The Journal of Systems and Software 152 (2019) 215–238

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

An empirical study on pareto based multi-objective feature selection for software defect prediction

Chao Ni[a], Xiang Chen[a,b,*], Fangfang Wu[a], Yuxiang Shen[b], Qing Gu[a,*]

[a] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
[b] School of Computer Science and Technology, Nantong University, Nantong, China

ARTICLE INFO

Article history:
Received 16 August 2018
Revised 8 February 2019
Accepted 14 March 2019
Available online 15 March 2019

MSC:
xx-xx
xx-xx

Keywords:
Software defect prediction
Search based software engineering
Feature selection
Multi-Objective optimization
Empirical study

ABSTRACT

The performance of software defect prediction (SDP) models depend on the quality of considered software features. Redundant features and irrelevant features may reduce the performance of the constructed models, which require feature selection methods to identify and remove them. Previous studies mostly treat feature selection as a single objective optimization problem, and multi-objective feature selection for SDP has not been thoroughly investigated. In this paper, we propose a novel method MOFES (Multi-Objective FEature Selection), which takes two optimization objectives into account. One optimization objective is to minimize the number of selected features, this objective is related to the cost analysis of this problem. Another objective is to maximize the performance of the constructed SDP models, this objective is related to the benefit analysis of this problem. MOFES utilizes Pareto based multi-objective optimization algorithms (PMAs) to solve this problem. In our empirical study, we design and conduct experiments on RELINK and PROMISE datasets, which are gathered from real open source projects. Firstly, we analyze the influence of different PMAs on MOFES and find that NSGA-II can achieve the best performance on both datasets. Then, we compare MOFES method with 22 state-of-the-art filter based and wrapper based feature selection methods, and find that MOFES can effectively select fewer but closely related features to construct high-quality models. Moreover, we also analyze the frequently selected features by MOFES, and these findings can be used to provide guidelines on gathering high-quality SDP datasets. Finally, we analyze the computational cost of MOFES and find that MOFES only needs 107 seconds on average.

1. Introduction

Software defects are introduced during the coding process of developers unconsciously. The reasons may come from misunderstanding of the software requirements, unreasonable development process, or the lack of development experience. Software with defects will produce unexpected results or behaviors after the software deployment, even will cause huge economic loss for enterprises in worst cases. Therefore, project managers want to use software testing or code inspection to detect as many defects as possible. However, testing resources are overwhelmingly limited, project managers hope that they can utilize effective methods to identify potential defective modules as early as possible and then allocate enough testing resources on them. Software defect prediction (SDP) (Hall et al., 2012; Kamei and Shihab, 2016; Yang et al., 2015; Yan et al., 2017) is one of such effective methods. It constructs SDP models by mining software repositories (such as version control systems, bug tracking systems, developer emails) and uses the constructed SDP models to predict potential defective modules.

During the process of gathering SDP datasets, researchers have designed different software features[1] to measure extracted modules and these features often have strong correlation with software defects. These software features are designed based on the analysis of code complexity or development process (Chidamber and Kemerer, 1994; Nagappan and Ball, 2005; Moser et al., 2008; Hassan, 2009; Radjenovic et al., 2013; Rahman and Devanbu, 2013). However, not all the features are beneficial to the construction of the SDP models. In particular, redundant features and irrelevant features will increase the model construction time, and even sometimes decrease the performance of the constructed models (Ghotra et al., 2017; Xu et al., 2016a). This problem is called

* Corresponding authors.
E-mail addresses: jacknichao920209@gmail.com (C. Ni), xchencs@ntu.edu.cn (X. Chen), 2402724397@qq.com (F. Wu), shenyxcs@gmail.com (Y. Shen), guq@nju.edu.cn (Q. Gu).

[1] In the literatures of software defect prediction, software features are also called software metrics.

https://doi.org/10.1016/j.jss.2019.03.012
0164-1212/

the curse of dimensionality. Designing novel and effective feature selection methods is a potential way to solve this problem. Moreover, fewer features can be helpful for the interpretation of the models and the visualization of the models. Previous studies have shown the ubiquitous existence of irrelevant features and redundant features in SDP datasets (Khoshgoftaar et al., 2012; Ghotra et al., 2017). For example, Khoshgoftaar et al. (2012) showed that the prediction performance of their proposed methods got worse when the SDP datasets contained incomplete or irrelevant features. Ghotra et al. (2017) conducted a dataset redundancy characteristic analysis by using principal component analysis. They found that to account for 95% of the data variance, the NASA datasets need an average of 31% of the components while the PROMISE datasets need an average of 59% of the components.

Nowadays, researchers have applied feature selection to software defect prediction and made some progress (Menzies et al., 2007; Gao et al., 2011; Wang et al., 2011; Song et al., 2011; Shivaji et al., 2013; Liu et al., 2014b, 2015, 2016; Xu et al., 2016a, 2016b; Ni et al., 2017a, 2017b). However, none of previous studies viewed this issue as a multi-objective optimization problem. To the best of our knowledge, we are the first to thoroughly evaluate multi-objective feature selection for SDP. In particular, we mainly take into account two optimization objectives. One optimization objective is to minimize the number of selected features, this objective is related to the cost analysis of this problem. Another objective is to maximize the performance of constructed SDP models, this objective is related to the benefit analysis of this problem. To address this problem, we propose MOFES method and this method utilizes Pareto based multi-objective optimization algorithms (PMAs) to solve this problem. To verify the effectiveness of our proposed method, we design and conduct a series of empirical studies. We choose RELINK and PROMISE datasets, which are gathered from real open source projects, as our experimental subjects. We use four different classifiers as the model construction methods. In particular, they are decision tree based classifier (J48), lazy based classifier (*K* Nearest Neighbor, KNN), function based classifier (logistic regression, LR), and probability based classifier (naive bayes, NB). We use AUC as the performance evaluation measure.

The findings of our empirical studies can be summarized as follows. First, we analyze the influence of different PMAs on MOFES and find NSGA-II (Deb et al., 2002) can achieve the best performance in terms of hypervolume quality indicator. Then, we compare MOFES method with 22 state-of-the-art filter based and wrapper based feature selection methods, final results show that MOFES can obtain better performance while choosing fewer features. Later, we analyze the frequently selected features by MOFES and find that features in different feature categories may obtain different performances in the context of SDP. MOFES proposed in this paper can make good use of the features from different feature categories, which is help for constructing high-quality SDP models. Finally, we analyze the computational cost of our method and show that MOFES only needs 107 seconds on average.

This paper extends our previous study (Chen et al., 2017) by considering more datasets from real-world open source projects, more Pareto based multi-object optimization algorithms as well as by conducting more extensive empirical studies. In particular, firstly, we further consider RELINK datasets to show the generality of our previous empirical results. Secondly, MOFES considers other 4 PMAs (i.e., MOCell, SPEA2, PAES and SMSEMOA). Thirdly, we compare MOFES with 22 state-of-the-arts wrapper based or filter based feature selection methods and most of these baseline methods are not considered in our previous study. Then we analyze the features which are frequently chosen by MOFES to provide guidelines for gathering high-quality SDP datasets in the future. Finally, we investigate the computational cost of MOFES and other feature selection baseline methods.

The main contributions of this paper can be summarized as follows:

- To the best of our knowledge, we are the first to investigate the performance of applying multi-objective feature selection to software defect prediction. Though recently there are two large-scale empirical studies (Ghotra et al., 2017; Xu et al., 2016a), which analyze the impact of different feature selection methods on SDP, these empirical studies do not consider feature selection methods using multi-objective optimization. Therefore, our study is an important supplement to previous studies on this issue.
- We consider a wide range of Pareto based multi-objective optimization algorithms (i.e., NSGA-II, MOCell, SPEA2, PAES and SMSEMOA) for our method MOFES. We compare the performance of these different PMAs in terms of hypervolume quality indicator and find that NSGA-II can achieve the best performance.
- We compare MOFES based on NSGA-II with 22 state-of-the-art wrapper based and filter based feature selection methods. Final comparison results verify the effectiveness of MOFES.

The rest of this paper is organized as follows. Section 2 introduces the background of software defect prediction and previous studies on applying feature selection to software defect prediction. Section 3 describes our proposed method MOFES in detail. Section 4 reports our empirical setup, including experimental subjects, feature selection baseline methods, evaluation performance measures and experimental design. Section 5 discusses the results of our experiments and analyzes the potential threats to validity for our empirical results. Section 6 concludes the paper with some future work.

2. Background and related work

2.1. Background of software defect prediction

Software defect prediction (SDP) (Hall et al., 2012; Kamei and Shihab, 2016; Tantithamthavorn et al., 2017; Lewis et al., 2013; Chen et al., 2019) is an active research issue in software engineering data mining domain. It can be used to identify potential defective modules in advance, and then allocate more testing resources on these modules. The process of SDP is shown in Fig. 1. This process can be divided into two phases (i.e., SDP model construction phase and SDP model application phase). In the model construction phase, it first mines software historical repositories (such as version control systems, bug tracking systems, developer emails) to extract and label program modules (here red blocks represent defective modules and green blocks represent non-defective modules). The granularity of the modules can be set as component, file, class or code change as needed (Menzies et al., 2007; Jureczko and Madeyski, 2010; Kim et al., 2008; Kamei et al., 2013; Chen et al., 2018). Then, it designs software features (Chidamber and Kemerer, 1994; Nagappan and Ball, 2005; Moser et al., 2008; Hassan, 2009; Radjenovic et al., 2013; Rahman and Devanbu, 2013) to measure extracted modules. These software features are mainly designed on the analysis of code complexity or development process. Based on the above steps, it can gather defect prediction datasets and use a specific classifier (such as logistic regression, decision tree, support vector machine) (Lessmann et al., 2008; Ghotra et al., 2015) to construct defect prediction models. In the model application phase, it can use the constructed models to predict new program modules as defect-prone or non defect-prone.

Data preprocessing is an important step in the model construction phase and can help to improve the model performance. Commonly used data preprocessing methods include feature selection,

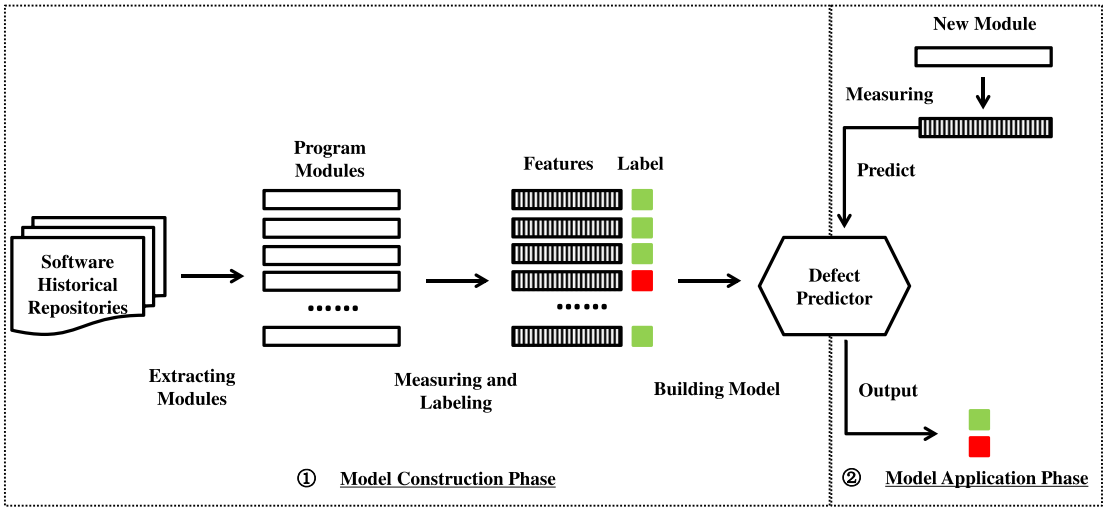

Fig. 1. The Process of Software Defect Prediction.

class imbalanced learning (Tan et al., 2015; Jing et al., 2017; Mahmood et al., 2015; Rodriguez et al., 2014; Bennin et al., 2017; Ozturk, 2017; Wang and Yao, 2013; Tantithamthavorn et al., 2018; Tantithamthavorn and Hassan, 2018), and noise identification and removing (Kim et al., 2011; Wu et al., 2011; Nguyen et al., 2012; Herzig et al., 2013; Tantithamthavorn et al., 2015). In this paper, we mainly focus on feature selection for software defect prediction.

2.2. Applying feature selection to software defect prediction

If program modules are measured by using many software features, the gathered datasets may have the problem of the curse of dimensionality. Until now, there are many methods have been proposed to solve the curse of dimensionality, which can be classified into two categories: feature selection methods and feature reduction methods. In particular, feature selection methods reduce the number of features by selecting a subset of the original features, while feature reduction methods reduce the number of features by combining original features into new features (Kondo et al., 2019). In this paper, we mainly focus on feature selection for alleviating curse of dimensionality, which aims to identify and remove irrelevant and redundant features as many as possible. In particular, an irrelevant feature is a feature, which has little correlation with the class. While a redundant feature is a feature, which contains information from one or more other features. Previous studies confirmed that the performance of SDP models may be hurt by irrelevant features or redundant features, since many classifiers are sensitive to these kinds of features (Menzies et al., 2007; Lessmann et al., 2008; Guo et al., 2004). Jiarpakdee et al. (2016) found that 10%∼67% of features in the 101 public SDP datasets are redundant.

However, feature selection is a challenge task if the search space is large. If a dataset has n features, the number of possible solutions is 2^n. Therefore, as the number of features n is increasing, the task is becoming more challenging. Feature selection is a challenge task also due to feature interaction problems. There are 2-way, 3-way or complex multi-way interactions among features. A feature, which is weakly relevant to the class by itself, could significantly improve the model performance if it is used together with some complementary features. While an individually relevant feature may become redundant when used together with other features. Therefore, selecting or removing these features may miss the optimal feature subsets.

Feature selection (Zhang et al., 2015; Nam et al., 2017; Hosseini et al., 2017; He et al., 2015; Catal and Diri, 2009; Yu et al., 2017) is an important data preprocessing step in SDP, which can reduce the dimensionality of the datasets, speedup the learning process, simplify the constructed models, and even increase the model performance. Until now, many feature selection methods have been proposed to solve this task. Existing feature selection methods can be classified into two categories: filter based methods and wrapper based methods. In particular, wrapper based methods evaluate the goodness of feature subsets by using the performance of the constructed models. While the filter based methods use general characteristics of datasets to evaluate the feature subsets. The computational cost of the latter methods is low, but the performance of the constructed models can not be guaranteed.

Nowadays, researchers applied feature selection to improve the performance of defect prediction models. Most of previous studies used filter based methods. Menzies et al. (2007) considered information gain based feature selection methods, which use forward selection strategy and exhaustive selection strategy. Gao et al. (2011) considered feature selection on a large-scale legacy software system in telecommunications. Their methods used different feature ranking and subset evaluation metrics. Wang et al. (2011) applied ensemble learning to combine different feature selection methods. Khoshgoftaar et al. (2012) examined 7 filter-based feature ranking methods for comparison based on 16 SDP datasets. These features include chi-squared (CS), information gain (IG), gain ratio (GR), symmetrical uncertainty (SU) and ReliefF with two variants (i.e., RF and RFW). We (Liu et al., 2014b) proposed a feature selection framework FECAR by using feature clustering and feature ranking. Then, we (Liu et al., 2016) further proposed a two-stage data preprocessing approach, which incorporates feature selection and instance reduction. Recently, we found that noises are inevitable when gathering SDP datasets. Therefore, we (Liu et al., 2015) proposed a framework FECS, which has a certain noise tolerance ability. Similar to FECAR (Liu et al., 2014b; Xu et al., 2016b) also proposed a cluster analysis based feature selection method MICHAC (i.e., maximal information coefficient with hierarchical agglomerative clustering), In particular, they used maximal information coefficient to rank candidate features to remove irrelevant features. Then, they grouped features by utilizing hierarchical agglomerative clustering and selected one feature from each cluster to remove redundant features. Shivaji et al. (2013) applied different feature selection methods to code change based defect prediction proposed by Kim et al. (2008). Wang et al. (2010) conducted a comprehensive empirical study on examining 17 different ensembles of feature ranking meth-

ods (i.e., rankers), which include 6 commonly-used feature ranking methods, a signal-to-noise filter method, and 11 threshold-based feature ranking methods. Experimental results indicated that ensemble methods based on a few rankers are effective and sometimes even better than ensemble methods based on many or all rankers. Muthukumaran et al. (2015) investigated 7 feature ranking methods, 2 wrapper based methods and an embedded method on the noisy NASA datasets and AEEEM datasets. They found that there have no significant difference among these methods. Liu et al. (2014a) proposed a new two-stage cost-sensitive learning method for SDP. This method utilized cost information not only in the classification stage but also in the feature selection stage. In particular, they developed 3 novel cost-sensitive feature selection methods. These methods are CSVS (Cost-Sensitive Variance Score), CSLS (Cost-Sensitive Laplacian Score) and CSCS (Cost-Sensitive Constraint Score). Experimental results demonstrated that the proposed CSCS method outperforms single-stage cost-sensitive learning methods, while the proposed cost-sensitive feature selection methods perform better than conventional cost-blind feature selection methods. Khoshgoftaar et al. (2010) proposed a method involving a feature selection method for selecting the important features and an instance sampling method for addressing class imbalanced problem. Empirical results suggested that feature selection on sampled datasets performs significantly better than feature selection based on the original datasets.

There are a few studies on applying wrapper based methods to SDP. Song et al. (2011) considered wrapper based feature selection in their general defect prediction framework. Xu et al. (2016a) investigated 32 different feature selection methods. These methods are divided into 5 families (i.e., filter based feature ranking methods, filter based subset selection methods, wrapper based subset selection methods, clustering based methods and extraction based methods). In their large-scale empirical studies, they found different conclusions can be drawn based on different datasets. Later, Ghotra et al. (2017) considered 30 feature selection methods and 21 classifiers. They found that the correlation-based filter-subset feature selection method with the BestFirst search strategy outperforms other feature selection methods. Laradji et al. (2015) firstly investigated 3 feature selection methods for software defect prediction and observed that selecting a few high-quality features can achieve much higher performance than others in terms of AUC. Besides, there also exist a few studies investigating the impacts of software features on the interpretation of defect prediction models (Jiarpakdee et al., 2018, 2019). The interpretation of such models is used to build empirical theories that are related to software quality (i.e., defect-prone or non defect-prone). That is, what software features share the strongest association with software quality.

Based on the above analysis, we can find that researchers have applied feature selection to software defect prediction and obtained some achievements. However, none of previous studies viewed this issue as a multi-objective optimization problem (i.e., this problem is only treated as a single objective optimization problem). Therefore, in this paper we want to apply wrapper based feature selection, which uses multi-objective optimization, to this problem. We notice that Canfora et al. (2015) applied multi-objective optimization to software defect prediction. But they mainly focused on the issue of cross-project defect prediction and wanted to achieve specific compromise between the number of likely defect-prone modules that the developers would likely discover, and lines of code to be analyzed/tested. Different from their study, we formalize a different problem (i.e., feature selection for software defect prediction) as a multi-objective optimization problem and this problem certainly has different optimization objectives.

3. Our proposed method MOFES

Our study can be put into the domain of search based software engineering (SBSE). SBSE concept was first proposed by Harman (Harman et al., 2012). It has become a hot research topic in recent software engineering research. SBSE has been applied to many problems throughout the software life cycle, from requirement analysis, software design, to software maintenance. SBSE is promising because it can provide automated or semi-automated solutions for software engineering problems with large-scale complex problem spaces, which have multiple competing or even conflicting objectives. For software defect prediction, Mark (Harman, 2010) also firstly suggested that SBSE (in particular multi-objective optimization) can be potentially used to construct defect prediction models.

If we formalize feature selection for software defect prediction as a multi-objective optimization problem, we can use state-of-the-art Pareto based Multi-Objective Algorithms (PMAs) to solve this problem. In particular, in this problem, we mainly consider two objectives. One objective is to choose features as fewer as possible, this is from the cost point of view to analyze this problem. Another objective is to improve the performance of the constructed defect prediction models as much as possible, this is from the benefit point of view to analyze this problem. There is an obvious conflict between these two objectives in most cases. In particular, selecting less features maybe decrease the model performance. On the contrary, improving the model performance maybe need to select more features. Therefore, we should make a compromise between these two conflict objectives. In this paper, we take PMAs into consideration and propose method MOFES (Multi-Objective FEature Selection). By using this method, we can obtain a series of non-dominated feature subsets and choose appropriate feature subsets according to actual requirements (i.e., preferring to choose less features or preferring to construct models with higher performance).

In this section, we first give some definitions for multi-objective optimization. Then, we introduce our framework by utilizing a specific PMA (i.e., NSGA-II proposed by Deb et al., 2002). Finally, we introduce other PMAs, which can also be utilized by our framework.

3.1. Preliminaries

For the convenience of the subsequent description, we first give some definitions for multi-objective optimization in the context of feature selection for software defect prediction.

Definition 1 (Pareto Dominance)**.** Supposing w_i and w_j are two feasible solutions for this problem, we call w_i is Pareto dominance on w_j (i.e., $w_i \succ w_j$), if and only if: $benefit(w_i) > benefit(w_j)$ and $cost(w_i) \leq cost(w_j)$ or $benefit(w_i) \geq benefit(w_j)$ and $cost(w_i) < cost(w_j)$

In this problem, a feasible solution denotes the selected feature subset. Function *benefit*() returns the prediction performance when using this feature subset to preprocess the dataset and then build the model. Function *cost*() returns the number of selected features.

Definition 2 (Pareto Optimal Solution)**.** A feasible solution w is a Pareto optimal solution, if and only if there is no other feasible solution w^*, which is Pareto dominance on w.

Definition 3 (Pareto Optimal Set)**.** This set is composed by all the Pareto optimal solutions.

Definition 4 (Pareto Front)**.** The surface composed by the vectors corresponding to all the Pareto optimal solutions is called Pareto front.

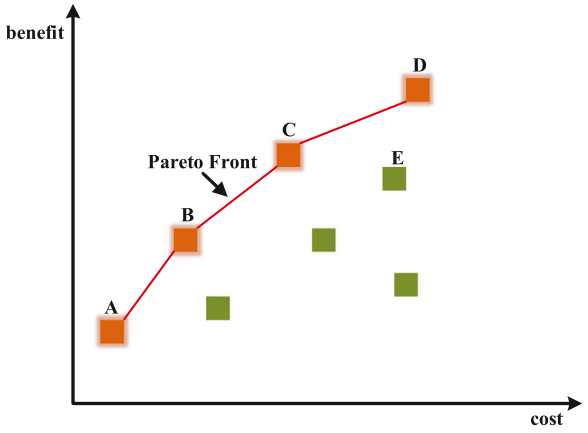

Fig. 2. Interpretation for Definitions in Multi-Objective Optimization.

We use a simple example to interpret these definitions. Supposing there are 7 candidate solutions for MOFES on a training set and these solutions are shown in Fig. 2. In this figure, x-axis denotes the cost value of the solution and y-axis denotes the benefit value of the solution. Based on Definition 1, we can find that solution C is Pareto dominance on solution E, since $benefit(C) > benefit(E)$ and $cost(C) < cost(E)$. Solutions A, B, C and D are pareto optimal solutions based on Definition 2, since there is no other solutions which are Pareto dominance on them. Therefore, the surface A-B-C-D constitutes Pareto front in these solutions.

3.2. MOFES by utilizing NSGA-II

Before utilizing MOAs, we first illustrate the encoding schema and the fitness value of the chromosome. For the chromosome encoding schema, we encode a feasible solution into n bit string if a dataset has n features. If the value of the i-th bit is set as 1, it means the i-th feature is selected. Otherwise if the value is set as 0, it means the i-th feature is not selected. Supposing there are 5 features $\{f_1, f_2, f_3, f_4, f_5\}$ and a initial population is {10010, 00100, 10110}. This means that the initial population include 3 chromosomes and these chromosomes correspond to these feature subsets (i.e., $\{f_1, f_4\}$, $\{f_3\}$ and $\{f_1, f_3, f_4\}$). There are two fitness values for each chromosome. From the benefit perspective, the fitness value of the chromosome returns the model performance on the training set by using the corresponding feature subset. More details on how to compute the fitness value can be found in the experimental design (in Section 4). From the cost perspective, the fitness value of the chromosome returns the size of the selected feature subset. If the selected feature subset is $\{f_1, f_4\}$, the corresponding fitness value is 2.

The algorithm of MOFES by utilizing NSGA-II can be found in Algorithm 1. In Algorithm 1, we first use *initPop*() function in Step 2 to initialize a population. The population has N chromosomes and each chromosome is randomly generated. For feature selection in SDP, there are three different heuristic initialize strategies (i.e., small initial strategy, large initial strategy and hybrid initial strategy). For the small initial strategy, chromosomes are randomly generated with fewer features (i.e., less than half of features). For the large initial strategy, chromosomes are randomly generated with more features (i.e., larger than half of features). For the hybrid initial strategy, chromosome are randomly generated. Half have fewer features and other half have more features. For MOFES, we use the small initial strategy to improve the performance of our method. More specifically, in our method each chromosome is generated by just randomly selecting only one feature.

Then, we use *makeNewPop*() function in Step 4 to generate new chromosomes by using classical evolutionary operators (such as crossover operator, mutation operator) in genetic algorithm. For example, the crossover operator will randomly choose two chromosomes according to crossover probability, perform crossover operation, and generate two new chromosomes. The mutation operator will randomly choose a chromosome according to mutation probability, perform mutation operation, and generate a new chromosome.

Algorithm 1 MOFES by Utilizing NSGA-II.

Input:
Population Size: N
Maximum Iteration Number: T
Output:
Pareto Optimal Set

1: $i \leftarrow 0$
2: $P_i \leftarrow initPop(N)$
3: **while** $i < T$ **do**
4: $C_i \leftarrow makeNewPop(P_i)$
5: $B_i \leftarrow P_i \cup C_i$
6: $F \leftarrow fastNondominatedSort(B_i)$
7: $P_{i+1} \leftarrow \emptyset$,
8: $j \leftarrow 1$
9: **while** $|P_{i+1}| + |F_j| \leq N$ **do**
10: $crowingDistanceAssign(F_j)$
11: $P_{i+1} \leftarrow P_{i+1} \cup F_j$
12: $j \leftarrow j + 1$
13: **end while**
14: $sort(F_j)$ //according to crowding distance
15: $P_{i+1} \leftarrow P_{i+1} \cup F_j[1 : (N - |P_{i+1}|)]$
16: $i \leftarrow i + 1$
17: **end while**
18: **return** Pareto optimal solutions in P_i

Later, we perform selection operation from Step 5 to Step 16 to select high-quality chromosomes into the new population. In particular, we first combine chromosomes in the previous population and new chromosomes generated by the mutation and crossover operations into B_i. Then, we use *fastNondominatedSort*() function to compute non-dominated ranks (NDR) for each candidate chromosome. First this function identifies all the non-dominated chromosomes in B_i, sets their NDR value as 1, puts these chromosomes into F_1, and removes these chromosomes from B_i. Then it continues to identify all the non-dominated chromosomes in B_i, sets their NDR value as 2, puts these chromosomes into F_2, and removes these chromosomes from B_i. When all the chromosomes have their NDR value, this process is terminated. Based on NDR value, we will select chromosomes with smaller NDR value as many as possible from Step 7 to Step 15. In Step 14, we use the concept of crowding distance. The crowding distance is the sum of the distance with other chromosomes, which have the same NDR value. Therefore, a chromosome with higher crowding distance will be in the low density regions of the search space. By considering crowding distance, we can avoid selecting chromosomes with high similarity. In the implementation of MOFES, we store the fitness value of each chromosome in the previous population to avoid repeated computation.

After sufficient population evolution, MOFES will satisfy the termination criterion and converge to stable solutions. Finally, MOFES returns all the Pareto optimal solutions in the current population.

3.3. Other PMAs

In addition to NSGA-II, we can also utilize other classical PMAs (Wang et al., 2016a). We consider the following PMAs: MOCell, SPEA2, PAES and SMSEMOA.

MOCell (Multi-objective cellular) (Nebro et al., 2007) is designed based on the cellular model of genetic algorithms. This PMA assumes that a chromosome only communicates with its neighbors during the evolutionary process. In addition, it uses an external archive to store a set of obtained non-dominated solutions.

SPEA2 (Improved strength Pareto evolutionary algorithm) (Zitzler and Thiele, 1999) computes the fitness value for each chromosome based on the objective function and the density estimation. This PMA uses the density estimation to maximize the population diversity by computing the distance between a solution and its nearest neighbors. SPEA also uses an archive to store a fixed number of best solutions.

Similar to MOCell and SPEA2, PAES (Pareto archived evolution strategy) (Knowles and Corne, 2000) uses an archive to store all the non-dominated solutions. The characteristic of PAES is that it uses a dynamic mutation operator for exploring the search space and aims to find optimal solutions.

SMSEMOA (Beume et al., 2007) combines ideas from other PMAs. It uses non-dominated sorting as a ranking criterion and applies hypervolume as the selection criterion to discard the chromosomes, which contribute the least hypervolume to the worst-ranked Pareto front.

Different from previous PMAs, we also consider RandomSearch. RandomSearch is used to represent the random search strategy. Both the evaluation and the selection functions are designed in a random way. This strategy is often chosen as the baseline method for other complex PMAs.

4. Experimental setup

To verify the effectiveness of our proposed MOFES, we design the following research questions.

RQ1: Among the Pareto-based multi-objective optimization algorithms, which one can achieve best performance for MOFES?

RQ2: Compared with state-of-the-art wrapper-based or filter-based feature selection methods, does MOFES have advantage in selecting fewer features while achieving better performance?

RQ3: Which software features are frequently chosen by MOFES?

RQ4: Does the computational cost of MOFES lower than that of commonly-used feature selection methods?

In this paper, we first investigate 6 classical PMAs and compare the performance of these PMAs in terms of hypervolume quality indicator. After finding the PMA, which can achieve the best performance, we then want to answer the following three RQs. In these RQs, we will compare MOFES method with 22 state-of-the-art feature selection methods. Then, we want to analyze the frequently selected features by MOFES method and these findings can be used to provide guidelines for the collection of SDP datasets with higher quality. Finally, we want to analyze the computational cost of MOFES method.

Before answering these RQs, we first introduce the experiment setup, including experimental subjects, feature categories, baseline feature selection methods, performance measures and experimental design.

4.1. Experimental subjects

To verify the effectiveness of our proposed method, we choose experimental subjects from real open source projects (i.e., RELINK and PROMISE datasets). These two datasets have been widely used in previous studies (Menzies et al., 2007; Tantithamthavorn et al., 2017; Lessmann et al., 2008; Ghotra et al., 2015; Wang et al., 2016b; Li et al., 2017; Nam and Kim, 2015; Liu et al., 2016; Xu et al., 2016a, 2016b; Liu et al., 2014b, 2015; Song et al., 2011) and can be downloaded from PROMISE repository[2].

Table 1
Characteristics of PROMISE Datasets.

Name	Granularity	# Modules	# Defective Modules
Ant-1.7	Class	745	166
Camel-1.6	Class	965	188
Ivy-2.0	Class	352	40
Jedit-4.0	Class	306	75
Lucene-2.4	Class	340	203
Poi-3.0	Class	442	281
Synapse-1.2	Class	256	86
Velocity-1.6	Class	229	78
Xalan-2.6	Class	885	411
Xerces-1.4	Class	588	437

Table 2
Characteristics of RELINK Datasets.

Name	Granularity	# Modules	# Defective Modules
Apache	File	194	98
Safe	File	56	22
ZXing	File	399	118

PROMISE datasets are provided by Jureczko and Madeyski (2010). These datasets were gathered from 10 different open source projects (such as ant, lucene, poi). The granularity of the program modules is set as class and they considered 20 metrics, such as CK metrics proposed by Chidamber and Kemerer (1994). These metrics take into account the encapsulation, inheritance and polymorphism of object-oriented programs. The characteristics of PROMISE datasets (such as project name, granularity, number of modules, and number of defective modules) are shown in Table 1.

RELINK datasets are gathered by Wu et al. (2011). They analyzed 3 open source projects (i.e., Apache HTTP Server, Safe and ZXing). These datasets considered 26 metrics, which are based on code complexity and can be measured by Understand tool. The characteristics of RELINK (such as project name, granularity, number of modules, and number of defective modules) are shown in Table 2.

4.2. Feature categories

The features (i.e., metrics) used by RELINK and PROMISE can be grouped into different categories. Table 3 shows the details of these feature categories. In RELINK (Wu et al., 2011), the considered features are grouped into two categories according to the definitions by Understand[3]. There are 12 complexity metrics (CPM) and 15 count metrics (CTM). The category to which the metric belongs is shown in the fourth column. Notice that the sum (=27) of the number of complexity metrics (=12) and the number of count metrics (=15) is greater than the total number of features (=26). The reason is that the feature **RatioCommentToCode** in ReLink belongs to two categories simultaneously. In PROMISE (Jureczko and Madeyski, 2010), the considered 20 features are also grouped into two categories: object-oriented metric (OOM) and complexity metric (CPM). In particular, OOM category can be further divided into 5 metric suites: 7 metrics suggested by Chidamber and Kemerer (1994), 2 metrics suggested by Henderson-Sellers (1995), 6 metrics suggested by Bansiya and Davis (2002), 4 metrics which are the quality oriented extension to Chidamber & Kemerer metric suite and 3 metrics suggested by Martin (1994). For CPM category, it only has McCabe's metric suite, which contains 3 metrics. Notice

[2] http://openscience.us/repo.

[3] https://scitools.com/support/metrics_list.

Table 3
Feature Categories for RELINK and PROMISE.

Dataset	Category	Description	Feature Name	# Feature
RELINK	Complexity Metric (CPM)	Complexity measures of the source code, e.g., McCabe Cyclomatic measure	AvgCyclomatic, AvgCyclomaticModified, AvgCyclomaticStrict, AvgEssential, MaxCyclomatic, MaxCyclomaticModified, MaxCyclomaticStrict, RatioCommentToCode, SumCyclomatic, SumCyclomaticModified, SumCyclomaticStrict, SumEssential	12
	Count Metric (CTM)	Quantitative counting of the source code,e.g., the number of all lines	AvgLine, AvgLineBlank, AvgLineCode, AvgLineComment, CountLine, CountLineBlank, CountLineCode, CountLineCodeDecl, CountLineCodeExe, CountLineComment, CountSemicolon, CountStmt, CountStmtDecl, CountStmtExe, RatioCommentToCode	15
PROMISE	Object Oriented Metric (OOM)	Chidamber and Kemerer (CK)	WMC, DIT, NOC, CBO, RFC, LCOM, LOC	7
		Henderson-Sellers (HS)	LCOM3, LOC	2
		Bansiy and Davis (HD)	NPM, DAM, MOA, MFA, CAM, LOC	6
		An extension to Chidamber and Kemerer metrics (ECK)	IC, CBM, AMC, LOC	4
		Martin (Martin)	CA, CE, LOC	3
	Complexity Metric (CPM)	Complexity measures of the source code, e.g., McCabe Cyclomatic measure	MAX(CC), Avg(CC), LOC	3

that the feature **LOC** in PROMISE belongs to all these metric suites simultaneously.

4.3. Baseline feature selection methods

In this paper, we consider 22 state-of-the-art feature selection methods (Ghotra et al., 2017; Xu et al., 2016a; Liu et al., 2014b) as baselines. These methods can be classified into four categories: filter based ranking methods, filter based subset methods, wrapper based subset selection methods and no feature selection method.

4.3.1. Filter based ranking methods

Filter based ranking methods, which use some measures to assign a score to each feature and present the users with a ranked list of features. This kind of methods can be further classified into 5 subcategories: statistic based methods, probability based methods, instance based methods, classifier based methods and clustering based methods.

For statistic based methods, we consider ChiSquared statistic (Liu and Setiono, 2012) to evaluate the importance of the feature.

For probability based methods, gain ratio (Quinlan, 1993) can mitigate the bias of information gain by penalizing the features with more values. Information gain (Cover and Thomas, 2012) is an entropy-based method. It measures the reduction in uncertainty of a class label after observing a feature. However, information gain is biased toward features with more values. Symmetrical uncertainty (Kannan and Ramaraj, 2010) evaluates the value of a set of features by measuring their symmetrical uncertainty with respect to another set of features.

For instance based methods, ReliefF (Kononenko, 1994) is an instance based ranking method. An instance from the dataset is randomly sampled and its nearest neighbour is located from the same or the opposite class. The relevance score of each feature is updated by comparing the values of the nearest neighbor features to the sampled instance.

For classifier based methods, one rule (Holte, 1993) generates a one-level decision rule for each individual feature and features are ranked according to their classification error rate. Support vector machine assigns a weight to each feature and uses the square of the assigned weight to rank the features.

For clustering based methods, FECAR is a representative method. FECAR (Liu et al., 2014b) firstly partitions original features into k clusters based on FF-Correlation measure (i.e., the correlation between any pair of features), then it selects relevant features from each cluster based on FC-Relevance measure (i.e., the relevance between a feature and the target class). In this paper, we only choose Symmetric Uncertainty as the FF-Correlation measure, and choose Information Gain, ChiSquare, ReliefF, Symmetric Uncertainty as the FC-Relevance measure respectively.

For these methods, the subset of features from the top-k of the ranking list is selected. Here, k is set to select $\lceil log_2 m \rceil$ features from the original feature set, which is recommended by Gao et al. (2011). Here m is the number of original features.

4.3.2. Filter based subset methods

Filter based subset selection methods use statistical measures on feature subsets to find the best one. A correlation-based feature subset method and a consistency-based feature subset method are considered in this paper.

Correlation-based feature subset selection (CfsSubset) (Hall, 2000) aims to identify a feature subset, in which these features have a high correlation with respect to the class label while having a low correlation within each other.

Consistency-based feature subset selection (ConsistencySubset) (Dash et al., 2000) uses an indicator (i.e., consistency) to mea-

sure the metric of a feature subset. This method aims to search for the minimal subset, whose consistency is equal to that of all the features.

4.3.3. Wrapper based methods

Wrapper based subset selection methods construct classification models using different subsets to find the ones which can achieve the best performance.

In this category, four classical classifiers (e.g., J48, KNN, LR and NB) are considered to validate the influence of different classifier on wrapper based methods. We also consider two types of search strategies in these methods: GFS (Greedy Forward Selection) and GBS (Greedy Backward Selection). In particular, GFS starts from an empty set (i.e., no features), then iteratively adds an optimal feature by using hill climbing until the performance of the model can not be further improved. On the contrary, GBS starts from all the features, then it iteratively removes a feature by using hill climbing until the performance of the model can not be further improved. Both GFS and GBS use greedy search strategies. Song et al. (2011) considered these two strategies in their proposed general defect prediction framework. However, these two search strategies exist nesting effect.

4.3.4. No feature selection method

This method does not consider feature selection. It means that we use all of original features to construct models and then evaluate the performance. We use **FULL** to denote this method.

To facilitate subsequent description for these baseline methods, we give abbreviation for each method. Table 4 provides an overview of all these baseline methods.

4.4. Performance evaluation measures

Previous studies argued that threshold-dependent performance measures (such as precision, recall, and F1) are problematic because they: (1) depend on an arbitrarily-selected threshold and (2) are sensitive to class imbalanced datasets. Instead, we use a more reasonable measure AUC (Area Under the receiver operator characteristic Curve) to measure the prediction performance of SDP models suggested by a recent study (Tantithamthavorn et al., 2016). Using ROC curve can consider different thresholds. In particular, horizontal axis represents the value of TPR (true positive rate), while vertical axis represents the value of FPR (false positive rate). According to different thresholds, the model has corresponding TPR value and FPR value and these two values can correspond to a point. All these points are then connected as a ROC curve. The value of AUC is the area under the ROC curve. The value of AUC is more close to 1, the better the performance of the constructed model. AUC performance measure is widely used in previous SDP studies (Xu et al., 2016a, 2016b; Liu et al., 2016, 2015, 2014b).

Instead of AUC performance measure, we also use quality indicators to evaluate the quality of Pareto fronts generated by different PMAs. These quality indicators can be classified into 4 categories: convergence, diversity, combination of convergence and diversity, and coverage (Wang et al., 2016a). In this paper, we select hypervolume (HV) (Zitzler and Thiele, 1999) because of its popularity in evaluation PMAs. HV can measure the volume in the objective space that is covered by a Pareto front. Therefore, it can measure both the convergence and the diversity of a Pareto front. A higher value of HV means a better quality of the Pareto front. More details of HV quality indicator can be found in Zitzler and Thiele (1999) and we use the implementation of HV quality indicator provided by JMetal packages (Durillo and Nebro, 2011a) to avoid the threats to internal validity.

4.5. Experimental design

We use 4 classical classifiers (i.e., J48, KNN, LR and NB) (Radjenovic et al., 2013; Kamei and Shihab, 2016; Hall et al., 2012; Ghotra et al., 2017; Khoshgoftaar et al., 2012; Rahman and Devanbu, 2013) to construct the defect prediction models after preprocessing the training set by using the selected features.

J48 is a decision tree based classifier. Decision trees use feature values for the classification of instances. A feature in an instance that has to be classified is represented by each node of the decision tree, while the assumption values taken by each node is represented by each branch. The classification of instances is performed by following a path through the tree from root node to leaf nodes by checking feature values against rules at each node. The root node is the node, which can best divide the whole training data.

KNN is a nearest neighbor technique. Nearest neighbour techniques are another category of statistical techniques. Nearest neighbour techniques take more time in the testing phase, while taking less time than techniques (such as decision trees, neural networks and Bayesian networks) in the training phase.

LR is a function based classifier and is short for Logistic Regression. LR is a linear regression after the normalization of the logistic function. It studies the fitting parameters from the training set, fits the target values into the range of [0,1], and then discretizes the target values for classification. The binary logistic model is often used in the context of SDP, which estimates the probability of a binary response based on one or more features.

NB is a simple probabilistic classifier, which assumes the features are statistically independent of each other. It can provide good enough classification results, even though some of the features are inter-related.

In our experiments, we use the implementation of these classifiers provided by Weka packages to avoid the internal threats to validity and use default value for hyperparameters of these classifiers.

To evaluate the model performance of different feature selection methods (Tantithamthavorn et al., 2017), we first use stratified sampling to generate training set *trainset* and testing set *testset* respectively. In particular, we randomly select 70% instances as the training set and the remaining 30% instances as the testing set. Then we use a specific feature selection method *fsm* on the *trainset* and *fsm* returns a selected feature subset f_{subset}. Latter, we use the chosen f_{subset} to preprocess *trainset* and *testset* simultaneously and then generate new training set *trainset*′ and new testing set *testset*′. Finally, we use a specific classifier to construct a SDP model on *trainset*′ and use *testset*′ to evaluate the performance of the constructed model. The model performance evaluation process is shown in Fig. 3. To reduce the randomness in splitting the dataset into the training set and the testing set, we perform this split process 10 times independently. Notice that all the features selection methods are only applied to *trainset*, since the label of each module in *testset* would not be available (Song et al., 2011). Some previous studies mistakenly applied feature selection methods to the whole data set and Menzies et al. thought this is one of the reasons for modeling violation issue (Menzies et al., 2007). Moreover, Jiarpakdee et al. (2018) and Tantithamthavorn et al. (2018) advocated that feature selection methods should be applied only on the training set to avoid producing optimistically biased performance estimation and interpretation.

Baseline feature selection methods only return a feature subset based on *trainset*, while our proposed method MOFES will return a Pareto optimal solution based on *trainset*. To generate the Pareto optimal solution, each chromosome in the population represents a chosen feature subset, its fitness value is computed based on *trainset* using 3-fold cross validation. In particular, for 3-fold cross val-

Table 4
Overview of State-of-the-art Feature Selection Baseline Methods.

Category	Subcategory	Method Name	Method Description	Abbreviation
Filter based Ranking Methods	Statistics based method	ChiSquared_Ranker	Evaluate the worth of a feature by computing the value of the chi-squared statistic with respect to the class	FR1
	Probability based methods	GainRatio_Ranker	Evaluate the worth of a feature by measuring the gain ratio with respect to the class	FR2
		InfoGain_Ranker	Evaluate the worth of a feature by measuring the information gain with respect to the class	FR3
		SymmetricalUncert_Ranker	Evaluate the worth of a feature by measuring the symmetrical uncertainty with respect to the class	FR4
	Instance based method	ReliefF_Ranker	Evaluate the worth of a feature by repeatedly sampling an instance and considering the value of the given feature for the nearest instance of the same and different class	FR5
	Classifier based methods	OneR_Ranker	Evaluate the worth of a feature by using the OneR classifier	FR6
		SVM_Ranker	Evaluate the worth of a feature by using an SVM classifier	FR7
	Clustering based methods (FECAR)	Cluster_ChiSquare_FeatureSelection	Use SU as FF-Correlation and ChiSquare as FC-Correlation in FECAR	FR8
		Cluster_IG_FeatureSelection	Use SU as FF-Correlation and Information Gain as FC-Correlation in FECAR	FR9
		Cluster_ReliefF_FeatureSelection	Use SU as FF-Correlation and ReliefF as FC-Correlation in FECAR	FR10
		Cluster_SU_FeatureSelection	Use SU as FF-Correlation and FC-Correlationin FECAR	FR11
Filter based Subset Methods	Correlation based feature subset selection	CfsSubset_GreegyStepwise	Evaluate the worth of a subset of features by considering the individual predictive ability of each feature along with the degree of redundancy between them	FS1
	Consistency based feature subset selection	ConsistencySubset_GreegyStepwise	Evaluate the worth of a subset of features by the level of consistency in the class values when the training instances are projected onto the subset of features	FS2
Wrapper based Subset Selection Methods	J48	WrapperSubset_GreegyStepwiseJ48	Evaluate feature subset by using J48 as the classifier and using the search forward strategy	WS1
		WrapperSubset_GreegyStepwiseJ48_Back	Evaluate feature subset by using J48 as the classifier and using the search backward strategy	WS2
	KNN	WrapperSubset_GreegyStepwiseKNN	Evaluate feature subset by using KNN as the classifier and using the search forward strategy	WS3
		WrapperSubset_GreegyStepwiseKNN_Back	Evaluate feature subset by using KNN as the classifier and using the search backward strategy	WS4
	LR	WrapperSubset_GreegyStepwiseLR	Evaluate feature subset by using LR as the classifier and using the search forward strategy	WS5
		WrapperSubset_GreegyStepwiseLR_Back	Evaluate feature subset by using LR as the classifier and using the search backward strategy	WS6
	NB	WrapperSubset_GreegyStepwiseNB	Evaluate feature subset by using NB as the classifier and using the search forward strategy	WS7
		WrapperSubset_GreegyStepwiseNB_Back	Evaluate feature subset by using NB as the classifier and using the search backward strategy	WS8
No Feature Selection Method	None	Full	Use all the original features to construct models	FULL

idation, we randomly split *trainset* into 3 folds by using stratified sampling. We use the 2/3 instances to build the model by a specific classifier, and then use the remaining 1/3 instances to evaluate the goodness of the selected feature subset. This process is repeated 3 times (i.e., each instance in *trainset* has been used to predict merely once). Finally, we use the average AUC value of these 3 runs as its fitness value. The main process is shown in Fig. 4.

Based on suggestions by previous studies on applying PMAs for numerical problem (Coello et al., 2007), parameter name and their value of PMAs used by MOFES are shown in Table 5. In this table, n represents the number of features.

5. Result analysis

5.1. Result analysis for RQ1

To investigate the influence of different PMAs on MOFES method, we take 6 state-of-the-art PMAs (i.e., NSGA-II, MOCell,

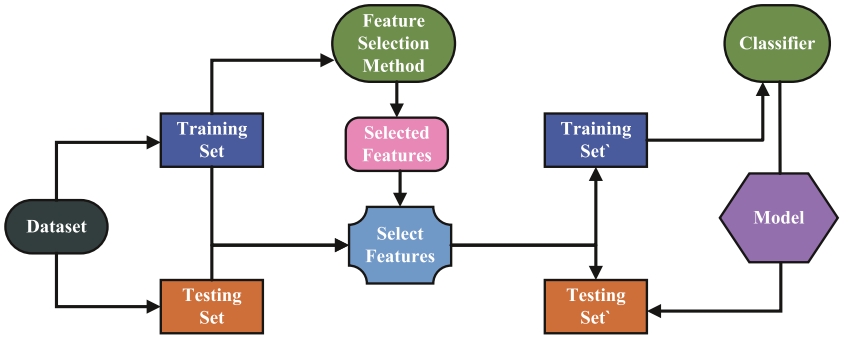

Fig. 3. The Process of Model Performance Evaluation.

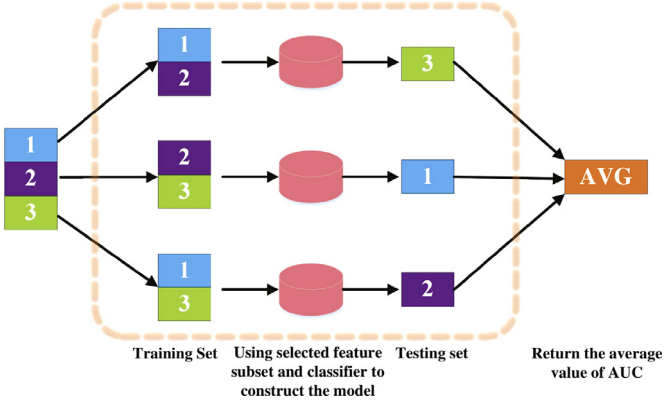

Fig. 4. The Fitness Value Computing Process for Each Chromosome.

Table 5
Parameter Setting of PMAs used by MOFES.

Parameter	Value
Population Size	100
Maximum Evaluation Number	3000
Selection Operator	Binary Tournament
Crossover Operation	Simulated Binary
Crossover Rate	0.9
Mutation Operator	Polynomial
Mutation Rate	$1/n$
Archive Size	100

SPEA2, PAES, SMSEMOA and RandomSearch) into consideration and want to find which one can achieve best performance for MOFES.

To reduce the randomness in splitting the dataset into the training set and the testing set, we perform this split process 10 times independently. The distribution of performance in terms of HV on RELINK by using different classifiers can be found in Figs. 5–8 respectively. The subfigures in each figure represent the results on different projects of RELINK and use boxplot to show the performance distribution. For each subfigure, x-axis represents different PMAs and y-axis represents the performance in terms of HV quality indicator. From Fig. 5, when using J48 as the classifier, NSGA-II is better than other PMAs based on median value. Moreover, NSGA-II can also achieve a stable performance on different projects. From Fig. 6, when using KNN as the classifier, NSGA-II is better than other PMAs based on median value in ZXing. While all PMAs except for RandomSearch have similar results in Apache and Safe. From Fig. 7, when using LR as the classifier, all PMAs except for RandomSearch have similar results in all projects. From Fig. 8, when using NB as the classifier, NSGA-II is better than other PMAs based on median value in Apache and Safe. However, all PMAs have poor performance in ZXing. In summary, we can find that when using different classifiers, all PMAs except for RandomSearc have almost the same and relative good performance on RELINK in terms of HV. The similar conclusions can also be drawn on PROMISE.

Table 6
Average Rank for Different PMAs on RELINK and PROMISE.

PMA	Rank on RELINK	Rank on PROMISE
NSGA-II	5.167	5.063
MOCell	4.917	4.313
SPEA2	3.958	4.325
SMSEMOA	3.292	2.425
PAES	2.458	3.875
RandomSearch	1.208	1.000

To further compare the performances of different PMAs, we firstly compare these PMAs based on the mean and standard deviation. Based on results on RELINK and PROMISE, we find that NSGA-II can achieve the best performance in most cases followed by MOCell. Then we compare these PMAs based on the median and IQR (interquartile range). Based on results on RELINK and PROMISE, we can find the same conclusions. Due to the length limitation of this paper, the detailed comparison results can be found in Appendix A and Appendix B.

Finally, we use Friedman test to analyze the ranks for different PMAs. The Friedman test assumes that the higher the rank, the better the performance in terms of the HV indicator. The results on RELINK and PROMISE datasets are shown in Table 6. In this table, the Friedman test considers reduction performance, which is distributed according to chi-square with 5 degrees of freedom. According to the results of Friedman test on two datasets, NSGA-II is the best since it has the highest rank score, while RandomSearch is the worst. Meanwhile, the top-3 PMAs on two datasets have the same ranking order, which indicates the stability of these three PMAs.

Summary for RQ1: In most cases, there is not much difference among NSGA-II, MOCell, SPEA2, PAES and SMSEMOA. However, RandomSearch always performs the worst on all datasets. Meanwhile, NSGA-II has an advantage over other PMAs when analyzing the mean value and median value in terms of HV indicator. It also has a highest rank score according to the results of the Friedman test. Therefore, we choose NSGA-II as the best PMA for implementing our method MOFES.

5.2. Result analysis for RQ2

In this RQ, we want to compared MOFES with 22 state-of-the-art wrapper-based and filter-based feature selection meth-

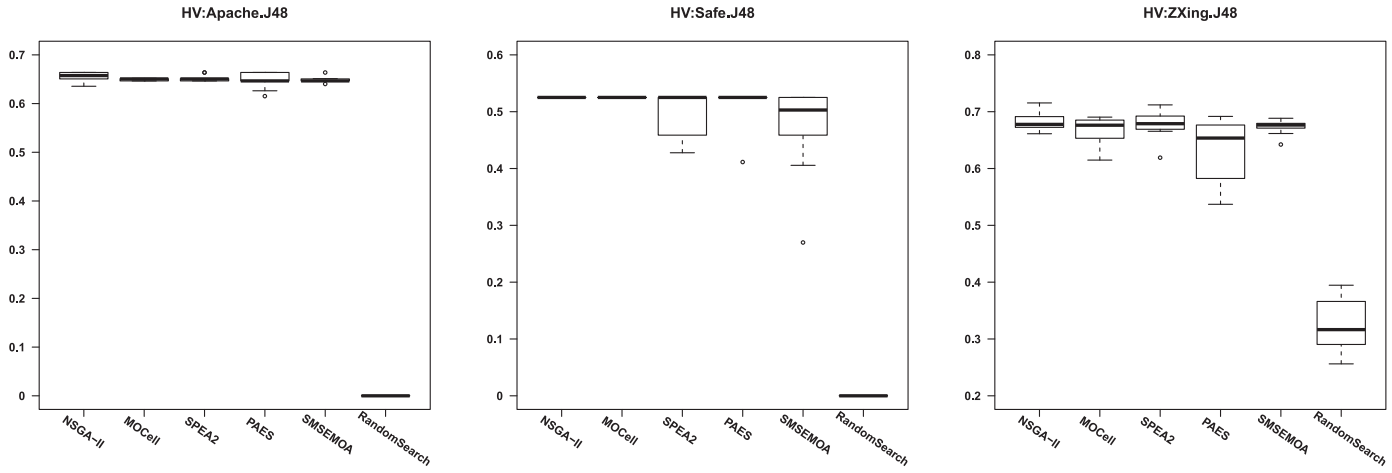

Fig. 5. The Distribution of Performance in terms of HV for Different PMAs on Relink by using J48 as the Classifier.

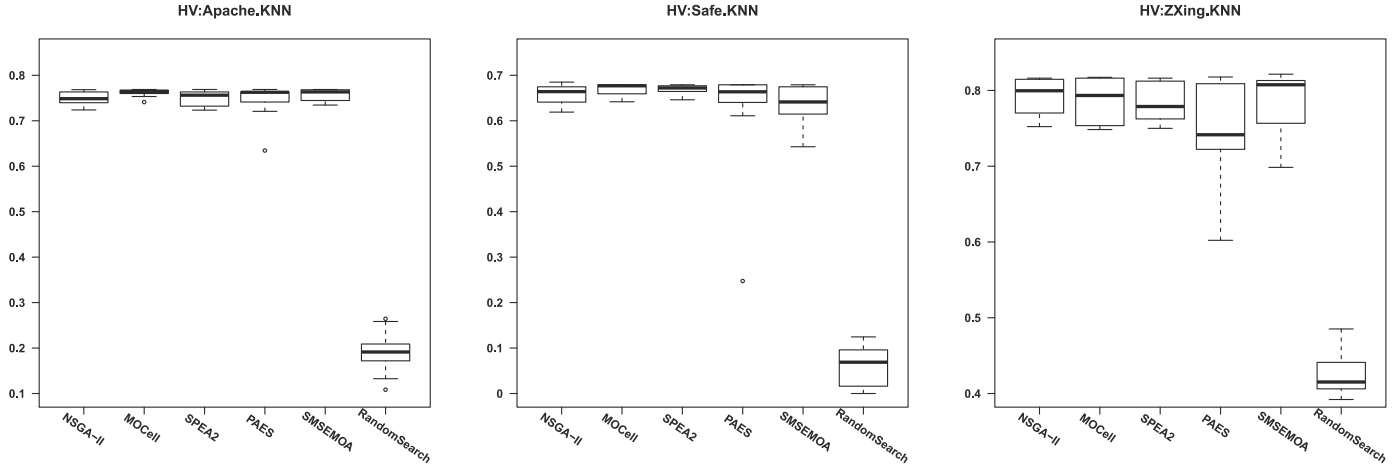

Fig. 6. The Distribution of Performance in terms of HV for Different PMAs on Relink by using KNN as the Classifier.

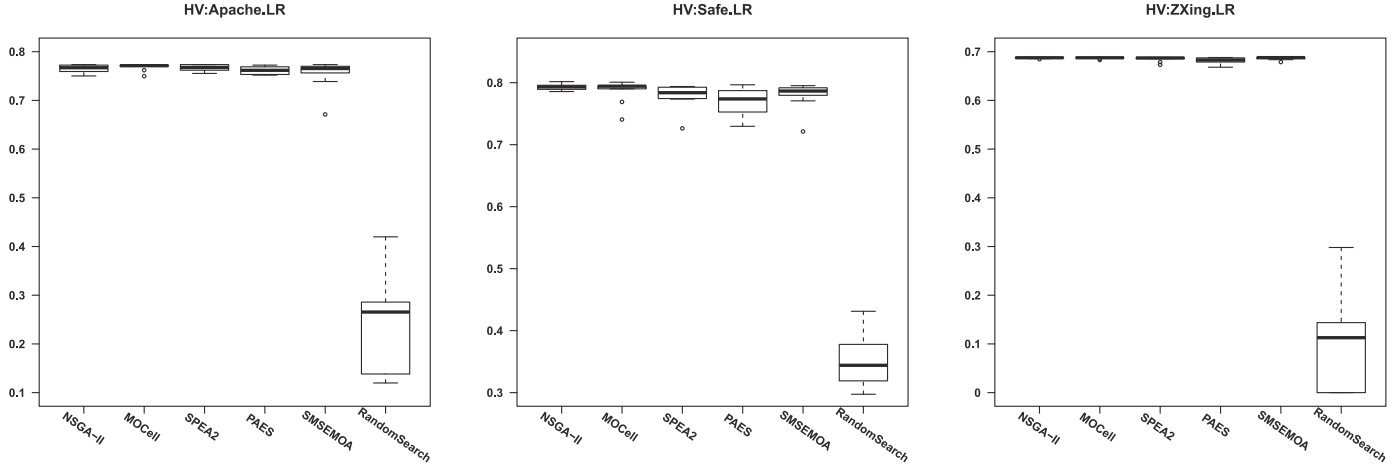

Fig. 7. The Distribution of Performance in terms of HV for Different PMAs on Relink by using LR as the Classifier.

ods (Ghotra et al., 2017; Xu et al., 2016a). Based on the result analysis for RQ1, we implement MOFES by using NSGA-II.

For a specific split of a dataset into the training set and the testing set, as there exists randomness in MOFES and some baseline methods, we independently run these methods 10 times. Therefore, MOFES will return 10 Pareto optimal sets in total on the training set. In our experiment, we first combine solutions in 10 Pareto optimal sets into one set T for further analysis. Then, we analyze the results of MOFES in two ways. In the first way, we divide all solutions in T into different groups according to the number of selected features. For every group, each solution in this group selects the same number of features. Notice that these solutions with the same number of selected features may select different features. Then, the average performance on the testing set in terms of AUC

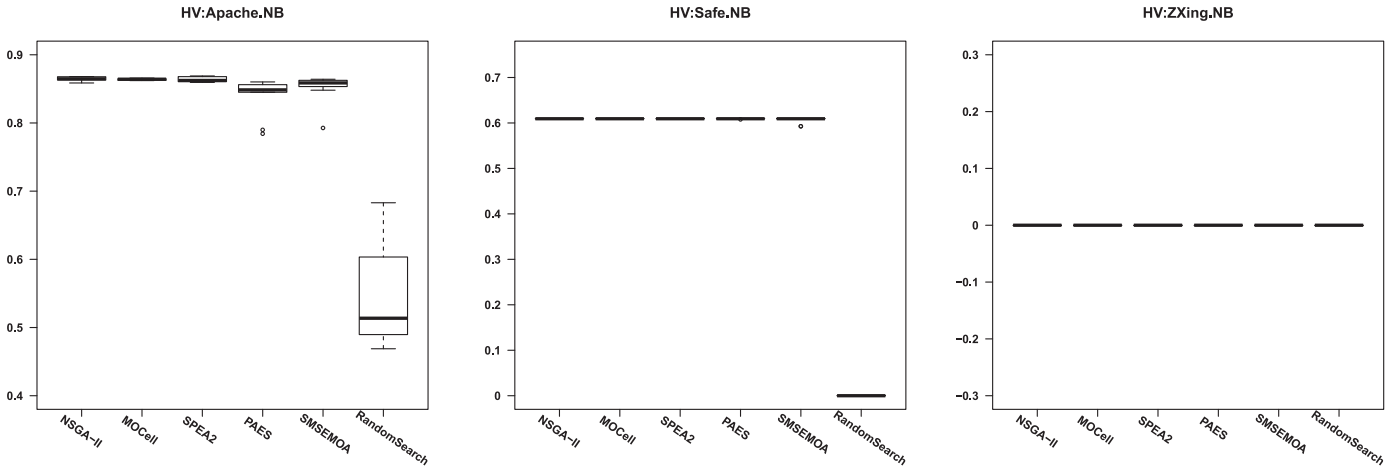

Fig. 8. The Distribution of Performance in terms of HV for Different PMAs on Relink by using NB as the Classifier.

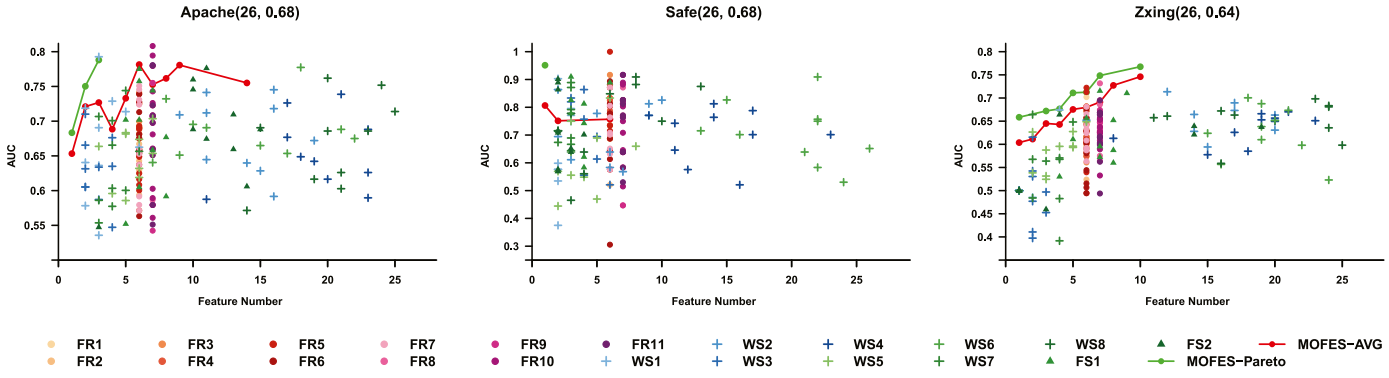

Fig. 9. Performance Comparison of Different Feature Selection Methods on RELINK when Using KNN as the Classifier.

will be computed for each group. We use line **MOFES-AVG** to represent this analysis way. In the second way, we choose the pareto front from the T based on the performance of these solutions on the testing set. We use line **MOFES-Pareto** to represent this analysis way.

When using KNN as the classifier, the comparison results on RELINK and PROMISE are shown in Figs. 9 and 10. In each subfigure, x-axis represents the number of selected feature by MOFES and all baseline methods, while y-axis represents AUC performance generated by different methods. At the top of each subfigure, there exists a text, which contains project name and a pair of two values enclosed in a parentheses. These two values represent the number of original features and average performance value in terms of AUC respectively when using no feature selection method (i.e., the baseline method FULL). For example, Apache (26, 0.68) indicates that Apache dataset has 26 original features, and the prediction model constructed by KNN when using all these features can achieve 0.68 on average in terms of AUC. Notice all the baseline methods will be executed 10 times independently. They may obtain the same feature subset in different runs. Therefore, for these method, they may have less than 10 points in some subfigures.

In Figs. 9 and 10, compared with FULL, all feature selection baseline methods can find some solutions, which are better than FULL among 3 projects in RELINK and 10 projects in PROMISE. It verifies that feature selection is useful in the context of SDP. Taking Apache project as an example, FULL (i.e., using all the original features) can only obtain 0.68. However, FR1 and FS1 can obtain the best value of 0.74 and 0.76 respectively by only using 6 features. WS1 can obtain the best value of 0.79 by only using 3 features. For our proposed method MOFES, it can also obtain the best value of 0.79 by only using 3 features. Meanwhile, almost in all projects, MOFES can select the fewest features while obtaining better performance.

We also analyze the number of features selected by all baseline methods. For some baseline methods in filter based ranking method category (i.e., FR1 to FR7), they select a specific number of features (i.e., 6 for RELINK and 7 for PROMISE), which is suggested by Gao et al. (2011). For FECAR (i.e., FR8 to FR11), they select a slightly different number of features and the reason is that features are selected by FECAR from each cluster proportionally on the cluster size and the ceil operator (i.e., $\lceil\rceil$) is used. For example, we assume the project has N features and we want to selected N_t features (N_t is set as 6 for RELINK and set as 7 for PROMISE) from the original features by using FECAR. Then for a cluster C, which has $|C|$ features in this cluster, FECAR will choose top $\lceil \frac{|C| \times N_t}{N} \rceil$ features from this cluster. For baseline methods in filter based subset method and wrapper based subset selection method categories (i.e., FS1, FS2 and WS1 to WS8), they select different number of features in different runs and the number of selected features distributes in a large value range. For MOFES, when selecting features, it has more flexibility since it can generate different Pareto optimal sets in different executions. In most cases, MOFES can find solutions, which can obtain better performance by selecting fewer features.

To further show the competitiveness of MOFES, we conduct Win/Tie/Loss analysis between our method and each baseline method. For example, supposing MOFES method returns Pareto optimal set {A, B, C, D} and a baseline method returns a solution E. If and only if there exist at least one solution in set {A, B, C, D}, which is Pareto dominance on E and the rest of the set are not

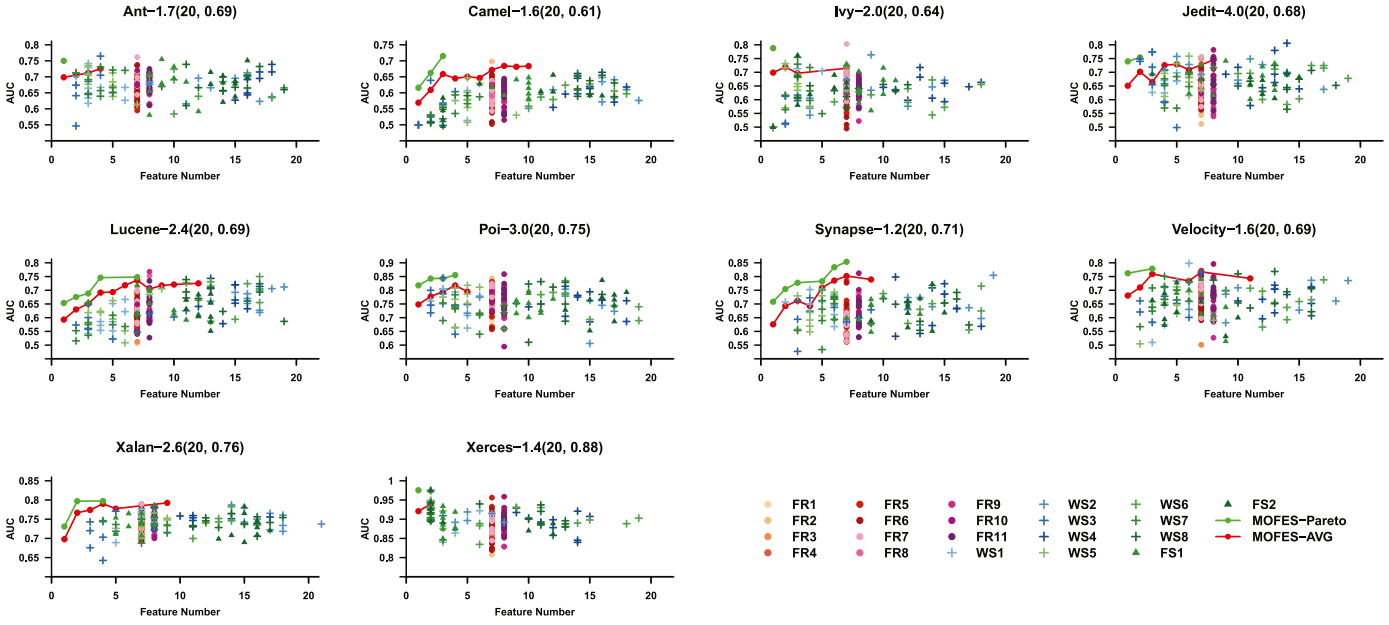

Fig. 10. Performance Comparison of Different Feature Selection Methods on PROMISE when Using KNN as the Classifier.

Pareto dominance by E, we mark MOFES method as a 'Win'. If and only if there exist at least one solution in set $\{A, B, C, D\}$, which is Pareto dominance by E and the rest of the set are not Pareto dominance by E, we mark MOFES as a 'Loss'. Otherwise, we mark MOFES method as a 'Tie'.

The overall Win/Tie/Loss evaluation results when comparing MOFES to all baselines for 4 classifiers can be found in Table 7. The first two columns represent the classifiers used in MOFES and the datasets used in experiments. The remaining columns represent 22 baseline methods. Each row refers to the result when comparing MOFES to other baseline methods given a specific classifier and a specific dataset. Since there are 10 independent runs for each method, it is obvious that the sum of Win/Tie/Loss are 100 for PROMISE (since PROMISE has 10 projects) and 30 for RELINK (since RELINK has 3 projects). For a specific classifier, we also give out the summarization of all the Win/Tie/Loss of MOFES on both datasets when compared to a specific baseline method.

In Table 7, it is not hard to find that MOFES can outperform all baseline methods on almost all the datasets when different classifiers are used. Due to the length limitation of this paper, the details of comparison results in terms of Win/Tie/Loss when using different classifiers can be found in Appendix C.

To conduct statistical analysis, we choose the solution with best performance in terms of AUC in the Pareto optimal set generated by MOFES for each run, since baseline feature selection methods only return one solution for each run. Then we use Wilcoxon signed-rank test to examine whether the performance difference between MOFES and a specific baseline feature selection method is statistically significant. Moreover, we use the Benjamini-Hochberg (BH) procedure to adjust p-values if we perform multiple comparisons. If the test shows a significant difference, we further compute Cliff's δ, which is a measure for non-parametric effect size, to examine whether the magnitude of the difference is substantial or not. The meaning of effectiveness level of Cliff's δ and its value range can be found in Table 8. In summary, MOFES performs significantly better or worse than the specific baseline feature selection method, if BH corrected p-value is less than 0.05 and the effectiveness level is not negligible based on Cliff;;s δ. While the difference between these two methods is not significant, if (1) p-value is not less than 0.05 or (2) p-value is less than 0.05 and the effectiveness level is negligible.

The overall statistical analysis results can be found in Table 9. It is not hard to find that MOFES can perform significantly better than all baseline methods in terms of p-value and Cliff's δ whether on PROMISE or on RELINK when different classifiers are used.

To comprehensively analyze the performance of MOFES, we also use F1 performance measure to make a comparison between MOFES and FULL. There are four possible outputs for an module classified in a target project: a module can be classified as defective when it is truly defective (true positive, TP); it can be classified as defective when it is actually non-defective (false positive, FP); it can be classified as non-defective when it is actually defective (false negative, FN); or it can be classified as non-defective and it is truly non-defective (true negative, TN). Based on above four possibilities, $F1$ can be computed as follow:

$$F1 = \frac{2 \times P \times R}{P + R} \tag{1}$$

where P $(= TP/(TP + FP))$ is the proportion of modules that are correctly predicted as defective among those predicted as defective; R $(= TP/(TP + FN))$ is the proportion of defective modules that are correctly predicted. $F1$ is the harmonic average of the P and R and this performance measure has also been widely used in previous SDP studies (Xia et al., 2016; Kim et al., 2008; Nam et al., 2013). The comparison results on RELINK are summarized in Table 10. In this table, the first column shows the project name and the remaining columns shows the comparison results for different classifiers. We analyze the results of MOFES in the first way (i.e., MOFES-AVG). In this way, we divide all solutions into different groups according to the number of selected features. For each group, the average performance in terms of AUC will be computed for each group. Moreover, some cells with light blue background indicate the average performance of MOFES is better than FULL when a specific classifier is used. From Table 10, we can find that MOFES wins FULL in almost all the datasets. Therefore, MOFES can not only effectively decrease the number of selected features but also increase the performance of later constructed models. For example, MOFES improves FULL by 31.1% when using LR as the clas-

Table 7
The Overall Summarization on Win/Tie/Loss Evaluation Results for Different Classifiers.

Classifier	Projects	Baselines																				
		FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	None
J48	PROMISE	93/7/0	93/7/0	95/5/0	95/5/0	92/7/1	93/7/0	89/9/2	98/2/0	98/2/0	99/1/0	100/0/0	97/3/0	96/4/0	94/6/0	99/1/0	94/6/0	98/2/0	93/7/0	99/0/1	89/11/0	99/1/0	99/1/0
	RELINK	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	29/1/0	30/0/0	29/1/0	29/1/0	30/0/0	29/1/0	30/0/0	29/1/0	30/0/0	29/0/1	30/0/0	30/0/0	29/1/0	30/0/0	28/1/1	30/0/0	30/0/0
	Total	**123/7/0**	**123/7/0**	**125/5/0**	**125/5/0**	**122/7/1**	**122/8/0**	**119/9/2**	**127/3/0**	**127/3/0**	**129/1/0**	**129/1/0**	**127/3/0**	**125/5/0**	**124/6/0**	**128/1/1**	**124/6/0**	**128/2/0**	**122/8/0**	**129/0/1**	**117/12/1**	**129/1/0**	**129/1/0**
KNN	PROMISE	100/0/0	100/0/0	100/0/0	100/0/0	99/1/0	100/0/0	97/3/0	99/1/0	99/1/0	96/4/0	100/0/0	99/1/0	99/1/0	93/7/0	98/2/0	92/8/0	98/2/0	94/6/0	98/2/0	96/4/0	100/0/0	98/2/0
	RELINK	28/2/0	28/2/0	27/3/0	27/3/0	27/3/0	29/1/0	28/2/0	27/3/0	27/3/0	28/2/0	26/4/0	28/2/0	29/1/0	27/3/0	30/0/0	29/1/0	30/0/0	30/0/0	30/0/0	29/1/0	30/0/0	30/0/0
	Total	**128/2/0**	**128/2/0**	**127/3/0**	**127/3/0**	**126/4/0**	**129/1/0**	**125/5/0**	**126/4/0**	**126/4/0**	**124/6/0**	**126/4/0**	**127/3/0**	**128/2/0**	**120/10/0**	**128/2/0**	**121/9/0**	**128/2/0**	**124/6/0**	**128/2/0**	**125/5/0**	**130/0/0**	**128/2/0**
LR	PROMISE	93/7/0	93/7/0	92/7/1	95/5/0	96/4/0	95/5/0	92/8/0	98/2/0	97/3/0	99/1/0	97/3/0	95/5/0	94/6/0	94/6/0	97/3/0	98/2/0	96/4/0	90/10/0	100/0/0	97/3/0	96/4/0	98/2/0
	RELINK	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	29/0/1	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0	30/0/0
	Total	**123/7/0**	**123/7/0**	**122/7/1**	**125/5/0**	**126/4/0**	**125/5/0**	**122/8/0**	**128/2/0**	**127/3/0**	**128/1/1**	**127/3/0**	**125/5/0**	**124/6/0**	**124/6/0**	**127/3/0**	**128/2/0**	**126/4/0**	**120/10/0**	**130/0/0**	**127/3/0**	**126/4/0**	**128/2/0**
NB	PROMISE	95/5/0	96/4/0	92/8/0	93/7/0	94/6/0	95/5/0	97/3/0	98/2/0	97/3/0	99/1/0	99/1/0	98/2/0	97/3/0	97/3/0	99/1/0	100/0/0	98/2/0	96/4/0	99/1/0	98/2/0	99/1/0	98/2/0
	RELINK	27/2/1	29/0/1	28/1/1	29/0/1	28/1/1	28/2/0	29/1/0	29/1/0	27/3/0	28/2/0	29/1/0	28/2/0	29/0/1	30/0/0	30/0/0	28/2/0	29/1/0	30/0/0	29/1/0	29/1/0	30/0/0	30/0/0
	Total	**122/7/1**	**125/4/1**	**120/9/1**	**122/7/1**	**122/7/1**	**123/7/0**	**126/4/0**	**127/3/0**	**124/6/0**	**127/3/0**	**128/2/0**	**126/4/0**	**126/3/1**	**127/3/0**	**129/1/0**	**128/2/0**	**127/3/0**	**126/4/0**	**128/2/0**	**127/3/0**	**129/1/0**	**128/2/0**

Table 8
Cliff's δ and Corresponding Effectiveness Level suggested by Benjamini and Hochberg (1995).

Cliff's δ	Effectiveness Level
$\lvert\delta\rvert < 0.147$	Negligible
$0.147 \leq \lvert\delta\rvert < 0.33$	Small
$0.33 \leq \lvert\delta\rvert < 0.474$	Medium
$0.474 \leq \lvert\delta\rvert$	Large

sifier on *Apache*. The similar conclusions can be also drawn on PROMISE too.

Summary for RQ2: MOFES can achieve better performance while selecting fewer features in most cases when compared to 22 state-of-the-art baseline feature selection methods.

5.3. Result analysis for RQ3

In this RQ, we want to analyze which features are frequently selected by MOFES, and these findings can provide guidelines for collecting SDP datasets with higher quality in the future. Tables 11 and 12 list the top-10 features frequently selected by MOFES on RELINK and PROMISE respectively. In these two tables, we divide all columns into four groups according to the used classifier. In each group, there has three columns, which denote feature name, category and selection proportion respectively.

For RELINK, the feature with the first rank are CountLineCodeExe, CountLineBlank, CountLineCodeExe and CountStmtDecl when the used classifiers are J48, KNN, LR and NB respectively. However, most of the top-1 features belong to the category of CTM (Count Metric), and the selection proportion of the feature with the first rank is almost two times as that of the feature with the second rank except for NB is used as the classifier. When analyzing top-5 features, most of these features are from the category of CTM except for LR is used as the classifier. However, when LR is used as the classifier, the selection proportion of the top-1 feature is 0.618 and this feature belongs to the category of CTM. In summary, features from the category of CTM are frequently selected for RELINK and these features help to construct high-quality models.

For AEEEM, most of the top-1 selected features are from the category of OOM (object oriented metrics) and the selection proportion is about 0.3. Moreover, features from the CK subcategory, Martin subcategory or ECK subcategory are frequently selected in top-10 features (i.e., 6/10, 7/10, 6/10, or 7/10 when J48, KNN, LR or NB is used as the classifier respectively). When analyze top-3 features, features from the CK subcategory or Martin subcategory are more frequently selected, which indicates these features from these two subcategories may more suitable for constructing high-quality models.

Summary for RQ3: Features in different feature categories may obtain different performances in the context of SDP. The proposed method MOFES can make good use of the features from different feature categories, which is help for constructing high-quality SDP models.

5.4. Result analysis for RQ4

In this RQ, we mainly analyze the computational cost of our method MOFES and make a comparison between MOFES and all

Table 9
Statistical Analysis Results for Different Classifiers in terms of p-value and Cliff's δ.

Category	Baseline	J48				KNN				LR				NB			
		PROMISE		RELINK		PROMISE		RELINK		PROMISE		RELINK		PROMISE		RELINK	
		p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ	p-value	δ
Filter based Ranking Methods	**FR1**	1.00E-04	0.625	1.14E-02	0.625	4.36E-05	0.725	5.60E-05	0.725	1.51E-05	0.798	1.60E-03	0.798	1.85E-03	0.575	3.50E-04	0.575
	FR2	1.07E-04	0.623	1.44E-02	0.623	2.67E-05	0.745	4.16E-05	0.745	2.35E-05	0.780	2.04E-03	0.780	1.07E-03	0.604	4.85E-04	0.604
	FR3	8.13E-05	0.633	1.57E-02	0.633	3.15E-05	0.738	3.10E-05	0.738	2.80E-05	0.773	2.92E-03	0.773	1.36E-03	0.591	4.85E-04	0.591
	FR4	7.33E-05	0.637	7.62E-03	0.637	2.56E-05	0.747	3.60E-05	0.747	1.72E-05	0.793	5.03E-04	0.793	1.32E-03	0.593	3.50E-04	0.593
	FR5	1.54E-04	0.608	1.71E-02	0.608	3.28E-05	0.737	3.09E-05	0.737	3.43E-06	0.856	1.07E-04	0.856	9.29E-04	0.611	6.67E-04	0.611
	FR6	1.22E-04	0.617	1.92E-03	0.617	4.36E-05	0.725	3.09E-05	0.725	6.94E-06	0.829	8.51E-04	0.829	8.66E-04	0.615	1.51E-04	0.615
	FR7	1.60E-04	0.607	3.98E-03	0.607	1.08E-04	0.687	1.24E-05	0.687	1.88E-05	0.789	1.15E-04	0.789	8.66E-04	0.615	1.79E-04	0.615
	FR8	1.31E-04	0.615	2.13E-02	0.615	4.02E-05	0.728	2.26E-04	0.728	2.06E-05	0.785	1.10E-03	0.785	3.32E-03	0.542	2.51E-04	0.542
	FR9	1.55E-04	0.608	9.07E-03	0.608	4.19E-05	0.727	2.95E-04	0.727	3.62E-05	0.762	2.59E-03	0.762	2.74E-03	0.553	4.13E-04	0.553
	FR10	2.79E-04	0.584	9.80E-03	0.584	1.65E-04	0.668	3.37E-04	0.668	7.27E-06	0.827	5.90E-05	0.827	1.27E-03	0.595	5.70E-04	0.595
	FR11	1.40E-04	0.612	1.33E-02	0.612	6.26E-05	0.710	3.84E-04	0.710	3.62E-05	0.762	4.61E-03	0.762	2.49E-03	0.558	2.51E-04	0.558
Filter based Subset Methods	**FS1**	8.71E-05	0.631	2.02E-03	0.631	4.02E-05	0.728	1.72E-04	0.728	1.97E-05	0.787	1.60E-03	0.787	2.74E-03	0.553	2.12E-04	0.553
	FS2	3.87E-05	0.661	1.28E-02	0.661	4.54E-05	0.723	1.90E-05	0.723	1.72E-05	0.793	4.05E-03	0.793	2.11E-03	0.567	1.03E-04	0.567
Wrapper based Subset Selection Methods	**WS1**	3.08E-05	0.669	2.19E-03	0.669	1.35E-04	0.677	4.17E-05	0.677	3.88E-06	0.851	6.73E-04	0.851	5.46E-05	0.744	7.67E-05	0.744
	WS2	6.37E-05	0.643	2.30E-03	0.643	1.26E-04	0.680	3.10E-05	0.680	9.58E-06	0.816	1.92E-04	0.816	1.51E-03	0.585	6.17E-05	0.585
	WS3	2.48E-05	0.677	1.93E-03	0.677	4.72E-05	0.722	5.58E-06	0.722	2.56E-06	0.867	7.48E-04	0.867	6.54E-05	0.736	2.12E-04	0.736
	WS4	6.15E-05	0.644	8.53E-04	0.644	1.13E-04	0.685	4.03E-06	0.685	7.98E-06	0.824	2.62E-06	0.824	1.85E-03	0.575	2.12E-04	0.575
	WS5	1.35E-04	0.613	8.16E-04	0.613	8.25E-05	0.698	1.21E-07	0.698	1.51E-05	0.798	1.78E-04	0.798	1.85E-03	0.575	8.87E-05	0.575
	WS6	9.66E-05	0.627	3.68E-03	0.627	7.94E-05	0.700	4.04E-06	0.700	5.50E-06	0.838	2.51E-05	0.838	1.07E-03	0.604	4.13E-04	0.604
	WS7	1.45E-04	0.611	2.25E-03	0.611	2.90E-05	0.742	1.06E-05	0.742	8.35E-06	0.822	5.76E-04	0.822	4.85E-04	0.644	6.17E-05	0.644
	WS8	7.32E-05	0.637	2.30E-03	0.637	1.08E-04	0.687	1.14E-04	0.687	1.44E-05	0.800	6.36E-05	0.800	1.91E-03	0.573	8.87E-05	0.573
No Feature Selection	**FULL**	2.50E-05	0.677	5.04E-04	0.677	1.78E-04	0.665	4.03E-06	0.665	6.04E-06	0.835	7.35E-06	0.835	2.11E-03	0.567	1.26E-04	0.567

Table 10
Comparison Results on RELINK between MOFES and FULL in terms of F1.

Projects	J48				KNN				LR				NB			
	MOFES		FULL		MOFES		FULL		MOFES		FULL		MOFES		FULL	
	# Features	Avg	# Features	Avg	# Features	Avg	# Features	Avg	# Features	Avg	# Features	Avg	# Features	Avg	# Features	Avg
Apache	1	0.698	26	0.671	1	0.640	26	0.665	1	0.723	26	0.670	1	0.703	26	0.706
	2	0.718			2	0.718			2	0.742			2	0.720		
	3	0.744			3	0.747			3	0.787			3	0.750		
	4	0.703			4	0.738			4	0.723			4	0.743		
	5	0.700			5	0.722			5	0.757			5	0.730		
	7	0.667			6	0.760			7	0.800			7	0.750		
	10	0.759			7	0.743			8	0.776			9	0.754		
					8	0.721			9	0.879						
Safe	1	0.817	26	0.741	1	0.747	26	0.719	1	0.796	26	0.708	1	0.788	26	0.731
	2	0.818			2	0.799			2	0.817			4	0.917		
	4	0.889			5	0.857			4	0.727			5	0.696		
									5	0.870						
									9	0.900						
ZXing	1	0.835	26	0.768	1	0.816	26	0.748	1	0.818	26	0.794	1	0.812	26	0.781
	2	0.818			2	0.789			2	0.828			2	0.801		
	3	0.828			3	0.796			3	0.822			4	0.824		
	5	0.824			4	0.757			4	0.827						
					5	0.759			5	0.830						
					6	0.749			7	0.834						
					7	0.833			9	0.831						
					8	0.773										
					9	0.766										

Table 11
Top 10 Features Frequently Selected by MOFES on RELINK.

J48			KNN			LR			NB		
Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion
CountLineCodeExe	CTM	0.451	CountLineBlank	CTM	0.552	CountLineCodeExe	CTM	0.618	CountStmtDecl	CTM	0.292
CountLineBlank	CTM	0.279	MaxCyclomaticStrict	CPM	0.296	AvgEssential	CPM	0.366	AvgCyclomatic	CPM	0.280
AvgEssential	CPM	0.255	CountLineCodeExe	CTM	0.245	CountLineBlank	CTM	0.315	RatioCommentToCode	CTM	0.268
AvgLineBlank	CTM	0.237	CountStmtDecl	CTM	0.243	MaxCyclomaticStrict	CPM	0.250	AvgLineComment	CTM	0.246
MaxCyclomaticModified	CPM	0.204	AvgLineComment	CTM	0.180	SumEssential	CPM	0.147	CountLine	CTM	0.225
SumCyclomaticModified	CPM	0.140	CountLineCodeDecl	CTM	0.169	CountStmtExe	CTM	0.136	CountStmtExe	CTM	0.198
MaxCyclomaticStrict	CPM	0.133	SumCyclomaticStrict	CPM	0.143	MaxCyclomaticModified	CPM	0.135	CountLineCodeDecl	CTM	0.186
CountLineCode	CTM	0.126	MaxCyclomaticModified	CPM	0.142	SumCyclomaticStrict	CPM	0.134	AvgCyclomaticStrict	CPM	0.166
AvgCyclomaticStrict	CPM	0.116	SumCyclomaticModified	CPM	0.139	AvgCyclomaticStrict	CPM	0.124	MaxCyclomaticStrict	CPM	0.155
CountStmt	CTM	0.095	AvgLine	CTM	0.135	SumCyclomatic	CPM	0.123	CountLineCodeExe	CTM	0.153

Table 12
Top 10 Features Frequently Selected by MOFES on PROMISE.

J48			KNN			LR			NB		
Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion	Feature Name	Category	Selection Proportion
CE	Martin	0.365	MFA	HD	0.306	CE	Martin	0.348	LCOM3	HS	0.364
WMC	CK	0.270	CBO	CK	0.252	CA	ECK	0.336	CE	Martin	0.306
CA	Martin	0.263	CE	Martin	0.240	CBO	CK	0.324	RFC	CK	0.272
NPM	HS	0.240	DAM	HD	0.176	RFC	CK	0.322	MAX(CC)	CPM	0.246
RFC	CK	0.238	WMC	CK	0.159	NPM	HD	0.263	CA	Martin	0.216
MFA	BD	0.194	NOC	CK	0.147	AMC	ECK	0.209	CAM	HD	0.209
AMC	ECK	0.185	RFC	CK	0.142	IC	ECK	0.180	AMC	ECK	0.198
LOCM3	HS	0.183	DIT	CK	0.136	LCOM3	HS	0.177	CBM	ECK	0.179
CBO	CK	0.182	NPM	HD	0.133	CAM	HD	0.168	CBO	CK	0.155
LOC	CPM&OOM	0.165	CBM	ECK	0.133	MAX(CC)	CPM	0.168	IC	ECK	0.152

baseline methods. We collect the sum of 10 independent execution running time for each baseline method and MOFES on 13 projects. Notice that the gathered running time includes feature selection, data preprocessing, model construction on the training set and the model application on the testing set. To conduct empirical studies, we implement all the methods by Java programming language with Weka packages and JMetal packages (Durillo and Nebro, 2011b). All the methods are executed on CentOS Linux 7 (Memory: 32 GB, Processor: Intel Xeon(R) CPU E5-2609 v2 @ 2.50GHz × 8). Moreover, we use the multi-thread technology supported by JMetal to speed up the execution of MOFES. The final results can be found in Table 13.

In Table 13, all the columns are separated into six groups. The first group lists the project name. The second group includes our proposed method MOFES. The next three groups include methods in filter based ranking method category, filter based subset method category, wrapper based subset selection method category respectively. The last group is the baseline method using no feature selection. For each project, the cell is emphasized in bold for the corresponding method with the highest computational cost for different groups. Moreover, the cell is filled with grey color for the corresponding method with the highest computational cost in all the methods.

From Table 13, for filter based feature ranking method category, FR7 needs highest computational cost. For filter based subset method category, FS2 needs highest computational cost. For wrapper based subset method category, WS6 needs highest computational cost. The computational cost of our proposed method MOFES need 120.799 seconds on average on PROMISE and need 64.743 seconds on average on RELINK. It is not hard to find that computational cost of MOFES is higher than filter based feature selections methods, but lower than most of wrapper based feature selection methods.

Summary for RQ4: The average computational cost of MOFES is higher than filter based feature selection methods, but is competitive in wrapper based feature selection methods. The computational cost of MOFES is 32.336 ∼ 100.198 seconds on RELINK and 50.97 ∼ 262.524 seconds on PROMISE.

5.5. Threats to validity

In this subsection, we mainly discuss the potential threats to validity of our empirical studies.

Threats to external validity are about whether the observed experimental results can be generalized to other subjects. To guarantee the representative of our empirical subjects, we chose RELINK and PROMISE datasets, which have been widely used by previous SDP empirical studies (Menzies et al., 2007; Tantithamthavorn et al., 2017; Lessmann et al., 2008; Ghotra et al., 2015; Wang et al., 2016b; Nam and Kim, 2015; Liu et al., 2016; Xu et al., 2016a, 2016b; Liu et al., 2014b, 2015; Song et al., 2011). In addition, we choose four classical classifier (i.e., J48, KNN, LR, and NB) as the classifiers for MOFES, which are also widely used by previous research for software defect prediction (Radjenovic et al., 2013; Kamei and Shihab, 2016; Hall et al., 2012; Ghotra et al., 2017; Khoshgoftaar et al., 2012; Rahman and Devanbu, 2013; Xia et al., 2016).

Threats to internal validity are mainly concerned with the uncontrolled internal factors that might have influence on the experimental results. To minimizes the internal threats, we implement these methods by pair programming. Test cases are designed and then used to verify the correctness of our implemented MOFES and other baseline feature selection methods. Moreover we utilize mature third-party libraries (such as Weka packages) to im-

Table 13
Computational Cost of Different Feature Selection Methods on RELINK And PROMISE (Unit: Second).

Project Name	MOFES	Filter based Ranking Methods										Filter based Subset Methods		Wrapper based Subset Selection Methods							None
		FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	
Ant-1.7	164.793	2.155	0.592	0.523	0.511	5.733	3.455	**24.852**	1.446	1.589	8.651	1.456	1.227	**3.124**	58.107	1420.672	171.858	572.73	346.669	2173.33	31.599	294.61	0.969
Camel-1.6	262.524	2.463	0.651	0.649	0.564	11.186	3.599	**165.905**	1.844	1.649	9.038	1.278	1.245	**2.429**	54.963	2188.037	207.592	913.973	456.704	2274.889	30.322	406.96	1.52
Ivy-2.0	69.122	1.664	0.526	0.383	0.246	1.279	1.856	**25.71**	0.921	0.583	2.042	0.393	0.515	**0.772**	10.225	395.55	27.791	204.122	132.257	1167.808	16.225	215.513	0.406
Jedit-4.0	63.871	1.644	0.47	0.385	0.209	0.975	1.96	**33.219**	0.878	0.576	1.64	0.652	0.549	**1.368**	32.889	452.451	50.892	229.296	275.183	1152.375	32.952	155.586	0.403
Lucene-2.4	75.077	1.702	0.495	0.362	0.227	1.227	1.592	**12.528**	0.695	0.6	2.287	0.598	0.547	**1.489**	53.475	571.447	95.908	209.355	249.129	1048.872	18.134	145.439	0.326
Poi-3.0	93.14	1.766	0.614	0.356	0.301	1.988	2.183	**11.296**	1.322	1.039	2.46	0.482	0.395	**1.605**	40.484	745.007	82.545	286.577	288.271	1481.567	43.606	220.951	0.637
Synapse-1.2	55.023	1.514	0.46	0.375	0.236	0.737	1.364	**12.17**	0.508	0.408	1.262	0.304	0.291	**0.989**	40.154	345.456	55.898	152.527	157.856	816.122	23.455	126.47	0.219
Velocity-1.6	50.97	1.292	0.443	0.3	0.21	0.618	1.404	**10.658**	0.637	0.447	1.122	0.36	0.326	**0.57**	29.787	309.818	35.394	147.021	165.724	942.349	14.88	125.705	0.276
Xalan-2.6	216.699	2.584	0.674	0.847	0.931	6.931	4.181	**30.085**	1.878	1.954	8.286	0.917	1.178	**4.429**	421.273	1636.743	296.55	887.322	745.94	2110.435	70.955	232.527	0.91
Xerces-1.4	156.771	1.978	0.596	0.466	0.425	3.204	2.5	**13.134**	0.95	0.861	3.697	1.105	0.616	**1.301**	24.026	659.235	82.701	550.482	171.461	3404.765	12.008	260.861	0.78
Apache	61.695	1.635	0.433	0.348	0.242	0.64	1.437	**11.133**	1.031	0.617	0.835	0.389	0.217	**0.495**	21.923	634.059	33.199	170.684	294.901	3150.522	21.365	169.805	0.404
Safe	32.336	1.132	0.351	0.267	0.163	0.353	1.354	**13.989**	0.518	0.368	0.493	0.254	0.149	**0.178**	7.39	165.654	19.722	46.575	64.856	367.353	4.885	84.466	0.083
Zxing	100.198	1.66	0.496	0.356	0.254	2.073	2.752	**62.356**	0.911	0.837	2.944	0.61	0.611	**1.165**	9.973	1912.725	63.333	523.129	238.674	2764.166	20.065	277.061	0.664
Average on PROMISE	120.799	1.876	0.552	0.465	0.386	3.388	2.409	33.956	1.108	0.971	4.049	0.755	0.689	1.808	76.538	872.442	110.713	415.341	298.919	1657.251	29.414	218.462	0.645
Average on RELINK	64.743	1.476	0.427	0.324	0.220	1.022	1.848	29.159	0.820	0.607	1.424	0.418	0.326	0.613	13.095	904.146	38.751	246.796	199.477	2094.014	15.438	177.111	0.384
Average on ALL	107.863	1.784	0.523	0.432	0.348	2.842	2.280	32.849	1.041	0.887	3.443	0.677	0.605	1.532	61.898	879.758	94.106	376.446	275.971	1758.043	26.189	208.920	0.584

plement classifiers and use jmetal[4] packages to implement PMAs. Besides, different values of hyperparameters may generate different results for the same classifier. For the convenience of experiment replication, we use default values for hyperparameters of classifiers in Weka. In the future, we want to investigate the performance of MOFES by considering hyperparameter optimization (Tantithamthavorn et al., 2016). Meanwhile, many quality indicators are proposed for evaluate the quality of Pareto fronts generated by different PMAs, such as IGD, epsilon. This paper only consider HV indicator, since it is mostly used in evaluation of PMAs in previous studies.

Threat to construct validity are about whether the performance metrics used in the empirical studies reflect the real-world situation. To minimize the threads, we used AUC to evaluate the performance of the prediction model. This performance metric has been widely used in current software defect prediction research (Tantithamthavorn et al., 2017; Zhang et al., 2016; Tantithamthavorn et al., 2016; Xu et al., 2016b, 2016a; Liu et al., 2016, 2015; Ghotra et al., 2015; Liu et al., 2014b; Lessmann et al., 2008).

Threats to conclusion validity focus on the used statistical analysis methods. In this paper, the Friedman test is used to ranking PMAs, which helps to ensure the confidence of performance comparison for different PMAs. Besides, Win/Tie/Loss analysis is also used for comparison between MOFES and all baseline methods.

6. Conclusion and future work

In this paper, we formalize feature selection for software defect prediction as a multi-objective optimization problem and propose MOFES method. Based on our empirical research, we find that multi-objective feature selection is very useful for SDP and MOFES can effectively select fewer but useful features to build high-quality SDP models.

In the future, we firstly want to consider more projects from open source community or commercial enterprises to verify whether our empirical results can be generalized. Secondly, we want to optimize the performance of our method and consider other recently proposed PMAs. Thirdly, we want to incorporate class imbalanced learning methods into MOFES, since the number of defective modules is less than the number of non-defective modules in most SDP datasets. Fourthly, we only consider wrapper based multi-objective optimization algorithms in this paper since its performance is better than the filter based multi-objective optimization algorithms in most cases. In the next step, we will analyze this type of method in depth as a supplement to this study. Fifthly, we want to consider more classifiers (such as decision tree, logistic regression) to verify the effectiveness of MOFES. Finally, recently a few studies (Jiarpakdee et al., 2018, 2019; Tantithamthavorn et al., 2018) have been conducted to investigate the impacts of feature selection methods on the interpretation of defect prediction models and the interpretation of prediction models can be used to formulate empirical theories related to software quality, which are essential to chart quality improvement and maintenance plans. Therefore we want to investigate the impact of our proposed feature selection method on the interpretation of prediction models in the future.

[4] https://jmetal.github.io/jMetal/.

Acknowledgment

The authors would like to thank the editors and the anonymous reviewers for their insightful comments and suggestions, which can substantially improve the quality of this work. This work was supported in part by National Natural Science Foundation of China (Grant Nos. 61373012, 61202006, 61602267), The Open Project of State Key Laboratory for Novel Software Technology at Nanjing University (Grant Nos. KFKT2016B18) and China Scholarship Council (Grant No. 201806190172), which supports overseas studies for Chao Ni to Monash University in Australia. Chao Ni and Xiang Chen contributed equally for this work. Xiang Chen and Qing Gu are the corresponding authors.

Appendix A. Comparison Results based on the Mean and Standard Deviation in terms of HV

Tables A.1 and A.2 show the results based on mean and standard deviation in terms of HV on RELINK and PROMISE respectively. In these two tables, some cells have gray background. In particular, a darker one means the corresponding PMA can achieve the best performance while a lighter one means the corresponding PMA can achieve the second best performance. Based on these two tables, we find that NSGA-II can achieve the best performance in most cases followed by MOCell. In particular, when analyzing results on RELINK in Table A.1, we can find NSGA-II can achieve the best performance in 8 times. While MOCell can achieve the best performance in 3 times and achieve the second best performance in 6 times. When analyzing results on PROMISE in Table A.2, we can find NSGA-II can achieve the best performance in 27 times and achieve the second best performance in 10 times. While MOCell can achieve the best performance in 4 times and achieve the second best performance in 20 times.

Table A.1
Comparison Results for Different PMAs based on the Mean and Standard Deviation in terms of HV on RELINK.

	NSGA-II	MOCell	SPEA2	PAES	SMSEMOA	RandomSearch
Apache.KNN	$7.50e-01_{1.4e-02}$	$7.62e-01_{8.2e-03}$	$7.49e-01_{1.7e-02}$	$7.44e-01_{3.9e-02}$	$7.58e-01_{1.2e-02}$	$1.91e-01_{4.6e-02}$
Safe.KNN	$6.59e-01_{2.0e-02}$	$6.69e-01_{1.4e-02}$	$6.69e-01_{1.0e-02}$	$6.19e-01_{1.3e-01}$	$6.34e-01_{4.2e-02}$	$6.07e-02_{4.2e-02}$
ZXing.KNN	$7.91e-01_{2.3e-02}$	$7.88e-01_{2.9e-02}$	$7.85e-01_{2.4e-02}$	$7.40e-01_{7.4e-02}$	$7.87e-01_{3.8e-02}$	$4.24e-01_{2.8e-02}$
Apache.LR	$7.65e-01_{7.5e-03}$	$7.69e-01_{7.0e-03}$	$7.66e-01_{6.5e-03}$	$7.61e-01_{7.3e-03}$	$7.54e-01_{2.9e-02}$	$2.37e-01_{8.9e-02}$
Safe.LR	$7.93e-01_{4.8e-03}$	$7.87e-01_{1.8e-02}$	$7.79e-01_{1.9e-02}$	$7.71e-01_{2.0e-02}$	$7.80e-01_{2.1e-02}$	$3.49e-01_{4.0e-02}$
ZXing.LR	$6.87e-01_{1.1e-03}$	$6.87e-01_{1.4e-03}$	$6.85e-01_{4.7e-03}$	$6.82e-01_{5.8e-03}$	$6.86e-01_{2.8e-03}$	$1.07e-01_{1.1e-01}$
Apache.J48	$6.56e-01_{9.2e-03}$	$6.50e-01_{2.1e-03}$	$6.52e-01_{6.4e-03}$	$6.47e-01_{1.5e-02}$	$6.49e-01_{5.8e-03}$	$0.00e+00_{0.0e+00}$
Safe.J48	$5.25e-01_{7.5e-09}$	$5.25e-01_{7.5e-09}$	$5.01e-01_{3.7e-02}$	$5.14e-01_{3.4e-02}$	$4.71e-01_{7.7e-02}$	$0.00e+00_{0.0e+00}$
ZXing.J48	$6.83e-01_{1.6e-02}$	$6.69e-01_{2.2e-02}$	$6.77e-01_{2.4e-02}$	$6.31e-01_{5.5e-02}$	$6.73e-01_{1.2e-02}$	$3.25e-01_{4.3e-02}$
Apache.NB	$8.64e-01_{3.0e-03}$	$8.64e-01_{1.2e-03}$	$8.64e-01_{3.5e-03}$	$8.39e-01_{2.6e-02}$	$8.52e-01_{2.0e-02}$	$5.44e-01_{7.1e-02}$
Safe.NB	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$6.09e-01_{4.1e-04}$	$6.06e-01_{6.6e-03}$	$0.00e+00_{0.0e+00}$
ZXing.NB	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$

Table A.2
Comparison Results for Different PMAs based on the Mean and Standard Deviation in terms of HV on PROMISE.

	NSGA-II	MOCell	SPEA2	PAES	SMSEMOA	RandomSearch
Ant-1.7.J48	$8.46e-01_{2.1e-03}$	$8.44e-01_{3.1e-03}$	$8.44e-01_{1.3e-03}$	$8.40e-01_{7.9e-03}$	$8.44e-01_{3.5e-03}$	$5.95e-01_{5.0e-02}$
Camel-1.6.J48	$6.62e-01_{4.3e-02}$	$6.61e-01_{4.6e-02}$	$6.39e-01_{5.2e-02}$	$5.92e-01_{7.2e-02}$	$6.62e-01_{3.1e-02}$	$4.73e-01_{2.7e-02}$
Ivy-2.0.J48	$7.09e-01_{1.1e-02}$	$6.99e-01_{5.7e-03}$	$7.07e-01_{1.0e-02}$	$6.70e-01_{4.2e-02}$	$7.00e-01_{1.8e-02}$	$4.78e-01_{2.6e-02}$
Jedit-4.0.J48	$7.99e-01_{1.2e-02}$	$8.06e-01_{1.3e-02}$	$7.97e-01_{7.2e-03}$	$7.64e-01_{4.8e-02}$	$7.99e-01_{8.9e-03}$	$5.91e-01_{2.1e-02}$
Lucene-2.4.J48	$7.77e-01_{5.4e-03}$	$7.80e-01_{3.1e-03}$	$7.77e-01_{4.3e-03}$	$7.67e-01_{9.9e-03}$	$7.76e-01_{7.8e-03}$	$5.54e-01_{2.6e-02}$
Poi-3.0-J48	$8.13e-01_{3.4e-03}$	$8.09e-01_{7.3e-03}$	$8.12e-01_{5.0e-03}$	$7.94e-01_{3.7e-02}$	$8.10e-01_{4.5e-03}$	$5.96e-01_{4.9e-02}$
Synapse-1.2.J48	$8.09e-01_{9.4e-04}$	$7.98e-01_{1.4e-02}$	$8.03e-01_{5.5e-03}$	$7.76e-01_{3.5e-02}$	$7.97e-01_{1.2e-02}$	$5.99e-01_{2.9e-02}$
Velocity-1.6.J48	$6.64e-01_{3.3e-03}$	$6.65e-01_{3.8e-03}$	$6.60e-01_{9.8e-03}$	$6.46e-01_{2.4e-02}$	$6.63e-01_{6.3e-03}$	$4.12e-01_{4.0e-02}$
Xalan-2.6.J48	$8.65e-01_{6.2e-03}$	$8.63e-01_{6.8e-03}$	$8.62e-01_{7.4e-03}$	$8.61e-01_{7.2e-03}$	$8.62e-01_{6.3e-03}$	$6.95e-01_{3.0e-02}$
Xerces-1.4.J48	$8.08e-01_{0.0e+00}$	$8.08e-01_{0.0e+00}$	$8.08e-01_{0.0e+00}$	$8.05e-01_{4.5e-03}$	$8.06e-01_{2.5e-03}$	$4.51e-01_{7.4e-02}$
Ant-1.7.KNN	$6.48e-01_{2.4e-03}$	$6.49e-01_{3.2e-04}$	$6.49e-01_{3.6e-04}$	$6.48e-01_{2.2e-03}$	$6.48e-01_{3.8e-03}$	$2.12e-01_{6.8e-02}$
Camel-1.6.KNN	$6.62e-01_{5.1e-03}$	$6.58e-01_{5.4e-03}$	$6.60e-01_{3.6e-03}$	$6.50e-01_{2.5e-02}$	$6.63e-01_{4.3e-03}$	$4.06e-01_{5.9e-02}$
Ivy-2.0.KNN	$7.33e-01_{7.3e-03}$	$7.34e-01_{6.6e-03}$	$7.35e-01_{2.7e-03}$	$7.26e-01_{2.7e-02}$	$7.31e-01_{6.3e-03}$	$3.86e-01_{1.0e-01}$
Jedit-4.0.KNN	$7.52e-01_{0.0e+00}$	$7.52e-01_{0.0e+00}$	$7.52e-01_{1.0e-03}$	$7.22e-01_{6.1e-02}$	$7.52e-01_{0.0e+00}$	$3.49e-01_{5.7e-02}$
Lucene-2.4.KNN	$7.80e-01_{6.6e-03}$	$7.79e-01_{1.3e-02}$	$7.81e-01_{6.3e-03}$	$7.35e-01_{4.5e-02}$	$7.77e-01_{7.3e-03}$	$5.65e-01_{6.2e-02}$
Poi-3.0.KNN	$6.96e-01_{1.1e-08}$	$6.96e-01_{1.1e-08}$	$6.96e-01_{1.1e-08}$	$6.71e-01_{7.6e-02}$	$6.95e-01_{3.4e-03}$	$2.91e-01_{1.1e-01}$
Synapse-1.2.KNN	$7.36e-01_{3.4e-03}$	$7.17e-01_{1.9e-02}$	$7.21e-01_{1.1e-02}$	$6.83e-01_{6.4e-02}$	$7.16e-01_{1.8e-02}$	$4.67e-01_{4.8e-02}$
Velocity-1.6.KNN	$4.09e-01_{5.3e-09}$	$4.09e-01_{5.3e-09}$	$4.09e-01_{5.3e-09}$	$4.09e-01_{5.3e-09}$	$4.09e-01_{5.3e-09}$	$0.00e+00_{0.0e+00}$
Xalan-2.6.KNN	$8.13e-01_{5.0e-03}$	$8.10e-01_{7.5e-03}$	$8.13e-01_{5.0e-03}$	$7.73e-01_{5.9e-02}$	$8.14e-01_{2.3e-03}$	$6.11e-01_{4.1e-02}$
Xerces-1.4.KNN	$8.20e-01_{1.2e-03}$	$8.20e-01_{0.0e+00}$	$8.20e-01_{1.4e-03}$	$8.19e-01_{1.8e-03}$	$8.20e-01_{2.1e-04}$	$4.98e-01_{3.9e-02}$
Ant-1.7.LR	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$0.00e+00_{0.0e+00}$
Camel-1.6.LR	$7.50e-01_{9.0e-04}$	$7.50e-01_{1.2e-03}$	$7.49e-01_{1.4e-03}$	$7.41e-01_{5.9e-03}$	$7.44e-01_{1.0e-02}$	$5.17e-01_{5.4e-02}$
Ivy2.0.LR	$7.81e-01_{6.1e-04}$	$7.80e-01_{7.7e-04}$	$7.81e-01_{6.8e-04}$	$7.80e-01_{6.1e-04}$	$7.81e-01_{1.1e-03}$	$3.64e-01_{6.6e-02}$
Jedit-4.0.LR	$8.35e-01_{1.3e-03}$	$8.33e-01_{4.2e-03}$	$8.32e-01_{4.5e-03}$	$8.13e-01_{2.8e-02}$	$8.30e-01_{5.3e-03}$	$6.09e-01_{4.2e-02}$
Lucene-2.4.LR	$8.19e-01_{1.3e-04}$	$8.18e-01_{8.7e-04}$	$8.19e-01_{6.9e-04}$	$8.05e-01_{2.6e-02}$	$8.19e-01_{5.2e-04}$	$6.27e-01_{3.9e-02}$
Poi-3.0.LR	$8.86e-01_{8.4e-04}$	$8.86e-01_{1.1e-03}$	$8.85e-01_{1.9e-03}$	$8.85e-01_{7.6e-04}$	$8.85e-01_{2.1e-03}$	$6.99e-01_{2.9e-02}$
Synapse-1.2.LR	$8.32e-01_{1.8e-08}$	$8.31e-01_{7.9e-04}$	$8.31e-01_{6.0e-04}$	$8.30e-01_{1.7e-03}$	$8.31e-01_{3.9e-04}$	$6.22e-01_{6.8e-02}$
Velocity-1.6.LR	$7.98e-01_{9.6e-04}$	$7.97e-01_{1.1e-03}$	$7.98e-01_{9.4e-04}$	$7.97e-01_{1.4e-03}$	$7.98e-01_{1.2e-03}$	$5.41e-01_{6.1e-02}$
Xalan-2.6.LR	$8.05e-01_{0.0e+00}$	$8.05e-01_{0.0e+00}$	$8.05e-01_{0.0e+00}$	$8.03e-01_{4.8e-03}$	$7.99e-01_{8.0e-03}$	$5.22e-01_{9.3e-02}$
Xerces-1.4.LR	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$3.64e-01_{4.0e-02}$
Ant-1.7.NB	$7.56e-01_{1.5e-08}$	$7.56e-01_{1.5e-08}$	$7.56e-01_{2.5e-04}$	$7.52e-01_{8.7e-03}$	$7.56e-01_{3.6e-04}$	$3.94e-01_{9.4e-02}$
Camel-1.6.NB	$7.56e-01_{6.2e-04}$	$7.54e-01_{4.7e-03}$	$7.56e-01_{3.7e-04}$	$7.55e-01_{1.4e-03}$	$7.55e-01_{1.0e-03}$	$5.38e-01_{3.3e-02}$
Ivy-2.0.NB	$7.10e-01_{1.1e-08}$	$7.10e-01_{1.1e-08}$	$7.10e-01_{1.1e-08}$	$7.05e-01_{1.1e-02}$	$7.10e-01_{1.1e-08}$	$3.03e-01_{1.5e-01}$
Jedit-4.0.NB	$7.34e-01_{2.0e-03}$	$7.32e-01_{2.5e-03}$	$7.32e-01_{2.5e-03}$	$7.28e-01_{3.5e-03}$	$7.33e-01_{2.7e-03}$	$4.66e-01_{3.8e-02}$
Lucene-2.4.NB	$7.54e-01_{1.1e-08}$	$7.54e-01_{3.2e-04}$	$7.54e-01_{7.3e-04}$	$7.43e-01_{2.4e-02}$	$7.48e-01_{6.2e-03}$	$5.00e-01_{3.9e-02}$
Poi-3.0.NB	$7.73e-01_{1.8e-08}$	$7.73e-01_{1.8e-08}$	$7.73e-01_{1.8e-08}$	$7.73e-01_{1.8e-08}$	$7.73e-01_{1.8e-08}$	$3.94e-01_{5.6e-02}$
Synapse-1.2.NB	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$7.69e-01_{2.6e-04}$	$7.69e-01_{0.0e+00}$	$4.44e-01_{6.3e-02}$
Velocity-1.6.NB	$6.91e-01_{1.3e-08}$	$6.91e-01_{1.3e-08}$	$6.91e-01_{1.3e-08}$	$6.91e-01_{1.3e-08}$	$6.91e-01_{1.3e-08}$	$2.62e-01_{9.6e-02}$
Xalan-2.6.NB	$8.07e-01_{1.9e-03}$	$8.07e-01_{1.8e-03}$	$8.04e-01_{7.4e-03}$	$8.03e-01_{8.5e-03}$	$7.99e-01_{9.7e-03}$	$5.72e-01_{3.5e-02}$
Xerces-1.4.NB	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$2.61e-01_{6.8e-02}$

Appendix B. Comparison Results based on the Median and IRQ in terms of HV

Tables B.1 and B.2 show the results based on median and IRQ in terms of HV on RELINK and PROMISE respectively. In these two tables, some cells have gray background. In particular, a darker one means the corresponding PMA can achieve the best performance while a lighter one means the corresponding PMA can achieve the second best performance. Based on these two tables, we also find that NSGA-II can achieve the best performance in most cases followed by MOCell. In particular, when analyzing results on RELINK in Table B.1, we can find NSGA-II can achieve the best performance in 4 times and achieve the second best performance in 3 times. While MOCell can achieve the best performance in 4 times and achieve the second best performance in 5 times. When analyzing results on PROMISE in Table B.2, we can find NSGA-II can achieve the best performance in 30 times and achieve the second best performance in 9 times. While MOCell can achieve the best performance in 7 times and achieve the second best performance in 20 times.

Table B.1
Comparison Results for Different PMAs based on the Median and IQR in terms of HV on RELINK.

	NSGA-II	MOCell	SPEA2	PAES	SMSEMOA	RandomSearch
Apache.KNN	$7.49e-01_{2.4e-02}$	$7.64e-01_{7.5e-03}$	$7.56e-01_{3.1e-02}$	$7.63e-01_{2.2e-02}$	$7.63e-01_{2.4e-02}$	$1.91e-01_{3.7e-02}$
Safe.KNN	$6.64e-01_{3.4e-02}$	$6.77e-01_{2.0e-02}$	$6.72e-01_{1.2e-02}$	$6.64e-01_{3.9e-02}$	$6.41e-01_{6.0e-02}$	$6.89e-02_{8.0e-02}$
ZXing.KNN	$8.00e-01_{4.4e-02}$	$7.93e-01_{6.3e-02}$	$7.79e-01_{5.0e-02}$	$7.42e-01_{8.7e-02}$	$8.08e-01_{5.6e-02}$	$4.15e-01_{3.5e-02}$
Apache.LR	$7.67e-01_{1.3e-02}$	$7.71e-01_{2.4e-03}$	$7.68e-01_{1.2e-02}$	$7.62e-01_{1.6e-02}$	$7.65e-01_{1.4e-02}$	$2.65e-01_{1.5e-01}$
Safe.LR	$7.93e-01_{6.5e-03}$	$7.94e-01_{4.6e-03}$	$7.84e-01_{1.8e-02}$	$7.74e-01_{3.5e-02}$	$7.87e-01_{1.2e-02}$	$3.44e-01_{5.9e-02}$
ZXing.LR	$6.88e-01_{1.1e-03}$	$6.88e-01_{4.8e-04}$	$6.87e-01_{2.1e-03}$	$6.83e-01_{8.5e-03}$	$6.88e-01_{2.1e-03}$	$1.13e-01_{1.4e-01}$
Apache.J48	$6.58e-01_{1.3e-02}$	$6.51e-01_{4.6e-03}$	$6.51e-01_{4.6e-03}$	$6.47e-01_{1.7e-02}$	$6.47e-01_{4.1e-03}$	$0.00e+00_{0.0e+00}$
Safe.J48	$5.25e-01_{0.0e+00}$	$5.25e-01_{0.0e+00}$	$5.25e-01_{6.6e-02}$	$5.25e-01_{0.0e+00}$	$5.03e-01_{6.6e-02}$	$0.00e+00_{0.0e+00}$
ZXing.J48	$6.78e-01_{1.9e-02}$	$6.76e-01_{3.2e-02}$	$6.79e-01_{2.3e-02}$	$6.54e-01_{9.4e-02}$	$6.77e-01_{8.4e-03}$	$3.16e-01_{7.6e-02}$
Apache.NB	$8.65e-01_{4.8e-03}$	$8.64e-01_{2.3e-03}$	$8.62e-01_{7.3e-03}$	$8.49e-01_{1.1e-02}$	$8.59e-01_{9.1e-03}$	$5.14e-01_{1.1e-01}$
Safe.NB	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$6.09e-01_{0.0e+00}$	$0.00e+00_{0.0e+00}$
ZXing.NB	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$	$0.00e+00_{0.0e+00}$

Table B.2
Comparison Results for Different PMAs based on the Median and IQR in terms of HV on PROMISE.

	NSGA-II	MOCell	SPEA2	PAES	SMSEMOA	RandomSearch
Ant-1.7.J48	$8.47e-01_{3.8e-03}$	$8.45e-01_{4.5e-03}$	$8.44e-01_{1.5e-03}$	$8.42e-01_{5.9e-03}$	$8.44e-01_{4.5e-03}$	$5.79e-01_{4.4e-02}$
Camel-1.6.J48	$6.80e-01_{3.8e-02}$	$6.92e-01_{6.3e-02}$	$6.51e-01_{1.1e-01}$	$5.73e-01_{1.2e-01}$	$6.70e-01_{4.3e-02}$	$4.75e-01_{4.0e-02}$
Ivy-2.0.J48	$7.07e-01_{2.0e-02}$	$6.99e-01_{1.3e-02}$	$7.05e-01_{2.8e-03}$	$6.99e-01_{6.7e-02}$	$7.02e-01_{1.4e-02}$	$4.77e-01_{3.2e-02}$
Jedit-4.0.J48	$8.04e-01_{2.0e-02}$	$8.12e-01_{2.1e-02}$	$7.98e-01_{1.5e-02}$	$7.83e-01_{3.9e-02}$	$7.97e-01_{1.5e-02}$	$5.90e-01_{3.4e-02}$
Lucene-2.4.J48	$7.79e-01_{2.9e-03}$	$7.80e-01_{5.6e-03}$	$7.78e-01_{6.1e-03}$	$7.68e-01_{7.1e-03}$	$7.79e-01_{4.8e-03}$	$5.43e-01_{4.2e-02}$
Poi-3.0-J48	$8.13e-01_{5.4e-03}$	$8.12e-01_{1.3e-02}$	$8.11e-01_{8.6e-03}$	$8.06e-01_{1.0e-02}$	$8.11e-01_{7.0e-03}$	$5.86e-01_{6.2e-02}$
Synapse-1.2.J48	$8.09e-01_{3.1e-04}$	$8.09e-01_{2.9e-02}$	$8.05e-01_{9.7e-03}$	$7.90e-01_{5.2e-02}$	$8.01e-01_{2.1e-02}$	$5.88e-01_{3.1e-02}$
Velocity-1.6.J48	$6.66e-01_{2.2e-03}$	$6.66e-01_{0.0e+00}$	$6.65e-01_{8.8e-03}$	$6.60e-01_{4.0e-02}$	$6.66e-01_{2.2e-03}$	$4.08e-01_{7.7e-02}$
Xalan-2.6.J48	$8.65e-01_{9.6e-03}$	$8.61e-01_{1.2e-02}$	$8.62e-01_{9.9e-03}$	$8.61e-01_{1.2e-02}$	$8.61e-01_{9.1e-03}$	$6.93e-01_{5.3e-02}$
Xerces-1.4.J48	$8.08e-01_{0.0e+00}$	$8.08e-01_{0.0e+00}$	$8.08e-01_{0.0e+00}$	$8.07e-01_{3.6e-03}$	$8.08e-01_{5.3e-03}$	$4.57e-01_{4.8e-02}$
Ant-1.7.KNN	$6.49e-01_{0.0e+00}$	$6.49e-01_{0.0e+00}$	$6.49e-01_{7.9e-04}$	$6.49e-01_{7.9e-04}$	$6.49e-01_{7.9e-04}$	$1.85e-01_{7.8e-02}$
Camel-1.6.KNN	$6.61e-01_{8.3e-03}$	$6.58e-01_{3.8e-03}$	$6.60e-01_{5.9e-03}$	$6.59e-01_{1.2e-02}$	$6.65e-01_{6.2e-03}$	$3.83e-01_{6.5e-02}$
Ivy-2.0.KNN	$7.36e-01_{1.1e-02}$	$7.35e-01_{7.3e-03}$	$7.36e-01_{1.4e-03}$	$7.35e-01_{2.4e-03}$	$7.31e-01_{9.4e-03}$	$3.59e-01_{1.9e-01}$
Jedit-4.0.KNN	$7.52e-01_{0.0e+00}$	$7.52e-01_{0.0e+00}$	$7.52e-01_{4.4e-04}$	$7.52e-01_{3.1e-05}$	$7.52e-01_{0.0e+00}$	$3.38e-01_{1.2e-01}$
Lucene-2.4.KNN	$7.82e-01_{1.2e-02}$	$7.82e-01_{7.2e-03}$	$7.81e-01_{1.1e-02}$	$7.50e-01_{2.2e-02}$	$7.79e-01_{8.7e-03}$	$5.40e-01_{7.2e-02}$
Poi-3.0.KNN	$6.96e-01_{0.0e+00}$	$6.96e-01_{0.0e+00}$	$6.96e-01_{0.0e+00}$	$6.96e-01_{0.0e+00}$	$6.96e-01_{0.0e+00}$	$3.02e-01_{2.3e-01}$
Synapse-1.2.KNN	$7.39e-01_{6.2e-03}$	$7.16e-01_{4.3e-02}$	$7.18e-01_{2.1e-02}$	$7.05e-01_{4.1e-02}$	$7.18e-01_{1.5e-02}$	$4.75e-01_{7.3e-02}$
Velocity-1.6.KNN	$4.09e-01_{0.0e+00}$	$4.09e-01_{0.0e+00}$	$4.09e-01_{0.0e+00}$	$4.09e-01_{0.0e+00}$	$4.09e-01_{0.0e+00}$	$0.00e+00_{0.0e+00}$
Xalan-2.6.KNN	$8.14e-01_{6.3e-03}$	$8.11e-01_{8.5e-03}$	$8.15e-01_{4.3e-03}$	$7.99e-01_{3.6e-02}$	$8.15e-01_{2.7e-03}$	$6.06e-01_{8.1e-02}$
Xerces-1.4.KNN	$8.20e-01_{0.0e+00}$	$8.20e-01_{0.0e+00}$	$8.20e-01_{0.0e+00}$	$8.20e-01_{1.5e-03}$	$8.20e-01_{0.0e+00}$	$4.95e-01_{6.0e-02}$
Ant-1.7.LR	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$4.94e-01_{0.0e+00}$	$0.00e+00_{0.0e+00}$
Camel-1.6.LR	$7.50e-01_{1.5e-03}$	$7.50e-01_{2.3e-03}$	$7.50e-01_{2.3e-03}$	$7.41e-01_{8.5e-03}$	$7.47e-01_{8.5e-03}$	$5.14e-01_{8.6e-02}$
Ivy2.0.LR	$7.81e-01_{0.0e+00}$	$7.80e-01_{1.5e-03}$	$7.81e-01_{1.3e-03}$	$7.80e-01_{1.9e-04}$	$7.81e-01_{1.5e-03}$	$3.36e-01_{3.8e-02}$
Jedit-4.0.LR	$8.35e-01_{8.5e-04}$	$8.36e-01_{4.5e-03}$	$8.33e-01_{6.1e-03}$	$8.25e-01_{2.8e-02}$	$8.31e-01_{1.1e-02}$	$6.09e-01_{6.5e-02}$
Lucene-2.4.LR	$8.19e-01_{0.0e+00}$	$8.19e-01_{1.8e-03}$	$8.19e-01_{9.6e-04}$	$8.17e-01_{2.2e-03}$	$8.19e-01_{3.0e-04}$	$6.22e-01_{5.6e-02}$
Poi-3.0.LR	$8.87e-01_{1.2e-03}$	$8.85e-01_{1.4e-03}$	$8.86e-01_{1.4e-03}$	$8.85e-01_{0.0e+00}$	$8.85e-01_{1.4e-03}$	$6.89e-01_{9.5e-03}$
Synapse-1.2.LR	$8.32e-01_{0.0e+00}$	$8.32e-01_{6.3e-04}$	$8.31e-01_{6.3e-04}$	$8.30e-01_{2.0e-03}$	$8.32e-01_{6.3e-04}$	$6.33e-01_{7.3e-02}$
Velocity-1.6.LR	$7.98e-01_{4.1e-04}$	$7.97e-01_{1.7e-03}$	$7.98e-01_{1.6e-03}$	$7.96e-01_{2.1e-03}$	$7.98e-01_{1.7e-03}$	$5.45e-01_{6.1e-02}$
Xalan-2.6.LR	$8.05e-01_{0.0e+00}$	$8.05e-01_{0.0e+00}$	$8.05e-01_{0.0e+00}$	$8.05e-01_{3.9e-03}$	$8.05e-01_{1.6e-02}$	$5.15e-01_{6.7e-02}$
Xerces-1.4.LR	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$7.83e-01_{0.0e+00}$	$3.79e-01_{5.0e-02}$
Ant-1.7.NB	$7.56e-01_{0.0e+00}$	$7.56e-01_{0.0e+00}$	$7.56e-01_{0.0e+00}$	$7.56e-01_{9.5e-04}$	$7.56e-01_{0.0e+00}$	$4.02e-01_{1.3e-01}$
Camel-1.6.NB	$7.56e-01_{8.5e-05}$	$7.56e-01_{0.0e+00}$	$7.56e-01_{4.3e-04}$	$7.55e-01_{2.1e-03}$	$7.55e-01_{2.1e-03}$	$5.42e-01_{5.5e-02}$
Ivy-2.0.NB	$7.10e-01_{0.0e+00}$	$7.10e-01_{0.0e+00}$	$7.10e-01_{0.0e+00}$	$7.10e-01_{3.5e-04}$	$7.10e-01_{0.0e+00}$	$2.80e-01_{1.5e-01}$
Jedit-4.0.NB	$7.35e-01_{0.0e+00}$	$7.32e-01_{5.0e-03}$	$7.32e-01_{5.0e-03}$	$7.27e-01_{2.9e-03}$	$7.35e-01_{5.0e-03}$	$4.61e-01_{5.6e-02}$
Lucene-2.4.NB	$7.54e-01_{0.0e+00}$	$7.53e-01_{6.6e-04}$	$7.54e-01_{0.0e+00}$	$7.51e-01_{6.3e-03}$	$7.51e-01_{5.3e-03}$	$5.09e-01_{6.3e-02}$
Poi-3.0.NB	$7.73e-01_{0.0e+00}$	$7.73e-01_{0.0e+00}$	$7.73e-01_{0.0e+00}$	$7.73e-01_{0.0e+00}$	$7.73e-01_{0.0e+00}$	$4.09e-01_{8.7e-02}$
Synapse-1.2.NB	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$7.69e-01_{0.0e+00}$	$4.25e-01_{7.3e-02}$
Velocity-1.6.NB	$6.91e-01_{0.0e+00}$	$6.91e-01_{0.0e+00}$	$6.91e-01_{0.0e+00}$	$6.91e-01_{0.0e+00}$	$6.91e-01_{0.0e+00}$	$2.52e-01_{1.7e-01}$
Xalan-2.6.NB	$8.08e-01_{1.6e-04}$	$8.08e-01_{0.0e+00}$	$8.08e-01_{3.7e-04}$	$8.07e-01_{2.8e-03}$	$8.05e-01_{2.1e-02}$	$5.68e-01_{6.4e-02}$
Xerces-1.4.NB	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$7.23e-01_{0.0e+00}$	$2.45e-01_{7.7e-02}$

Appendix C. Detailed Comparison Results in terms of Win/Tie/Loss for Different Classifiers

The detailed comparison results in terms of Win/Tie/Loss when comparing MOFES with all baseline methods for different classifiers can be found in Tables C.1–C.4. The first column represents project name in two datasets. The remaining columns represent 22 baseline methods. Each row refers to the result in terms of Win/Tie/Loss for each dataset. The last row shows the summarization of all the Win/Tie/Loss when comparing to a specific baseline method.

Table C.1
Comparison Results in terms of Win/Tie/Loss when using J48 as the Classifier.

Projects	Baselines-J48																				
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	FULL
Ant-1.7	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0
Camel-1.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Ivy-2.0	6/4/0	6/4/0	7/3/0	6/4/0	8/1/1	7/3/0	7/1/2	8/2/0	8/2/0	10/0/0	10/0/0	9/1/0	9/1/0	9/1/0	9/1/0	10/0/0	9/1/0	8/2/0	9/0/1	6/4/0	10/0/0	9/1/0
Jedit-4.0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Lucene-2.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Poi-3.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Synapse-1.2	9/1/0	7/3/0	9/1/0	9/1/0	8/2/0	8/2/0	6/4/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	8/2/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	7/3/0	10/0/0	10/0/0
Velocity-1.6	8/2/0	10/0/0	9/1/0	10/0/0	7/3/0	8/2/0	7/3/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	9/1/0	8/2/0	10/0/0	9/1/0	10/0/0	6/4/0	10/0/0	8/2/0	10/0/0	10/0/0
Xalan-2.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	7/3/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0
Xerces-1.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Apache	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0
Safe	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/0/1	10/0/0	10/0/0	10/0/0	10/0/0	9/0/1	10/0/0	10/0/0
Zxing	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Total	**123/7/0**	**123/7/0**	**125/5/0**	**125/5/0**	**122/7/1**	**122/8/0**	**119/9/2**	**127/3/0**	**127/3/0**	**129/1/0**	**129/1/0**	**127/3/0**	**125/5/0**	**124/6/0**	**128/1/1**	**124/6/0**	**128/2/0**	**122/8/0**	**129/0/1**	**117/12/1**	**129/1/0**	**129/1/0**

Table C.2
Comparison Results in terms of Win/Tie/Loss when using KNN as the Classifier.

Projects	Baselines-KNN																				
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	FULL
Ant-1.7	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0
Camel-1.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Ivy-2.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Jedit-4.0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	9/1/0	8/2/0	9/1/0	8/2/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0
Lucene-2.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	9/1/0
Poi-3.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Synapse-1.2	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	6/4/0	10/0/0	7/3/0	10/0/0	6/4/0	9/1/0	6/4/0	10/0/0	10/0/0
Velocity-1.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xalan-2.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xerces-1.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Apache	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	8/2/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Safe	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Zxing	8/2/0	8/2/0	7/3/0	7/3/0	8/2/0	9/1/0	8/2/0	7/3/0	7/3/0	10/0/0	6/4/0	8/2/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0
Total	**128/2/0**	**128/2/0**	**127/3/0**	**127/3/0**	**126/4/0**	**129/1/0**	**125/5/0**	**126/4/0**	**126/4/0**	**124/6/0**	**126/4/0**	**127/3/0**	**128/2/0**	**120/10/0**	**128/2/0**	**121/9/0**	**128/2/0**	**124/6/0**	**128/2/0**	**125/5/0**	**130/0/0**	**128/2/0**

Table C.3
Comparison Results in terms of Win/Tie/Loss when using LR as the Classifier.

Projects	Baselines-LR																					
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	FULL
Ant-1.7	8/2/0	9/1/0	9/1/0	10/0/0	10/0/0	8/2/0	8/2/0	8/2/0	9/1/0	10/0/0	8/2/0	8/2/0	10/0/0	8/2/0	10/0/0	10/0/0	9/1/0	8/2/0	10/0/0	10/0/0	9/1/0	10/0/0
Camel-1.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	10/0/0
Ivy-2.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Jedit-4.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	9/1/0
Lucene-2.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0
Poi-3.0	7/3/0	6/4/0	4/5/1	6/4/0	6/4/0	9/1/0	5/5/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	6/4/0	10/0/0	8/2/0	10/0/0	10/0/0
Synapse-1.2	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	8/2/0	9/1/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0
Velocity-1.6	9/1/0	9/1/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xalan-2.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xerces-1.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	7/3/0	8/2/0	10/0/0	8/2/0	10/0/0	8/2/0	10/0/0	10/0/0	10/0/0	10/0/0
Apache	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Safe	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Zxing	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/0/1	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Total	**123/7/0**	**123/7/0**	**122/7/1**	**125/5/0**	**126/4/0**	**125/5/0**	**122/8/0**	**128/2/0**	**127/3/0**	**128/1/1**	**127/3/0**	**125/5/0**	**124/6/0**	**124/6/0**	**127/3/0**	**128/2/0**	**126/4/0**	**120/10/0**	**130/0/0**	**127/3/0**	**126/4/0**	**128/2/0**

Table C.4
Comparison Results in terms of Win/Tie/Loss when using NB as the Classifier.

Projects	Baselines-NB																					
	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	FR9	FR10	FR11	FS1	FS2	WS1	WS2	WS3	WS4	WS5	WS6	WS7	WS8	FULL
Ant-1.7	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Camel-1.6	5/5/0	6/4/0	4/6/0	4/6/0	5/5/0	5/5/0	7/3/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	7/3/0	10/0/0	10/0/0	10/0/0	6/4/0	10/0/0	8/2/0	9/1/0	10/0/0
Ivy-2.0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Jedit-4.0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Lucene-2.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0
Poi-3.0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0
Synapse-1.2	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0
Velocity-1.6	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xalan-2.6	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	9/1/0	9/1/0	9/1/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Xerces-1.4	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Apache	8/1/1	9/0/1	8/1/1	9/0/1	8/1/1	9/1/0	9/1/0	9/1/0	9/1/0	9/1/0	9/1/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0
Safe	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	9/1/0	9/0/1	10/0/0	10/0/0	9/1/0	9/1/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0
Zxing	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	9/1/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0
Total	**122/7/1**	**125/4/1**	**120/9/1**	**122/7/1**	**122/7/1**	**123/7/0**	**126/4/0**	**127/3/0**	**124/6/0**	**127/3/0**	**128/2/0**	**126/4/0**	**126/3/1**	**127/3/0**	**129/1/0**	**128/2/0**	**127/3/0**	**126/4/0**	**128/2/0**	**127/3/0**	**129/1/0**	**128/2/0**

References

Bansiya, J., Davis, C.G., 2002. A hierarchical model for object-oriented design quality assessment. IEEE Trans. Softw. Eng. 28 (1), 4–17.

Benjamini, Y., Hochberg, Y., 1995. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J. R. Stat. Soc. Ser. B (Methodol.) 57 (1), 289–300.

Bennin, K.E., Keung, J., Phannachitta, P., Monden, A., Mensah, S., 2017. Mahakil:diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Trans. Softw. Eng. 44 (6), 534–550.

Beume, N., Naujoks, B., Emmerich, M., 2007. Sms-emoa: multiobjective selection based on dominated hypervolume. Eur. J Oper. Res. 181 (3), 1653–1669.

Canfora, G., Lucia, A.D., Penta, M.D., Oliveto, R., Panichella, A., Panichella, S., 2015. Defect prediction as a multiobjective optimization problem. Softw. Test., Verific. Reliab. 25 (4), 426–459.

Catal, C., Diri, B., 2009. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Inf. Sci. 179 (8), 1040–1058.

Chen, X., Shen, Y., Cui, Z., Ju, X., 2017. Applying feature selection to software defect prediction using multi-objective optimization. In: Proceedings of the Annual International Computer Software and Applications Conference, pp. 54–59.

Chen, X., Zhang, D., Zhao, Y., Cui, Z., Ni, C., 2019. Software defect number prediction: unsupervised vs supervised methods. Inf. Softw. Technol. 106, 161–181.

Chen, X., Zhao, Y., Wang, Q., Yuan, Z., 2018. Multi: multi-objective effort-aware just-in-time software defect prediction. Inf. Softw. Technol. 93, 1–13.

Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Trans. Softw. Eng. 20 (6), 476–493.

Coello, C.A.C., Lamont, G.B., Van Veldhuizen, D.A., et al., 2007. Evolutionary Algorithms for Solving Multi-Objective Problems, vol. 5. Springer.

Cover, T.M., Thomas, J.A., 2012. Elements of Information Theory, second edition. John Wiley & Sons, Inc.

Dash, M., Liu, H., Motoda, H., 2000. Consistency based feature selection. In: Proceedings of the Pacific-Asia conference on knowledge discovery and data mining, pp. 98–109.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: nsga-ii. IEEE Trans. Evol. Comput. 6 (2), 182–197.

Durillo, J.J., Nebro, A.J., 2011. Jmetal: a java framework for multi-objective optimization. Adv. Eng. Softw. 42 (10), 760–771.

Durillo, J.J., Nebro, A.J., 2011. Jmetal: a java framework for multi-objective optimization. Adv. Eng. Softw. 42 (10), 760–771.

Gao, K., Khoshgoftaar, T.M., Wang, H., Seliya, N., 2011. Choosing software metrics for defect prediction: an investigation on feature selection techniques. Softw. Pract. Experience 41 (5), 579–606.

Ghotra, B., McIntosh, S., Hassan, A.E., 2015. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proceedings of the International Conference on Software Engineering, pp. 789–800.

Ghotra, B., Mcintosh, S., Hassan, A.E., 2017. A large-scale study of the impact of feature selection techniques on defect classification models. In: Proceedings of the International Conference on Mining Software Repositories, pp. 146–157.

Guo, L., Ma, Y., Cukic, B., Singh, H., 2004. Robust prediction of fault-proneness by random forests. In: Proceedings of International Symposium on Software Reliability Engineering, pp. 417–428.

Hall, M.A., 2000. Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the International Conference on Machine Learning, pp. 359–366.

Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S., 2012. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. Softw. Eng. 38 (6), 1276–1304.

Harman, M., 2010. The relationship between search based software engineering and predictive modeling. In: Proceedings of the International Conference on Predictive Models in Software Engineering, pp. 1:1–1:13.

Harman, M., Mansouri, S.A., Zhang, Y., 2012. Search-based software engineering: trends, techniques and applications. ACM Comput. Surv. 45 (1), 11:1–11:61.

Hassan, A.E., 2009. Predicting faults using the complexity of code changes. In: Proceedings of the International Conference on Software Engineering, pp. 78–88.

He, P., Li, B., Liu, X., Chen, J., Ma, Y., 2015. An empirical study on software defect prediction with a simplified metric set. Inf. Softw. Technol. 59, 170–190.

Henderson-Sellers, B., 1995. Object-Oriented Metrics: Measures of Complexity. Prentice-Hall, Inc.

Herzig, K., Just, S., Zeller, A., 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the International Conference on Software Engineering, pp. 392–401.

Holte, R.C., 1993. Very simple classification rules perform well on most commonly used datasets. Mach. Learn. 11 (1), 63–90.

Hosseini, S., Turhan, B., Mantyla, M., 2017. A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction. Inf. Softw. Technol. 95, 296–312.

Jiarpakdee, J., Tantithamthavorn, C., Hassan, A.E., 2019. The impact of correlated metrics on the interpretation of defect models. IEEE Trans. Softw. Eng. 1.

Jiarpakdee, J., Tantithamthavorn, C., Ihara, A., Matsumoto, K., 2016. A study of redundant metrics in defect prediction datasets. In: Proceedings of the International Symposium on Software Reliability Engineering Workshops, pp. 51–52.

Jiarpakdee, J., Tantithamthavorn, C., Treude, C., 2018. Autospearman: automatically mitigating correlated software metrics for interpreting defect models. In: Proceedings of International Conference on Software Maintenance and Evolution, pp. 92–103.

Jing, X.Y., Wu, F., Dong, X., Xu, B., 2017. An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems. IEEE Trans. Softw. Eng. 43 (4), 321–339.

Jureczko, M., Madeyski, L., 2010. Towards identifying software project clusters with regard to defect prediction. In: Proceedings of the International Conference on Predictive Models in Software Engineering, pp. 9:1–9:10.

Kamei, Y., Shihab, E., 2016. Defect prediction: Accomplishments and future challenges. In: Proceedings of International Conference on Software Analysis, Evolution, and Reengineering, pp. 33–45.

Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N., 2013. A large-scale empirical study of just-in-time quality assurance. IEEE Trans. Softw. Eng. 39 (6), 757–773.

Kannan, S.S., Ramaraj, N., 2010. A novel hybrid feature selection via symmetrical uncertainty ranking based local memetic search algorithm. Knowl Based Syst 23 (6), 580–585.

Khoshgoftaar, T.M., Gao, K., Napolitano, A., 2012. An empirical study of feature ranking techniques for software quality prediction. Int. J. Softw. Eng. Knowl. Eng. 22 (02), 161–183.

Khoshgoftaar, T.M., Gao, K., Seliya, N., 2010. Attribute selection and imbalanced data: Problems in software defect prediction. In: Proceedings of the International Conference on Tools with Artificial Intelligence, pp. 137–144.

Kim, S., Jr., E.J.W., Zhang, Y., 2008. Classifying software changes: clean or buggy? IEEE Trans. Softw. Eng. 34 (2), 181–196.

Kim, S., Zhang, H., Wu, R., Gong, L., 2011. Dealing with noise in defect prediction. In: Proceedings of the International Conference on Software Engineering, pp. 481–490.

Knowles, J.D., Corne, D.W., 2000. Approximating the nondominated front using the pareto archived evolution strategy. Evol. Comput. 8 (2), 149–172.

Kondo, M., Bezemer, C.-P., Kamei, Y., Hassan, A.E., Mizuno, O., 2019. The impact of feature reduction techniques on defect prediction models. Empir. Softw. Eng. 1–39.

Kononenko, I., 1994. Estimating attributes: analysis and extensions of relief. In: Proceedings of European Conference on Machine Learning on Machine Learning, pp. 171–182.

Laradji, I.H., Alshayeb, M., Ghouti, L., 2015. Software defect prediction using ensemble learning on selected features. Inf. Softw. Technol. 58, 388–402.

Lessmann, S., Baesens, B., Mues, C., Pietsch, S., 2008. Benchmarking classification models for software defect prediction: a proposed framework and novel findings. IEEE Trans. Softw. Eng. 34 (4), 485–496.

Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R., Whitehead, E.J., 2013. Does bug prediction support human developers? findings from a Google case study. In: Proceedings of International Conference on Software Engineering, pp. 372–381.

Li, J., He, P., Zhu, J., Lyu, M.R., 2017. Software defect prediction via convolutional neural network. In: Proceedings of the International Conference on Software Quality, Reliability and Security, pp. 318–328.

Liu, H., Setiono, R., 2012. Chi2: feature selection and discretization of numeric attributes. In: Proceedings of the International Conference on Tools with Artificial Intelligence, pp. 388–391.

Liu, M., Miao, L., Zhang, D., 2014. Two-stage cost-sensitive learning for software defect prediction. IEEE Trans. Reliab. 63 (2), 676–686.

Liu, S., Chen, X., Liu, W., Chen, J., Gu, Q., Chen, D., 2014. Fecar: a feature selection framework for software defect prediction. In: Proceedings of Annual Computer Software and Applications Conference, pp. 426–435.

Liu, W., Liu, S., Gu, Q., Chen, J., Chen, X., Chen, D., 2016. Empirical studies of a two-stage data preprocessing approach for software fault prediction. IEEE Trans. Reliab. 65 (1), 38–53.

Liu, W., Liu, S., Gu, Q., Chen, X., Chen, D., 2015. Fecs: a cluster based feature selection method for software fault prediction with noises. In: Proceedings of Annual Computer Software and Applications Conference, pp. 276–281.

Mahmood, Z., Bowes, D., Lane, P.C.R., Hall, T., 2015. What is the impact of imbalance on software defect prediction performance? In: Proceedings of the International Conference on Predictive Models and Data Analytics in Software Engineering, pp. 4:1–4:4.

Martin, R., 1994. Oo design quality metrics. Anal. depend. 12, 151–170.

Menzies, T., Greenwald, J., Frank, A., 2007. Data mining static code attributes to learn defect predictors. IEEE Trans. Softw. Eng. 33 (1), 2–13.

Moser, R., Pedrycz, W., Succi, G., 2008. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the International Conference on Software Engineering, pp. 181–190.

Muthukumaran, K., Rallapalli, A., Murthy, N.L.B., 2015. Impact of feature selection techniques on bug prediction models. In: Proceedings of the India Software Engineering Conference, pp. 120–129.

Nagappan, N., Ball, T., 2005. Use of relative code churn measures to predict system defect density. In: Proceedings of the International Conference on Software Engineering, pp. 284–292.

Nam, J., Fu, W., Kim, S., Menzies, T., Tan, L., 2017. Heterogeneous defect prediction. IEEE Trans. Softw. Eng. PP (99), 1.

Nam, J., Kim, S., 2015. Heterogeneous defect prediction. In: Proceedings of the joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, pp. 508–519.

Nam, J., Pan, S.J., Kim, S., 2013. Transfer defect learning. In: Proceedings of the International Conference on Software Engineering, pp. 382–391.

Nebro, A.J., Durillo, J.J., Luna, F., Alba, E., 2007. Design issues in a multiobjective cellular genetic algorithm. In: Proceedings of International Conference on Evolutionary Multi-Criterion Optimization, pp. 126–140.

Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N., 2012. Multi-layered approach for recovering links between bug reports and fixes. In: Proceedings of the International Symposium on the Foundations of Software Engineering, pp. 63:1–63:11.
Ni, C., Liu, W., Gu, Q., Chen, X., Chen, D., 2017. Fesch: a feature selection method using clusters of hybrid-data for cross-project defect prediction. In: Proceedings of The Annual International Computers, Software and Applications Conference, pp. 51–56.
Ni, C., Liu, W.-S., Chen, X., Gu, Q., Chen, D.-X., Huang, Q.-G., 2017. A cluster based feature selection method for cross-project software defect prediction. J. Comput. Sci. Technol. 32 (6), 1090–1107.
Ozturk, M.M., 2017. Which type of metrics are useful to deal with class imbalance in software defect prediction? Inf. Softw. Technol. 92, 17–29.
Quinlan, J.R., 2014. C4.5: Programs for Machine Learning. Elsevier.
Radjenovic, D., Hericko, M., Torkar, R., Zivkovic, A., 2013. Software fault prediction metrics: a systematic literature review. Inf. Softw. Technol. 55 (8), 1397–1418.
Rahman, F., Devanbu, P., 2013. How, and why, process metrics are better. In: Proceedings of the International Conference on Software Engineering, pp. 432–441.
Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J., Riquelme, J.C., 2014. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, pp. 43:1–43:10.
Shivaji, S., Whitehead, E.J., Akella, R., Kim, S., 2013. Reducing features to improve code change-based bug prediction. IEEE Trans. Softw. Eng. 39 (4), 552–569.
Song, Q., Jia, Z., Shepperd, M., Ying, S., Liu, J., 2011. A general software defect-proneness prediction framework. IEEE Trans. Softw. Eng. 37 (3), 356–370.
Tan, M., Tan, L., Dara, S., Mayeux, C., 2015. Online defect prediction for imbalanced data. In: Proceedings of the International Conference on Software Engineering, pp. 99–108.
Tantithamthavorn, C., Hassan, A.E., 2018. An experience report on defect modelling in practice: Pitfalls and challenges. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice. ACM, pp. 286–295.
Tantithamthavorn, C., Hassan, A.E., Matsumoto, K., 2018. The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. Trans. Softw. Eng. 1.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Ihara, A., Matsumoto, K., 2015. The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proceedings of the International Conference on Software Engineering, pp. 812–823.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2016. Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the International Conference on Software Engineering, pp. 321–332.
Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K., 2017. An empirical comparison of model validation techniques for defect prediction models. IEEE Trans. Softw. Eng. 43 (1), 1–18.
Wang, H., Khoshgoftaar, T., Hulse, J.V., Gao, K., 2011. Metric selection for software defect prediction. Int. J. Softw. Eng.Knowl. Eng. 21 (2), 237–257.
Wang, H., Khoshgoftaar, T.M., Napolitano, A., 2010. A comparative study of ensemble feature selection techniques for software defect prediction. In: Proceedings of the International Conference on Machine Learning and Applications, pp. 135–140.
Wang, S., Ali, S., Yue, T., Li, Y., Liaaen, M., 2016. A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering. In: Proceedings of International Conference on Software Engineering, pp. 631–642.
Wang, S., Liu, T., Tan, L., 2016. Automatically learning semantic features for defect prediction. In: Proceedings of the International Conference on Software Engineering, pp. 297–308.
Wang, S., Yao, X., 2013. Using class imbalance learning for software defect prediction. IEEE Trans. Reliab. 62 (2), 434–443.
Wu, R., Zhang, H., Kim, S., Cheung, S.-C., 2011. Relink: recovering links between bugs and changes. In: Proceedings of the joint meeting of the European Software Engineering Conference and the Symposium on the Foundations of Software Engineering, pp. 15–25.
Xia, X., Lo, D., Pan, S.J., Nagappan, N., Wang, X., 2016. Hydra: massively compositional model for cross-project defect prediction. IEEE Trans. Softw. Eng. 42 (10), 977–998.
Xu, Z., Liu, J., Yang, Z., An, G., Jia, X., 2016. The impact of feature selection on defect prediction performance: An empirical comparison. In: Proceedings of the International Symposium on Software Reliability Engineering, pp. 309–320.
Xu, Z., Xuan, J., Liu, J., Cui, X., 2016. Michac: Defect prediction via feature selection based on maximal information coefficient with hierarchical agglomerative clustering. In: Proceedings of the International Conference on Software Analysis, Evolution, and Reengineering, pp. 370–381.
Yan, M., Fang, Y., Lo, D., Xia, X., Zhang, X., 2017. File-level defect prediction: unsupervised vs. supervised models. In: Empirical Software Engineering and Measurement (ESEM), 2017 ACM/IEEE International Symposium on. IEEE, pp. 344–353.
Yang, X., Tang, K., Yao, X., 2015. A learning-to-rank approach to software defect prediction. IEEE Trans. Reliab. 64 (1), 234–246.
Yu, Q., Jiang, S., Zhang, Y., 2017. A feature matching and transfer approach for cross–company defect prediction. J. Syst. Softw. 132, 366–378.
Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E., 2016. Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the International Conference on Software Engineering, pp. 309–320.
Zhang, Y., Lo, D., Xia, X., Sun, J., 2015. An empirical study of classifier combination for cross-project defect prediction. In: Proceedings of the Annual Computer Software and Applications Conference, pp. 264–269.
Zitzler, E., Thiele, L., 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. 3 (4), 257–271.

Chao Ni received his B.S. degree in computer science from Nantong University, Nantong, in 2014. Then he received his M.S. degree in computer science from Nanjing University, Nanjing, in 2017. Now he is a Ph.D. candidate of State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests are mainly in software defect prediction and machine learning.

Xiang Chen received his B.S. degree in the School of Management from Xi'an Jiaotong University, Xi'an, in 2002. Then he received his M.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, in 2008 and 2011, respectively. Now he is an associate professor in the School of Computer Science and Technology, Nantong University, Nantong. His research interests are mainly in software testing, such as software defect prediction, combinatorial testing, regression testing, and software fault localization. He has published over 40 papers in referred journals and conferences.

Fangfang Wu received her B.S. degree in computer science from Nantong University, Nantong, in 2014. Now she is a Master candidate of State Key Laboratory for Novel Software Technology and the Department of Computer Science and Technology, Nanjing University, Nanjing. Her research interests are mainly in software defect prediction and machine learning.

Yuxiang Shen is a bachelor candidate of school of computer science and technology from Nantong University, Nantong, in 2014. His research interests are mainly in software defect prediction and machine learning.

Qing Gu received his Ph.D. degree in computer science from Nanjing University, Nanjing. He is a professor of the State Key Laboratory of Novel Software Technology, and the Department of Computer Science and Technology, Nanjing University, Nanjing. His research interests include software testing, quality and process improvement, software maintenance and evolution, and complex network.