# A Multi-Objective Learning Method for Building Sparse Defect Prediction Models

1st Xin Li
*School of Data and Computer Science*
*Sun Yat-Sen University*
Guangzhou, China
lixin49@mail2.sysu.edu.cn

2nd Xiaoxing Yang
*College of Big Data and Internat*
*Shenzhen Technology University*
Shenzhen, China
yangxiaoxing@sztu.edu.cn

3rd Jianmin Su
*School of Intelligent Manufacturing*
*Wuyi University*
Jiangmen, China
jackysura@163.com

4th Wushao Wen
*School of Data and Computer Science*
*Sun Yat-Sen University*
Guangzhou, China
wenwsh@mail.sysu.edu.cn

***Abstract*—Software defect prediction constructs a model from the previous version of a software project to predict defects in the current version, which can help software testers to focus on software modules with more defects in the current version. Most existing methods construct defect prediction models through minimizing the defect prediction error measures. Some researchers proposed model construction approaches that directly optimized the ranking performance in order to achieve an accurate order. However, the model complexity is also considered. Therefore, defect prediction can be seen as a multi-objective optimization problem and should be solved by multi-objective approaches. And hence, in this paper, we employ an existing multi-objective evolutionary algorithm and propose a new multi-objective learning method based on it, to construct defect prediction models by simultaneously optimizing more than one goal. Experimental results over 30 sets of cross-version data show the effectiveness of the proposed multi-objective approaches.**

***Index Terms*—software defect prediction, multi-objective learning method, ranking task, cross-version defect prediction**

## I. Introduction

Software defect prediction employs software metrics (also referred to as features or attributes) of software modules (such as classes, files, packages, etc) to construct defect prediction models, in order to predict defect information of new software modules [16]. It can support software testing activities, by helping software testers to focus on software modules with defects, in order to timely detect and repair defects before releasing [14]. Cross-version defect prediction constructs a model from the previous version of a software project to predict defects in the current version, so it can help software testers to focus on software modules with more defects in the current version [12].

There are two most frequently investigated goals of software defect prediction: to predict an order of software modules based on the predicted defect numbers (which we call as ranking task) [11], [16], and to predict whether a software module has defects or not (which we call as classification task) [9]. In this paper, the former goal is studied. The detailed process is as follows [16]: first, data is obtained from software modules according to metrics such as lines of code, and response for a class [5]; secondly, a model is constructed based on the data from modules with known defect numbers; and finally, the model is used to predict defects of software modules with unknown defect numbers, so that an order of these modules based on the predicted defect numbers is obtained.

Most existing methods, such as linear regression [10], negative binomial regression [11], recursive partitioning, Bayesian additive regression trees, and random forest [13], construct defect prediction models by maximum likelihood estimation or least squares, focusing on the fitting of each sample. Considering that prediction models constructed by these methods might fail to give an accurate order (because of the difficulty to predict the exact number of defects in a software module due to noisy data), Yang et al. [15] proposed an effective learning-to-rank method, which constructed defect prediction models by directly optimizing the ranking performance.

However, in some cases, the model complexity is also considered. That is, when performance is the same, a simple model is better than a complex model. Furthermore, in some cases, both a good order of software modules and a high prediction accuracy are desired. In other words, an accurate order is needed to help software testers to allocate testing resources (for instance, more testing resources for modules with more defects), and the predicted defect numbers should be reasonable or meaningful (for example, if true defect number is 3, a predicted defect number around 3 might be better than 30000). Therefore, defect prediction for the ranking task can be seen as a multi-objective optimization problem. And hence, in this paper, we investigate the application of multi-objective learning methods to construct defect prediction models by simultaneously optimizing both ranking performance and model complexity.

This work is supported by National Natural Science Foundation of China (Grants No. 61602534) and Fundamental Research Funds for the Central Universities (No.17lgpy122). (Corresponding author: Xiaoxing Yang )

The rest of this paper is organized as follows. Section 2 presents related work. In Section 3, an overview of the investigated multi-objective optimization methods are described, including the original one and the proposed one. In Section 4, we detail the experimental methodologies. Experimental results are reported in Section 5. Section 6 presents the threats to validity, and Section 7 draws the conclusions.

## II. Related Work

Software defect prediction for the ranking task can help software testers to focus on software modules with more defects, which attracts many researchers. For example, Gao et al. [4] compared eight count models, and they concluded that zero-inflated negative binomial regression, hurdle negative binomial regression and hurdle Poisson regression with threshold 2 were more effective. Weyuker et al. [13] compared four approaches (negative binomial regression, random forest, recursive partitioning and Bayesian additive regression trees) to predict the number of defects in software modules, and they concluded that negative binomial regression and random forest models performed better than the other two models. Yang and Wen [16] investigated two penalized regression methods for constructing prediction models, and they concluded that both penalized regression performed better than linear regression and negative binomial regression for cross-version defect prediction because they could handle the multi-collinearity problems in the datasets.

The above-mentioned methods construct models by minimizing prediction error measures such as average, relative, or mean square errors, which might not provide a direct insight into the ranking performance [7]. Therefore, some researchers employed evolutionary algorithm to directly optimize the ranking performance. For example, Khoshgoftaar et al. [7] used a GP-based approach to optimize tree size and performances for the 95%, 90%, 80%, and 70% cutoff percentile values to obtain prediction models, and they concluded better performance of the proposed models. Yang et al. [15] proposed a learning-to-rank approach to directly optimize the ranking performance measure (fault-percentile-average), and the experimental results showed that the learning-to-rank approach and random forest were better than other methods to construct defect prediction models for the ranking task.

Realizing that single-objective approaches might be insufficient for solving software defect prediction problems, some researchers proposed multi-objective learning algorithms to construct models [8]. For example, in 2004, Khoshgoftaar et al. [7] used a GP-based approach to optimize five objectives, in order to obtain a better ranking. To be noted, the five objectives were in order, i.e., the first objective had priority over other objectives, and then came the second objective, etc. In 2007, Khoshgoftaar et al. [6] proposed a multi-objective optimization method to construct models to fit different applications, which simultaneously optimized a balance of classification accuracy, available resources and model simplicity. In order to deal with the imbalanced problem in software defect prediction, Carvalho et al. [1] applied multi-objective particle swarm optimization to obtain a set of rules with sensitivity and specificity as the objectives, and these rules composed a classifier. Shukla et al. [12] compared a multi-objective logistic regression method with four traditional machine learning algorithms for cross-version defect prediction for the classification task, and they concluded better performance of the multi-objective logistic regression. Li et al. [8] compared a multi-objective approach that optimized two objectives and a single-objective approach that directly optimized a trade-off of the two objectives for classification software defect prediction, and they pointed out that when the trade-off of objectives was known, it might be better to choose single-objective approaches to directly optimize the trade-off, instead of using multi-objective approaches.

## III. Multi-Objective Optimization

According to related work, existing studies about software defect prediction for the ranking task constructed software defect prediction models by optimizing prediction error measures, or the ranking performance, ignoring model complexity. In some cases, a simple model is desired, and both a good order to help testers to allocate testing resources, and a good predicted accuracy as reference are desired. Therefore, defect prediction can be seen as a multi-objective optimization problem, and hence we employ a multi-objective evolutionary algorithm to solve it.

In this section, we firstly introduce the involved objectives (fitness functions). Subsequently, the employed multi-objective evolutionary approach and the revised method are described.

### *A. Fitness Functions*

As mentioned above, all three objectives (a good order, a high prediction accuracy, and a simple model) might be desired in some cases. Hence, three fitness functions are described in this subsection.

In this study, we adopt fault-percentile-average (FPA) [13] as the ranking performance measure. Considering $m$ modules $f_1, f_2, \ldots, f_m$, listed in increasing order of predicted defect number, $s_i$ as the actual defect number in the module $f_i$, and $s$=$s_1$+$s_2$+...+$s_m$ as the total number of defects in all modules, the proportion of actual defects in the top $t$ predicted modules (i.e. top $t$ modules predicted to have most defects) to the whole defects is $\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$. Then FPA is defined as follows [13]:

$$\frac{1}{m}\sum_{t=1}^{m}\frac{1}{s}\sum_{i=m-t+1}^{m} s_i$$

FPA is an average of the proportions of actual defects in the top $i$ ($i$: $1$ to $m$) predicted modules. Larger FPA means better ranking performance, and can help allocate testing resources better on the whole.

Mean square error (MSE) is adopted as prediction accuracy in this paper. Considering $m$ modules $f_1, f_2, \ldots, f_m$, $s_i$ as the actual defect number in the module $f_i$, and $p_i$ as the predicted defect number in the module $f_i$, MSE is simply computed as follows:

$$\frac{1}{m}\sum_{i=1}^{m}(p_i - s_i)^2$$

Smaller MSE means better prediction accuracy.

In this paper, we adopt a linear model as a base predictor for multi-objective approaches. Given n training metric vectors of software modules $\mathbf{x_i}= (x_{i,1}, x_{i,2}, \cdots\cdots, x_{i,d})$ (i: 1 to n, $\mathbf{x_i}$ is the vector of the $i^{th}$ software module, $x_{i,j}$ is the $j^{th}$ metric value in vector $\mathbf{x_i}$, and d is number of metrics), which compose a metric matrix $\mathbf{X}$, and corresponding defect number $y_i$, the predicted defect number of testing module $f(\mathbf{z_i})$ is computed as follows:

$$f(\mathbf{z_i}) = \sum_{j=1}^{d}\alpha_j z_{i,j} + \alpha_0 \tag{1}$$

where $\alpha_j$s (including $\alpha_0$) are the corresponding parameters obtained by training. Once $\alpha_j$s are fixed, the model is learned. If more $\alpha_j$s equal to zero, it costs less computation, and the model is simpler. Therefore, we use the number of nonzero $\alpha_j$s to measure the model complexity, which is denoted as NNZ. Smaller NNZ means a simpler model.

## B. Multi-Objective Approaches

Multi-objective approaches are used to construct a set of Pareto-optimal models. In this study, we adopt a linear model as a base predictor, and Non-dominated Sorting Genetic Algorithm II (NSGA-II) [2] as the multi-objective learning algorithm. As mentioned in the above subsection, the predicted defect number of testing module $f(\mathbf{z_i})$ is computed as $f(\mathbf{z_i}) = \sum_{j=1}^{d}\alpha_j z_{i,j} + \alpha_0$. The goal of NSGA-II is to find out a set of $\alpha_j$s, which optimizes two or three objectives (FPA, MSE, or/and NNZ). The detailed process of NSGA-II is as follows (more details in [2]).

- Initialize: Set population size (number of solutions) to N, and randomly generate N solutions that compose population $P_0$. Sort the solution in $P_0$ using fast non-dominated sorting, and compute the non-dominated rank value of each solution.
  Set the generation number $t = 0$.
- While $t < t_{max}$ (maximal generation number, used to decide the termination condition), do
  - Use binary tournament selection to select individuals from $P_t$ for crossover and mutation to generate the offspring population $Q_t$.
  - Combine solutions in $P_t$ and $Q_t$ to get $R_t = P_t U Q_t$.
  - Sort $R_t$ based on non-domination rank value and crowding distance, and select N elitist individuals to compose the new parent population $P_{t+1}$.
- EndWhile and Return Pareto-optimal solutions in $P_{t+1}$.

Because it is difficult for NSGA-II to achieve zero $\alpha_j$s using crossover and mutation operation, we revise the process of NSGA-II by setting part of parameters to zero randomly. The detailed process of the revised NSGA-II is as follows.

- Initialize: Set population size (number of solutions) to N, and randomly generate N solutions that compose population $P_0$. Sort the solution in $P_0$ using fast non-dominated sorting, and compute the non-dominated rank value of each solution.
  Set the generation number $t = 0$.
  Set the certain ratio to $Ratio$, which is from 0 to 1.
- While $t < t_{max}$ (maximal generation number, used to decide the termination condition), do
  - Use binary tournament selection to select individuals from $P_t$ for crossover and mutation to generate the offspring population $Q_t$.
  - for every solution $q = (\alpha_0, \alpha_1, ..., \alpha_d)$ in $Q_t$

```
for $i = 0$ to $d$
    Produce a number $randi$ from 0 to 1.
    If $randi < Ratio$, set $\alpha_i = 0$
```

  - Combine solutions in $P_t$ and $Q_t$ to get $R_t = P_t U Q_t$.
  - Sort $R_t$ based on non-domination rank value and crowding distance, and select N elitist individuals to compose the new parent population $P_{t+1}$.
- EndWhile and Return Pareto-optimal solutions in $P_{t+1}$.

## IV. Experimental Setup

In this section, we detail the research questions, data sets, and implementation respectively.

## A. Research Questions

In this paper, we mainly investigate the ability of multi-objective learning methods to build sparse defect prediction models. Since we adopt a linear model as the base predictor, we compare linear models optimized by multi-objective learning methods with linear models achieved by other methods, including ordinary linear regression [13], learning-to-ranking approach [15], lasso regression and ridge regression [16]. Therefore, the main research questions in this paper are as follows.

* **RQ1**: Compared with linear models achieved by a single-objective evolutionary algorithm (the learning-to-ranking approach) [15], how do linear models obtained by multi-objective methods perform?
* **RQ2**: Compared with other linear mdoels (linear regression [13], lasso regression and ridge regression [16]), how do our linear models perform?

In essence, both research questions evaluate whether multi-objective methods bring benefits. The learning-to-rank method constructs defect prediction models by directly optimizing the ranking performance, which has shown its effectiveness in Yang et al. work [15]. Linear regression, lasso regression and ridge regression focus on the fitting of each sample because they construct models by minimizing mean square errors. Their good ranking performance for software defect prediction has also been shown [13], [16]. Therefore, we respectively compare the multi-objective methods with these two kinds of algorithms. Different from previous work [13], [15], [16], because we investigate the ability of multi-objective learning methods to build sparse defect prediction models, we consider not only the ranking performance, but also the model simplicity.

### B. Datasets

In order to facilitate others to reproduce results, 41 data sets from 11 open-source projects in PROMISE repository [12], [15][1] are used. Following Yang et al.'s work [15], we use the defect number as the dependent variable.

The characteristics of these experimental datasets are shown in Table I. The column of 'faulty modules' records the number of modules having defects (with the percentages of faulty modules in the subsequent brackets), the column of 'range of defects' records ranges of defect numbers in the corresponding datasets, and the column of 'total defects' records the total number of defects in all modules of the corresponding datasets.

TABLE I
EXPERIMENTAL DATASETS

| Datasets name | module number | metric number | faulty modules | range of defects | total defects |
|---|---|---|---|---|---|
| ant-1.3 | 125 | 20 | 20(16%) | [0,3] | 33 |
| ant-1.4 | 178 | 20 | 40(22.5%) | [0,3] | 47 |
| ant-1.5 | 293 | 20 | 32(10.9%) | [0,2] | 35 |
| ant-1.6 | 351 | 20 | 92(26.2%) | [0,10] | 184 |
| ant-1.7 | 745 | 20 | 166(22.3%) | [0,10] | 338 |
| lucene-2.0 | 195 | 20 | 91(46.7%) | [0,22] | 268 |
| lucene-2.2 | 247 | 20 | 144(58.3%) | [0,47] | 414 |
| lucene-2.4 | 340 | 20 | 203(59.7%) | [0,30] | 632 |
| xalan-2.4 | 723 | 20 | 110(15.2%) | [0,7] | 156 |
| xalan-2.5 | 803 | 20 | 387(48.2%) | [0,9] | 531 |
| xalan-2.6 | 885 | 20 | 411(46.4%) | [0,9] | 625 |
| xalan-2.7 | 909 | 20 | 898(98.8%) | [0,8] | 1213 |
| xerces-init | 162 | 20 | 77(47.5%) | [0,11] | 167 |
| xerces-1.2 | 440 | 20 | 71(16.2%) | [0,4] | 115 |
| xerces-1.3 | 453 | 20 | 69(15.2%) | [0,30] | 193 |
| xerces-1.4 | 588 | 20 | 437(74.3%) | [0,62] | 1596 |
| camel-1.0 | 339 | 20 | 13(3.8%) | [0,2] | 14 |
| camel-1.2 | 608 | 20 | 216(35.5%) | [0,28] | 522 |
| camel-1.4 | 872 | 20 | 145(16.6%) | [0,17] | 335 |
| camel-1.6 | 965 | 20 | 188(19.5%) | [0,28] | 500 |
| ivy-1.1 | 111 | 20 | 63(56.8%) | [0,36] | 233 |
| ivy-1.4 | 241 | 20 | 16(6.6%) | [0,3] | 18 |
| ivy-2.0 | 352 | 20 | 40(11.4%) | [0,3] | 56 |
| synapse-1.0 | 157 | 20 | 16(10.2%) | [0,4] | 21 |
| synapse-1.1 | 222 | 20 | 60(27.0%) | [0,7] | 99 |
| synapse-1.2 | 256 | 20 | 86(33.6%) | [0,9] | 145 |
| velocity-1.4 | 196 | 20 | 147(75.0%) | [0,7] | 210 |
| velocity-1.5 | 214 | 20 | 142(66.4%) | [0,10] | 331 |
| velocity-1.6 | 229 | 20 | 78(34.1%) | [0,12] | 190 |
| jedit-3.2 | 272 | 20 | 90(33.1%) | [0,45] | 382 |
| jedit-4.0 | 306 | 20 | 74(24.2%) | [0,23] | 226 |
| jedit-4.1 | 312 | 20 | 79(25.3%) | [0,17] | 217 |
| jedit-4.2 | 367 | 20 | 48(13.1%) | [0,10] | 106 |
| jedit-4.3 | 492 | 20 | 11(2.2%) | [0,2] | 12 |
| log4j-1.0 | 135 | 20 | 34(25.2%) | [0,9] | 61 |
| log4j-1.1 | 109 | 20 | 37(33.9%) | [0,9] | 86 |
| log4j-1.2 | 205 | 20 | 189(92.2%) | [0,10] | 498 |
| poi-1.5 | 237 | 20 | 141(59.5%) | [0,20] | 342 |
| poi-2.0 | 314 | 20 | 37(11.8%) | [0,2] | 39 |
| poi-2.5 | 385 | 20 | 248(64.4%) | [0,11] | 496 |
| poi-3.0 | 442 | 20 | 281(63.6%) | [0,19] | 500 |

These specific twenty metrics can be found Jureczko et al. 's work [5].

### C. Implementation

In order to simulate the actual situation, cross-version defect prediction is adopted. That is, software defect prediction models constructed according to one version are used to predict defects of the next version.

According to the research questions, we need to implement the learning-to-ranking approach, linear regression, ridge regression, lasso regression, and the two multi-objective approaches. The involved objectives included fault-percentile-average (FPA), mean square error (MSE) and number of nonzero (NNZ), which are described in Section III. All methods are implemented in Python. Specifically, the two multi-objective approaches, NSGA-II and the revised NSGA-II are implemented based on Geatpy2, which is a genetic and evolutionary algorithm toolbox for Python with high performance. The parameters for the multi-objective approach are simply set like the learning-to-rank approach [15]. The feasible solution space is set as $\Omega = \prod_{i=1}^{d}[-20, 20]$, and the population size and maximal generation are set to 100. The $Ratio$ of our revised NSGA-II is set as 0.2.

Besides, linear regression, ridge regression and lasso regression models are implemented based on scikit-learn, which is a popular software machine learning library for the python programming language. Default parameters are adopted for linear regression. In ridge regression model, the value of alpha has an impact on the performance of models. Thus, we use RidgeCV with alpha values ranges from 0 to 1000 at 0.1 interval, to build ridge regression models with a relatively suitable alpha. And in lasso regression model, the parameters are optimized by least angle regression. Like Ridge regression, the parameter alpha is also important. Thus, we use LassoLarsCV, which set the alpha parameter by 10-fold cross-validation to build lasso regression model.

## V. EXPERIMENTAL RESULTS

The experimental results are reported according to the research questions: RQ1 and RQ2. That is, we respectively compare the multi-objective methods with the single-objective evolutionary algorithm (the learning-to-rank approach), and other classic linear models.

### A. RQ1: Multi-Objective VS. Learning-to-Rank

In this subsection, we compare the multi-objective approaches (original NSGA-II and revised NSGA-II) with the learning-to-rank approach over 30 sets of cross-version data. The multi-objective approaches optimize three objectives (fault-percentile-average (FPA), mean square error (MSE), and number of nonzero (NNZ)) simultaneously, and the learning-to-rank approach optimizes FPA. For both multi-objective and single-objective algorithms, all Pareto-Optimal models (all optimal models for single-objective method) in final population over previous version of data are used to predict the next version, so there may be more than one result for each method. Because there are a total of three objectives and the learning-to-rank approach only optimizes FPA, in order to show results more intuitively, we compare their results using different figures (two figures record FPA and MSE, and the other two record FPA and NNZ).

[1]http://openscience.us/repo/defect/ck/

The compared results of FPA and MSE are shown in Figures 1 and 2, and the compared results of FPA and NNZ are shown in Figures 3 and 4. Since the MSE values of the learning-to-rank approach are very large compared with multi-objective methods, we use $log(MSE)$ (natural logarithm based on $e$) instead of MSE as the $y$ axis.

In Figures 1 to 4, over most sets of data, there exist models by revised NSGA-II, which dominate (achieve both smaller MSE/NNZ and larger FPA than) the learning-to-rank models. This indicates the good performance of revised NSGA-II.

The MSE values of original NSGA-II models are between the learning-to-rank models and models achieved by revised NSGA-II. NNZ values of models obtained by both the learning-to-rank approach and original NSGA-II are 21, which means no zero parameters for these models. Although original NSGA-II optimizes three objectives, but it is really difficult for it to achieve zero parameters using crossover and mutation operation. Nevertheless, it can improve MSE compared with single-objective approach.

As a whole, the revised NSGA-II can achieve good performance according to all three objectives, and it can obtain models dominating models by NSGA-II and the learning-to-rank approach over most sets of data. Considering the advantage of obtaining diverse models simultaneously for different applications, multi-objective methods perform well compared with the single-objective approach, and the revised NSGA-II performs best.

## B. RQ2: Multi-Objective VS. Other Linear Models

In the above subsection, we mentioned that multi-objective methods have the advantages of obtaining multiple models simultaneously for different applications. However, as pointed by Li et al. [8], for each specific application, only one model of Pareto-optimal is desired. Therefore, in this subsection, we choose only one model for each method and compare their performance over 30 sets of cross-version data. To be specific, for multi-objective methods and the learning-to-rank approach, we choose the model with largest weighted target (in this implementation, we use (4*FPA-2NNZ-MSE) with normalized FPA/NNZ/MSE values as the weighted target) from Pareto-optimal training models to predict the testing data (the next version of data). For linear regression, ridge regression and lasso regression methods, they obtain only one training model, so we can use it directly to predict the testing data. The compared results are shown in Table II.

From Table II, we can see that lasso regression and the revised multi-objective approach can achieve simpler models, while other approaches have no zero parameter. The lasso shrinks the ordinary least-squares estimator towards zero and potentially sets $\alpha_i$ to zero for some $i$ [3], so it may build sparse defect prediction models. The revised multi-objective approach, which is modified to make more zero parameters, can achieve simpler models than lasso regression (having more zero parameters). Therefore, when model complexity is considered, we can use the revised multi-objective approach to construct sparse defect prediction models.

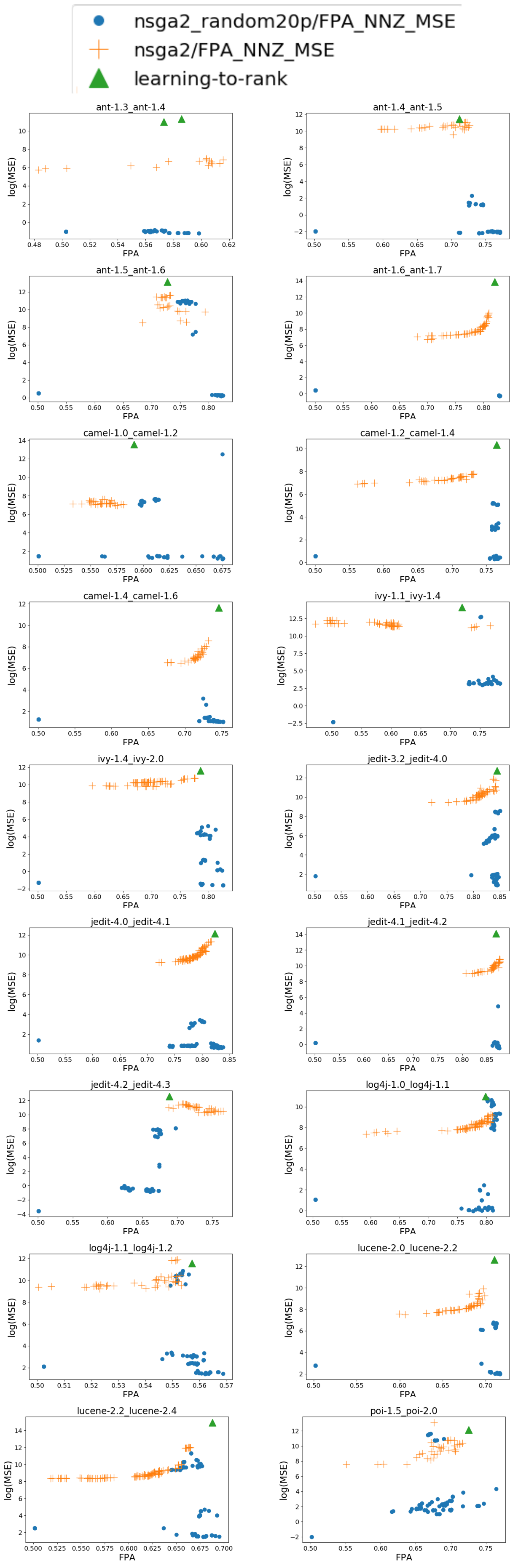

Fig. 1. Multi-Objective VS. Learning-to-Rank According to FPA and MSE

TABLE II
RESULTS OF ALL LINEAR MODELS

| *filename* | nsga2 | | | revised-nsga2 | | | learning-to-rank | | | lasso regression | | | ridge regression | | | linear regression | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *target* | *fpa* | *mse* | *nnz* | *fpa* | *mse* | *nnz* | *fpa* | *mse* | *nnz* | *fpa* | *mse* | *nnz* | *fpa* | *mse* | *nnz* | *fpa* | *mse* | *nnz* |
| *ant*-1.3-1.4 | 0.6035 | 9.4e+02 | 21 | 0.5981 | 0.3 | 1 | 0.573 | 6.0e+04 | 21 | 0.5781 | 0.3 | 4 | 0.6018 | 0.4 | 21 | 0.5412 | 0.5 | 21 |
| *ant*-1.4-1.5 | 0.7022 | 4.7e+04 | 21 | 0.762 | 0.1 | 2 | 0.7117 | 9.5e+04 | 21 | 0.7695 | 0.1 | 5 | 0.7591 | 0.1 | 21 | 0.7178 | 0.2 | 21 |
| *ant*-1.5-1.6 | 0.7108 | 3.9e+04 | 21 | 0.8122 | 1.4 | 2 | 0.728 | 5.2e+05 | 21 | 0.817 | 1.3 | 8 | 0.8077 | 1.3 | 21 | 0.7838 | 1.3 | 21 |
| *ant*-1.6-1.7 | 0.8046 | 6.0e+03 | 21 | 0.8274 | 0.8 | 1 | 0.8196 | 1.1e+06 | 21 | 0.8283 | 0.7 | 4 | 0.8218 | 0.7 | 21 | 0.8136 | 0.8 | 21 |
| *camel*-1.0-1.2 | 0.5723 | 1.4e+03 | 21 | 0.6749 | 3.2 | 2 | 0.5912 | 7.5e+05 | 21 | 0.5008 | 4.3 | 1 | 0.6098 | 4.2 | 21 | 0.6123 | 4.2 | 21 |
| *camel*-1.2-1.4 | 0.7085 | 1.7e+03 | 21 | 0.7628 | 1.7 | 1 | 0.765 | 3.2e+04 | 21 | 0.7593 | 1.7 | 5 | 0.6949 | 2.7 | 21 | 0.6772 | 3.1 | 21 |
| *camel*-1.4-1.6 | 0.7171 | 1.3e+03 | 21 | 0.7391 | 3.4 | 1 | 0.7461 | 1.2e+05 | 21 | 0.7544 | 3.0 | 6 | 0.7489 | 2.9 | 21 | 0.7458 | 2.9 | 21 |
| *ivy*-1.1-1.4 | 0.6005 | 9.0e+04 | 21 | 0.7697 | 22.1 | 1 | 0.7199 | 1.3e+06 | 21 | 0.7683 | 125.4 | 9 | 0.7732 | 147.8 | 21 | 0.7905 | 127.9 | 21 |
| *ivy*-1.4-2.0 | 0.7031 | 3.0e+04 | 21 | 0.8066 | 0.2 | 2 | 0.7856 | 1.1e+05 | 21 | 0.7481 | 0.2 | 4 | 0.506 | 0.3 | 21 | 0.4351 | 0.3 | 21 |
| *jedit*-3.2-4.0 | 0.8415 | 6.2e+04 | 21 | 0.8398 | 4.0 | 1 | 0.8447 | 3.5e+05 | 21 | 0.8382 | 6.8 | 15 | 0.8353 | 7.6 | 21 | 0.8354 | 7.6 | 21 |
| *jedit*-4.0-4.1 | 0.7971 | 2.9e+04 | 21 | 0.8216 | 2.0 | 1 | 0.8231 | 1.9e+05 | 21 | 0.7947 | 1.5 | 15 | 0.804 | 1.6 | 21 | 0.804 | 1.6 | 21 |
| *jedit*-4.1-4.2 | 0.8669 | 2.4e+04 | 21 | 0.864 | 1.2 | 1 | 0.8686 | 1.3e+06 | 21 | 0.8805 | 2.6 | 21 | 0.8826 | 2.5 | 21 | 0.8826 | 2.5 | 21 |
| *jedit*-4.2-4.3 | 0.7166 | 7.9e+04 | 21 | 0.6651 | 0.5 | 1 | 0.6885 | 2.9e+05 | 21 | 0.6506 | 0.7 | 21 | 0.6479 | 0.7 | 21 | 0.6506 | 0.7 | 21 |
| *log4j*-1.0-1.1 | 0.794 | 4.4e+03 | 21 | 0.7568 | 1.2 | 1 | 0.7984 | 6.1e+04 | 21 | 0.8008 | 1.0 | 12 | 0.8033 | 1.0 | 21 | 0.7786 | 1.0 | 21 |
| *log4j*-1.1-1.2 | 0.5484 | 2.8e+04 | 21 | 0.5607 | 4.4 | 2 | 0.5572 | 1.0e+05 | 21 | 0.5622 | 8.8 | 8 | 0.5656 | 11.3 | 21 | 0.5671 | 9.2 | 21 |
| *lucene*-2.0-2.2 | 0.6908 | 6.0e+03 | 21 | 0.7162 | 8.0 | 1 | 0.7103 | 3.1e+05 | 21 | 0.7193 | 6.8 | 9 | 0.7191 | 6.9 | 21 | 0.7188 | 6.9 | 21 |
| *lucene*-2.2-2.4 | 0.6415 | 1.2e+04 | 21 | 0.6951 | 4.5 | 1 | 0.6881 | 3.1e+06 | 21 | 0.6942 | 5.9 | 6 | 0.6557 | 14.1 | 21 | 0.6358 | 11.2 | 21 |
| *poi*-1.5-2.0 | 0.6995 | 4.8e+04 | 21 | 0.7009 | 16.1 | 2 | 0.7255 | 1.9e+05 | 21 | 0.7086 | 4.4 | 5 | 0.6933 | 5.5 | 21 | 0.6623 | 6.8 | 21 |
| *poi*-2.0-2.5 | 0.5134 | 3.6e+04 | 21 | 0.4543 | 28.9 | 2 | 0.5024 | 1.3e+05 | 21 | 0.5013 | 3.3 | 1 | 0.4727 | 3.2 | 21 | 0.4766 | 3.3 | 21 |
| *poi*-2.5-3.0 | 0.6476 | 4.4e+04 | 21 | 0.6869 | 4.9 | 2 | 0.6603 | 1.3e+05 | 21 | 0.7015 | 2.7 | 2 | 0.6891 | 2.4 | 21 | 0.6725 | 2.5 | 21 |
| *synapse*-1.0-1.1 | 0.6426 | 5.0e+03 | 21 | 0.7105 | 0.8 | 1 | 0.6391 | 6.7e+04 | 21 | 0.7387 | 0.9 | 6 | 0.7236 | 0.9 | 21 | 0.6377 | 0.9 | 21 |
| *synapse*-1.1-1.2 | 0.6616 | 3.5e+03 | 21 | 0.6534 | 1.1 | 2 | 0.6231 | 2.3e+06 | 21 | 0.681 | 0.9 | 6 | 0.6749 | 0.9 | 21 | 0.6623 | 1.0 | 21 |
| *velocity*-1.4-1.5 | 0.6095 | 1.8e+04 | 21 | 0.6213 | 5.0 | 3 | 0.6476 | 3.4e+05 | 21 | 0.6202 | 4.6 | 20 | 0.6212 | 4.8 | 21 | 0.6195 | 4.8 | 21 |
| *velocity*-1.5-1.6 | 0.7437 | 6.5e+03 | 21 | 0.7125 | 3.1 | 2 | 0.7602 | 2.8e+05 | 21 | 0.737 | 2.6 | 9 | 0.7465 | 2.8 | 21 | 0.7466 | 2.9 | 21 |
| *xalan*-2.4-2.5 | 0.6334 | 3.3e+04 | 21 | 0.6333 | 1.0 | 1 | 0.6475 | 1.1e+06 | 21 | 0.6333 | 0.9 | 4 | 0.6468 | 0.9 | 21 | 0.6432 | 0.9 | 21 |
| *xalan*-2.5-2.6 | 0.6646 | 1.5e+04 | 21 | 0.6615 | 23.5 | 3 | 0.66 | 7.5e+05 | 21 | 0.6508 | 0.8 | 19 | 0.6529 | 0.8 | 21 | 0.6493 | 0.8 | 21 |
| *xalan*-2.6-2.7 | 0.5673 | 6.8e+04 | 21 | 0.5699 | 1.5 | 1 | 0.5689 | 7.2e+05 | 21 | 0.5588 | 0.9 | 6 | 0.5538 | 0.9 | 21 | 0.5533 | 0.9 | 21 |
| *xerces-init*-1.2 | 0.7025 | 2.2e+04 | 21 | 0.6046 | 38.5 | 3 | 0.6355 | 5.9e+05 | 21 | 0.5071 | 1.3 | 5 | 0.5699 | 1.5 | 21 | 0.6038 | 2.3 | 21 |
| *xerces*-1.2-1.3 | 0.6687 | 4.5e+03 | 21 | 0.6165 | 4.5 | 5 | 0.717 | 9.9e+04 | 21 | 0.7137 | 3.5 | 5 | 0.6986 | 3.2 | 21 | 0.6728 | 3.2 | 21 |
| *xerces*-1.3-1.4 | 0.675 | 3.2e+03 | 21 | 0.74 | 29.4 | 2 | 0.7142 | 7.0e+04 | 21 | 0.7524 | 26.0 | 4 | 0.6767 | 27.9 | 21 | 0.6711 | 27.8 | 21 |
| *average* | 0.6816 | 2.6e+04 | 21 | 0.7013 | 7.2 | 1.7 | 0.6974 | 5.6e+05 | 21 | 0.6990 | 7.5 | 8.2 | 0.6886 | 8.7 | 21 | 0.6754 | 8.0 | 21 |

According to average MSE, the revised multi-objective approach can achieve more accurate models than the original multi-objective approach and the learning-to-rank approach, and it can achieve models that are comparable to models by linear regression, ridge regression and lasso regression. Nevertheless, the best method for each set of data might be different. For example, for "jedit-4.1_jedit-4.2", revised NSGA-II achieves best MSE, while ridge regression achieves minimum MSE for "poi-2.0_poi-2.5". According to average FPA, the revised multi-objective approach performs as well as other linear models. Similarly, the best method for each set of data might be also different.

As a whole, the revised NSGA-II can construct sparse linear models, compared with other linear models. Furthermore, the revised multi-objective approach performs as well as the original multi-objective approach and other methods according to FPA over most sets of data. In some cases such as "ivy-1.4_ivy-2.0", they even achieve better results. According to MSE, the revised multi-objective approach performs better than the original multi-objective approach and the learning-to-rank approach, and performs as well as linear regression, lasso regression and ridge regression. These imply the good performance of the revised multi-objective methods.

## VI. THREATS TO VALIDITY

Eleven open-source projects including 41 versions in PROMISE repository [12] are employed to conduct experiments, and cross-version defect prediction is adopted in order to simulate the real application. Therefore, the obtained results are strongly related to the software defect prediction domain and the results are convincing. However, some potential threats to validity should be considered.

One threat is the datasets. Although we employ a large collection of publicly available datasets, these datasets are only a very small part of all datasets in real world. The conclusions over these 41 datasets might not hold for other datasets.

Another threat is the choice of compared methods and parameters. There are many methods that can construct linear defect prediction models. In this paper, only a few linear models are investigated. Furthermore, the parameters of the compared methods were simply set according to previous work [13], [15], [16]. The parameters for the multi-objective approach were simply set like the learning-to-rank approach [15]. These parameters might not be the best parameters, and the results might be different using different parameters.

The choice of performance measures can also be a threat. We use fpa, mse and nnz as the performance measures. However, there exist other performance measures, such as average absolute error. When different performance measures are adopted, the conclusion might be different.

## VII. CONCLUSIONS

Software defect prediction is very important because it can help software testers to allocate testing resources ef-

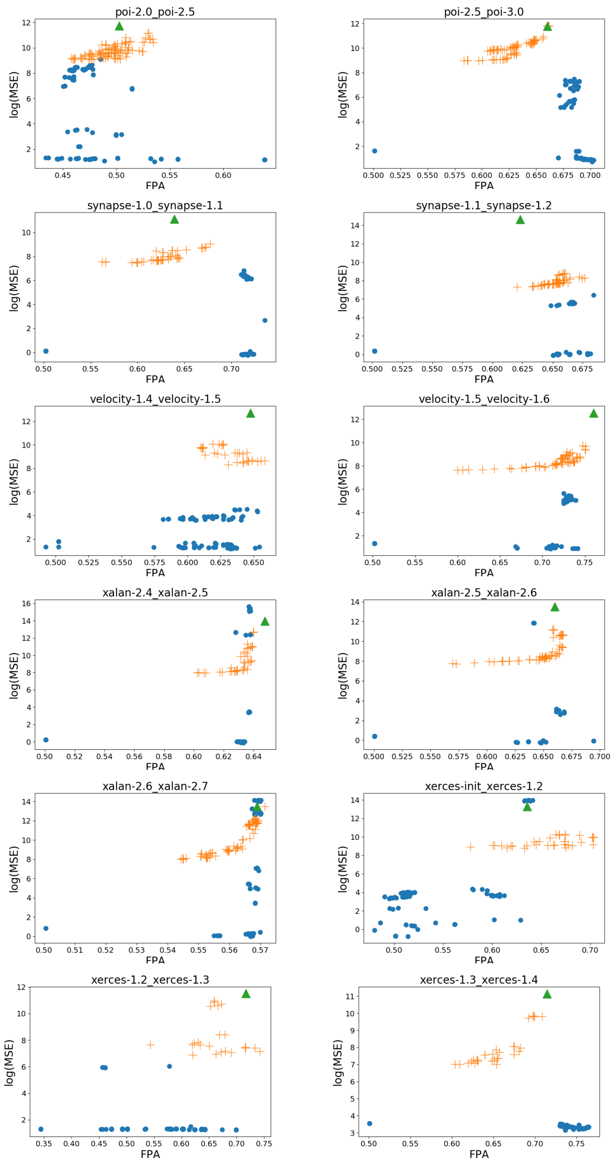

Fig. 2. Multi-Objective VS. Learning-to-Rank According to FPA and MSE

fectively and efficiently. Existing methods construct defect prediction models through minimizing the defect prediction error measures or optimizing the ranking performance. In some cases, the model complexity is also considered. Hence, defect prediction can be seen as a multi-objective optimization problem.

Therefore, we study a multi-objective evolutionary algorithm (NSGA-II) that construct a defect prediction model by simultaneously optimizing more than one goal, for cross-version defect prediction. Cross-version defect prediction constructs a model from the previous version of a software project to predict defects in the current version, which is similar to actual application. In order to build simpler models with more zero parameters, we also propose a revised multi-objective method (revised NSGA-II). Experimental results over 30 sets of cross-version data show that the multi-objective approach

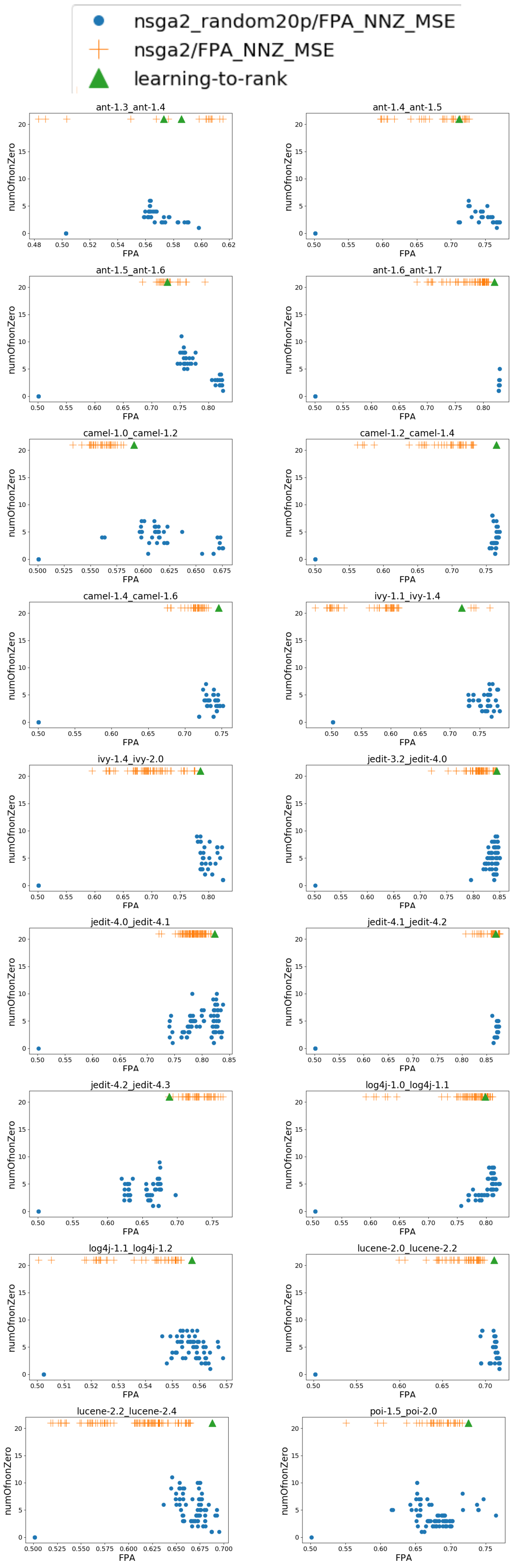

Fig. 3. Multi-Objective VS. Learning-to-Rank According to FPA and NNZ

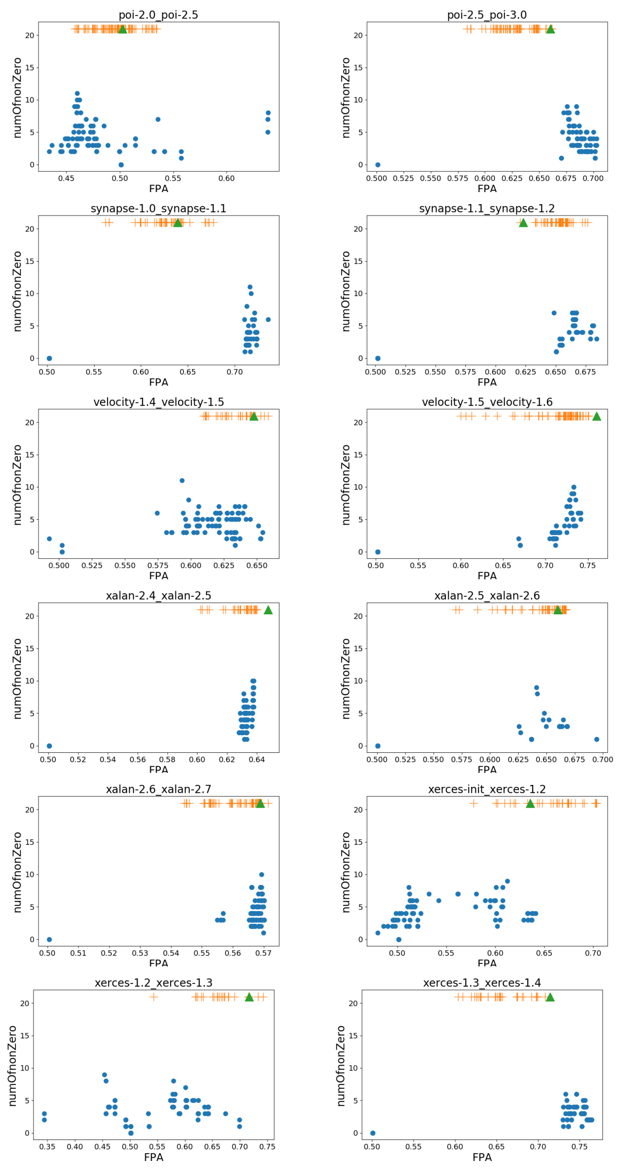

Fig. 4. Multi-Objective VS. Learning-to-Rank According to FPA and NNZ

and the revised multi-objective approach can construct diverse models to fit different applications. The revised NSGA-II can construct sparse linear models that have less zero parameters than models achieved by other compared methods, without performance loss according to FPA (ranking performance measure) and MSE (prediction error measure). In some cases such as ”ivy-1.4_ivy-2.0”, the revised NSGA-II even achieves better results. These imply the good performance of the revised multi-objective methods. Therefore, it is a good choice to use the revised multi-objective approach to build sparse defect prediction models.

## ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (Grants No. 61602534) and Fundamental Research Funds for the Central Universities (No.17lgpy122). This work is also supported by the Scientific Research Platforms and Projects in Universities in Guangdong Province under Grants 2019KTSCX204.

## REFERENCES

[1] A. B. Carvalho and S. R. Vergilio. A symbolic fault-prediction model based on multiobjective particle swarm optimization. *Systems and Software*, pages 868–882, 2010.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6:182–197, 2000.

[3] W. J. Fu. Penalized regressions: the bridge versus the lasso. *Journal of Computational and Graphical Statistics*, 7(3):397–416, 1998.

[4] K. Gao and T. Khoshgoftaar. A comprehensive empirical study of count models for software defect prediction. *IEEE Transactions on Reliability*, 56(2):223–236, 2007.

[5] M. Jureczko and L. Madeyski. Towards identifying software project clusters with regard to defect prediction. In *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, PROMISE ’10, pages 9:1–9:10, 2010.

[6] T. M. Khoshgoftaar and Y. Liu. A multi-objective software quality classification model using genetic programming. *IEEE Transactions on Reliability*, 56(2):237–245, 2007.

[7] T. M. Khoshgoftaar, Y. Liu, and N. Seliya. A multiobjective module-order model for software quality enhancement. *IEEE Transactions on Evolutionary Computation*, 8(6):593–608, 2004.

[8] Y. Li, J. Su, and X. Yang. Multi-objective vs. single-objective approaches for software defect prediction. In *International Conference on Management Engineering, Software Engineering and Service Sciences, ICMSS 2018*, pages 122–127, 2018.

[9] T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. *IEEE Transactions on Software Engineering*, 33(1):2–13, 2007.

[10] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transactions on Software Engineering*, 22(12):886–894, 1996.

[11] T. Ostrand, E. Weyuker, and R. Bell. Predicting the location and number of faults in large software systems. *IEEE Transactions on Software Engineering*, 31(4):340–355, 2005.

[12] S. Shukla, T. Radhakrishnan, and K. Muthukumaran. Multi-objective cross-version defect prediction. *Soft Computing*, pages 1–22, 2016.

[13] E. Weyuker, T. Ostrand, and R. Bell. Comparing the effectiveness of several modeling methods for fault prediction. *Empirical Software Engineering*, 15(3):277–295, 2010.

[14] Z. Xu, J. Liu, X. Luo, and T. Zhang. Cross-version defect prediction via hybrid active learning with kernel principal component analysis. In *25th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, pages 209–220, 2018.

[15] X. Yang, K. Tang, and X. Yao. A learning-to-rank approach to software defect prediction. *IEEE Transactions on Reliability*, 64(1):234–246, 2015.

[16] X. Yang and W. Wen. Ridge and lasso regression models for cross-version defect prediction. *IEEE Transactions on Reliability*, 67(3):885–896, 2018.