Thesis1

by Monika Singh

FILE	THESIS_FINAL.DOCX (550.68K)		
TIME SUBMITTED	02-JUL-2017 03:14AM	WORD COUNT	10960
SUBMISSION ID	828545624	CHARACTER COUNT	57198

CERTIFICATE

This is to certify that report entitled Monika Singh (2K15/CSE/10) has completed the thesis titled "**Analysing multi-objective Search Algorithms to predict faulty classes**" under my supervision in partial fulfilment of the MASTER OF TECHNOLOGY degree in Software Engineering at DELHI TECHNOLOGICAL UNIVERSITY.

Project Guide
Dr. Ruchika Malhotra
Assistant Professor
Department of Computer Science and Engineering
Delhi Technological University
Delhi -110042

DECLARATION

We hereby declare that the thesis work entitled "**Analysing multi-objective Search Algorithms to predict faulty classes**" which is being submitted to Delhi Technological University, in partial fulfilment of requirements for the award of degree of Master of Technology (Software Engineering) is a bonafide report of thesis carried out by me. The material contained in the report has not been submitted to any university or institution for the award of any degree.

Monika Singh
2K15/CSE/10

ACKNOWLEDGEMENT

I am very thankful to Dr. Ruchika Malhotra (Assistant Professor, Computer Science Eng. Dept.) and all the faculty members of the Computer Science Engineering Dept. of DTU. They all provided immense support and guidance for the completion of the project undertaken by me.

I would also like to express my gratitude to the university for providing the laboratories, infrastructure, testing facilities and environment which allowed me to work without any obstructions.

I would also like to appreciate the support provided by our lab assistants, seniors and peer group who aided me with all the knowledge they had regarding various topics.

Monika Singh
M. Tech. in Software Engineering
Roll No. 2K15/CSE/10

ABSTRACT

Many real world problems involve optimization of multiple objective functions on a feasible variable space. These objective functions are often conflicting and cannot be formulated as a scalar function. Such problems are known as multi-objective optimization (MOO) problems as there are multiple objectives which need to be optimized simultaneously. A recent MOO problem in software engineering domain is the prediction of faulty classes in a software. While faulty classes are predicted, finding a trade-off between two conflicting objectives is essential. The first one is minimizing the number of classes to be recommended and the second one is maximizing the relevance of the solution which is based on the history based and lexical based similarities between the Application program interface (API) document and the bug description. Evolutionary algorithms (EA) seem to be well suited to solve such MOO problems as they parallely generates a set of solutions, ultimately exploiting similarities of solutions by crossover. Previous studies have suggested that EAs show improved performance over other search algorithms for solving MOO problems. This study evaluates the use of two EA's namely the Non-dominated Sorting Genetic Algorithm (NSGA-II) and the Strength Pareto Evolutionary Algorithm 2 (SPEA2) to predict faulty classes. The results are empirically validated on six open source Java projects. They point towards the superiority of the NSGA II algorithm over the SPEA 2 algorithm.

Keywords:

Multi objective optimization (MOO), Evolutionary Algorithms (EA); fault prediction.

TABLE OF CONTENTS

CERTIFICATE **i**

DECLARATION

ii

ACKNOWLEDGEMENT

iii

ABSTRACT

iv

FIGURES AND TABLES

vi

CHAPTER 1 INTRODUCTION 1-4

1.1 Research Questions 2-3

1.2 Motivation of study 3

1.3 Organization of thesis 3-4

CHAPTER 2 LITERATURE REVIEW 5-7

2.1 Applications of MOEA 5-6

2.1.1 Software reliability prediction 5

2.1.2 Analysis of MOEA 5

2.1.3 Cross Project Defect Prediction 6

2.1.4 Maintainability defect prediction 6

2.2 Defect Prediction using Information Retrieval 6-8

2.2.1 BugLocator 7

2.2.2 Debug Advisor 7

2.2.3 Bug Scout 7

2.2.4 BLUiR 7

CHAPTER 3 RESEARCH METHODOLOGY ……..9-14
3.1 Overview of the Recommendation Framework ……9-10
3.2 NSGA II ……10-12
3.3 SPEA 2 ……12-14

CHAPTER 4 EMPIRICAL STUDY DESIGN ……...15-20
4.1 Description of Data Sets ……15-16
4.2 Validation Method ……16
4.3 Solution Representation ……16-17
4.4 Description of Objectives ……17-19
4.5 Performance Evaluation ……19-20

CHAPTER 5 RESULT AND ANALYSIS……21-30
5.1 Results of RQ1 ……21-25
5.2 Results of RQ2 ……25-28
5.3 Results of RQ3 ……28-30

CHAPTER 6 THREATS TO VALIDITY ……31-32
6.1 Internal Validity ……31
6.2 External Validity ……31-32

CHAPTER 7 CONCLUSION AND FUTURE WORK……33-34
7.1 Summary ……..33
7.2 Proposed Work ……34

Appendix ……35-36
References ……..……37-41

FIGURES AND TABLES

Figure 1 Approach overview 9
Figure 2: NSGA II 10
Figure 3: NSGA II flowchart and algorithm 11
Figure 4: Flowchart of SPEA 2 algorithm 13
Figure 5: SPEA 2 Algorithm 14
Figure 6: An eclipse bug report (Bug ID: 384108) 17
Figure 7: Median performance metrices values of 30 runs for k=5 22
Figure 8: Median Performance metrics values of 30 runs for k=10 23
Figure 9: Median Performance metrices values of 30 runs for k=15 23
Figure 10: Median Performance metrices values of 30 runs for k=20 24
Figure 11: Median of different mono objective technique for k =5 26
Fig 12: Execution time (ms) of NSGA II and SPEA 2 30
Table 1: Data Set Description 16
Table 2: Friedman test ranking for k=5 27
Table 3: Friedman test ranking for k=10 27
Table 4: Friedman test ranking for k=15 27
Table 5: Friedman test ranking for k=20 28
Table 6: Wilcoxon test result 29
Table A.1: Median of different mono-objective technique for k =10, 15 and 20 35-36

CHAPTER 1

Introduction

Software testing is used to detect bugs in a software [1]. A bug is an erroneous behaviour of the software which leads to an unexpected result. Bug reports typically consist of the description of the bugs and are very useful for the maintenance of the software. It helps the developer to understand the problem, which was encountered while using the software. Bug reports inform the developer about those software components which need maintenance or correction [3]. Thus, a developer relies upon the bug report's description to find the origination of a specific bug. However, due to insufficient and unclear information provided in the bug report the task of the developer becomes tedious. This study automates the approach of determining the code segments, which need to be investigated to eliminate bugs in a software. This automation would make the process of correcting bugs less time consuming and would improve the productivity of the developer [2]. Most of the present approaches to this problem are based on lexical matching, but there is difference in the language of the code and the bug descriptions, which limits the efficiency of existing approaches.

The approach used in this study for predicting relevant components of a software for bug correction overcomes the problems of existing approaches. It is based on the following observations: The first observation is that the API of the classes and the methods is more useful to derive the similarity between the bug description and the code fragment [2]. Secondly, the classes which have been fixed recently based on any bug report are more prone to contain a bug if the bug reports are similar. Thirdly, the classes which have been fixed recently are more prone to contain bugs than the class fixed long time back. Fourthly, the classes fixed frequently are more prone to be erroneous. Apart from these observations, it should also be noted that the number of recommended classes should be minimized in order to make the task of bug location easier.

To solve this bug localization problem and provide an effective result, multi objective optimization have been used. There is a need to balance two main conflicting objectives, i.e. maximization of lexical based and history based similarities and minimization of the number of classes to be recommended. We investigate two EA for this purpose, namely NSGA II and

SPEA 2. The two algorithms are considered better than the other evolutionary algorithms on the basis of better distribution and broader range of solutions [30, 43]. Both the investigated algorithms are validated on six large open source projects which consist of over 2000 bug reports and their descriptions. The results of the study indicate that both the investigated EA's perform better than mono objective algorithms. However, the NSGA II algorithm outperformed SPEA 2 for prediction of relevant classes. The efficiency of the approach is evaluated by assessing the correctness of the recommended classes for any bug report.

1.1 *Research Questions*

Thus, we explore the subsequent research questions in this study:

- *RQ1:* What is the predictive performance of NSGA II and SPEA 2 for determining relevant classes based on bug reports?
  In this question, we validate the performance of NSGA II and SPEA 2 on six open source projects and estimate their accuracy using five performance metrics (precision, recall, F-measure g-mean and balance).

- *RQ2:* What is the comparative performance of NSGA II and SPEA2 algorithm to mono-objective approaches?
  To answer this question the performance metrics mentioned in RQ1 are used. The performance of the NSGA II and SPEA 2 algorithm is statistically compared with the mono objective algorithms which aggregates all the objectives into a single one with equal weights. The comparison of the multi objective algorithm with the mono objective is not as simple as it seems because Multi objective returns a set of solution whereas mono objective returns a single optimal solution. Hence, we use the nearest solution [19] as a candidate solution for multi objective algorithm, which is compared with mono objective algorithm's solution.

- *RQ3:* Which is a better evolutionary algorithm amongst NSGA II and SPEA 2 for predicting relevant classes?
  The main objective of the research question is to articulate which algorithm is better for the purpose of predicting faulty classes. Wilcoxon statistical test is used to compare the two EA in order to determine the variations in the performance of

both the algorithms. We also compared the two algorithms with respect to CPU time consumption. CPU time is determined by the execution time taken by a technique on all the six open source codes.

1.2 *Motivation of study*

As we know that bug reports are hard to be managed and even more difficult to provide a feasible solution to those bugs. In today's world it is of utmost importance to solve those bugs as soon as possible in order to evolve the software according to users' needs. The main motivation behind the study is the increasing use of EA in the field of multi objective optimization. Therefore in this study we used two multi objective EAs in order to predict the relevant classes in order to fix the bug according to the bug report provided by the user. This study is closely related to the one performed by Rafi Almhana et al. [2]. They analysed the working of NSGA II multi objective evolutionary algorithm for predicting faulty classes in a software. However, this study is better in comparison to the existing one in terms of the following parameters:

Rafi Almhana et al. [2] used only one EA algorithm for the prediction of the faulty classes whereas in our analysis, two different EAs are evaluated and compared. This is important in order to determine the suitability of EA to the task of recommending appropriate faulty classes.

Rafi Almhana et al. [2] used precision, recall and accuracy metrices as the performance measure for analysing the results. However, literature studies have criticized the use of accuracy as an effective measure to determine the correctness of working of the algorithm [44]. Therefore, we have additionally used two robust performance measures namely, G-mean and balance as they are recommended by researchers. These measures are advocated in literature studies [36, 43, 48, 49].Thus, our results are profoundly more generalizable. This additionally strengthens the outcome of the study. The study will help the developer of the software during the maintenance phase of the software model. They will be able to detect the faulty classes and the results are then used to remove those bugs from the software efficiently.

1.3 *Organization of thesis*

The paper is organized in various sections as follows: Section 2 gives an overview of the related work of the study that is what is the various research works have been done in this area and how all those work helped in evolution of our study, Section 3 summarized research methodologies used in this paper including overview of the recommended framework and the description of both the EAs. Section 4 describes empirical study design of the dataset description, validation method, solution representation, objectives and performance evaluation methods which are used in our studies. Section 5 states result of the study which has been discussed with correspondence to each RQ. Section 6 states the threats to validity and at last the section 7 summarizes the research work under conclusion and suggests some future work.

CHAPTER 2

Literature Review

This section gives an overview of the research work done with relation to our study and is further sub divided into 2 sections i.e. application of multi objective evolutionary algorithms in various software engineering domains and defect prediction studies which recommend relevant classes on the basis of bug reports.

2.1 *Applications of Multi Objective EA in Software Engineering*

The mechanism of natural selection and the process of optimization directed towards evolutionary algorithm development [17]. The main goal of evolutionary algorithms is to simulate the process of evolution in a computer. EA are well suited to solve the above mentioned MOO problems. Unlike conventional methods, EA calculates the performance of the candidate solution at different points simultaneously. The origin of evolutionary algorithms dates back to the late 1950s and, since 1970s, various evolutionary algorithms are developed to optimize multiple objectives. The main motivation behind using evolutionary algorithm for MOO is because it deals simultaneously with multiple solutions which allows to calculate the Pareto optimal set by running the algorithm just once, whereas in traditional algorithms a series of separate runs to arrive to a particular solution is used.

Applications on different fields:

- *Software reliability prediction:*

EA have been used in a various software engineering domains. Aljahdali and Telbany used Genetic algorithm (GA) technique for software reliability prediction. They predicted the extent up to which a software can be relied upon [44]. GA is used in the study to overcome uncertainties which may arise due to multi objective functions in order to achieve the best

result. The results prove that the use of multi objective algorithm has better performance in comparison with the average method [44].

- *Analysis of Multi-objective Evolutionary Algorithms*

Another application of multi objective EA is for training ensemble models. In this research work, different sets of performance measures are used to train ensembles and the outcomes of these measures are then compared. The result shows that out of all the performance measures Logarithmic Standard Deviation, Mean Magnitude of the Relative Error and Percentage of predictions, these three measures outperformed all the others [47].

- *Cross project defect prediction using multi objective algorithms:*

In the other application of multi objective EA Cross project defect prediction is evaluated. This research work is used to predict the models which are prone to defects. They tries to identify effective multi objective methods over cross-project (CP) environment. A harmonic search meta-heuristic algorithm has been used to effectively summarize three conflicting objectives in the context of class imbalance. The results are of this defect prediction models binary in nature and just signifies the presence of defect in any model, Results are also promising and shows that algorithm can be effectively used in various prediction model [46].

- *Maintainability defect prediction and correction*

Next application of multi-objective EA is Maintainability defect prediction and correction. In this research genetic algorithm is used to predict defect and then correct them as a two-step automated process [45]. The first step uses genetic algorithm in order to automatic rule generation for defect detection reducing the efforts to do it manually. And in the second step multiple objectives are compromised using NSGA II. The results shows that the algorithm was successful in detecting most of the defect and then correctly fixing them with insignificant exertion.

2.2 Defect prediction using Information Retrieval

Software defect prediction is a standout amongst the most research ranges in software engineering [8]. Defect or bug prediction model effectively contributes towards the development of the software by providing software reliability [12]. Defect prediction refers to

predicting the area of the code which is more prone to bugs [6]. Various works previously done based on this information retrieval concept are explained below.

Different tools and approaches used to predict faulty classes are summarized as follows:

BugLocator

It is a method based on information retrieval which recommends relevant files in order to fix a bug [16]. It ranks all the files based on the textual similarity between the source code and the bug report description. It uses revised Vector Space Model (RVSM) and information about the similar bugs which are fixed before in order to derive the lexical similarity between the report of the bug and the source code and to improve the ranking performance. It works better than methods based on Vector Space Model (VSM) and Latent Dirichlet Allocation (LDA).

DebugAdvisor

It is a system which investigates the bug by using the complete bug report as a text query and then mine the bug repository to derive the required results [4]. It uses a fat query which contains structured and unstructured data in kilobytes of size. This query contains all the information about the issue being debugged, which includes natural language text, the output of the debugger etc. The accuracy of this system depends upon the accuracy of the bug report.

BugScout

It is an automated approach which has been used to minimize the search space in order to find the faulty code segment [13]. BugScout assumes that there is some textual similarity between the contents of the report of the bug and the source code which can be used to locate the buggy source files. It uses the Vector Space Model (VSM) and similar information about the bug to derive the lexical similarity between the report of the bug and the source code and rank the code fragments accordingly. The drawback of this method is that it solely depends upon the bug report entered by the user which is not always framed appropriately.

BLUiR

It is an open source toolkit available for everyone [5]. The key feature of their algorithm is that they have used structures information retrieval based on the constructs of the code such as the name of the class and method and enables more accurate bug localization. This method is also based on comparison between source code and the report of the bug to retrieve the information based on the class name method names or the code construct.

These studies are successful in establishing a relationship between the multi objective problems and the evolutionary algorithms. The EAs can be widely used with MOO problems because of the ability of the EAs to solution set in a single run of the algorithm. Most of the work in defect predication is done using single objectives. Only one study conducted by Rafi Almhana et al. [2] used a multi objective algorithm to predict the bugs. Therefore, there is a need to conduct more studies which analyze the effective of multi objective evolutionary algorithms for predicting relevant classes corresponding to a bug report. This study also takes into consideration different objectives and uses them to provide improved result as compared to mono objective algorithms.

CHAPTER 2

Research Methodologies

This section first describes the framework used for prioritizing code fragment which need to be investigated corresponding to a bug report. In the next segment the EAs used in the study are described.

3.1 Overview of the Recommendation Framework

The approach aims at recommending faulty classes from the bug reports provided by the user. A substantial search space is to be examined so as to locate the relevant classes. The developers can then use these reports to fix the bugs. Also, sometimes more than one class is to be rectified in order to remove the bug from the system.

A heuristic based approach is suggested based on two conflicting objectives to be simplified. The first objective is correctness function which consists of two sub functions. The first sub function maximizes the textual similarity between the API description of the classes and the description of the bug. And the second sub function maximizes the history based similarities i.e. when the class was last fixed and how often. The second goal is to limit the quantity of classes to be recommended.

It clearly visible that the above two objectives are conflicting in nature. Therefore, in this study the task of recommending relevant classes is modeled as a MOO problem. Two different EA are used to tackle the problem, namely NSGA-II and the SPEA2. The results are then compared to ascertain which algorithm is better.

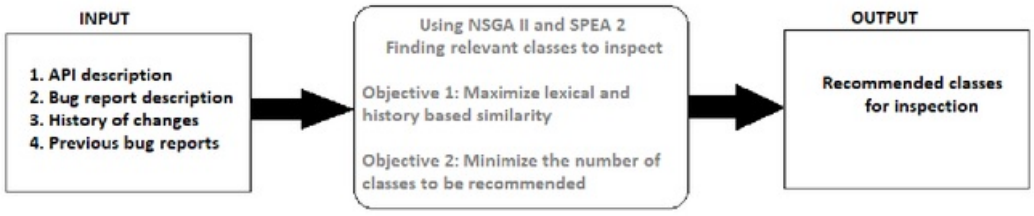


Figure 1: Approach overview

A summarization of the approach used is described in figure 1. This algorithm takes a series of input, i.e. API description of the source code, bug report description, history of the changes made in the previous classes and the previous bug reports and the output is a sequence of rank classes to be inspected for bug maximizing the lexical and history based similarity and minimizing the classes to be inspected. Two algorithms have been used to implement the above mentioned approach, i.e. NSGA-II and SPEA2 which recommends the classes for the inspection of bug as an output

3.2 NSGA II

A non-dominated EA is a multi-objective optimization evolutionary algorithm. The algorithm was proposed by Deb et al [39] in 2002. The NSGA algorithm was first proposed by Srinivas and Deb [39], in 1995 and which was further extended as NSGA II.

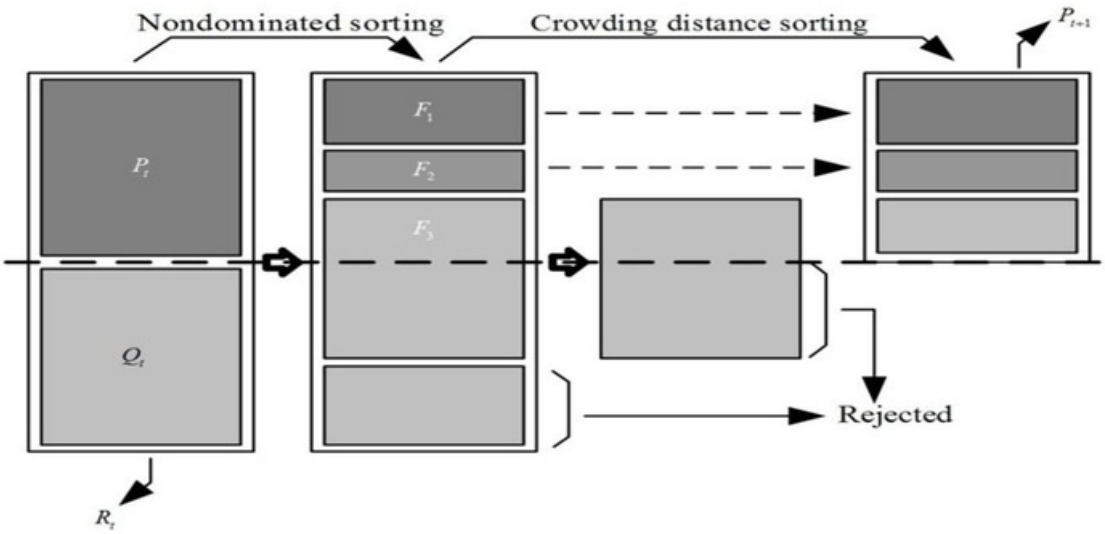
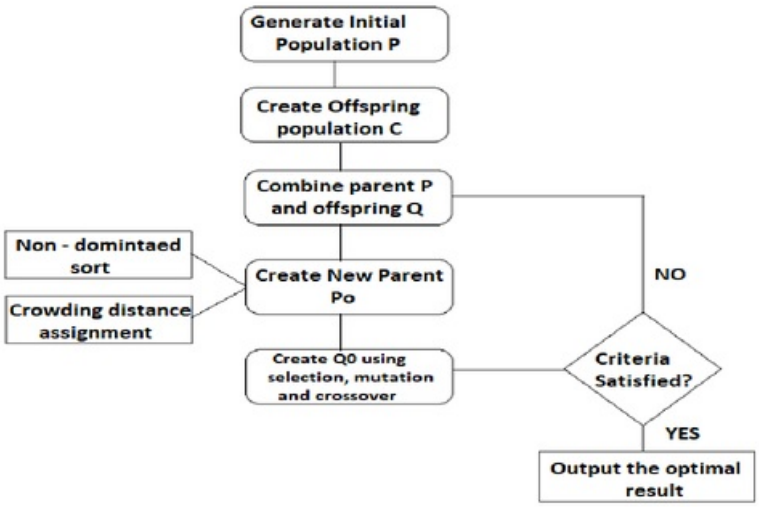


Figure 2: NSGA II

NSGA II uses various genetic operators such as crossover and mutation. Along with this it also uses two specialized multi objective operators and mechanism, i.e. non-dominated sorting and crowding distance to give a better generalization of different conflicting objectives as shown in figure 2. The population is sorted and separated into different fronts in non-dominated sorting where the primary front always denotes the Pareto front whereas crowding distance is a parameter which is used to rank the members of the front based on dominance



Algorithm 1 NSGA II

1. $Pop_0 \leftarrow$ Initialize population
2. $C_0 \leftarrow$ Create offspring Population
3. **while** (┐ stopping criteria)
    a. $U_t \leftarrow$ Union of P_0 and C_0
    b. f ← sorting based on non-dominance (U_t)
    c. $Pop_{t+1} \leftarrow$ Ó
    d. j = 1
    e. **while** $| Pop_{t+1} | + | F_i | \leq N$ **do**
        i. Crowding-distance assignment (F_i)
        ii. $Pop_{t+1} \leftarrow Pop_{t+1} \cup f_i$
        iii. j ← j++

f. **end while**

g. Sort (fi ≺ S)

h. Pop_{t+1} ← split and choose first (N- | Pop_{t+1} |) elements

i. C_{t+1} ← create-new-population(Pop_{t+1})

j. t ← t++

4. end while

Figure 3: NSGA II flowchart and algorithm

As shown in figure 3 NSGA II algorithm begins by creating a new population randomly known as the parent population, Pop_o. The population consist of N individuals and non-dominance sorting is used to sort the population into different fronts. Fronts are then to be minimized by eliminating the solutions which are not further useful. The reduction of the front is based on dominance and in case if the dominance of the two population is similar then the crowding distance between the two is used. Furthermore, every solution is assigned with a rank or front which is based on its non-domination level with minimum rank as the best rank and so on. Then a new population C_0 (offspring population) of size S is created using selection, recombination and mutation operators. A recombined population of size 2S is created as U_t. Then further the population is sorted according to non-dominance. This selection of the population is continued till the front is populated with S solutions. To choose exactly S population members from the best non dominated set i.e. F1 crowded comparison operator ≺ is used to sort the population in descending order and out of those, the best solution is used to fill the population. Front t is then split and sorted in descending order. Then a new child population t+1 is created using mutation, selection and crossover. The above mentioned process is repeatedly executed.

3.3 SPEA 2

The Strength Pareto evolutionary algorithm is used to solve MOO problems to find an approximate Pareto solution. SPEA2 was proposed by Zitzler [10] and has shown good performance result. SPEA2 comprises of different operations such as a good fitness function and density estimation. The algorithm is also able to obtain the population with both "precision" and "diversity" [9].

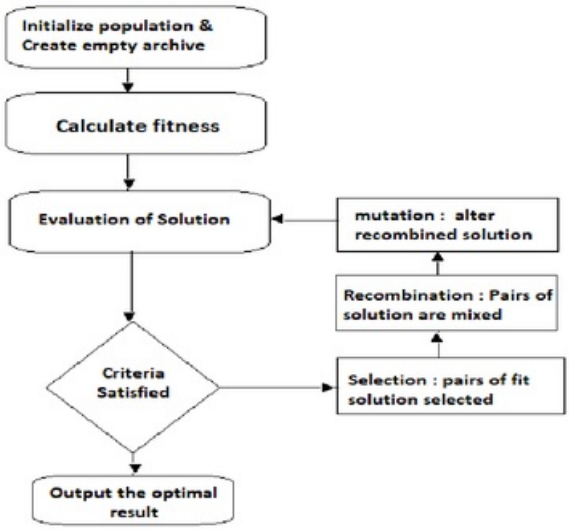


Figure 4: Flowchart of SPEA 2 algorithm

According to the flowchart given in figure 4 SPEA 2 starts with an initial population P_0 and repeats the following steps for each iteration. SPEA 2 as shown in figure 5 starts with, an empty archive (external set) is created along with the initial population. Fitness value is then initialized to both population and archive i.e. P_0 and C_o. All the non-dominated members of P_0 and C_o are copied to the archive C_{t+1} and if any member is dominated or is repeated then those members are removed from the archive. Also, if the members count exceeds a limit N than those elements are truncated using clustering techniques otherwise if the size is fewer than N then we fill C_{t+1} with dominated individuals. These steps are repeated until the maximum size of generation N is reached or other stopping criteria is met. In the next step, selection is performed using binary tournaments. And at last after applying recombination and mutation a new population is created which replaces the old population.

Algorithm 2 Pseudo code SPEA 2

1. $P_0 \leftarrow$ Initialize the population
2. $C_o \leftarrow$ Ó Create an empty archive
3. For each t -> 0 to N
    a. Calculate fitness values of each individual in Pt and Ct
    b. $C_{t+1} \leftarrow$ Union of Pt and Ct

c. If $|C_{t+1}| > N$

i. Truncate C_{t+1}

d. Else

i. Insert dominated individuals

e. If $t \geq T$

i. V ← vectors of non- dominated individuals in

4. Perform binary tournament selection with replacement C_{t+1}
5. C_{t+1} ← New Population by recombination and mutation operator
6. t ← t + 1

Figure 5: SPEA 2 Algorithm

In SPEA 2 density information is the key attribute to fitness assignment. Also, the magnitude of the archive is fixed and if the number of non-dominated individuals are not as much as the archive size than dominated individuals are to be filled in the empty space. And if the number of individuals increases, truncation is the option.

CHAPTER 4

Empirical Study Design

This section discuss different design consideration of our study.

4.1 Description of Data Sets

For the evaluation of the two algorithms NSGA II and SPEA 2 the following six open source projects have been used:

1. *AspectJ* :

   An aspect – oriented programming extension for java

2. *Birt*:

   An Eclipse-based business intelligence and reporting tool.

3. *Eclipse Platform UI*:

   The user interface of an integrated development platform.

4. *JDT*:

   A suite of Java development tools for Eclipse.

5. *SWT*:

   A widget toolkit for Java.

6. *Tomcat*:

   A web application server and servlet container.

AspectJ which is a java programming extension, Birt which is an Eclipse based tool, Eclipse Platform UI which is a UI for integrated development, JDT is the tool for development of java, Tomcat which is a web server and SWT which is a widget toolkit.

Table 1 shows statistics of the six systems along with including features such as bug reports count for each open source project, number of API file's description of the classes, time range of the bug report and the number of classes that are fixed per bug report is described in the table.

Project	Bug reports	# API	Time	# fixed files/ classes per bug report

Eclipse UI	2135	1314	10/2001 – 01/2014	2
Bert	3584	957	06/2005 – 12/2013	1
JDT	5978	1329	10/2001 – 01/2014	2
AspectJ	524	54	03/2002 – 01/2014	1
Tomcat	987	389	07/2002 – 01/2014	1
SWT	3578	161	02/2002 – 01/2014	3

The count of the project files on which the algorithm has been run is more than 5,000 for six open source projects. For the purpose of validating the source code just before the bug was registered is used.

Table 1: Data Set Description

4.2 Validation Method

The collected data is categorized into two sets: the training data set and the test data set. The bug is organized according to the time on which they are reported and then are divided into 10 folds. The first fold contains the newest bug report, whereas the tenth fold contains the oldest. The last fold is used as the training set and just the previous fold as the testing data set and the process is repeated for all the ten sets. Finally, the output is compared to the expected solution of the problem.

4.3 Solution Representation

The candidate solution representation is done using a vector. Classes to be inspected are represented in each element of the vector. The main aim is to minimize the number of classes to be recommended on the basis of multi objective algorithms used. The solution is in the form of a list of classes to be inspected for the bug report. The classes are to be inspected in the same order as the class being the first element of the vector has a higher probability of having a bug.

Figure 6 represents the bug report from the Eclipse UI Project (Bug ID 34810) description using which we need to find out the solution that is the classes to be inspected. A bug report is a summary of what actually the bug is. This bug report describes a bug about no progress

shown while executing test in the Junit view icon. The solution would consist of a series of classes to be inspected.

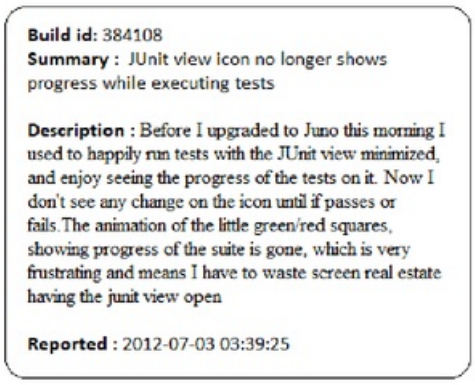


Figure 6: An eclipse bug report (Bug ID: 384108)

4.4 Description of Objectives

Correctness Objective: This objective is just the mean of lexical based similarity (LS) and the history based similarity (HS) between the bug report and the classes.

$$f1 = \frac{HS+LS}{2} \qquad (1)$$

The lexical similarity refers to the textual similarity between the API and the bug report. The lexical similarity is calculated by computing the cosine similarity between the API and the bug report description. A series of steps are followed to calculate the lexical similarity between the two. The steps are described below:

I. Camel Case Splitter is used to perform the tokenization process to access the identifiers.
II. Stop word reduction is used in order to remove the irrelevant information form the source code as well as the bug report description.
III. Porter Stemmer is used to reduce the words to its stem in order to eliminate the redundant information on any particular word.

IV. Finally, the cosine similarity is used to compare the API and the bug described in order to derive the lexical similarity between the two.

Cosine similarity between two vectors measures the angle of cosine between them.

$$P.Q = ||P||_2 \, ||Q||_2 \cos\theta \qquad (2)$$

$$\text{Similarity} = \cos(\Theta)$$

$$\cos\theta = \frac{P.Q}{||P||2 \, . \, ||Q||2}$$

$$\cos\theta = \frac{\sum_1^n P_i Q_i}{\sqrt{\sum_1^n P_i^2}\sqrt{\sum_1^n Q_i^2}} \qquad (3)$$

Where P_i and Q_i are elements of the two vectors P and Q respectively as shown in equation 3. The weights of each element in vector P_i and Q_i can be calculated using an information retrieval technique known as Term frequency – Inverse Term Frequency (TF-IDF) method. The lexical similarity between the source codes i.e. the API description of the classes and the bug report is calculated using the cosine similarity function.

The API description of the source code consists of various words matching to the bug description so the use of API description is more beneficial for our purpose to derive the lexical similarity. However, use of just the lexical similarity between the two will not be enough.

The other function which comes under correctness objective is history based similarity. The history based similarity computation consists of three functions and the final result value is considered to be the average of the three functions.

$$H_1 = \frac{\sum_{i=1}^{size(S)} NoBugFixed\,(C_i)}{Size(S) * Max(NoBugFixed(C_i))} \qquad (4)$$

The first function is used to compute that how many times any given class has been previously fixed in order to derive the value for the elimination of bugs based on the history of the class and the formula for its computation is shown in equation 4. In the equation the submission of number of bugs fixed for a class is divided by the product of the size of the solution set S and the maximum number of times any class was fixed.

$$H_2 = \frac{\sum_{i=1}^{Size(S)} \frac{1}{report.date - last.report(C_i)+1}}{Size(S)} \quad (5)$$

The second function checks whether the class which has been recommended has been recently fixed. If the class have been recently fixed or modified there are higher chances of the class being buggy. The time period can be computed by taking the difference between the date on which the bug was reported and the last time the class was modified. If the date of class modification and bug reporting is same then the value becomes 1. The formula for the second function is defined in equation 5.

$$H_3 = \frac{Cir}{Size(S)} \quad (6)$$

The third and the last function defines consistency among the recommended classes on the basis of the previous bug report. The class has higher chances of having a bug involving all the recommended classes if those classes are recommended previously for the similar bug. This value can be calculated by using equation 6. Cir is the value corresponding to the intersection of a set of classes between set of classes recommended and the solution S.

4.5 Performance Evaluation

The outcomes of the developed models is evaluated using precision, recall. F-measure, g-mean and balance. Further Friedman test along with Wilcoxon test has been used to statistically analyze the results.A confusion matrix is a 2 x 2 matrix which helps in visualizing the performance of the algorithms. The metrices to study the performance of this study are described below:

- Precision: It refers to the fraction of correctly recommended files in top k of files which are recommended to the minimum number of files to be inspected. It can also be defined as out of the number of classes selected, what percentage of them is the algorithm classify correctly.

$$Precision\ (P) = \frac{|\{Relevent\ classes\} \cap \{Retreived\ classes\}|}{Retrieved\ classes} \quad (7)$$

- Recall: It refers to the fraction of correctly recommended files in top k of files which are recommended to the total number of expected files to be recommended. It is also known as probability of detection (PD).

$$Recall\ (R) = \frac{|\{Relevant\ classes\} \cap \{\ Retreived\ classes\}|}{Relevant\ Classes} \quad (8)$$

- F-measure: It is a weighted harmonic mean of recall and precision.

$$F_1 = 2 * \frac{Precision * Recall}{Precision + R} \quad (9)$$

- Probability of false alarm (PF): It refers to the number of non-relevant and non-retrieved classes divided by classes which do not contain bug.

$$PF = \frac{|\{Non-Relevant\ classes\} \cap \{retrieved\ classes\}|}{Non\ relevant\ classes} * 100 \quad (11)$$

- G-mean: G-mean is used maintain a balance between high positive as well as high negative accuracy. It is calculated as a geometric mean of positive and negative accuracy. Higher the value of g-mean better the performance of the algorithm.

$$gmean = \sqrt{P * \frac{|\{non\ relevant\ classes\} \cap \{non-retreived\ classes\}|}{Non-retrieved\ classes}} \quad (12)$$

- Balance: In balance measure Euclidean distance is found between the percentage of classes which are correctly detected as buggy know as probability of detection (PD) to the percentage of classes which are incorrectly detected to be non-buggy classes known as probability of false alarm (PD). Higher value of balance implies better performance of the algorithms.

$$PF = 1 - \frac{\sqrt{(0-(\frac{PF}{100}))^2 + (1-(\frac{PD}{100}))^2}}{\sqrt{2}} \quad (13)$$

CHAPTER 5

Result and Analysis

5.1 *Studied Projects:*

This section elaborates the result of our research work and answer different RQs mentioned in section I of the paper.

5.1 RQ1: *What is the predictive performance of NSGA II and SPEA 2 for determining relevant classes based on bug reports?*

In order to validate the correctness of the two algorithms to recommend relevant classes, we evaluate precision, recall, f-measure, g-mean and balance performance metrics. The values in figure represents the median value of ten-fold cross validation results of all the 30 runs executed for each technique. Each bar in the figure represent median value of precision, recall, f-measure, g-mean and balance for NSGA II and SPEA 2 for the 6 open source projects respectively for top 5 of the recommended files i.e. k=5. The values corresponding to NSGA II algorithm are depicted using dark grey bars whereas for SPEA 2 algorithm light grey bars are used.

The median value of precision ranges from 75% to 54% for both the EAs. The best precision value is for the EclipseUI data set whereas the lowest precision value is for AspectJ dataset. The lowest value of the precision is around 54% for k=5 which is still considered good enough because for each bug it is not necessary that we have to inspect multiple classes [2]. Similarly, the value of recall for NSGA II ranges from 68% to 51% and for SPEA 2 ranges from 63% to 44%. After analyzing the precision and recall results we can say that both the algorithm are successful in predicting the classes. However, we need a more meticulous method for the comparison, so just the precision and recall are not enough. Thus, we use some other performance metrices for the verification of the result.

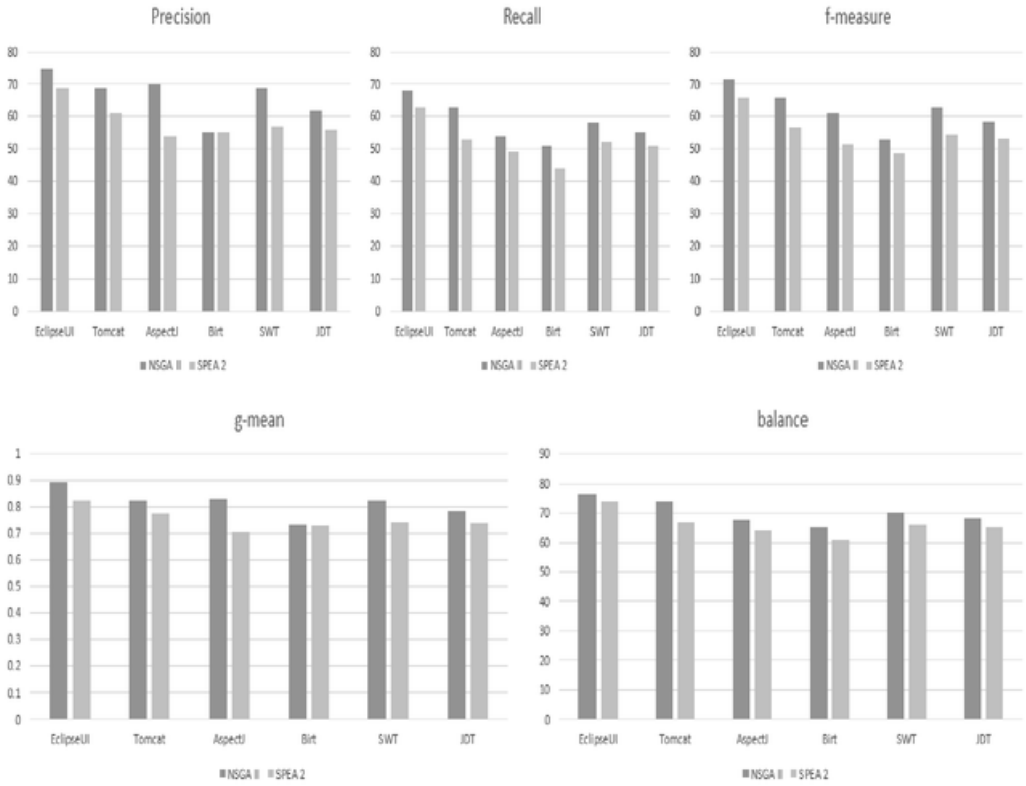


Figure 7: Median performance metrices values of 30 runs for k=5

The other performance measures used are f-measure, g-mean and balance. From figure 7 we can conclude that performance metric values for NSGA II ranges from 71% to 52% for f-measure and 76% to 65% corresponding to balance. The performance of SPEA 2 ranges from 65% to 51% for f-measure and 74.01% to 60.64% corresponding to balance. After analyzing it can be declared that NSGA II and SPEA 2 both gives appropriate balance values and can be successfully used for recommending relevant classes. Figure 7 also shows the G-mean value for all the data sets. After analysis it can be said that the highest value of G-mean is for NSGA II algorithm that is 0.89 whereas for SPEA 2 algorithm the value is 0.82 for EclipseUI data set. So this feature along with all the performance metrices discussed above clearly shows that most of the buggy classes were recommended correctly using both NSGA II and SPEA 2 algorithms.

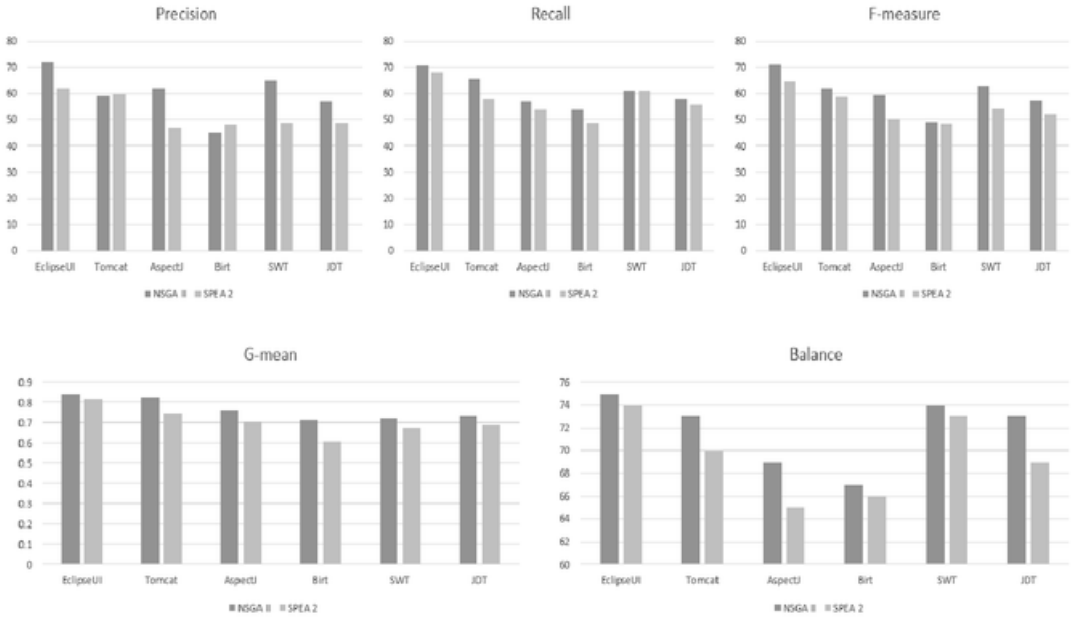


Figure 8: Median Performance metrics values of 30 runs for k=10

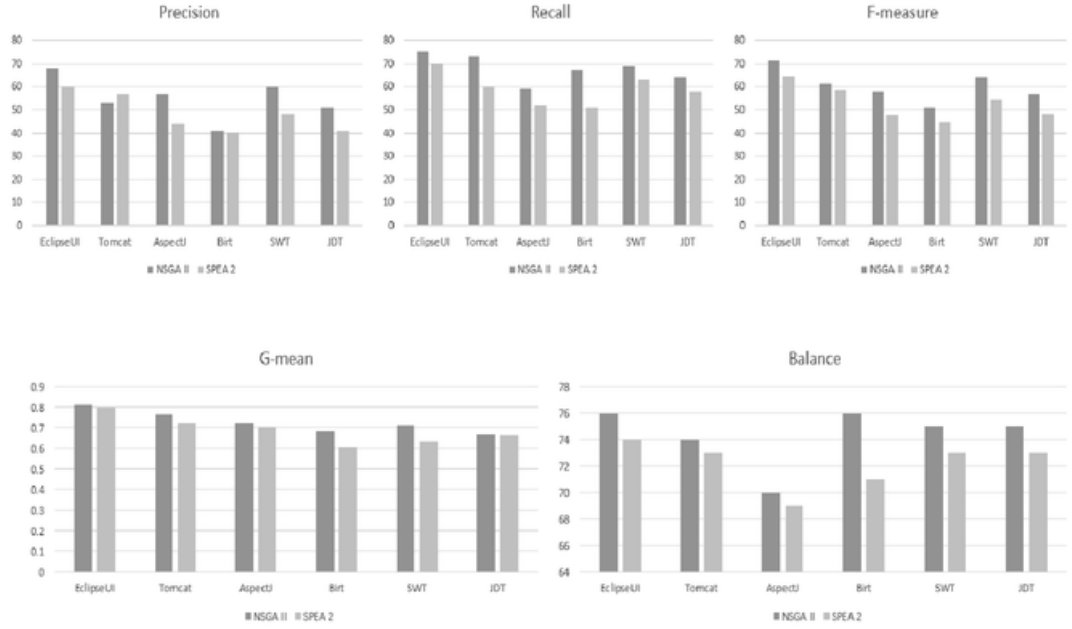


Figure 9: Median Performance metrices values of 30 runs for k=15

Figure 8,9,10 summarizes the result of performance metrices for k=10, 15, 20 of both the investigated algorithms where K= n means top n of the recommended files. The cumulative range for k=10,15 and 20 in majority of the cases lies between 75%-51% for precision , 75%-52% for recall , 71%-50% for f-measure , 0.89-0.62 for g-mean and 76%-65% for balance with respect to NSGA II algorithm. Whereas for SPEA2 the range lies between 69-53% for precision, 74%-55% for recall, 66%-54% for f-measure, 0.82-0.57 for g-mean and 75%-60% for balance. In certain cases, the SPEA2 gave better performance metrics than NSGA II. Figure 6 demonstrates that for the data set tomcat SPEA2 gives better precision value than NSGA II, and for data set SWT precision values obtained by both the algorithms were same.

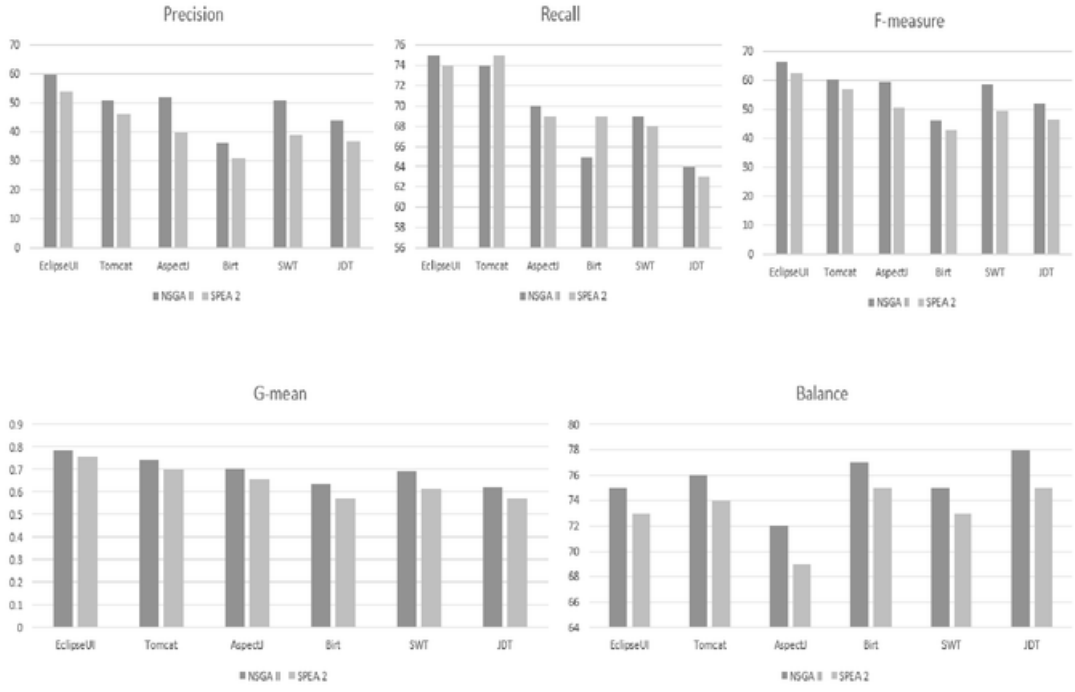


Figure 10: Median Performance metrices values of 30 runs for k=20

Figure 7 depicts that for the data set Tomcat, SPEA2 gives better precision value than NSGA II. Figure 8 shows that for Tomcat and SWT dataset SPEA2 obtains better recall values than NSGA II. In all other cases shown in figure 8 to 10 NSGA II is performing better than SPEA 2. Other than one or two cases NSGA II more precisely predict faulty classes using bug reports. These results confirm the results obtained by Rafi Almhana et al. [2], which advocate NSGA II for predicting relevant classes corresponding to a bug

report. However, our study additionally evaluated SPEA2 which was also found to be competent in prediction of relevant classes.

The performance of our bug detection model was evaluated using different performance metrics. The investigated EAs are effective in predicting relevant classes to localize the bug based on the description of the bug report. The G-mean values for k=5 range from 0.89-0.73 for NSGA II and 0.82-0.70 for SPEA 2 and Balance values for NSGA II ranges from 76%-65% and for SPEA 2 it is 74%-60%.

5.2 RQ2: *What is the comparative performance of NSGA II and SPEA2 algorithm to mono-objective approaches?*

The results in RQ1 indicate the effectiveness of NSGA II and SPEA 2 for predicting faulty classes. Further, we validate whether these multi objective EAs are able to perform better than mono objective algorithm. The multi objective EAs are compared with three mono-objective algorithms based on five metrics i.e. precision, recall, F-measure, g-mean and balance in all the 6 open sources projects. The results are then validated using Friedman test.

The 3 mono objective algorithms can be defined as:

I. **LS:** It is a mono – objective which is based on lexical similarity.
II. **HS:** Mono-objective defining history based similarity.
III. **GA:** It aggregates both LS and HS objectives.

The results of the multi objective algorithms are compared with the mono objective algorithms LS i.e. the lexical similarity and the HS i.e. the history based similarity and finally the GA which is also mono objective which takes the average of both the objectives. As indicated in figure 11, the median value of precision for mono objective algorithms range from 64% to 45%, value of recall, f-measure, g-mean and balance ranges from 60%-51% , 61.93% to 44.04% , 0.77 to 0.66 and 71.72% to as low as 55.71%

respectively. From these observations, we can clearly state that the performance of our multi objective EAs are much better than those of mono objective.

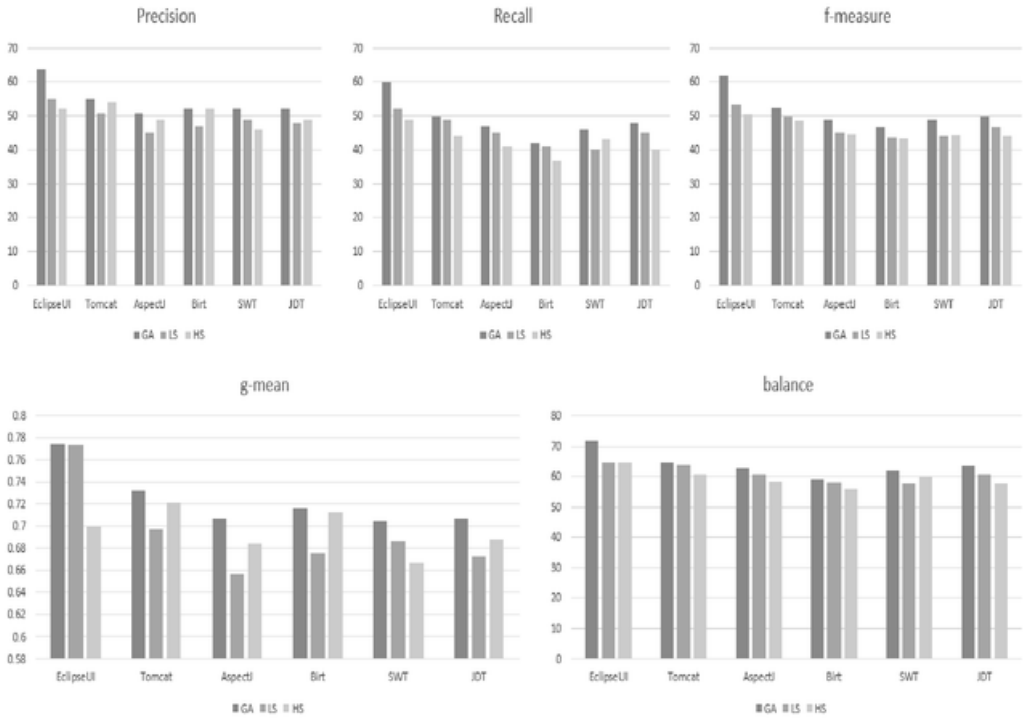


Figure 11: Median value of 30 runs on different mono objective technique for k =5

We can also deduce that the performance of GA i.e. the aggregate of both the objective is also better than that of the history based similarity or lexical similarity. We can say that the average g-mean for LS and HS is 0.6941 and 0.6954 whereas combining both the objectives into GA the g-mean becomes 0.7236 which shows the performance on combining the two objectives into one improves the performance significantly. The values of various performance metrices for k=10, 15 and 20 also shows the similar results as that of k=5 and the result tables are shown in Appendix A.

In order to assess the superiority of the employed EAs, we performed Friedman test. The test was performed on all the evaluated performance measures using six open source java data sets. It allocates a mean rank to each technique. The lower the rank, the better the algorithm.

The test is on the basis of chi square distribution. Performance of these algorithms are then compared for 6 datasets on the basis of median of 30 runs of the investigated performance metrics.

Algorithms	Precision	Recall	F-measure	G-mean	Balance
NSGA II	**1.08**	**1.00**	**1.25**	**1.00**	**1.42**
SPEA 2	1.92	2.17	1.83	2.00	1.58
GA	3.08	2.83	3.00	3.00	3.00
LS	4.67	4.17	4.58	4.17	4.17
HS	4.25	4.83	4.33	4.83	4.83

Table 2: Friedman test ranking for k=5

Algorithms	Precision	Recall	F-measure	G-mean	Balance
NSGA II	**1.58**	**1.08**	**1.00**	**1.00**	**1.00**
SPEA 2	1.67	2.42	2.00	2.00	2.00
GA	3.21	3.67	3.42	3.00	3.08
LS	4.83	4.83	5.00	4.00	4.00
HS	3.71	3.00	3.58	5.00	4.92

Table 3: Friedman test ranking for k=10

Algorithms	Precision	Recall	F-measure	G-mean	Balance
NSGA II	**1.25**	**1.00**	**1.00**	**1.00**	**1.00**
SPEA 2	1.92	2.08	2.00	2.00	2.00
GA	2.83	3.25	3.00	3.00	3.00
LS	4.58	4.67	4.83	4.17	4.17
HS	4.42	4.00	4.17	4.83	4.83

Table 4: Friedman test ranking for k=15

Algorithms	Precision	Recall	F-measure	G-mean	Balance
NSGA II	**1.00**	2.50	**1.00**	**1.00**	**1.00**
SPEA 2	2.08	**1.50**	2.00	2.00	2.17

GA	3.17	2.83	3.17	3.00	2.92
LS	4.17	3.83	4.17	4.00	4.17
HS	4.58	4.33	4.67	5.00	4.75

Table 5: Friedman test ranking for k=20

As shown in the table 2-5, the finest rank is obtained by NSGA II followed by SPEA 2. From these ranks we can also derive that both the multi objective EAs are better in predicting faulty classes than mono objective algorithms. On the other hand, after multi objective EAs, GA is statistically better in comparison to LS and HS. Table 3, 4 and 5 also demonstrates similar results that are demonstrated in figure 2 i.e. the rank of NSGA II is the finest among all the algorithms compared. However, in just one case i.e. when k=20, SPEA2 algorithm attains a better rank than NSGA II corresponding to values of recall. NSGA II still dominates SPEA 2 in all the other cases. The Friedman results are significant for all performance metrics at α= 0.05. From all the above observations we can say that our multi objective EAs outperform the mono objective algorithms significantly, thus concluding that our formulation of the approach was adequate.

The statistical result of Friedman test advocates that both the investigated EA are ranked better than the mono objective algorithms. This result indicates that in order to predict relevant faulty classes the conflicting objectives can be properly addressed only by using EA. Since, conflicting objectives cannot be properly represented by mono-objective algorithms, they give poor results.

5.3 RQ3*: Which is the better evolutionary algorithms amongst NSGA II and SPEA 2 for predicting relevant classes?*

We concluded that the NSGA II and SPEA 2 algorithms are better than the three mono objective algorithm namely GA, LS, HS. However, the main aim of the paper is to determine, which of the two EA's among NSGA II and SPEA 2 is better, giving an edge to one algorithm over the other. To validate this result, we have used Wilcoxon test. This test is used two compare two samples and determine and rank the samples according to the absolute values. The comparison test is performed at significance level of 0.05 using all the

performance metrics used in the study i.e. precision recall, F-measure, g-mean and balance. The Wilcoxon test results are shown in Table 6. The results are tested for different values of k i.e. 5, 10, 15 and 20.

NSGA II vs SPEA 2	Precision	Recall	f-measure	g-mean	balance
K=5	S+	S+	S+	S+	NS+
K=10	NS+	S+	S+	S+	S+
K=15	NS+	S+	S+	S+	S+
K=20	S+	S+	S+	S+	S+

Table 6: Wilcoxon test result

To represent the result of the Wilcoxon test four variables are used:

S+ -> NSGA II algorithm outperform SPEA II algorithm significantly.

S- -> SPEA II algorithm outperform NSGA II algorithm significantly.

NS+ -> NSGA II outperform SPEA 2 but not significantly.

NS- -> SPEA 2 outperform NSGA II but not significantly.

According to table 6 NSGA II significantly outperform SPEA II in seventeen out of twenty cases for all the evaluated performance metrics values. In the other three cases also, the performance of NSGA II was better, although it was not significant. Thus the statistic results of Wilcoxon advocate that NSGA II algorithm is the best technique for defect prediction.

We also compare the CPU times of the two EA's. Figure 8 presents the execution time performance of our multi objective approaches. The average execution time of NSGA II on all the six databases was around 950millisecond and for SPEA 2 the execution times aggregates to 2145 millisecond. In terms of execution time too, the NSGA II gives a better running time as compared to SPEA 2. The execution time was reasonable. Also, the execution time depends upon number of files in our dataset and the history of the bug report.

Fig 8: Execution time (ms) of NSGA II and SPEA 2

Comparing both the algorithms, NSGA II performs better in noisy environment for latter generations. Regardless of the noise present in the environment NSGA II is able to give appropriate results [42]. NSGA II also have a broader range of solution sets as compared to SPEA II. The solutions found by NSGA II closer to pareto optimal font whereas SPEA II perform better in case of high dimensional objective spaces [42]. Also, SPEA2 consumes more time as compared to SPEA 2 so we can say that SPEA 2 is computationally expensive [30].

The pair wise comparison of NSGA II algorithm with SPEA 2 algorithm is statistically evaluated using Wilcoxon test to determine which algorithm performs better for determining relevant classes. The results clearly advocates that NSGA II outperform SPEA in almost all the cases. And for almost all the cases NSGA II perform significantly better than SPEA II as derived from Wilcoxon test. The comparison on the bases of CPU time indicates that NSGA II is faster in terms of predicting buggy classes as compared to SPEA 2 algorithm.

CHAPTER 4

Empirical Study Design

Factors which may result in affecting the outcomes of our study refers to threats to validity. Threats to validity can be classified into two types: Internal validity, External Validity.

Internal Validity:

Internal validity ensures that the study follows the principle of cause and effect on independent variables. In our study, we have used EAs which are stochastic in nature therefore the results vary for each run of the algorithm. This can be considered as one of the internal threat. In order to overcome this threat, this study performs 30 runs for each data set. Moreover, the results of the study are statistically analyzed using Friedman test and Wilcoxon test with a confidence level of 95% (alpha = 5%). The other threat to the validity of our experiment is that not much of the work is done in this area of bug localization using multi objective EAs. So, the validation of the result was a difficult task. Thus, we compared mono objective algorithms as well as another multi objective algorithm in order to verify the results.

External Validity:

External Validity of our study refers to generalization of the results. It can also be defined as to what degree the outcomes of any study can be generalized to different situations. This validity is not easy to be achieved and is the basis on which we can say whether the study is successful or not. External Validity can be classified into two main types that is population validity and ecological validity. Both the types of external validity is useful in determining the strong point of the experimental design.

Threats to external validity:

Aptitude: The example may have certain components that may collaborate with the autonomous variable, restricting generalizability.

Situation: These threats are specific to a particular situations. This can be used to generalize the potential of the research work.

Pre-test effects: If only after performing the pre-test we are able to find out the cause effect relationship, then it can limit the generalizability of our work to a great extent.

Post-test effects: If only after performing the post-test we are able to find out the cause effect relationship, then it can limit the generalizability of our work to a great extent.

To overcome this treat we can perform the studies repeatedly on multiple data sets and then compare the results. In this study we have used six different open source java projects for validation of the result. However, in all the dataset the programming language used is java, which implies that the results cannot be generalized for all the programming languages. To reduce the external threats to validity, some additional comparable studies needs to be conducted for a better generalizable result.

CHAPTER 6

Conclusion and future work

6.1 *Conclusion*

We have proposed an approach to automate the process of finding the relevant classes corresponding to bug reports in this study. Two multi objective algorithms are evaluated to find a trade-off between minimizing the number of classes to be recommended and maximizing the correctness of the recommended classes. The correctness function includes lexical and the history based similarity. The two algorithms used for this purpose are NSGA II and SPEA 2 The results is performed using 30 independent runs for six open source datasets. The results are then statistically verified using Friedman test with 95% confidence level. We also used Wilcoxon test to pair wise give a comparison study of both the algorithms.

The results of our study are summarized as follows. In our study we validated the performance of two algorithms NSGAII and SPEA 2. The results verified that both the multi objective EAs were able to successfully predict the relevant classes to be inspected. The average g-mean for all the dataset under the value of k ranging from 5 to 20 is 0.75 for NSGA II and 0.70 for SPEA 2 and for balance the values are 75% for NSGA II and 72% for SPEA2 and for f-measure the values are 61% for NSGA II and 56% for SPEA2. The results shows that the performance is improved by 19%-12% for g-mean, 16%-12% for balance and 22%-12% in most of the cases in comparison to mono-objective algorithm. We also concluded using Friedman test that both the algorithms were better than mono objective algorithms and

thus applying multi objective EAs for solving this problem is a successful approach. The Friedman results shows that NSGA II was ranked best in all the cases except one where SPEA 2 gave a better result. Finally, the pair wise comparison of both the algorithms clearly stated that NSGA II outperformed SPEA 2 and the when the results were statistically validated using Wilcoxon test. The comparison on the basis of CPU time also advocated NSGA II. Therefore, the bug prediction model used in our study shows that the proposed approaches are promising and can be effectively applied to predict relevant classes to be inspected with respect to the description of the bug report. The results can be efficiently used by the researchers and software developers for efficient planning and resource allocation.

6.2 Proposed Work

And as a part of the future work, the algorithms can further be implemented in different languages and then the result can be evaluated. Also, the bug descriptions of the same bug with different user are almost similar but are registered as a different but so we can also work on eliminating the duplicate bug reports and further prioritizing can also be considered.

Appendix

This appendix represents statistical results of mono objective algorithms for all the values of k i.e. 10,15 and 20. Table A.1 represents the results of 5 performance metrices: precision, recall, g-mean, f-measure and balance for all the six open source data sets.

Precision	**GA K=10**	**LS K=10**	**HS K=10**	**GA K=15**	**LS K=15**	**HS K=15**	**GA K=20**	**LS K=20**	**HS K=20**
EclipseUI	58	48	47	56	45	40	51	45	38
Tomcat	50	38	49	53	35	47	32	31	29
AspectJ	45	35	44	43	32	40	35	32	40
Birt	46	34	46	39	31	31	29	28	26
SWT	47	38	42	45	35	33	36	35	30
JDT	43	35	44	41	32	37	35	32	31
Recall	**GA K=10**	**LS K=10**	**HS K=10**	**GA K=15**	**LS K=15**	**HS K=15**	**GA K=20**	**LS K=20**	**HS K=20**
EclipseUI	66	58	68	69	59	68	70	68	67
Tomcat	56	55	54	59	56	56	68	65	65
AspectJ	53	51	56	51	50	49	65	61	59
Birt	48	47	48	47	47	48	64	64	64
SWT	52	48	53	63	51	62	69	68	68
JDT	56	51	57	57	53	57	65	63	62
f-measure	**GA K=10**	**LS K=10**	**HS K=10**	**GA K=15**	**LS K=15**	**HS K=15**	**GA K=20**	**LS K=20**	**HS K=20**
EclipseUI	62	53	50	62	51	50	59	54	48

Tomcat	52	49	48	55	43	51	43	41	40
AspectJ	49	45	44	47	39	44	45	41	47
Birt	46	44	43	43	37	38	39	38	36
SWT	48	44	44	52	41	43	47	46	41
JDT	50	46	44	48	40	45	45	42	41
g-mean	**GA K=10**	**LS K=10**	**HS K=10**	**GA K=15**	**LS K=15**	**HS K=15**	**GA K=20**	**LS K=20**	**HS K=20**
Eclipse UI	0.774	0.744	0.739	0.724	0.714	0.709	0.692	0.661	0.641
Tomcat	0.713	0.677	0.651	0.683	0.677	0.651	0.663	0.637	0.621
AspectJ	0.696	0.656	0.634	0.666	0.636	0.618	0.626	0.602	0.574
Birt	0.596	0.586	0.572	0.596	0.586	0.572	0.524	0.511	0.501
SWT	0.644	0.626	0.619	0.614	0.577	0.579	0.565	0.523	0.514
JDT	0.676	0.652	0.647	0.56627	0.532	0.517	0.521	0.500	0.472
Balance	**GA K=10**	**LS K=10**	**HS K=10**	**GA K=15**	**LS K=15**	**HS K=15**	**GA K=20**	**LS K=20**	**HS K=20**
EclipseUI	72	65	65	71	64	66	69	65	64
Tomcat	67	66	60	72	70	64	73	72	69
AspectJ	62	61	58	68	65	60	70	67	65
Birt	63	63	60	70	66	62	71	70	68
SWT	65	63	61	65	64	63	67	66	64
JDT	63	62	57	64	62	59	65	64	65

Table A.1 : Median value of 30 runs on different mono objective technique for k =10,15 and 20

Reference

1. Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.
2. Almhana, R., Mkaouer, W., Kessentini, M., & Ouni, A. (2016, August). Recommending relevant classes for bug reports using multi-objective search. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering* (pp. 286-295). ACM.
3. Bettenburg, N., Just, S., Schröter, A., Weiss, C., Premraj, R., & Zimmermann, T. (2008, November). What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering* (pp. 308-318). ACM.
4. Saha, R. K., Lease, M., Khurshid, S., & Perry, D. E. (2013, November). Improving bug localization using structured information retrieval. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on* (pp. 345-355). IEEE.
5. Saha, R. K., Lease, M., Khurshid, S., & Perry, D. E. (2013, November). Improving bug localization using structured information retrieval. In *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on* (pp. 345-355). IEEE.
6. Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R., & Whitehead Jr, E. J. (2013, May). Does bug prediction support human developers? findings from a google case study.

In *Proceedings of the 2013 International Conference on Software Engineering* (pp. 372-381). IEEE Press.

7. Harman, M., McMinn, P., De Souza, J. T., & Yoo, S. (2012, January). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification* (pp. 1-59). Springer-Verlag..
8. Nam, J. (2014). Survey on software defect prediction. *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology, Tech. Rep*.
9. Kim, M., Hiroyasu, T., Miki, M., & Watanabe, S. (2004, September). SPEA2+: Improving the performance of the strength Pareto evolutionary algorithm 2. In *PPSN* (pp. 742-751).
10. Zitzler, E., Laumanns, M., & Thiele, L. (2001). SPEA2: Improving the strength Pareto evolutionary algorithm.
11. Salton, G., Wong, A., & Yang, C. S. (1975). A Vector Space Model for Automatic Indexing "Communications of the ACM Vol. 18.
12. Gupta, V., Ganeshan, N., & Singhal, T. K. (2015). Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicators. *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, *6*(2), 60-65.
13. Nguyen, A. T., Nguyen, T. T., Al-Kofahi, J., Nguyen, H. V., & Nguyen, T. N. (2011, November). A topic-based approach for narrowing the search space of buggy files from a bug report. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on* (pp. 263-272). IEEE.
14. Ye, X., Bunescu, R., & Liu, C. (2014, November). Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 689-699). ACM.
15. Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of machine Learning research*, *3*(Jan), 993-1022.
16. Zhou, J., Zhang, H., & Lo, D. (2012, June). Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 14-24). IEEE Press.

17. Zitzler, E., Laumanns, M., & Bleuler, S. (2004). A tutorial on evolutionary multiobjective optimization. *Metaheuristics for multiobjective optimisation*, 3-37.
18. Arcuri, A., & Fraser, G. (2013). Parameter tuning or default values? An empirical investigation in search-based software engineering. *Empirical Software Engineering*, *18*(3), 594-623.
19. Bettenburg, N., Premraj, R., Zimmermann, T., & Kim, S. (2008, September). Duplicate bug reports considered harmful… really?. In *Software maintenance, 2008. ICSM 2008. IEEE international conference on* (pp. 337-345). IEEE..
20. Dreyton, D., Araújo, A. A., Dantas, A., Freitas, Á., & Souza, J. (2015, September). Search-based bug report prioritization for kate editor bugs repository. In *International Symposium on Search Based Software Engineering* (pp. 295-300). Springer, Cham.
21. Dumais, S. T. (2004). Latent semantic analysis. *Annual review of information science and technology*, *38*(1), 188-230..
22. Enslen, E., Hill, E., Pollock, L., & Vijay-Shanker, K. (2009, May). Mining source code to automatically split identifiers for software analysis. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on* (pp. 71-80). IEEE..
23. Fischer, M., Pinzger, M., & Gall, H. (2003, November). Analyzing and relating bug report data for feature tracking. In *WCRE* (Vol. 3, p. 90).
24. Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. *Foundations of genetic algorithms*, *1*, 69-93.
25. Harman, M., & Jones, B. F. (2001). Search-based software engineering. *Information and software Technology*, *43*(14), 833-839.
26. Harman, M., Mansouri, S. A., & Zhang, Y. (2012). Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, *45*(1), 11.
27. Henard, C., Papadakis, M., & Le Traon, Y. (2014, August). Mutation-Based Generation of Software Product Line Test Configurations. In *SSBSE* (pp. 92-106).
28. Nguyen, A. T., Nguyen, T. T., Al-Kofahi, J., Nguyen, H. V., & Nguyen, T. N. (2011, November). A topic-based approach for narrowing the search space of buggy files from a bug report. In *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on* (pp. 263-272). IEEE.

29. Kubat, M., & Matwin, S. (1997, July). Addressing the curse of imbalanced training sets: one-sided selection. In *ICML* (Vol. 97, pp. 179-186).
30. King, R. A., Deb, K., & Rughooputh, H. C. S. (2010). Comparison of NSGA-II and SPEA2 on the Multiobjective Environmental/Economic Dispatch Problem. *University of Mauritius Research Journal*, *16*(1), 485-511.
31. Rao, S., & Kak, A. (2011, May). Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *Proceedings of the 8th Working Conference on Mining Software Repositories* (pp. 43-52). ACM.
32. Saha, R. K., Lawall, J., Khurshid, S., & Perry, D. E. (2014, September). On the effectiveness of information retrieval based bug localization for c programs. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on* (pp. 161-170). IEEE.
33. Salton, G., Wong, A., & Yang, C. S. (1975). A Vector Space Model for Automatic Indexing "Communications of the ACM Vol. 18.
34. Shelburg, J., Kessentini, M., & Tauritz, D. R. (2013, August). Regression testing for model transformations: A multi-objective approach. In *International Symposium on Search Based Software Engineering* (pp. 209-223). Springer, Berlin, Heidelberg.
35. Sun, C., Lo, D., Wang, X., Jiang, J., & Khoo, S. C. (2010, May). A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1* (pp. 45-54). ACM.
36. He, H., & Garcia, E. A. (2009). Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, *21*(9), 1263-1284.
37. Wong, C. P., Xiong, Y., Zhang, H., Hao, D., Zhang, L., & Mei, H. (2014, September). Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on* (pp. 181-190). IEEE.
38. Ye, X., Bunescu, R., & Liu, C. (2014, November). Learning to rank relevant files for bug reports using domain knowledge. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 689-699). ACM.

39. Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, *6*(2), 182-197.
40. Zhou, J., Zhang, H., & Lo, D. (2012, June). Where should the bugs be fixed?-more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering* (pp. 14-24). IEEE Press.
41. Deb, K., & Sundar, J. (2006, July). Reference point based multi-objective optimization using evolutionary algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation* (pp. 635-642). ACM.
42. Kunkle, D. (2005). A summary and comparison of MOEA algorithms. In *Internal Report*. College of Computer and Information Science, Northeastern University. .
43. Malhotra, R., & Khanna, M. (2016). An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Automated Software Engineering*, *3*(24), 673-717.
44. Aljahdali, S. H., & El-Telbany, M. E. (2009, May). Software reliability prediction using multi-objective genetic algorithm. In *Computer Systems and Applications, 2009. AICCSA 2009. IEEE/ACS International Conference on* (pp. 293-300). IEEE.
45. Minku, L. L., & Yao, X. (2013, October). An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation. In *Proceedings of the 9th international conference on predictive models in software engineering*(p. 8). ACM.
46. Ryu, D., & Baik, J. (2016). Effective multi-objective naïve bayes learning for cross-project defect prediction. *Applied Soft Computing*, *49*, 1062-1077.
47. Ouni, A., Kessentini, M., Sahraoui, H., & Boukadoum, M. (2013). Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering*, 1-33.
48. Harman, M., Islam, S. S., Jia, Y., Minku, L. L., Sarro, F., & Srivisut, K. (2014, August). Less is More: Temporal Fault Predictive Performance over Multiple Hadoop Releases. In *SSBSE* (pp. 240-246).

49. Menzies, T., Greenwald, J., & Frank, A. (2007). Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering*, *33*(1), 2-13.

Thesis1

ORIGINALITY REPORT

%10	%	%10	%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Lecture Notes in Computer Science, 2012. Publication	%1
2	Lecture Notes in Computer Science, 2016. Publication	%1
3	Ye, Xin, Razvan Bunescu, and Chang Liu. "Learning to rank relevant files for bug reports using domain knowledge", Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014, 2014. Publication	%1
4	Ruchika Malhotra, Megha Khanna. "An exploratory study for software change prediction in object-oriented systems using hybridized techniques", Automated Software Engineering, 2016 Publication	%1
5	"Evolutionary Multi-Criterion Optimization", Springer Nature, 2017 Publication	%1
6	Lecture Notes in Computer Science, 2013. Publication	<%1

7	Ye, Xin, Razvan Bunescu, and Chang Liu. "Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-grained Benchmark, and Feature Evaluation", IEEE Transactions on Software Engineering, 2015. Publication	<%1
8	Zhou, Jian, Hongyu Zhang, and David Lo. "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports", 2012 34th International Conference on Software Engineering (ICSE), 2012. Publication	<%1
9	Minku, Leandro L., and Xin Yao. "An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation", Proceedings of the 9th International Conference on Predictive Models in Software Engineering - PROMISE 13, 2013. Publication	<%1
10	Ouni, Ali, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Mohamed Salah Hamdi. "Improving multi-objective code-smells correction using development history", Journal of Systems and Software, 2015. Publication	<%1

11	Niu, Haoran, Iman Keivanloo, and Ying Zou. "Learning to rank code examples for code search engines", Empirical Software Engineering, 2016. Publication	<%1
12	Ouni, Ali, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Kalyanmoy Deb. "Multi-Criteria Code Refactoring Using Search-Based Software Engineering : An Industrial Case Study", ACM Transactions on Software Engineering and Methodology, 2016. Publication	<%1
13	Saha, Ripon K., Lingming Zhang, Sarfraz Khurshid, and Dewayne E. Perry. "An Information Retrieval Approach for Regression Test Prioritization Based on Program Changes", 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 2015. Publication	<%1
14	Panduro, M.A.. "Design of electronically steerable linear arrays with evolutionary algorithms", Applied Soft Computing Journal, 200801 Publication	<%1
15	L. Diosan. "A multi-objective evolutionary approach to the portfolio optimization problem", International Conference on	<%1

Computational Intelligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and Internet Commerce (CIMCA-IAWTIC 06), 2005
Publication

---

16 Saha, Ripon K., Matthew Lease, Sarfraz Khurshid, and Dewayne E. Perry. "Improving bug localization using structured information retrieval", 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013. <%1
Publication

---

17 Lattarulo, Valerio, and Geoffrey T. Parks. "A preliminary study of a new multi-objective optimization algorithm", 2012 IEEE Congress on Evolutionary Computation, 2012. <%1
Publication

---

18 Liagkouras, K. Metaxiotis K.. "A new probe guided mutation operator for more efficient exploration of the search space: an experime", International Journal of Operational Res, Dec 31 2015 Issue <%1
Publication

---

19 Ogawa, Hideto, Makoto Ichii, Fumihiro Kumeno, and Toshiaki Aoki. "Experimental Fault Analysis Process Implemented Using Model Extraction and Model Checking", 2015 IEEE 39th Annual Computer Software and <%1

Applications Conference, 2015.
Publication

20	Communications in Computer and Information Science, 2012. Publication	<%1
21	Lecture Notes in Computer Science, 2005. Publication	<%1
22	Everett, M. E., and S. Constable. "Electric dipole fields over an anisotropic seafloor: theory and application to the structure of 40Ma Pacific Ocean lithosphere", Geophysical Journal International, 1999. Publication	<%1
23	Denizart, M.. "Triode and tetrode guns for field emission electron microscopes", Ultramicroscopy, 1981 Publication	<%1
24	Jindaluang, Wattana, Varin Chouvatut, and Sanpawat Kantabutra. "Under-sampling by algorithm with performance guaranteed for class-imbalance problem", 2014 International Computer Science and Engineering Conference (ICSEC), 2014. Publication	<%1
25	Shihab, Emad, Akinori Ihara, Yasutaka Kamei, Walid M. Ibrahim, Masao Ohira, Bram Adams, Ahmed E. Hassan, and Ken-ichi Matsumoto. "Studying re-opened bugs in open source software", Empirical Software	<%1

Engineering, 2013.
Publication

---

26 Deo Vidyarthi. "Multi-objective optimization for channel allocation in mobile computing using NSGA-II", International Journal of Network Management, 05/2011 <%1
Publication

---

27 F. Marcelloni. "A Comparison of Multi-Ojective Evolutionary Algorithms in Fuzzy Rule-Based Systems Generation", NAFIPS 2006 - 2006 Annual Meeting of the North American Fuzzy Information Processing Society, 06/2006 <%1
Publication

---

28 Davies, Steven, Marc Roper, and Murray Wood. "Using Bug Report Similarity to Enhance Bug Localisation", 2012 19th Working Conference on Reverse Engineering, 2012. <%1
Publication

---

29 C.P. Loizou, M. Pantziaris, I. Seimenis, C.S. Pattichis. "Brain MR image normalization in texture analysis of multiple sclerosis", 2009 9th International Conference on Information Technology and Applications in Biomedicine, 2009 <%1
Publication

---

30 Biggers, Lauren R., Cecylia Bocovich, Riley Capshaw, Brian P. Eddy, Letha H. Etzkorn, <%1

and Nicholas A. Kraft. "Configuring latent Dirichlet allocation based feature location", Empirical Software Engineering, 2014.
Publication

31	Lecture Notes in Computer Science, 2014. Publication	<%1
32	Ilhem Boussaïd, Patrick Siarry, Mohamed Ahmed-Nacer. "A survey on search-based model-driven engineering", Automated Software Engineering, 2017 Publication	<%1
33	Vachhani, Vimal L., Vipul K. Dabhi, and Harshadkumar B. Prajapati. "Survey of multi objective evolutionary algorithms", 2015 International Conference on Circuits Power and Computing Technologies [ICCPCT-2015], 2015. Publication	<%1
34	"Fundamental Approaches to Software Engineering", Springer Nature, 2017 Publication	<%1
35	Software and Systems Traceability, 2012. Publication	<%1
36	Nilam Kaushik. "A Comparative Study of the Performance of IR Models on Duplicate Bug Detection", 2012 16th European Conference on Software Maintenance and Reengineering, 03/2012 Publication	<%1

37	Jun Yang, Hongbing Qian. "Defect Prediction on Unlabeled Datasets by Using Unsupervised Clustering", 2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 Publication	<%1
38	Ouni, A., M. Kessentini, and H. Sahraoui. "Search-Based Refactoring Using Recorded Code Changes", 2013 17th European Conference on Software Maintenance and Reengineering, 2013. Publication	<%1
39	Armitage. "Front Matter", Networking and Online Games, 04/04/2006 Publication	<%1
40	Hans-Otto Georgii. "Gibbs Measures and Phase Transitions", Walter de Gruyter GmbH, 2011 Publication	<%1
41	Malhotra, Ruchika, and Ankita Jain. "Software Effort Prediction using Statistical and Machine Learning Methods", International Journal of Advanced Computer Science and Applications, 2011. Publication	<%1

42	Tian, Yuan, Nasir Ali, David Lo, and Ahmed E. Hassan. "On the unreliability of bug severity data", Empirical Software Engineering, 2015. Publication	<%1
43	Lian, Xiaoli, and Li Zhang. "Optimized feature selection towards functional and non-functional requirements in Software Product Lines", 2015 IEEE 22nd International Conference on Software Analysis Evolution and Reengineering (SANER), 2015. Publication	<%1
44	Fernandez, A.. "A study of the behaviour of linguistic fuzzy rule based classification systems in the framework of imbalanced data-sets", Fuzzy Sets and Systems, 20080916 Publication	<%1
45	Ouni, Ali, Marouane Kessentini, Slim Bechikh, and Houari Sahraoui. "Prioritizing code-smells correction tasks using chemical reaction optimization", Software Quality Journal, 2015. Publication	<%1
46	Ali T. Al-Awami. "Stochastic Dispatch of Power Grids with High Penetration of Wind Power Using Pareto Optimization", Green Energy and Technology, 2010 Publication	<%1

47 Tantithamthavorn, Chakkrit, Akinori Ihara, and Ken-Ichi Matsumoto. "Using Co-change Histories to Improve Bug Localization Performance", 2013 14th ACIS International Conference on Software Engineering Artificial Intelligence Networking and Parallel/Distributed Computing, 2013.
Publication
<%1

48 Rohaninejad, Mohamad, Amirsaman Kheirkhah, Parviz Fattahi, and Behdin Vahedi-Nouri. "A hybrid multi-objective genetic algorithm based on the ELECTRE method for a capacitated flexible job shop scheduling problem", The International Journal of Advanced Manufacturing Technology, 2015.
Publication
<%1

49 Dongsun Kim, , Yida Tao, Sunghun Kim, and Andreas Zeller. "Where Should We Fix This Bug? A Two-Phase Recommendation Model", IEEE Transactions on Software Engineering, 2013.
Publication
<%1

50 Ryu, Duksan, and Jongmoon Baik. "Effective multi-objective na&#65533;ve Bayes learning for cross-project defect prediction", Applied Soft Computing, 2016.
Publication
<%1

51 Pei Chann Chang. "MOEA/D for flowshop

scheduling problems", 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), 06/2008
Publication

<%1

52 Zhang, Tao, He Jiang, Xiapu Luo, and Alvin T.S. Chan. "A Literature Review of Research in Bug Resolution: Tasks, Challenges and Future Directions", The Computer Journal, 2016.
Publication

<%1

53 Scanniello, Giuseppe, Andrian Marcus, and Daniele Pascale. "Link analysis algorithms for static concept location: an empirical assessment", Empirical Software Engineering, 2015.
Publication

<%1

54 Basu, M.. "Economic environmental dispatch using multi-objective differential evolution", Applied Soft Computing Journal, 201103
Publication

<%1

55 Eduardo Lupiani. "An Evolutionary Multiobjective Constrained Optimisation Approach for Case Selection: Evaluation in a Medical Problem", Lecture Notes in Computer Science, 2011
Publication

<%1

56 Li, Miqing, Shengxiang Yang, Jinhua Zheng, and Xiaohui Liu. "ETEA: A Euclidean

<%1

Minimum Spanning Tree-Based Evolutionary Algorithm for Multi-Objective Optimization", Evolutionary Computation, 2014.
Publication

---

57 Shin, Kyoung Seok Kim, Jun Hyuk Kim, Yeo. "A two-leveled multi-objective symbiotic evolutionary algorithm for the hub and spoke location proble", Journal of Advanced Transportation, Winter 2009 Issue <%1
Publication

---

58 Evolving Software Systems, 2014. <%1
Publication

---

59 Cote, P.. "Multi-objective optimization of an ecological assembly model", Ecological Informatics, 20070101 <%1
Publication

---

60 Lin, Meng-Jie, Cheng-Zen Yang, Chao-Yuan Lee, and Chun-Chang Chen. "Enhancements for duplication detection in bug reports with manifold correlation features", Journal of Systems and Software, 2016. <%1
Publication

---

61 Malhotra, Ruchika, and Ankita Jain. "Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality", Journal of Information Processing Systems, 2012. <%1
Publication

EXCLUDE QUOTES	OFF	EXCLUDE MATCHES	OFF
EXCLUDE BIBLIOGRAPHY	ON		