потпрограма.

Пример примене метрике приказан је на Слика 6.25. Са леве стране приказана је процедура у коју је преводилац већ уградио једну позивану процедуру. У горњем десном делу приказана је оригинална процедура P1 која позива процедуру P2, такође приказану на слици. У доњем десном углу приказан је скуп инструкција које ефективно остају након примене четврте метрике. Применом технике у посматраном случају парцијална мера сличности се увећала до максимума, што у општем случају не мора да важи. У случају да неки преводилац ради уграђивање процедура не место позива у више нивоа, тада би посматрану технику било добро применити на процедуре које се у графу позива налазе до одређене дубине у односу на

позивајућу процедуру, чиме се може контролисати величина околине процедуре која се користи при поређењу те процедуре са неком другом.

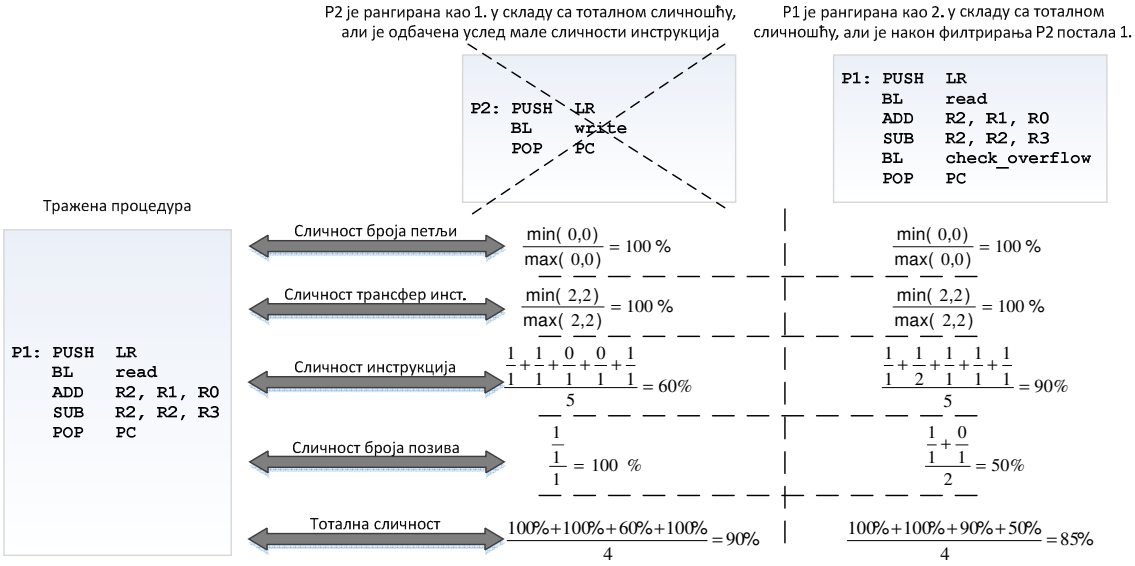


Слика 6.25 Пример примене технике симулације уграђивања процедура на место позива (четврта техника).

6.3.6 Филтрирање значајно различитих процедура

Високо апстрактан процес поређења и измене које настају у процесу превођења доводе до ситуације да се процедура која не потиче од изворног кода тражене процедуре прогласи сличнијом од процедуре која потиче од изворног кода тражене процедуре. Разлог томе је што већина парцијалних мера сличности које носе малу количину информација, буду велике и доминантне над малим бројем парцијалних мера сличности које носе велику количину информација и сугеришу да посматрана процедура мање вероватно потиче од изворног кода тражене процедуре. Применом пете технике могуће је ублажити ову аномалију тако што се одређене процедуре пре рангирања елиминишу из даљег разматрања. Процедура се елиминише из разматрања уколико је вредност парцијалне мере сличности, рачуната на

основу метрике која пребројава појављивања инструкција, мања од унапред одабране вредности. На тај начин се из разматрања одбацују процедуре које са великом вероватноћом не потичу од изворног кода тражене процедуре. Пример примене пете технике приказан је на Слика 6.26. У датом примеру



Слика 6.26 Пример примене технике која из разматрања елиминише процедуре које се по инструкцијама значајно разликују од тражене процедуре (пета техника).

граница за одбацивање је постављена на 70%. На примеру се види да је пре примене технике процедура *P2*, која не потиче од изворног кода тражене процедуре, била рангирана испред процедуре *P1* која потиче од изворног кода тражене процедуре.

6.4 Евалуација предложених техника

Циљ евалуације јесте да се испита допринос техника и да се основни алгоритам проширен техникама упореди са постојећим приступима и алатима. За поређење ће се овај пут користити приступи који раде над бинарним кодом директно, док ће представник постојећих алата бити предложени приступ који је дао већи одзив од свих упоређених постојећих алата. За разлику од поређења предложеног приступа са постојећим алатима, за одабране постојеће приступе не постоје адекватни алати, па су због тога за потребе поређења имплементирани алгоритми који су описани у одговарајућим радовима. За поређење су коришћени следећи приступи: *ACD* (*CD*), *Binary Analysis Tool - BAT* (*LV1*), приступ предложен у овом раду (*LV2*), *Opcode sequence* (*MD*) и приступ који предлаже *Dullien* (*VA*).

Евалуације је спроведена кроз три експеримента, при чему се сваки састоји од већег броја тестова. Иако приступи и алати који се пореде припадају различитим доменима, предвиђени су за рад под различитим околностима и имају различите алате, у евалуацији су сви коришћени на исти начин, у складу са основним циљем овог рада. Тестови су припремљени на исти начин као и при евалуацији предложеног приступа, користећи истих пет преводилаца са четири различита оптимизациона нивоа: најмањи ниво оптимизација, највиши ниво оптимизација, оптимизација за брзину и оптимизација за простор. Сваки тест се састоји од једне процедуре која се тражи и скупа процедура међу којима се сигурно налази процедура која потиче од изворног кода тражене процедуре. Скуп процедура по правилу

припада једном бинарном програму, па су све преведене на исти начин. Процеси превођења тражене процедуре и скупа процедура су независни и произвољни унутар поменутих алата и нивоа оптимизација. Основна разлика у припреми тестова је што сви приступи који се користе у наредним експериментима на улазу очекују бинарни код, евентуално у дисасемблираном облику. Табела 6.4 приказује основне карактеристике експеримената.

Табела 6.4 Приказ основних карактеристика експеримената.

Експеримент	Коришћени приступи	Извор тражене процедуре	Извор претраживаних процедура	Број тестова у експерименту	Просечан број процедура у претраживаном програму
1.	предложени	STAMP	STAMP	1510	285
2.	постојећи и предложени	STAMP	STAMP	1510	285
3.	постојећи и предложени	MatrixSsl	BusyBox	1532	1949

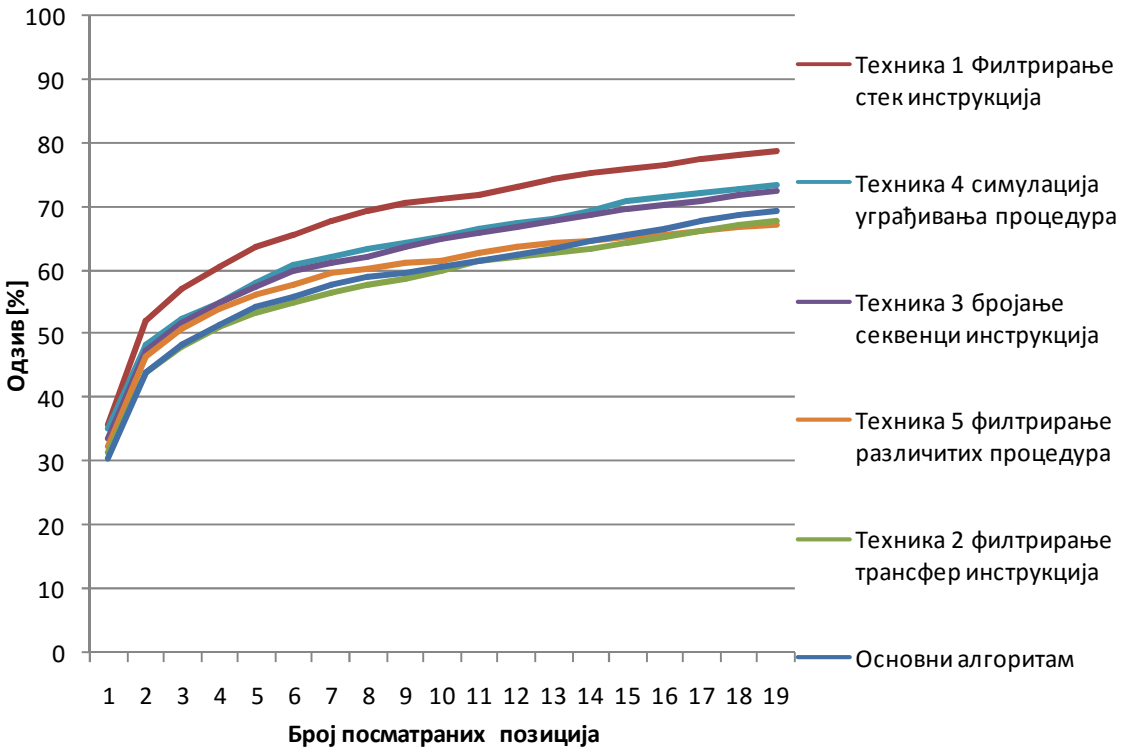
Прва два експеримента изведена су над тестовима формираним од пет програма *STAMP* окружења који деле одређени скуп процедура и имају за циљ да одговоре на истраживачка питања. Први експеримент има за циљ да одговори на истраживачка питање 4. и 5. која питају да ли примена сваке од техника појединачно доприноси повећању одзива и да ли се приликом примене свих техника заједно појављује синергистички ефекат. Други експеримент треба да одговори на последње истраживачко питање које

испитује да ли основни алгоритам проширен предложеним техникама даје најбољи резултат у погледу одзива, прецизности и *F* мере. Трећи експеримент представља верификацију другог експеримента и изведен је над тестовима који представљају реалан случај употребе библиотеке *MatrixSSL*, издате под двоструком лиценцом, унутар софтверског пакета *BusyBox*.

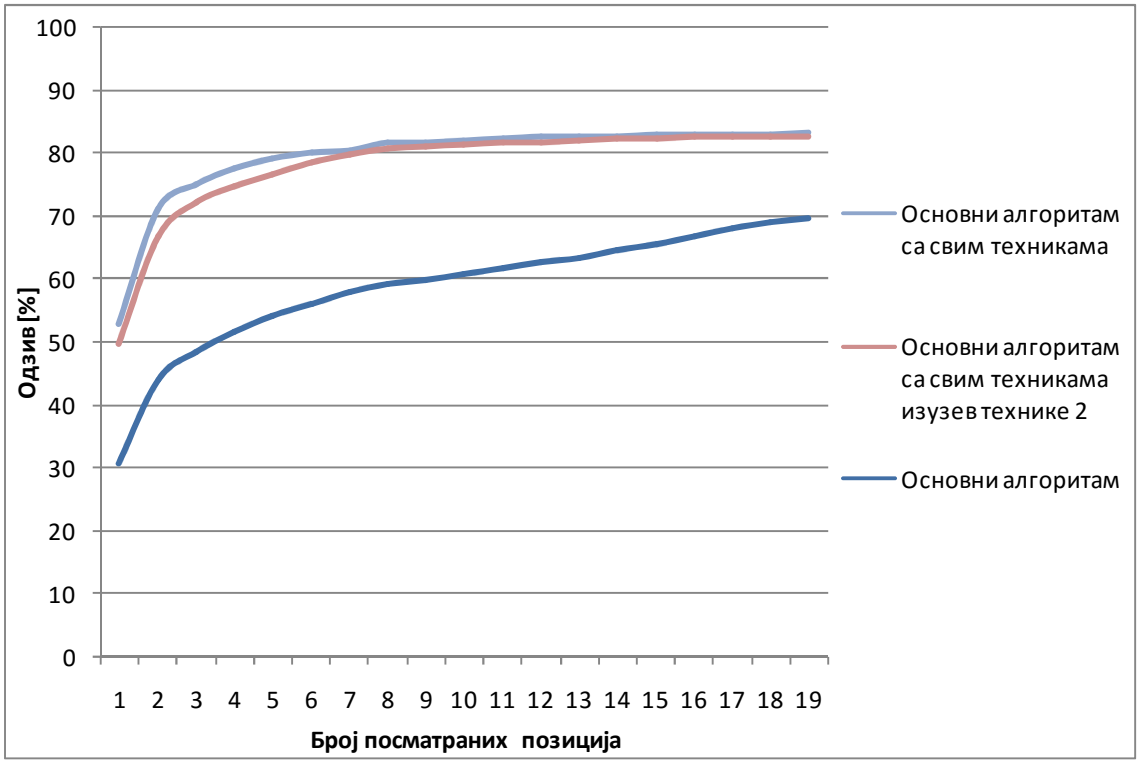
6.4.1 Експеримент 2.1

Први експеримент је изведен тако што су мерени одзиви основног алгоритма, прво без техника, а затим уз додатак сваке од техника појединачно и на крају са свим техникама заједно. За потребе границе за одбацивање процедура у петој техници испробане су све вредности од 0% до 100% са кораком од 1%. Највећи одзив у случају када се посматра само прва позиција постигнут је за вредност 71%, и та граница је одабрана за употребу у преосталим мерењима и експериментима. На Слика 6.27 је приказан одзив који постиже основни приступ без техника и за сваку од техника појединачно, све у зависности од броја позиција које се посматрају. Анализа приказаних резултата показује да скоро све предложене технике повећавају одзив основног алгоритма. За очекивати је да допринос технике значајно зависи од вероватноће да се ситуација за коју је техника намењена заиста и догоди. У складу с тим, било је за очекивати да прва техника има највећи допринос јер већина процедура има параметре. Четврта техника је следећа најбоља техника, што сугерише да одлука о уграђивању процедуре у великој мери зависи од процеса превођења. Једина техника чији је допринос упитан када се примени самостално је друга техника. Позитиван допринос остварује

само у случају када се посматра прва позиција. Ипак, на Слика 6.28 приказани су одзиви основног алгоритма без техника, са свим техникама осим друге технике и са свим предложеним техникама. Може се приметити да је одзив без друге технике мањи од одзива који се постиже са свим техникама, што води закључку да и друга техника има позитиван допринос ако се примени после преостале четири технике. Такође, са Слика 6.27 иСлика 6.28 се види и да основни алгоритам проширен свим техникама постиже већи одзив него када се прошири са било којом од предложених техника појединачно. Шта више, применом свих техника заједно постиже се и синергистички ефекат, па је одзив већи за 6.7% до 9.3% у односу на збир доприноса које имају појединачне технике ако се посматра до првих пет позиција.



Слика 6.27 Одзив који основни алгоритам постиже без техника и са сваком од техника појединачно, у зависности од броја позиција које се посматрају.

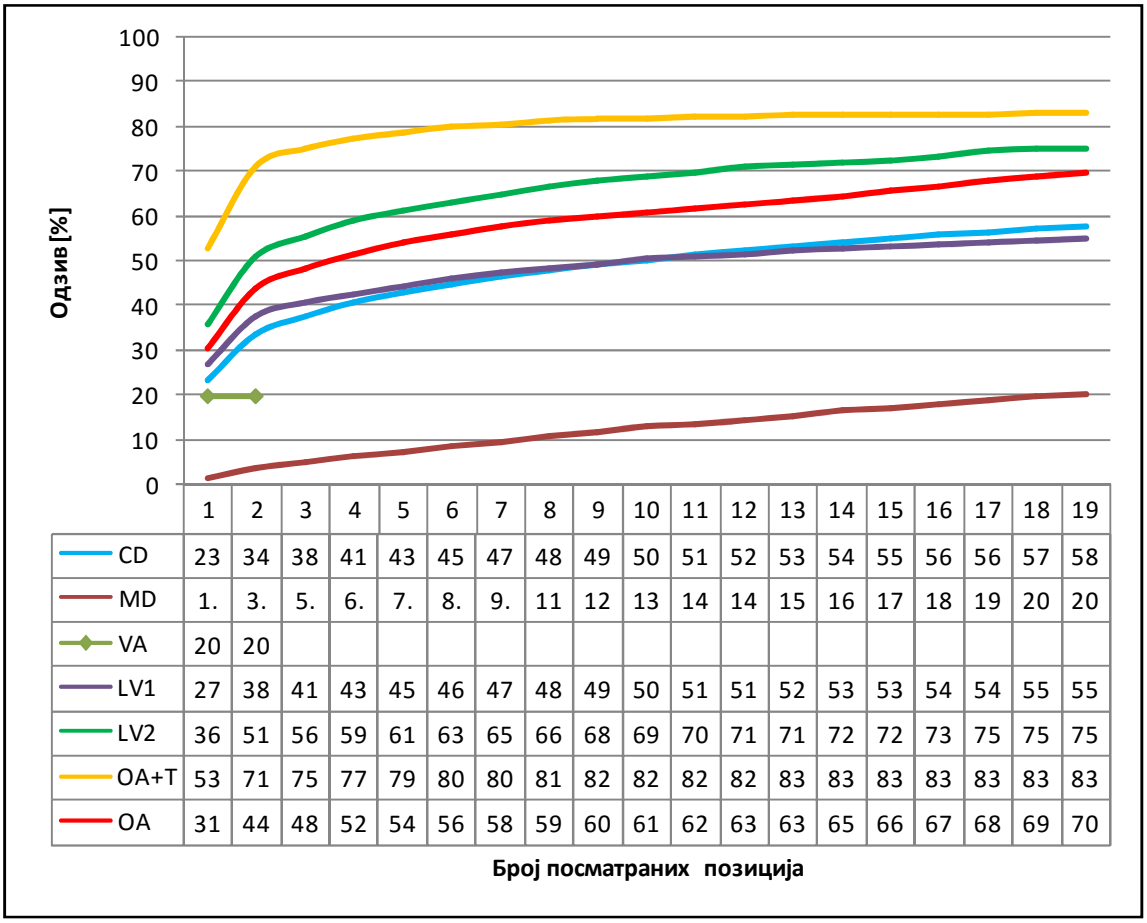


Слика 6.28 Одзив који постиже основни алгоритам без техника, са свим техникама осим друге и са свим техникама, све у зависности од броја позиција које се посматрају.

6.4.2 Експеримент 2.2

У другом експерименту упоређене су имплементације постојећих приступа, направљене за потребе овог рада и основног алгоритма проширеног свим предложеним техникама. Експеримент је изведен над тестовима заснованим на *STAMP* окружењу. Одзиви свих приступа приказани су на Слика 6.29. Основни алгоритам постиже највећи одзив. Посматрајући само прву позицију, одзив је 53%, док већ за посматрање прве две позиције одзив расте на 71%. Треба приметити да је увођење техника дало знатно боље резултате и од предложеног приступа који је био најбољи при поређењу са постојећим алатима. Такође, примећује се да приступ означен са

VA има одзив само ако се посматра прва или прве две позиције. Разлог томе је што поменути приступ не одређује сличност, већ само упарује процедуре, па се рангирање код тог приступа своди на само прве две позиције.

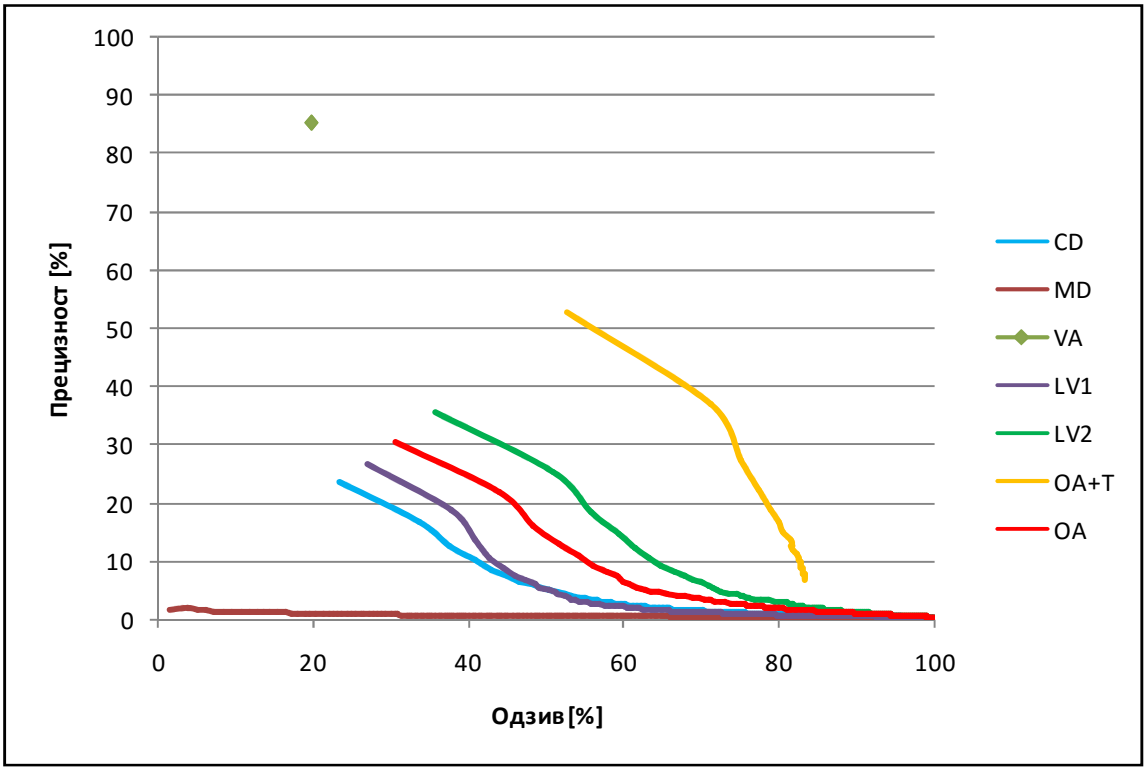


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	23	34	38	41	43	45	47	48	49	50	51	52	53	54	55	56	56	57	58
MD	1.	3.	5.	6.	7.	8.	9.	11	12	13	14	14	15	16	17	18	19	20	20
VA	20	20																	
LV1	27	38	41	43	45	46	47	48	49	50	51	51	52	53	53	54	54	55	55
LV2	36	51	56	59	61	63	65	66	68	69	70	71	71	72	72	73	75	75	75
OA+T	53	71	75	77	79	80	80	81	82	82	82	82	83	83	83	83	83	83	83
OA	31	44	48	52	54	56	58	59	60	61	62	63	63	65	66	67	68	69	70

Слика 6.29 Одзив основног алгоритма проширеног техникама и одабраних постојећих приступа, случај *STAMP* (AO+T - основни алгоритам проширен техникама, OA - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

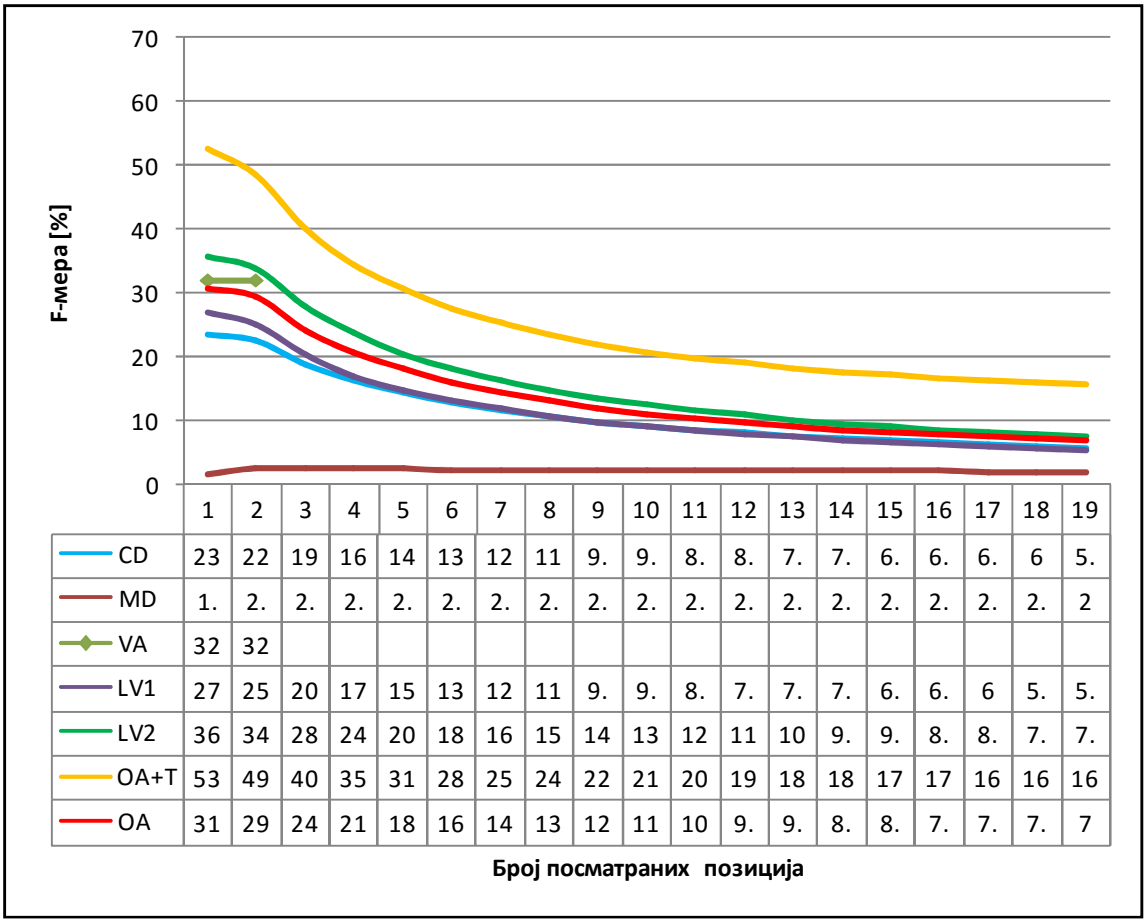
На Слика 6.30 приказане су криве које представљају однос прецизности и одзива. Истичу се два приступа. Први је означен са *VA* (приступ предложен од *Dullien*) и за њега се примећује да има изузетно

високу прецизност од чак 85%, али знатно мањи одзив од свега 20%. Други приступ који се истиче је основни алгоритам проширен техникама који



Слика 6.30 Зависност прецизности од одзива основног алгоритма проширеног техникама и постојећих приступа, случај *STAMP* (OA+T - основни алгоритам проширен техникама, OA - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

постиже највећу прецизност од 53% за исто толики одзив. Такође важан резултат је и прецизност од 37% за одзив од 71%, што значи да је могуће наћи више од 2/3 тражених процедура, а да се при томе више од 1/3 експертског времена утроши на верификацију да процедуре потичу од истог изворног кода, што је из угла експерта изузетно велика помоћ.

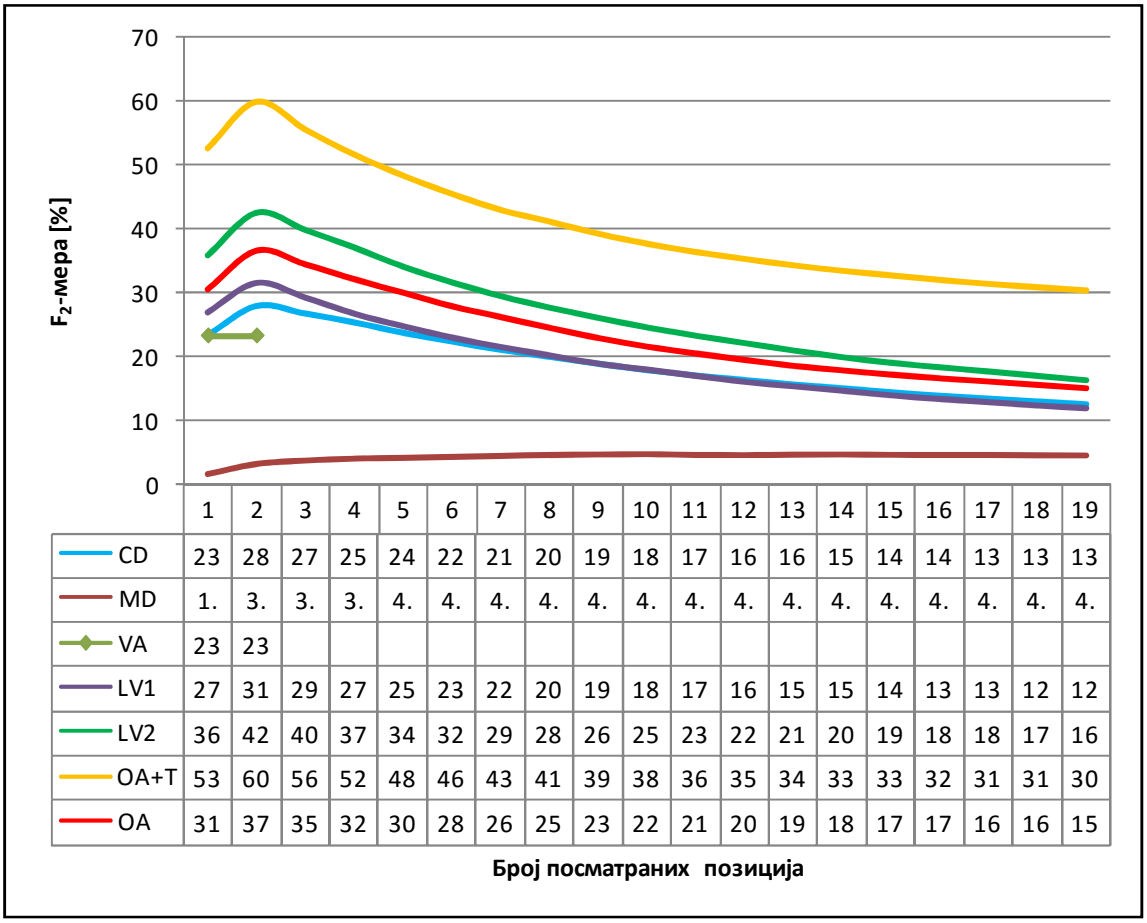


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	23	22	19	16	14	13	12	11	9.	9.	8.	8.	7.	7.	6.	6.	6.	6	5.
MD	1.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2
VA	32	32																	
LV1	27	25	20	17	15	13	12	11	9.	9.	8.	7.	7.	7.	6.	6.	6	5.	5.
LV2	36	34	28	24	20	18	16	15	14	13	12	11	10	9.	9.	8.	8.	7.	7.
OA+T	53	49	40	35	31	28	25	24	22	21	20	19	18	18	17	17	16	16	16
OA	31	29	24	21	18	16	14	13	12	11	10	9.	9.	8.	8.	7.	7.	7.	7

Слика 6.31 *F* мера основног алгоритма проширеног техникама и постојећих приступа у зависности од броја позиција које се посматрају, случај *STAMP* (ОА+Т - основни алгоритам проширен техникама, ОА - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

На Слика 6.31 је приказана *F* мера коју постижу основни алгоритам проширен техникама и остали постојећи приступи. Највеће вредности постиже основни алгоритам проширен техникама за сваки број посматраних позиција. Највећу вредност *F* мере постиже ако се посматра само прва позиција, што сугерише да се најбољи ефекат по питању броја нађених процедура и утрошка времена експерта постиже баш у том случају. Други најбољи приступ је предложени приступ (конфигурација која је коришћена

за поређење са постојећим алатима). Приступ означен са *VA* се у погледу *F* мере не истиче иако постиже велику прецизност што је последица релативно малог одзива поменутог приступа.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	23	28	27	25	24	22	21	20	19	18	17	16	16	15	14	14	13	13	13
MD	1.	3.	3.	3.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.	4.
VA	23	23																	
LV1	27	31	29	27	25	23	22	20	19	18	17	16	15	15	14	13	13	12	12
LV2	36	42	40	37	34	32	29	28	26	25	23	22	21	20	19	18	18	17	16
OA+T	53	60	56	52	48	46	43	41	39	38	36	35	34	33	33	32	31	31	30
OA	31	37	35	32	30	28	26	25	23	22	21	20	19	18	17	17	16	16	15

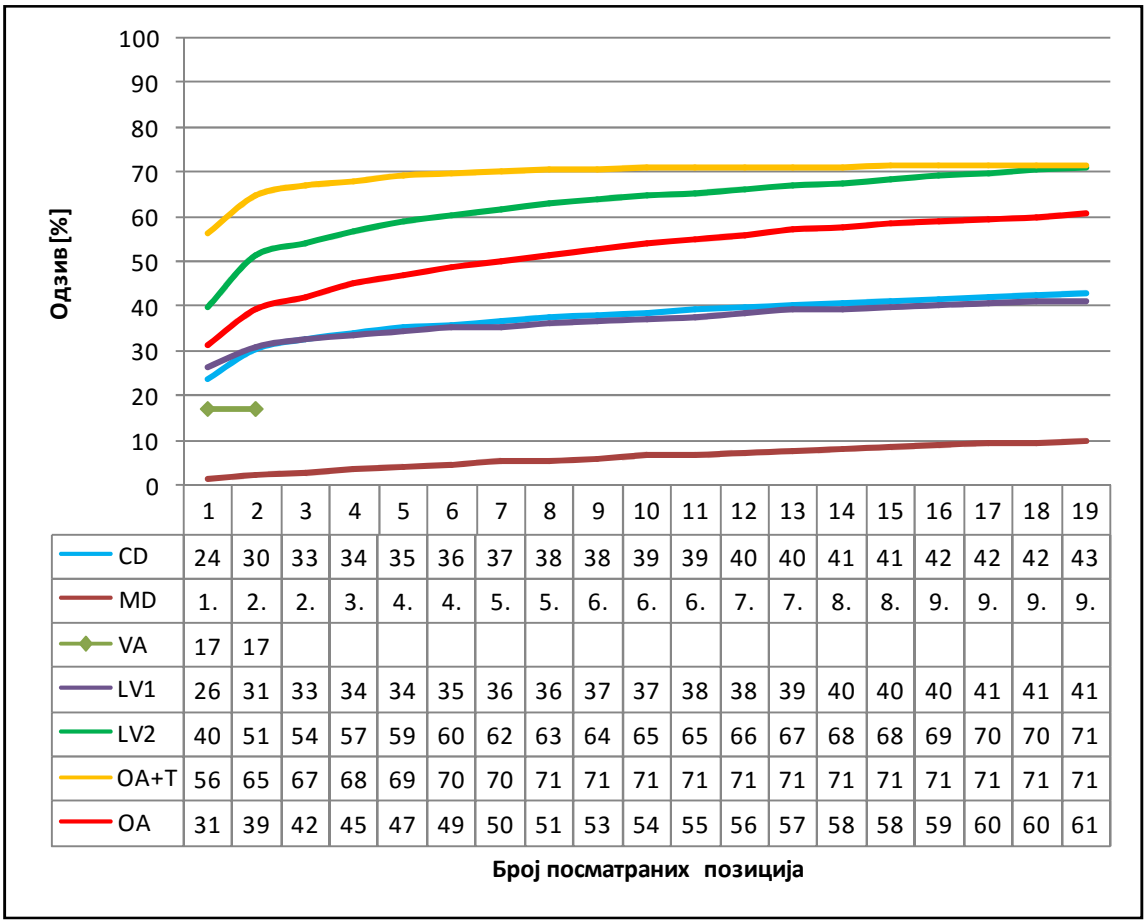
Слика 6.32 F_2 мера основног алгоритма проширеног техникама и постојећих приступа у зависности од броја позиција које се посматрају, случај *STAMP* (ОА+Т - основни алгоритам проширен техникама, ОА - основни алгоритам, *CD - ACD, LV1 - BAT, LV2* - предложени приступ, *MD - Opcode sequence, VA - Dullien*).

На Слика 6.32 је приказана F_2 мера основног алгоритма проширеног техникама и постојећих приступа. У погледу F_2 мере предложене технике се

још више истичу. Ипак, када се да предност одзиву, што F_2 мера и ради, највећа вредност се постиже када се посматрају прве две позиције.

6.4.3 Експеримент 2.3

У трећем експерименту поновљено је поређење имплементација постојећих приступа које су за ову прилику направљене и основног алгоритма проширеног свим предложеним техникама. Експеримент је



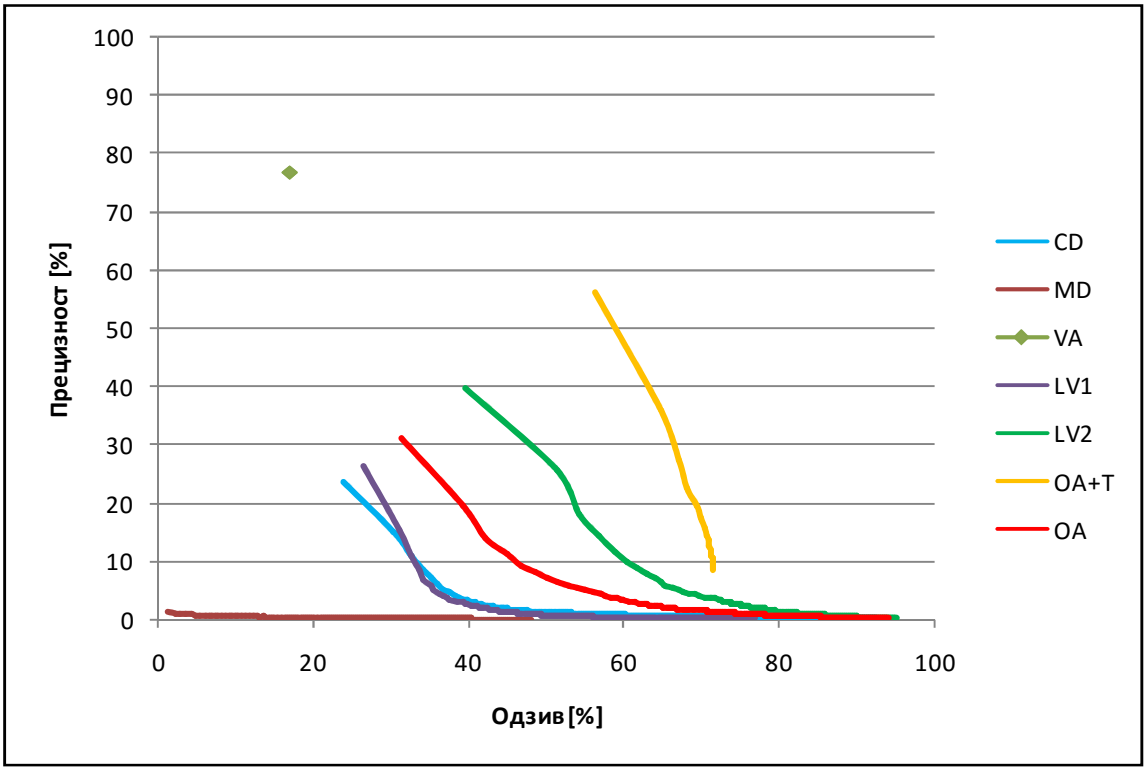
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	24	30	33	34	35	36	37	38	38	39	39	40	40	41	41	42	42	42	43
MD	1.	2.	2.	3.	4.	4.	5.	5.	6.	6.	6.	7.	7.	8.	8.	9.	9.	9.	9.
VA	17	17																	
LV1	26	31	33	34	34	35	36	36	37	37	38	38	39	40	40	40	41	41	41
LV2	40	51	54	57	59	60	62	63	64	65	65	66	67	68	68	69	70	70	71
OA+T	56	65	67	68	69	70	70	71	71	71	71	71	71	71	71	71	71	71	71
OA	31	39	42	45	47	49	50	51	53	54	55	56	57	58	58	59	60	60	61

Слика 6.33 Одзив основног алгоритма проширеног техникама и одабраних постојећих приступа, случај *BusyBox* (OA+T - основни алгоритам проширен техникама, OA - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

изведен над тестовима заснованим на *BusyBox* окружењу. Одзиви свих приступа приказани су на Слика 6.33. Резултати као и однос упоређених приступа су слични резултатима постигнутим на тестовима базираним на *STAMP*. Примећује се да ако се посматра прва позиција, предложене технике су помогле да основни алгоритам постигне одзив 56%. Међутим, приликом посматрања већег броја позиција, одзив основног приступа проширеног предложеним техникама улази у засићење. Ако се посматра првих 19 позиција, примећује се да одзив предложеног приступа (*LV2*) сустиже одзив до којег се долази захваљујући техникама. Разлог уласка одзива у засићење је пета техника која елиминише одређене процедуре из разматрања, међу којима се у овом случају налази и одређени број оних које потичу од изворног кода тражене процедуре. Разлог томе би могао бити просечна величина процедуре која је у случају *BusyBox* пакета већа више од три пута у односу на *STAMP* окружење. Стога би требало испитати да ли и како величина процедуре утиче на вредност границе која се користи у петој техници.

На Слика 6.34 приказане су криве које представљају однос прецизности и одзива у случају тестова базираних на *BusyBox* пакету. И у овом случају се истичу два приступ. Први приступ, означен са *VA* (приступ који је предложио *Dullien*), постиже високу прецизност, мада нешто мању него у случају тестова базираних на *STAMP* окружењу. Други приступ је основни алгоритам проширен техникама и код њега се примећује незнатан

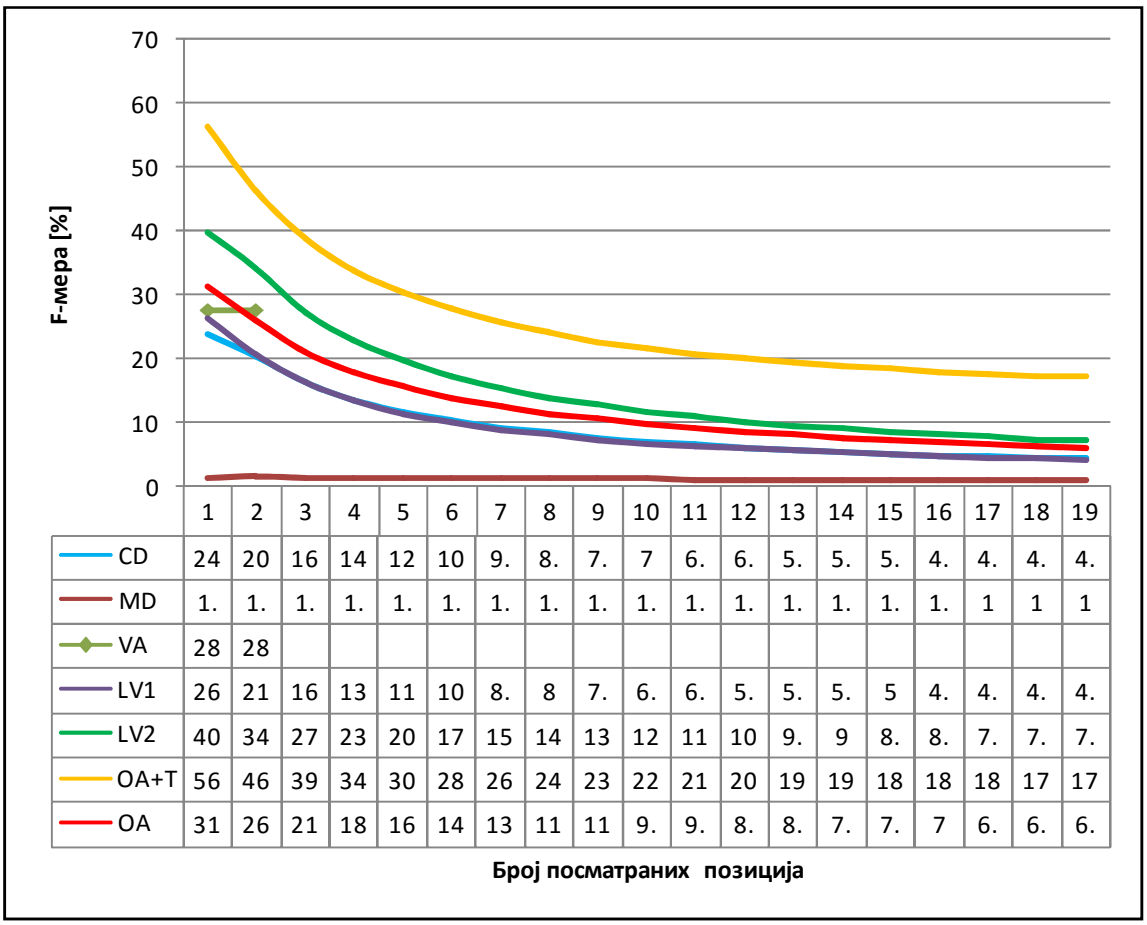
пораст прецизности и одзива у односу на тестове базиране на *STAMP* окружењу.



Слика 6.34 Зависност прецизности од одзива основног алгоритма проширеног техникама и постојећих приступа, случај *BusyBox* (OA+T - основни алгоритам проширен техникама, OA - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

На Слика 6.35 приказана је *F* мера коју постижу основни алгоритам проширен техникама и остали постојећи приступи у случају тестова базираних на *BusyBox* пакету. Поређењем са Слика 6.31 примећује се сличност резултата. Највеће вредности и у овом случају постиже основни алгоритам проширен техникама за сваки број посматраних позиција. Највећу вредност *F* мере постиже основни алгоритам проширен техникама ако се

посматра само прва позиција, што је у сагласности са резултатима и закључком добијеним у претходном експерименту.

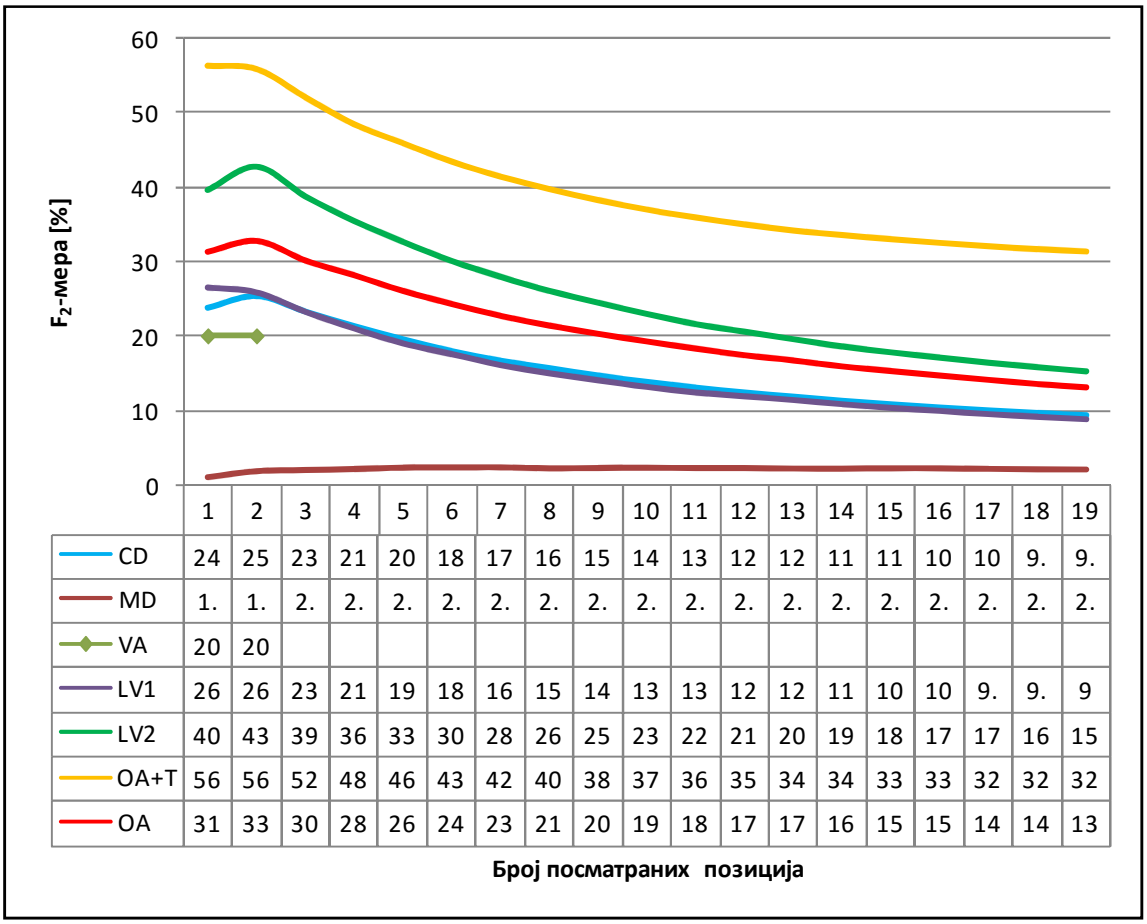


	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	24	20	16	14	12	10	9.	8.	7.	7	6.	6.	5.	5.	5.	4.	4.	4.	4.
MD	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1.	1	1	1
VA	28	28																	
LV1	26	21	16	13	11	10	8.	8	7.	6.	6.	5.	5.	5.	5	4.	4.	4.	4.
LV2	40	34	27	23	20	17	15	14	13	12	11	10	9.	9	8.	8.	7.	7.	7.
OA+T	56	46	39	34	30	28	26	24	23	22	21	20	19	19	18	18	18	17	17
OA	31	26	21	18	16	14	13	11	11	9.	9.	8.	8.	7.	7.	7	6.	6.	6.

Слика 6.35 *F* мера основног алгоритма проширеног техникама и постојећих приступа у зависности од броја позиција које се посматрају, случај *BusyBox* (ОА+Т - основни алгоритам проширен техникама, ОА - основни алгоритам, *CD - ACD, LV1 - BAT, LV2* - предложени приступ, *MD - Opcode sequence, VA - Dullien*).

На Слика 6.36 је приказана F_2 мера основног алгоритма проширеног техникама и постојећих приступа. Резултати су слични резултатима из другог експеримента и опет је приметно велика разлика постигнућа предложених техника у односу на други најбољи алат. Ипак постоји мала

разлика ако се посматрају две највеће вредности F_2 мере основног алгоритма проширеног техникама. Наиме, ове две вредности су приближно исте, па би се могао донети закључак да је подједнако ефикасно посматрати само прву или прве две позиције.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
CD	24	25	23	21	20	18	17	16	15	14	13	12	12	11	11	10	10	9.	9.
MD	1.	1.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.	2.
VA	20	20																	
LV1	26	26	23	21	19	18	16	15	14	13	13	12	12	11	10	10	9.	9.	9
LV2	40	43	39	36	33	30	28	26	25	23	22	21	20	19	18	17	17	16	15
OA+T	56	56	52	48	46	43	42	40	38	37	36	35	34	34	33	33	32	32	32
OA	31	33	30	28	26	24	23	21	20	19	18	17	17	16	15	15	14	14	13

Слика 6.36 F_2 мера основног алгоритма проширеног техникама и постојећих приступа у зависности од броја позиција које се посматрају, случај *BusyBox* (OA+T - основни алгоритам проширен техникама, OA - основни алгоритам, *CD - ACD*, *LV1 - BAT*, *LV2* - предложени приступ, *MD - Opcode sequence*, *VA - Dullien*).

6.5 Одговори на истраживачка питања

Евалуације предложеног приступа и предложених техника су понудиле одговоре на истраживачка питања постављена на крају Секције 3. Одговор на прво питање које пита, да ли додавање нових метрика доприноси повећању одзива предложеног приступа у односу на употребу метрика које су инспирисане одговарајућим метрикама намењеним вишим програмским језицима, добијен је у експерименту 1.1 евалуације предложеног приступа (Секција 6.2.1). Експеримент је показао да је одзив већи за све бројеве посматраних позиција. Највећи релативни пораст је био у случају када се посматра само прва позиција и износио је 44%.

Одговор на друго питање које пита, да ли одзив који постиже предложени приступ зависи од компајлера, нивоа оптимизације и контекста у којем је преведена тражена процедура, добијен је у експерименту 1.2 евалуације предложеног приступа. У посматраним експериментима се показало да избор преводиоца којим ће процедура бити преведена има утицаја на резултате и да се употребом *Keil* преводиоца постиже највећи одзив. Такође, знатно већи одзив добија се уколико се погоди исти преводилац којим је преведена процедура. По питању нивоа оптимизације, незнатне разлике се појављују тек за већи број посматраних позиција и то у прилог употреби нултог нивоа оптимизације. Показало се и да избор окружења у којем се преводи тражена процедура такође има утицаја на процес поређења и да одзив може да варира за 15% по апсолутној вредности.

Одговор на треће питање које пита, да ли предложени приступ постиже већи одзив од постојећих алата, добијено је у експерименту 1.3 евалуације предложеног приступа и још једном верификовано у експерименту 1.4. Оба пута највећи одзив је добијен употребом предложеног приступа. Ако се погледају и експерименти 2.2 и 2.3 евалуације предложених техника, може се приметити да предложени приступ чак и без техника даје већи одзив од свих тестираних постојећих приступа.

Одговор на четврто питање, које пита да ли је појединачним додавањем нових техника за ублажавање негативних последица поређења на основу предложених софтверских метрика могуће добити већи одзив од одзива предложеног приступа, добијен је у експерименту 2.1 евалуације предложених техника. Четири технике су дале несумњив допринос, док је четврта техника имала скроман допринос само када се посматра прва позиција. Ипак, поменута техника даје допринос за сваки број посматраних позиција уколико се примени заједно са преостале четири. Тада долази до синергистичког ефекта па је укупно повећање одзива веће од збира доприноса појединачних техника, што је уједно и одговор на пето питање које пита да ли је додавањем свих техника заједно могуће добити синергистички ефекат.

Одговор на шесто питање, које пита да ли је уз употребу предложених техника могуће постићи већи одзив од одзива постојећих алата и приступа, добијен је у експериментима 2.2 и 2.3 евалуације предложених техника. Из експеримената се види да предложене технике примењене на основни

алгоритам постижу већи одзив од свих тестираних приступа укључујући и евалуирану конфигурацију предложеног приступа. У комбинацији са резултатима експеримената 1.3 и 1.4 евалуације предложеног приступа може се закључити да је одзив до којег су довеле предложене технике значајно већи и од одзива свих тестираних постојећих алата.

Поред приказане евалуације спроведена су иницијална истраживања и случају када је код преведен за *x86* архитектуру [70]. Прилагођен је и евалуиран само основни приступ. Евалуација је спроведена аналогно евалуацији спроведеној у овом раду, с тим да су коришћени преводиоци за *x86* архитектуру. Резултати евалуације за случај *x86* архитектуре су у складу са резултатима приказаним у овом раду, одакле се закључује да је предложени приступ применљив и на *x86* архитектуру.

7. Закључак

Енорман раст софтверске индустрије довео је до мноштва различитих модела пословања софтверских компанија. Један од видова пословања је развој софтвера са јавно доступним изворним кодом, за који је неопходно платити накнаду у случају комерцијалне употребе. Међутим, неретко се дешава да се такав софтвер, најчешће у облику софтверских библиотека, употреби на начин који није у складу са лиценцом под којом је софтвер издат. Откривање да је до дошло до употребе софтверске библиотеке у случају када је доступан само бинарни код производа представља веома комплексан задатак, што је последица непознатог процеса превођења изворног кода у бинарни код. У првом делу рада направљен је преглед области у којима постоји потреба за поређењем кода, као и приступа и алата из поменутих области. Анализом је утврђено да постојећи алати и приступи постижу скромне резултате ако се примене на откривање употребе софтверске библиотеке. Стога се приступило развоју идеје о увођењу новог приступа којим би се у поменутом случају добили знатно бољи резултати.

Предложен је нови приступ намењен за поређење процедура на бинарном нивоу који треба да убрза процес откривања употребе софтверске библиотеке доступне у изворном коду. Приступ процењује сличност

процедура на основу софтверских метрика и рангира процедуре из бинарног кода програма у складу са процењеним сличностима према траженој процедури. Приступ предлаже низ софтверских метрика, компаратора, трансформатора и формула. Предложено је 19 софтверских метрика од којих су неке инспирисане метрикама намењеним за више програмске језике, док је остатак осмишљен наменски за примену на бинарном коду. Уведене су скаларне метрике које се састоје од само једне вредности и векторске, које се састоје од низа вредности са ознакама. За поређење вредности скаларних метрика предложен је један компаратор, док су за поређење векторских метрика предвиђена два различита компаратора. Резултат компаратора је парцијална мера сличности коју је даље могуће трансформисати једним од 6 предложених трансформатора у вредност која верније репрезентује ниво сличности упоређених процедура. Трансформисане вредности се даље комбинују у јединствену вредност која представља тоталну меру сличности две упоређене процедуре. За рачунање тоталне мере сличности предложено је седам различитих формула, при чему се формуле могу примењивати на два начина. Поред предложених формула испитане су могућности примене три технике машинског учења, и то машине са векторима подршке (енг. *Support Vector Machine - SVM*), Наивни Бајес(енг. *Naive Bayes - NB*) и к најближих суседа (енг. *k - Nearest Neighbors - kNN*).

Евалуација је показала да предложени приступ има већи одзив од свих постојећих алата са којима је упоређен. Ипак, уочено је да поређење на нивоу предложених метрика има и недостатака, па је предложен скуп од пет техника од којих се очекује да значајно повећају одзив. Три предложене

технике су намењене ублажавању негативних ефеката које на поређење имају различити начини рада са аргументима, локалним променљивим и регистрима, као и уграђивање процедура на место позива. Преостале две технике ублажавају негативне ефекте поређења процедура на основу софтверских метрика. Једна од њих додатно издваја парцијалне информације о редоследу инструкција, док друга елиминише процедуре које се по одређеном критеријуму разликују од тражене процедуре преко одређене границе.

Евалуацијом предложених техника утврђено је да се помоћу њих постиже знатно већи одзив него са било којим другим постојећим приступом и алатом коришћеним у поређењу, укључујући и претходно предложени приступ. Највећи постигнути одзив је 53% у случају тестова заснованих на *STAMP* окружењу, док је у случају тестова заснованих на *BusyBox* окружењу постигнут одзив од 56%. Ипак, примећује се да алат *CCFinder* и приступ који предлаже *Dullien* постижу највећу прецизност, али одзив остаје знатно мањи од одзива који се постиже уз помоћ предложених техника. Стога су упоређене и вредности *F* мера. Највеће вредности *F* мере за сваки број посматраних позиција постигнуте су уз помоћ предложених техника. Највећа вредност добијена је ако се као успех посматра само проналазак тражене процедуре на првој позицији, што сугерише да је за најефикаснију примену предложеног приступа потребно посматрати само прву позицију. Пошто је проналазак процедуре веома важан, па и по цену нешто већих улагања у експертско време, посматрана је и F_2 мера која одзиву даје већи значај од прецизности. И по питању ове мере, уз помоћ предложених техника се добија највећа

вредност за сваки број посматраних позиција. За разлику од *F* мере, највећа вредност F_2 мере се постиже ако се посматрају прве две позиције, сугеришући да у случају када је важније наћи процедуру и по цену мање ефикасно утрошеног експертског времена, претрагу треба проширити и на другу позицију.

Иако је предложени приступ, заједно са предложеним техникама дао знатно бољи одзив од других алата и приступа са којима је упоређен, истраживање је покрило само један мали део домена проблема. Усавршавања и даљи правци развоја су разноврсни. Један веома значајан правац, који је сам по себи преобиман, јесте систематична анализа оптимизација већег броја преводилаца у циљу откривања непроменљивих делова кода који би се при поређењу могли користити. Таква анализа би могла резултовати и у препознавању преводиоца који је при превођењу коришћен, као и опција које су том приликом употребљене и тиме значајно олакшати процес поређења. Ипак, не може се гарантовати да неки од преводилаца неће променити скуп оптимизација које користи, као и да се неће појавити нови преводилац са другим скупом оптимизација и тиме довести до ситуација које претходна анализа није обрадила.

Поред анализе оптимизација и увођења нових, бољих метрика један правац може бити и испитивање ефеката различитих начина тумачења појединих метрика и начина комбиновања резултата који се од метрике добију. У том смислу требало би испитати како одређене карактеристике процедуре, попут величине, утичу на поређење. Такође, у предложеном

приступу све метрике су имале линеаран, и за одабрану конфигурацију увек исти утицај на рачунање тоталне мере сличности упоређених процедура. Интересантно би било испитати и да ли одређене вредности неке метрике имају утицај на значај вредности те исте или неке друге метрике.

Још један интересантан правац истраживања јесте додатна трансформација бинарног кода у циљу уклањања разлика насталих превођењем изворног кода. Као трансформацију, могуће је применити нормализацију кода која има за циљ да различито преведене делове кода сведе на исти, предефинисани облик. За ту намену се могу користити и различите оптимизације које преводиоци иначе користе, попут разних трансформација петљи и елиминације вишеструког рачунања истог подизраза. Такође, као трансформацију је могуће применити и пребацивање кода у неки други облик са једноставнијим инструкцијским скупом, попут *SSA* (енг. *Static Single Assingment*) форме, за коју постоје разрађени алгоритми за оптимизацију и даљу трансформацију.

Један од недостатака предложеног приступа једним малим делом надомештен је техником која пребројава секвенце инструкција одређене унапред дефинисане дужине. Ипак, на тај начин задржана је само информација о редоследу инструкција у меморији, али не и о њиховим зависностима по графу контроле тока и графу тока података. Анализа поменутих зависности и начина на које се из зависности могу издвојити битни делови који се једноставно могу упоредити, има велики значај и представља озбиљан изазов за истраживаче. Један једноставан пример била

би анализа тока података у циљу откривања броја параметара процедуре, што представља информацију на коју коришћени преводилац и опције коришћене приликом превођења имају ограничен утицај.

Анализа зависности инструкција по подацима и контроли тока може бити веома интересантна. Ипак, суштина проблема се може посматрати и кроз питање да ли две процедуре обављају исти посао? Такво питање се сусреће и у другим областима у којима се ради поређење кода. У области одржавања софтвера помињу се клонови типа 4 који представљају семантичке клонове. У области откривања злонамерног кода било би веома погодно када би се могле наћи функционалности које се појављују у неком познатом злонамерном коду. Уколико би било могуће из процедуре издвојити семантику и на исти начин описати произвољан алгоритам, тада би се добио веома моћан приступ који би поред поменутих области, примену могао наћи и у процесу провере нарушавања патентних права.

8. Референце

[1] S. Comino and F. Manenti, Dual licensing in open source software markets, Information Economics and Policy, Vol. 23, No. 3, pp. 234-242, 2011.

[2] P.E. Chaudhry and A. Zimmerman, Protecting your intellectual property rights: Understanding the role of management, governments, consumers and pirates, Springer Science & Business Media, 2012.

[3] C. Cifuentes and K. J. Gough, Decompilation of binary programs, Software: Practice and Experience, Vol. 25, No. 7, pp. 811-829, 1995.

[4] 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, 2015.

[5] Z. Radivojević, M. Cvetanović, and S. Stojanović, Comparison of Binary Procedures: A Set of Techniques for Evading Compiler Transformations, The Computer Journal, Vol. n/a, No. n/a, pp. n/a, n/a (bezuslovno prihvaćen).

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

[7] C.K. Roy, and J.R. Cordy, A survey on software clone detection research, Technical Report 541, Queen's University at Kingston, 2007.

[8] 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.

[9] J. Hage, P. Rademaker, and N. van Vugt, A comparison of plagiarism detection tools, Technical Report, Utrecht University at Utrecht, The Netherlands (2010).

[10] 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.

[11] C.D. Manning, P. Raghavan, and H. Schütze, Introduction to information retrieval, Cambridge university press, 2008.

[12] C.K. Roy, M.F. Zibran, and R. Koschke, The vision of software clone management: Past, present, and future (keynote paper), Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE, 2014.

[13] I.J. Davis and M.W. Godfrey, From whence it came: Detecting source code clones by analyzing assembler, 17th Working Conference on Reverse Engineering (WCRE), pp. 242-246, IEEE, October 2010.

[14] A. Hemel, K.T. Kalleberg, R. Vermaas, and E. Dolstra, Finding software license violations through binary code clone detection, Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 63-72, ACM, May 2011.

[15] I. Santos, F. Brezo, J. Nieves, Y.K. Penya, B. Sanz, C. Laorden, and P.G. Bringas, Idea: Opcode-sequence-based malware detection, In Engineering Secure Software and Systems, Springer Berlin Heidelberg, pp. 35-43, 2010.

[16] T. Dullien and R. Rolles, Graph-based comparison of executable objects (english version), Proceedings of SSTIC 05, Rennes, France, pp. 1-13, 2005.

[17] S. Ducasse, M. Rieger, and S. Demeyer, A language independent approach for detecting duplicated code, Proceedings of IEEE International Conference on Software Maintenance (ICSM'99), IEEE, Oxford, England, pp. 109-118, 1999.

[18] M.S. Uddin, C.K. Roy, K. Schneider, and A. Hindle, On the effectiveness of simhash for detecting near-miss clones in large scale software systems, 18th Working Conference on Reverse Engineering (WCRE), IEEE, pp. 13-22, 2011.

[19] B.S. Baker, Finding clones with dup: Analysis of an experiment, IEEE Transactions on Software Engineering, Vol. 33, No. 9, pp. 608-621, 2007.

[20] 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, 2002.

[21] E. Merlo, Detection of plagiarism in university projects using metrics-based spectral similarity, Internat, Begegnungs-und Forschungszentrum für Informatik, 2007.

[22] N. Davey , P. Barson, S. Field, R. Frank, and D. Tansley, The development of a software clone detector, International Journal of Applied Software Technology, Vol. 1, No. 3/4, pp. 219-236, 1995.

[23] R. Falke, P. Frenzel, and R. Koschke, Empirical evaluation of clone detection using syntax suffix trees, Empirical Software Engineering, Vol. 13, No. 6, pp. 601-643, 2008.

[24] L. Jiang, G. Misherghi, Z. Su, and S. Glondu, Deckard: Scalable and accurate tree-based detection of code clones, In Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society, pp. 96-105, 2007.

[25] 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, ACM, pp. 872-881 , 2006.

[26] J.Krinke and S. Breu, Control-flow-graph-based aspect mining, 1st Workshop on Aspect Reverse Engineering, 2004.

[27] J.R. Cordy and C.K. Roy, The NiCad clone detector, 19th International Conference on Program Comprehension, IEEE, pp. 219-220, 2011.

[28] U. Manber, Finding Similar Files in a Large File System, Usenix Winter, Vol. 94, pp. 1-10, 1994.

[29] J.H. Johnson, Identifying redundancy in source code using fingerprints, Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering, IBM Press, Vol. 1, 1993.

[30] H.J. Johnson, Visualizing textual redundancy in legacy source, Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 1994.

[31] S. Ducasse, M. Rieger, and S. Demeyer, A language independent approach for detecting duplicated code, Proceedings of IEEE International Conference on Software Maintenance 1999 (ICSM'99), IEEE, 1999.

[32] M. Rieger, Effective clone detection without language barriers, PhD thesis at the University of Bern, Switzerland, 2005.

[33] R. Wettel, and R. Marinescu, Archeology of code duplication: Recovering duplication chains from small duplication fragments, Seventh International Symposium on Symbolic and Numeric Algorithms for Scientific Computing 2005 (SYNASC 2005), IEEE, 2005.

[34] A. Marcus and J. Maletic, Identification of high-level concept clones in source code, Proceedings of 16th Annual International Conference on Automated Software Engineering 2001 (ASE 2001), IEEE, 2001.

[35] B.S. Baker and R. Giancarlo, Sparse dynamic programming for longest common subsequence from fragments, Journal of algorithms Vol. 42, No. 2, pp. 231-254, 2002.

[36] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code, OSDI, Vol. 4, No. 19, pp. 289-302, 2004.

[37] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, CP-Miner: Finding copy-paste and related bugs in large-scale software code, IEEE Transactions on Software Engineering, Vol. 32, No. 3, pp. 176-192, 2006.

[38] R. Agrawal and R. Srikant, Mining sequential patterns, Proceedings of the Eleventh International Conference on Data Engineering 1995, IEEE, 1995.

[39] S. Schleimer, D.S. Wilkerson, and A. Aiken, Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, ACM, 2003.

[40] L. Prechelt, G. Malpohl, and M. Philippsen, Finding plagiarisms among a set of programs with JPlag, Journal of Universal Computer Scienc, Vol. 8, No. 11, pp. 1016-1038, 2002.

[41] I.D. Baxter, A. Yahin, L. Moura, M.S. Anna, and L. Bier, Clone detection using abstract syntax trees, Proceedings of International Conference on Software Maintenance 1998, IEEE, pp. 368-377, 1998.

[42] W. Yang, Identifying syntactic differences between two programs, Software: Practice and Experience, Vol. 21, No. 7, pp. 739-755, 1991.

[43] V. Wahler, D. Seipel, J.W. von Gudenberg, and G. Fischer, Clone detection in source code by frequent itemset techniques, Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation (SCAM'04), IEEE, pp. 128-135, 2004

[44] J. Han, M. Kamber, and J. Pei, Data mining: concepts and techniques, Elsevier, 2011.

[45] W.S. Evans, C.W. Fraser, and F. Ma, Clone detection via structural abstraction, Software Quality Journal, Vol. 17, No. 4, pp. 309-330, 2009.

[46] R. Komondoor and S. Horwitz, Using slicing to identify duplication in source code, Static Analysis, Springer Berlin Heidelberg, pp. 40-56, 2001.

[47] R. Komondoor, Automated duplicated-code detection and procedure extraction, PhD thesis at University of Wisconsin–Madison, 2003.

[48] J. Krinke, Identifying similar code with program dependence graphs, Proceedings of Eighth Working Conference on Reverse Engineering 2001, IEEE, 2001.

[49] J. Mayrand, C. Leblanc, and E.M. Merlo, Experiment on the automatic detection of function clones in a software system using metrics, Proceedings of International Conference on Software Maintenance 1996, IEEE, 1996.

[50] J.-F. Patenaude, E. Merlo, M. Dagenais, and B. Laguë, Extending software quality assessment techniques to java systems, Proceedings of Seventh International Workshop on Program Comprehension 1999, IEEE, pp. 49-56, 1999.

[51] K.A. Kontogiannis, R. DeMori, E. Merlo, M. Galler, and M. Bernstein, Pattern matching for clone and concept detection, In Reverse engineering, Springer US, pp. 77-108, 1996.

[52] G.A. Di Lucca, M. Di Penta, A.R. Fasolino, and P. Granato, Clone analysis in the web era: An approach to identify cloned web pages, Seventh IEEE Workshop on Empirical Studies of Software Maintenance, IEEE, pp. 107-113, 2001.

[53] F. Lanubile and T. Mallardo, Finding function clones in web application, Proceedings. Seventh European Conference on Software Maintenance and Reengineering 2003, IEEE, 2003.

[54] F. Calefato, F. Lanubile, and T. Mallardo, Function clone detection in web applications: a semiautomated approach, Journal of Web Engineering, Vol. 3, pp. 3-21, 2004.

[55] B. De Sutter, B. De Bus, and K. De Bosschere, Sifting out the mud: low level C++ code reuse, ACM SIGPLAN Notices, Vol. 37, No. 11, pp. 275-291, 2002.

[56] K.D. Cooper and N. McIntosh, Enhanced code compression for embedded RISC processors, ACM SIGPLAN Notices, Vol. 34, No. 5, pp. 139-149, 1999.

[57] B. Smith, ARM and Intel battle over the mobile chip's future, Computer, Vol. 41, No. 5, pp. 15-18, 2008.

[58] V. Juričić, Detecting source code similarity using low-level languages, Proceedings of the 33rd International Conference on Information Technology Interfaces (ITI 2011), IEEE, 2011.

[59] C.C. Minh, J.W. Chung, C. Kozyrakis, and K. Olukotun, STAMP: Stanford transactional applications for multi-processing, IEEE International Symposium on Workload Characterization 2008 (IISWC 2008), IEEE, pp. 35-46, 2008.

[60] D. Abbott, Chapter 13 - BusyBox and Linux Initialization, In D. Abbott (eds), Linux for Embedded and Real-time Applications (Third Edition), Newnes, Oxford, 2013.

[61] PeerSecNetworks, MatrixSSL, http://www.matrixssl.org/, Pristup: 01.09.2015.

[62] Hex-Rays, IDA. Dostupno na: https://www.hex-rays.com/products/ida/index.shtml, Pristupljeno 25.08.2015.

[63] M. Merkle, Verovatnoća i statistika, Akademska misao, Beograd 2002.

[64] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Section 16.5. Support Vector Machines, in Numerical Recipes: The Art of Scientific Computing (3rd ed.), New York: Cambridge University Press, 2007.

[65] I. Rish, An empirical study of the naive Bayes classifier, Workshop on empirical methods in artificial intelligence (IJCAI 2001), IBM New York, Vol. 3, No. 22, 2001.

[66] P.A.N. Jeng-Shyang, Q.I.A.O. Yu-Long, and S.U.N. Sheng-He, A fast K nearest neighbors classification algorithm, IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. 87, No. 4, pp. 961-963, 2004.

[67] D. Bruschi, L. Martignoni, and M. Monga, Code normalization for self-mutating malware, IEEE Security & Privacy, Vol. 2, pp. 46-54, 2007.

[68] F. Bouchez, A. Darte, C. Guillon, and F. Rastello, Register allocation: what does the NP-completeness proof of Chaitin et al. really prove? or revisiting register allocation: why and how, In Languages and Compilers for Parallel Computing, Springer Berlin Heidelberg, pp. 283-298, 2007.

[69] M. Arnold, S. Fink, V. Sarkar, and P.F. Sweeney, A comparative study of static and profile-based heuristics for inlining, ACM SIGPLAN Notices, Vol. 35, No. 7, pp. 52-64, 2000.

[70] K. Berta, S. Stojanović, M. Cvetanović, and Z. Radivojević, Estimation of Similarity between Functions Extracted from x86 Executable Files, Serbian Journal of Electrical Engineering, Vol. 12, No. 2, pp. 253-262, 2015.

Биографија аутора

Саша Стојановић је рођен 27. септембра 1982. године у Фочи од оца Доброслава и мајке Руже. У Фочи је завршио основну и први разред гимназије. Даље школовање наставио је у Математичкој гимназији у Београду. За време основног и средњег школовања учествовао и освајао награде на многобројним такмичењима из математике, физике и информатике, међу којима су и два међународна такмичења светског ранга. Електротехнички факултет у Београду уписао је 2001. године. Саша Стојановић је завршио основне студије на Електротехничком факултету за мање од 5 година, при чему је сваки испит положио у првом року. Дипломирао је 15. маја 2006. године на Одсеку за рачунарску технику и информатику, са просечном оценом 9,64. Дипломски рад одбранио је са оценом 10. Докторске студије уписао је 2008. године на Смеру рачунарска техника и информатика.

Од септембра 2006. године ангажован је при Катедри за рачунарску технику и информатику. Дана 22. јануара 2007. изабран за сарадника у настави, а 06. априла 2009. изабран у звање Асистента. Стално је ангажован за потребе држања вежби на табли на већем броју курсева.

Објавио је неколико радова у домаћим и међународним часописима, као и на скуповима националног и међународног значаја. Од тога, један рад је објавио у признатом међународном часопису (СЦИ листа, категорија М22), два рада у међународном часопису (СЦИ листа, категорија М23). У домаћим

часописима има два објављена рада (један категорије М51, други категорије М53). На скуповима међународног значаја објавио је 4 рада, док је на скуповима националног значаја објавио 2 рада. Поред радова, учествовао је и у припреми и извођењу 4 туторијала, од чега су 2 презентована на скуповима међународног значаја (HiPEAC и ISCA), док су друга 2 презентована на скуповима националног значаја. Ангажован је и на једном пројекту Министарства просвете, науке и технолошког развоја, а учествовао је и на већем броју комерцијалних пројеката. Успешно је испратио 4 стручна курса.

Прилог 1.

Изјава о ауторству

Потписани-а Саша Стојановић

број индекса 5011/2007

Изјављујем

да је докторска дисертација под насловом

ПРОЦЕНА СЛИЧНОСТИ ПРОЦЕДУРА У БИНАРНОМ КОДУ

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 01.09.2015.

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Саша Стојановић

Број индекса 5011/2007

Студијски програм

Наслов рада ПРОЦЕНА СЛИЧНОСТИ ПРОЦЕДУРА У БИНАРНОМ КОДУ

Ментор др Вељко Милутиновић, редовни професор, ЕТФ, УБ

Потписани/а Саша Стојановић

Изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао/ла за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду.**

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања 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 докторанда

У Београду, 01.09.2015.

Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

ПРОЦЕНА СЛИЧНОСТИ ПРОЦЕДУРА У БИНАРНОМ 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 Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио/ла.

1. Ауторство

2. Ауторство - некомерцијално

3. Ауторство – некомерцијално – без прераде

4. Ауторство – некомерцијално – делити под истим условима

5. Ауторство – без прераде

6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 01.09.2015.

_____

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 прераде. 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 прераде. 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.