An Investigation of a Multi-Objective Evolutionary Algorithm for Cross-Version Defect Prediction

**

ABSTRACT

Cross-version 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. In more cases, we pay attention to both the order and the accuracy. In some cases, 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 investigate a multi-objective evolutionary algorithm and propose a revised multi-objective approach that construct defect prediction models by simultaneously optimizing more than one goal, for cross-version defect prediction. Experimental results over 30 sets of cross-version data show the effectiveness of the multi-objective approaches.

KEYWORDS

cross-version defect prediction, multi-objective evolutionary algorithm, ranking task

ACM Reference Format:
. 2020. An Investigation of a Multi-Objective Evolutionary Algorithm for Cross-Version Defect Prediction. In *EASE2020: The International Conference on Evaluation and Assessment in Software Engineering, April 15–17, 2020, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages.

1 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 [15]. 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 [13]. 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 [11].

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) [10, 15], and to predict whether a software module has defects or not (which we call as classification task) [8]. In this paper, the former goal is studied. The detailed process is as follows [15]: first, data is obtained from software modules according to metrics such as lines of code, and response for a class [4]; 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 [9], negative binomial regression [10], recursive partitioning, Bayesian additive regression trees, and random forest [12], 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. [14] proposed an effective learning-to-rank method, which constructed defect prediction models by directly optimizing the ranking performance.

In more 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, the predicted defect number should be around 3 instead of 30000). Furthermore, 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. Therefore, defect prediction for the ranking task can be seen as a multi-objective optimization problem. And hence, in this paper, we employ a multi-objective evolutionary algorithm to cross-version defect prediction, which

*

EASE2020, April 15–17, 2020, Trondheim, Norway

constructs defect prediction models by simultaneously optimizing more than one goal. To be specific, we combine the merits of Yang et al.'s learning-to-rank approach (optimizing ranking performance) and traditional methods (minimizing prediction errors).

The rest of this paper is organized as follows. Section 2 presents related work. In Section 3, an overview of the multi-objective optimization method is described, including the multi-objective evolutionary algorithm and our fitness functions. 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.

2 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. [3] 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. [12] 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 [15] 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 [6]. Therefore, some researchers employed evolutionary algorithm to directly optimize the ranking performance. For example, Khoshgoftaar et al. [6] 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. [14] 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 [7]. For example, in 2004, Khoshgoftaar et al. [6] 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. [5] 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. [11] 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. [7] 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.

3 MULTI-OBJECTIVE OPTIMIZATION

According to related work, existing studies constructed software defect prediction models by optimizing prediction error measures, or the ranking performance. Models by minimizing error measures might not give a good order [14], and models by optimizing the ranking performance might have a poor prediction accuracy. In more cases, both a good order of software modules and a good predicted accuracy are desired. We need an accurate order to help testers to allocate testing resources, and a predicted defect number as reference. Moreover, model complexity is also considered in some cases. 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. Subsequently, the employed multi-objective evolutionary approach is described.

3.1 The Objectives

As mentioned above, both ranking performance and prediction accuracy should be considered, and model complexity is also considered in some cases. Hence, three objectives are described in this subsection.

In this study, we adopt fault-percentile-average (FPA) [12] as the ranking performance measure. Considering m modules f_1, f_2 , ..., 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[12]:

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

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:

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

In order to compare with the learning-to-rank approach [14] directly, we also adopt a linear model as the base predictor for our multi-objective approach. 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:

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

where α_js (including α_0) are the corresponding parameters obtained by training. Once α_js are fixed, the model is learned. If more α_js equal to zero, the computation is simpler. Therefore, we use the number of nonzero α_js to measure the model complexity, which is denoted as NNZ. To be consistent with linear regression, there is an α_0, which is different from Yang et al.'s work [14].

3.2 Multi-Objective Approach

Multi-objective approaches are used to construct a set of Pareto-optimal models. In this study, we adopt linear regression as the 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 α_js, which optimizes two or three objectives (FPA, MSE, or 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 α_js using crossover and mutation operation, we revise the process of NSGA-II by setting parameters to zero when they are not larger than 5% of the maximum feasible value. Both original and revised NSGA-II are studied in this paper.

4 EXPERIMENTAL SETUP

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

4.1 Research Questions

The main research questions in this paper are as follows:

* **RQ1**: Compared with the learning-to-ranking approach [14], how does the multi-objective methods perform?
* **RQ2**: Compared with linear regression [12] and ridge regression [15], how does the multi-objective methods perform?
* **RQ3**: How complex are these achieved models?

In essence, both RQ1 and RQ2 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 [14]. Linear regression and ridge regression focus on the fitting of each sample because they construct models by minimizing least squares. Their good performance for software defect prediction has also been shown [12, 15]. Therefore, we respectively compare the multi-objective methods with these two kinds of algorithms. In some cases, model complexity is also considered, so we investigate the complexity of these achieved models in RQ3.

4.2 Datasets

In order to facilitate others to reproduce results, 41 data sets from 11 open-source projects in PROMISE repository [11, 14][1] are used. Following Yang et al.'s work[14], we use the defect number as the dependent variable.

The characteristics of these experimental datasets are shown in Table 1. 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.

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

Table 1: 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 [4].

4.3 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, and the two multi-objective approaches. The involved objectives included FPA, MSE and NNZ, which are described in Section 3. All methods are implemented in Python. The parameters of the compared methods are simply set according to previous work [12, 14, 15], and the parameters for the multi-objective approach are simply set like the learning-to-rank approach [14]. For example, 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.

5 EXPERIMENTAL RESULTS

The experimental results are reported according to the research questions: RQ1, RQ2, and RQ3.

5.1 RQ1: Multi-Objective VS. Learning-to-Rank

In this subsection, we compare the multi-objective approach with the learning-to-rank approach over 30 sets of cross-version data. The compared results are shown in Figures 1 and 2.

To be noted, there are four results of multi-objective approach. As mentioned above, we study both original NSGA-II and revised NSGA-II as the multi-objective learning algorithms. In these figures, "multi-objective/FPA_MSE" means using original NSGA-II to simultaneously optimize two objectives: FPA and MSE, and "multi-objective/FPA_NNZ_MSE" means using original NSGA-II to simultaneously optimize three objectives: FPA, NNZ and MSE. When optimizing two objectives (FPA and MSE), the results of the revised NSGA-II are denoted as "multi-objective-revised/FPA_MSE". When optimizing all three objectives, the results of the revised NSGA-II are denoted as "multi-objective-revised/FPA_NNZ_MSE".

From Figures 1 and 2, we can see that the mse values of the learning-to-rank approach are very large (larger than 10^10). By contrast, the mse values of all multi-objective approaches are much smaller. Furthermore, we can find models from multi-objective approaches, which dominate (achieve both smaller mse and larger fpa than) the learning-to-rank model, over most sets of data. This implies the good performance of multi-objective approaches. Another merit of multi-objective approaches is the advantage of obtaining diverse models simultaneously for different applications. As a whole, the multi-objective methods perform well compared with the learning-to-rank approach.

5.2 RQ2: Multi-Objective VS. Linear and Ridge Regression

In this subsection, we compare the multi-objective approach with linear regression and ridge regression over 30 sets of cross-version data. The compared results are shown in Figures 3 and 4. The denotations of multi-objective approaches are the same with Figure 1. "RidgeCV" denotes the results of ridge regression models.

From Figures.3 and 4, we can see that, for some multi-objective models, when they have relative big fpa, their mse values are quite big, compared with linear and ridge regression. In some cases such as "lucene-2.0_lucene-2.2", linear and ridge regression models dominate all multi-objective models. However, over some sets of data such as "camel-1.0_camel-1.2", we can find models from multi-objective approaches, which achieve comparable mse and larger fpa than linear and ridge regression models. As a whole, the multi-objective methods perform comparably with with linear regression and ridge regression. Nevertheless, the diversity of multi-objective approaches still holds.

Unexpectedly, the revised multi-objective approach, which is modified to make more zero parameters, performs as well as the original multi-objective approach according to mse

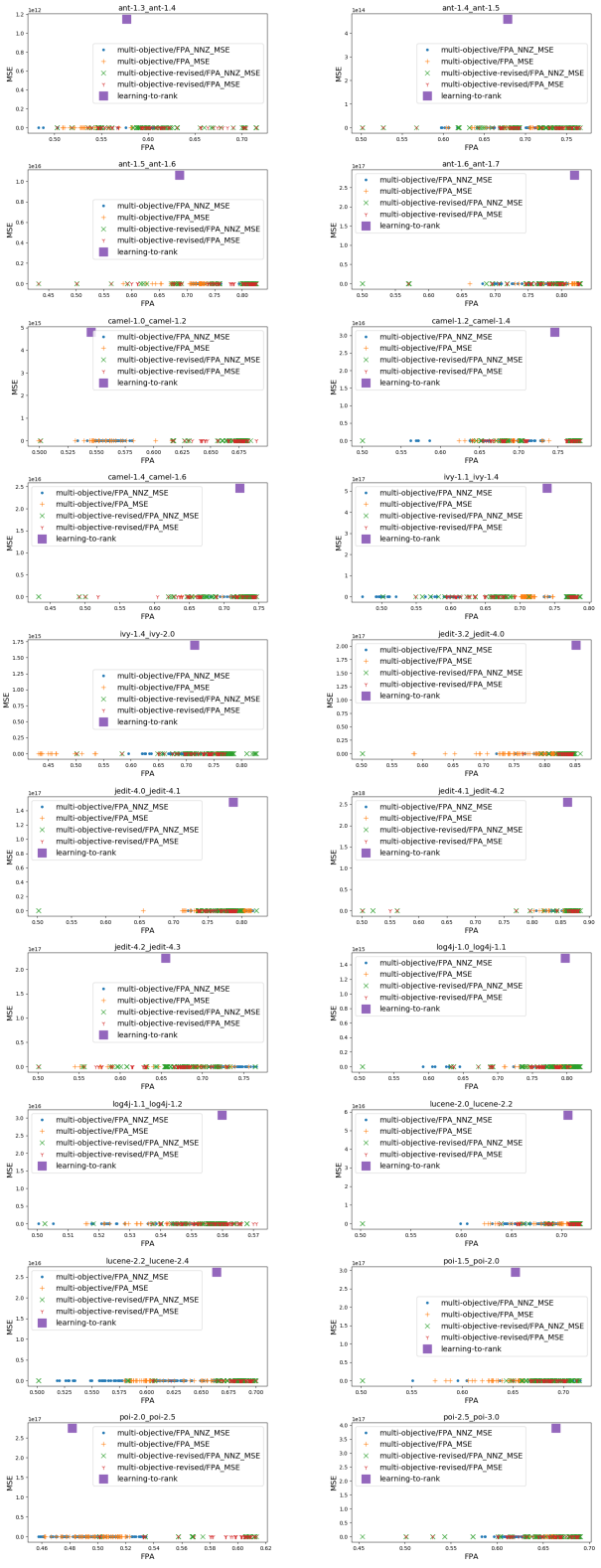

Figure 1: Multi-Objective VS. Learning-to-Rank

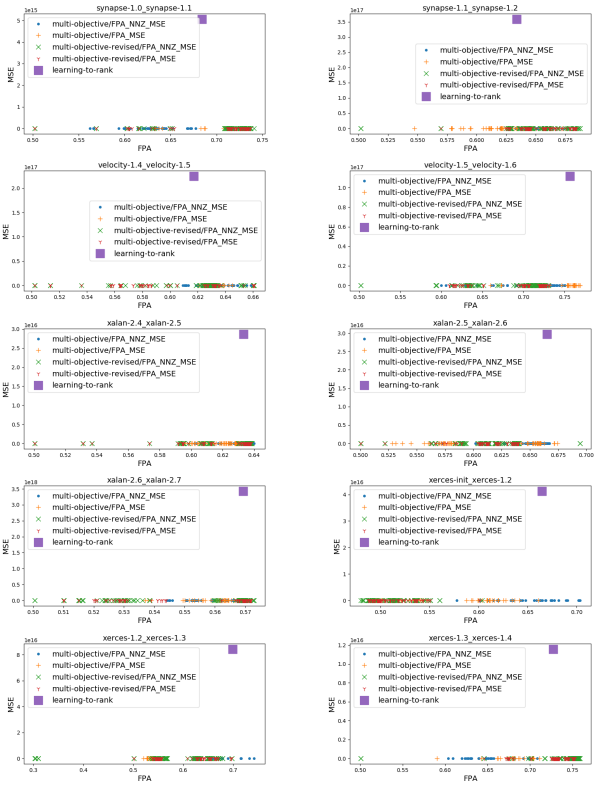

Figure 2: Multi-Objective VS. Learning-to-Rank

and fpa in most cases. In some cases such as "jedit-4.1_jedit-4.2", they even achieve better results. This implies simpler model might achieve better testing results.

5.3 RQ3: Model Complexity

In this subsection, we compute number of nonzero parameters (NNZ) of all compared models in order to analyze their complexity. The results are shown in Table 2. "LR" means linear regression, "RR" means ridge regression, and "LTR" means the learning-to-rank approach.

From Table 2, we can see that only the revised multi-objective approach can achieve simpler models, no matter optimizing two or three objective. That is, even the revised multi-objective approach does not optimize NNZ, it can obtain models with fewer nonzero parameters. All of other approaches have no zero parameter, including the original multi-objective approach. The answer to RQ3 is that the revised multi-objective approach can achieve simpler models. Therefore, when model complexity is considered, we can use the revised multi-objective approach.

6 THREATS TO VALIDITY

Eleven open-source projects including 41 versions in PROMISE repository [11] are employed to conduct experiments, and

Table 2: Summary of Number of Nonzero Parameters of All Models

filename	*nsga2*	*nsga2*	*nsga2*	*nsga2toZero*	*nsga2toZero*	*nsga2toZero*	*LR*	*RR*	*LTR*
targets	*fpa|nnz*	*fpa|nnz|mse*	*fpa|mse*	*fpa|nnz*	*fpa|nnz|mse*	*fpa|mse*			
$ant-1.3$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 4]	[0, 5]	21	21	21
$ant-1.4$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 4]	[0, 3]	21	21	21
$ant-1.5$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 5]	[0, 5]	21	21	21
$ant-1.6$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 4]	[1, 6]	21	21	21
$ant-1.7$	[21, 21]	[21, 21]	[21, 21]	[0, 2]	[0, 4]	[1, 5]	21	21	21
$camel-1$	[21, 21]	[21, 21]	[21, 21]	[0, 7]	[0, 8]	[0, 8]	21	21	21
$camel-1.2$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 6]	[1, 6]	21	21	21
$camel-1.4$	[21, 21]	[21, 21]	[21, 21]	[0, 7]	[0, 7]	[0, 10]	21	21	21
$camel-1.6$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 8]	[1, 8]	21	21	21
$ivy-1.1$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 5]	[1, 6]	21	21	21
$ivy-1.4$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 6]	[0, 5]	21	21	21
$ivy-2.0$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 6]	[0, 8]	21	21	21
$jedit-3.2$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 9]	[2, 8]	21	21	21
$jedit-4.0$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 4]	[1, 4]	21	21	21
$jedit-4.1$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 4]	[1, 3]	21	21	21
$jedit-4.2$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 5]	[0, 6]	21	21	21
$jedit-4.3$	[21, 21]	[21, 21]	[21, 21]	[0, 2]	[0, 3]	[0, 3]	21	21	21
$log4j-1.0$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 8]	[1, 7]	21	21	21
$log4j-1.1$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 5]	[1, 7]	21	21	21
$log4j-1.2$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 5]	[2, 6]	21	21	21
$lucene-2$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 7]	[1, 6]	21	21	21
$lucene-2.2$	[21, 21]	[21, 21]	[21, 21]	[0, 7]	[0, 8]	[1, 7]	21	21	21
$lucene-2.4$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 6]	[1, 10]	21	21	21
$poi-1.5$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 6]	[1, 9]	21	21	21
$poi-2.0$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 4]	[0, 7]	21	21	21
$poi-2.5$	[21, 21]	[21, 21]	[21, 21]	[1, 4]	[0, 4]	[1, 4]	21	21	21
$poi-3.0$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 7]	[1, 6]	21	21	21
$synapse-1.0$	[21, 21]	[21, 21]	[21, 21]	[0, 2]	[0, 4]	[0, 5]	21	21	21
$synapse-1.1$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 6]	[1, 7]	21	21	21
$synapse-1.2$	[21, 21]	[21, 21]	[21, 21]	[1, 5]	[0, 7]	[1, 6]	21	21	21
$velocity-1.4$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 3]	[1, 3]	21	21	21
$velocity-1.5$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 6]	[2, 7]	21	21	21
$velocity-1.6$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 4]	[1, 5]	21	21	21
$xalan-2.4$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 6]	[0, 8]	21	21	21
$xalan-2.5$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 5]	[1, 4]	21	21	21
$xalan-2.6$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 6]	[1, 7]	21	21	21
$xalan-2.7$	[21, 21]	[21, 21]	[21, 21]	[0, 5]	[0, 7]	[1, 11]	21	21	21
$xerces-init$	[21, 21]	[21, 21]	[21, 21]	[0, 3]	[0, 21]	[1, 21]	21	21	21
$xerces-1.2$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 4]	[0, 4]	21	21	21
$xerces-1.3$	[21, 21]	[21, 21]	[21, 21]	[0, 4]	[0, 7]	[1, 6]	21	21	21
$xerces-1.4$	[21, 21]	[21, 21]	[21, 21]	[0, 6]	[0, 7]	[1, 11]	21	21	21

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 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 [12, 14, 15]. The parameters for the multi-objective approach were simply set like the learning-to-rank approach [14]. These parameters might not be the best parameters, and the results might be different using different parameters.

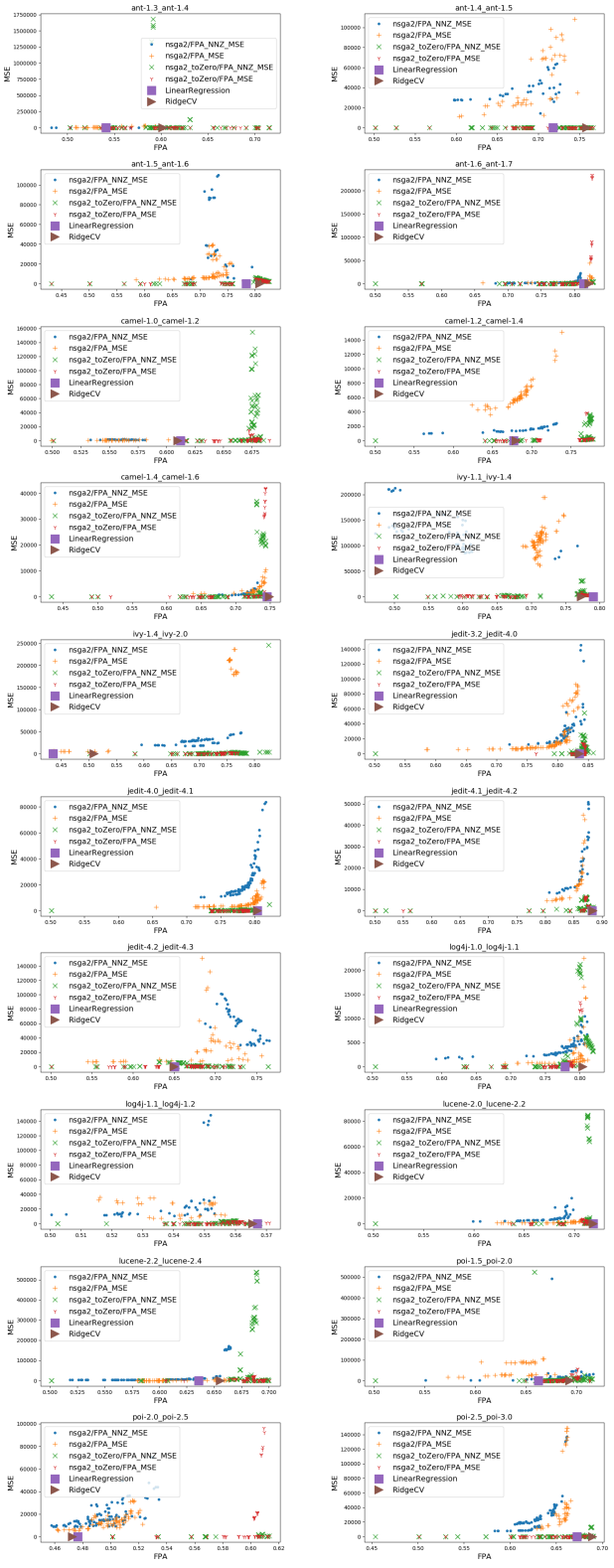

Figure 3: Multi-Objective VS. Linear and Ridge Regression

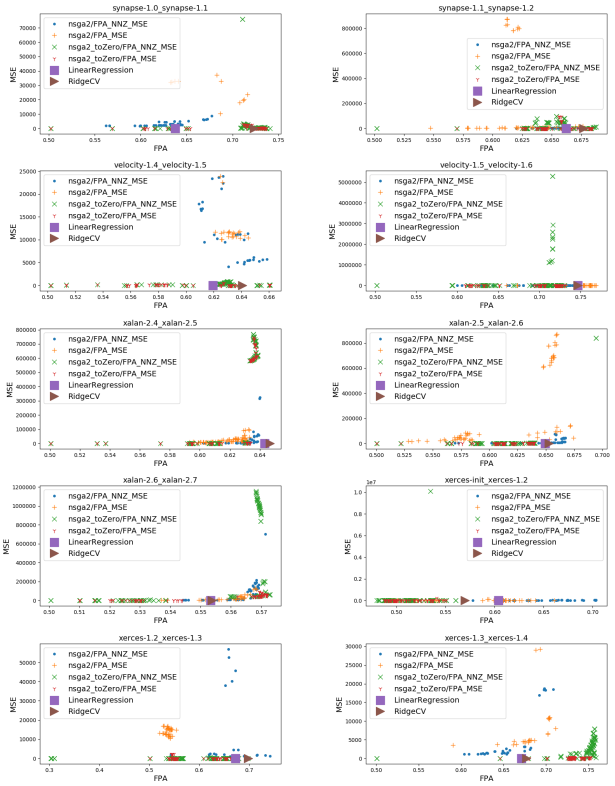

Figure 4: Multi-Objective VS. Linear and Ridge Regression

The choice of performance measures can also be a threat. We use fpa, mse and nnz as 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.

7 CONCLUSIONS

Software defect prediction is very important because it can help software testers to allocate testing resources effectively and efficiently. Existing methods construct defect prediction models through minimizing the defect prediction error measures or optimizing the ranking performance. In more cases, both a good order and a high prediction accuracy are desired. 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 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. Experimental results over 30 sets of cross-version data show that the effectiveness of the multi-objective approach and the revised multi-objective approach, which can construct diverse models to fit different applications. According to measures of fpa and mse, both multi-objective approaches perform well compared with the learning-to-rank approach, and perform comparably with linear regression and ridge regression. When model complexity is considered, the revised multi-objective approach can be chosen because it can achieve simpler models.

8 ACKNOWLEDGMENTS

This work is supported by .

REFERENCES

[1] A. B. Carvalho and S. R. Vergilio. 2010. A Symbolic Fault-Prediction Model Based on Multiobjective Particle Swarm Optimization. *Systems and Software* (2010), 868–882.

[2] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2000. A Fast Elitist Multi-Objective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6 (2000), 182–197.

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

[4] Marian Jureczko and Lech Madeyski. 2010. 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)*. Article 9, 10 pages.

[5] Taghi M Khoshgoftaar and Yi Liu. 2007. A Multi-Objective Software Quality Classification Model Using Genetic Programming. *IEEE Transactions on Reliability* 56, 2 (2007), 237–245.

[6] Taghi M Khoshgoftaar, Yi Liu, and Naeem Seliya. 2004. A Multiobjective Module-Order Model for Software Quality Enhancement. *IEEE Transactions on Evolutionary Computation* 8, 6 (2004), 593–608.

[7] Y. Li, J. Su, and X. Yang. 2018. Multi-Objective vs. Single-Objective Approaches for Software Defect Prediction. In *International Conference on Management Engineering, Software Engineering and Service Sciences, ICMSS 2018*. 122–127.

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

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

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

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

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

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

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

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