Software Defect Prediction Based on Competitive Organization CoEvolutionary Algorithm

Xiao-dong Mu, Rui-hua Chang, Li Zhang

Xi’an Research Inst. of Hi-Tech, Xi’an, 710025, China
mu_msn@msn.com, sxwcrh@163.com, Zhangli_522@126.com

Abstract

In order to improve the accuracy of prediction for software defect data sets, competitive organization coevolutionary algorithm is presented and applied for dealing with the software defect data. During this algorithm, mechanism of competition is introduced into coevolutionary algorithm. Then leagues are formed based on the importance of attributes among them. And three evolution operators which are reduced operator, allied operators and disturbed operators are developed. Furthermore, both the importance of attributes and evaluation from competition are used for the calculation of fitness function. Finally, five data sets from NASA MDP (Metrics Data Program) were used to validate the algorithm. The experimental results show that the proposed algorithm is effective for software defect prediction.

Keywords: Competition, Coevolutionary algorithm, Software defect, Prediction

1. Introduction

Along with the increase of software complexity, software quality is growing to be an important factor in the fields of software engineering. In order to raise the effectiveness and efficiency of testing, software defect prediction has been used to identify defect-prone modules in an upcoming version of a software system and help to allow the effort on those modules. Over the past decades years, several empirical studies have been carried out to predict the fault proneness models such as [1-6]. From a holistic point of view, software defect prediction studies can be categorized as statistical and machine learning (ML) approaches. And the use of machine learning approaches to fault prediction modeling is more popular[7]. Unfortunately, this problems of software defect prediction have not resolved thoroughly. And none of the techniques have achieved widespread applicability in the software industry due to several reasons, including the limitation of testing resource and budge, the lack of software tools to automate this software defect prediction, the unwillingness to collect the software defect data, many methods based on the private software data, and the other practical problems [6].

Genetic algorithm (GA) is an evolutionary computation technique[8-9] and has been applied into many areas including software defect prediction[5]. However, genetic algorithm is easy to fall into premature and slow convergence. Recently, a coevolutionary algorithm was proposed to resolve this problem. The coevolutionary algorithm considers the coordination relationships between population and environment. As the advantageous of coevolution algorithm, a growing number of researchers have studied it. However, to the author's knowledge, research on software defect prediction using coevolution is at the beginning. In this study, we proposed a new methods competitive organization coevolutionary algorithm (COCA), and applied to solve the problem of software defect prediction.

The rest of this paper is organized as follows: Firstly, we introduce related work about software defect prediction. Secondly, we present the competitive organization coevolutionary algorithm (COCA). Thirdly, we apply the COCA algorithm for prediction of software defect data. We simultaneously analyze and compared the results. Finally, we give our conclusion and works in the future.

2. Related Work of Software Defect Prediction

Until now, various machine learning models, such as linear regression, discriminate analysis, decision trees, neural networks and Naïve Bayes and so on, have been developed and applied to predict defects in software. These relatively sophisticated models are preferable to simple linear regression and
correlation models because the relationship between defects (response variable) and static measures (predictor variables) might not be a monotonous linear relationship [10].


Munson et al. [13] investigate linear regression models and discriminate analysis to conclude the performance of the latter is better.

Nagappan et al. [14] also used linear regression analysis with the STREW metric suite. This suite of metrics was extracted from the testing process and is used to estimate the post-release defects. They validate their approach on industrial, open source and student projects and find strong correlations between the proposed metric suite and post-release defects.

Catal et al. [15] applied artificial immune system to software defect prediction for pursue the high-performance models. In this paper, they reported that RF (Random Forests) gains the best prediction results for large datasets. And Naïve Byes is the best algorithm for small data sets. In 2011, Catal et al [16] developed an Eclipse-based software fault prediction tools for Java programs, and naïve bays chosen as the plug-in for the tools.

Norman Fenton et al. in Ref. [17] pointed out that traditional statistical approaches were inadequate for software defect prediction. In 2007, he used dynamic Bayesian nets for predicting software defects [18]. Rather than depending only on data from previous versions, his method makes use of causal models of the Project Manager’s understanding and covering mechanisms.

On open-source software, Denaro and Pezze [19] analyzed Apache using logistic regression with static code features and their 80% prediction performance pointed 50% of the modules to be inspected.

Olivier Vandecruys et al. [20] tackled software quality problems based on Ant Colony Optimization. And compared with C4.5, logistic regression and support vector machines, AntMiner+ model is superior to them.

Bullard et al. [21] employ a rule-based classification model in a telecommunication system and reported that their model produces lower false positives, which are considered as high cost classification errors.

In brief, those works present promising results. However, until now, the ML-based works show two main disadvantages: most prediction models are not easily interpreted by the programmers and testers; and most approaches require a pre-process step. Until now, despite so many effort have been putting into developing the software defect prediction models, there are few prediction models achieved widespread application in the fields of software engineering.

3. Competitive Organization Coevolutionary Algorithm for Defect Prediction

Software defect prediction is usually dealt with binary classification for a module: defective or non-defective. In Ref. [3], we have proposed a method that is a competitive organization coevolution algorithm for classification. Considering the similarity of them, in this section, we introduce the algorithm and applied it for software defect prediction.

3.1 Calculation of Fitness Function

Fitness function is one of the most important parts for COCA. In COCA, the individual’s fitness is calculated not only by a population but also relying on competition among species. And in this algorithm, population is divided into two competitive parts which are species training data (STRD) and species test data (STED).

3.1.1. Calculation of the Fitness Function for STED

The calculation of fitness function for STED is mainly through competition. In this section, we utilize Eq. (1) to calculate the reward factor as follows:

Supposed: the number of league of STRD is \(M\), and there are \(M\) rules. There are \(N\) test data in STED.

For STED, assuming that the number of test data correctly classified is \(N_i\), then the reward factor is
calculated as follows:

\[ \beta_{\text{STED}} = 1 / (N - M_j) \]  

(1)

From Eq. (1), we can find that the less number of individual defeated by opponents, the greater their access to the reward factor. That is because the value of 1 is allocated to the fewer individuals. On the contrast, it will receive a small reward factor.

Finally, the value of league’s fitness function is calculated by \( f_{\text{STED}}(x) = \beta_{\text{STED}} \).

3.1.2. Calculation of the Fitness Function for STRD

**Step1**: To calculate the impact on the classification attributes. For a discretional league, if an attribute is removed, the results of prediction will change. Hence, its importance of the attributes is big. On the other hand, it is small. It is defined just as the Eq. (2):

\[ \sigma(A') = \gamma_A(B) - \gamma_{A-A}(B) \]  

(2)

where \( \sigma \) is the importance of attributes. \( A \) and \( B \) respectively express condition attributes set and class attribute set. Subset of attribute \( A' \) is part of \( A \). \( \gamma \) indicate the dependence between attributes.

**Step2**: According to the importance of attributes, if the attribute’s importance is zero, the attribute will be removed. And it forms a new property set \( \{ C_1, C_2, \ldots, C_k \} \), where \( k \) is the number of attributes.

**Step3**: Calculation of reward factor and penalty factor as follows:

Supposed: the number of league in STRD is \( M \), and there are \( M \) rules, assuming that the number of test data by \( \text{League}_i \) successful identifies is \( M_j \).

Its reward factor is calculated as follows: \( \beta_{\text{STRD}} = 1 / (N - M_j) \).

**Step4**: The fitness is calculated based on Eq. (3).

\[ f_{\text{STRD}}(x) = \sum_{i=1}^{k} \sum_{j=1}^{\mathbb{U}_{\text{str}}} \sigma_j(A_j) + \beta_{\text{STRD}} \]  

(3)

where, \( \sigma_j(A_j) \) expresses importance of the \( i \)th attributes of \( j \)th individuals and \( | \mathbb{U}_{\text{str}} | \) expresses the number of effective attributes of league, \( | x | \) expresses the number of league.

Finally, the value of league’s fitness function is returned.

3.2 The Evolution Operators

In this section, according to the actual requests of the proposed method, we present reduced operator, allied operators and disturbed operators for increase the diversity of population.

Input: \( \text{League}_i = \begin{pmatrix} V_{C_1} & \cdots & V_{C_l} \\ \vdots & \ddots & \vdots \\ V_{C_k} & \cdots & V_{C_l} \end{pmatrix} \), \( \text{League}_j = \begin{pmatrix} V_{C_1} & \cdots & V_{C_l} \\ \vdots & \ddots & \vdots \\ V_{C_k} & \cdots & V_{C_l} \end{pmatrix} \)

where, each row \( (V_{C_1}, V_{C_2}, \ldots, V_{C_k}) \) expresses a member of league, a \( L \) is composted of multiple similar data sets.

Output: The new league data \( L' \).

**Reduced Operator**

**Step1**: For any \( L_i \), if it meets the conditions: \( | L_i | > 1 \) and \( V_{C_1} \cup V_{C_2} \cup \ldots \cup V_{C_k} = V_{C_l} \) then goes to Step 2;

**Setp2**: The \( L_i \) is amended as \( \text{League}_i' = \begin{pmatrix} * & V_{C_1} & \cdots & V_{C_l} \\ \vdots & \vdots & \ddots & \vdots \\ \vdots & V_{C_k} & \cdots & V_{C_l} \end{pmatrix} \), where *indicates that the property

is excluded;

**Setp3**: Return the new league data \( L' \).

**Allied Operator**

**Step1**: \( L \) and \( L' \) are randomly selected from the same category;
Step2: Form a new individual League = 
\[
\begin{pmatrix}
V_{C11} & \ldots & V_{C1t} \\
\vdots & \ddots & \vdots \\
V_{Ck1} & \ldots & V_{Ckt} \\
\vdots & \ddots & \vdots \\
V_{CgL} & \ldots & V_{Cgt}
\end{pmatrix};
\]

Step3: Return the new league data \(L_i'\).

Disturbed Operator

Initialization: if the attribution is discrete, then \(K = 0\); and if it is a continuous attribute, then \(K = 1\);

Step1: For any \(L_i\), the attribution is randomly selected as a disturbed gene;

Step2: When \(K = 0\), replaced the value from its domain and produce a new one; then go to Step 4.

Step3: When \(K = 1\), a small number was randomly generated, then the original value plus or minus the number to generate new individuals;

Step4: Return the new league data \(L_i'\).

3.3 Algorithms for Software Defect Prediction

Step1: Initialization and preprocessing of data including the balancing of both classes (defective or otherwise) and the removal of a large number of repeating instances. Initialization the number of evolution \(E\), competition number \(C\), and the disturbance probability \(\mu\). Set \(t = 0\), \(k = 1\), \(l = 0\) where \(t\) is the number of evolution, \(k\) is the number of population, \(l\) is the number of competition.

Step2: According to rule of 2-8, select the training data sets (STRD) and test data sets (STED). STRD is divided into \(|\text{Class}|\) sub-populations accordance with the number of Class type.

Step3: In the sub-populations, attribute significance are calculated, and leagues are initialized.

Step4: To select \(L_i\) and \(L_j\) randomly. Then the above evolution operators are performed on them. And the fitness function is calculated. If it meets the following condition:

\[
\begin{align*}
&f'_{\text{max}} = \max\{f'_{L_i}, f'_{L_j}\} \\
&f'_{\text{max}} = \max\{f'_{L_i}, f'_{L_j}\} \\
&\text{and } f'_{\text{max}} > f_{\text{max}}
\end{align*}
\]

then \(L_i'\) and \(L_j'\) replace \(L_i\) and \(L_j\) respectively. Otherwise, \(L_i\) and \(L_j\) are saved.

Step5: When \(k > |\text{Class}|\), then \(t = t + 1\), turn to Step6, otherwise turn to Step4.

Step6: \(k = k + 1\), while \(t > E\), extract and select the prediction rules. Otherwise turn to Step4.

Step7: To make STED and STRD "arms race". Then by referring to corresponding table of rules and, the fitness function value of STED and STRD is modified.

Step8: If the termination condition is met or \(l > C\), then the algorithm quit. Otherwise, set \(t = 0\), \(k = 1\), go to Step2.

4. Experiments and Analysis

4.1 Description of Data

As in any machine learning problem, software defect prediction models require a set of features (i.e. independent variables) to characterize the problem and to give estimation on the defect proneness of the system (i.e. dependent variable). In software quality, these attributes are referred to as software metrics. Metrics are the attributes that represent software; they are the raw data for software domain.
An effective management of any software development process requires monitoring and analysis of software metrics. The software metrics and dataset used in this study are five mission critical NASA software projects [22], which are all high assurance and complex real-time system. NASA makes extensive use of contractors from many other industries including government and commercial organizations. It is practical to leverage the useful information in order to predict the quality of an ongoing similar project. Table 1 summarizes the characteristic of five datasets used in this study. And Table 2 presents the part of metrics used in the five datasets considering the length of paper.

<table>
<thead>
<tr>
<th>Data</th>
<th>Language</th>
<th>Model</th>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KC3</td>
<td>JAVA</td>
<td>458</td>
<td>40</td>
<td>processing and delivery of satellite metadata</td>
</tr>
<tr>
<td>CM1</td>
<td>C</td>
<td>498</td>
<td>22</td>
<td>NASA spacecraft instrument</td>
</tr>
<tr>
<td>MC2</td>
<td>C++</td>
<td>161</td>
<td>40</td>
<td>Video guidance system</td>
</tr>
<tr>
<td>PC3</td>
<td>C</td>
<td>1563</td>
<td>38</td>
<td>Flight software for earth orbiting satellite</td>
</tr>
<tr>
<td>PC4</td>
<td>C</td>
<td>1458</td>
<td>38</td>
<td>Flight software for earth orbiting satellite</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metrics</th>
<th>Type</th>
<th>Metrics</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>V(g)</td>
<td>McCabe</td>
<td>T</td>
<td>DHalstead</td>
</tr>
<tr>
<td>EV(g)</td>
<td>McCabe</td>
<td>UniqOp</td>
<td>DHalstead</td>
</tr>
<tr>
<td>IV(g)</td>
<td>McCabe</td>
<td>UniqOpnd</td>
<td>DHalstead</td>
</tr>
<tr>
<td>LOC</td>
<td>McCabe</td>
<td>TotalOp</td>
<td>DHalstead</td>
</tr>
<tr>
<td>N</td>
<td>DHalstead</td>
<td>TotalOpnd</td>
<td>DHalstead</td>
</tr>
<tr>
<td>V</td>
<td>DHalstead</td>
<td>UniqOp</td>
<td>DHalstead</td>
</tr>
<tr>
<td>L</td>
<td>DHalstead</td>
<td>LOCcode</td>
<td>Line Count</td>
</tr>
<tr>
<td>D</td>
<td>DHalstead</td>
<td>LOCComment</td>
<td>Line Count</td>
</tr>
<tr>
<td>I</td>
<td>DHalstead</td>
<td>LOCBlank</td>
<td>Line Count</td>
</tr>
<tr>
<td>E</td>
<td>DHalstead</td>
<td>LOCCodeAndComment</td>
<td>Line Count</td>
</tr>
<tr>
<td>B</td>
<td>DHalstead</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

4.2 Prediction Performance Measures

Evaluation measures [23] play a crucial role in both assessing the classification performance and guiding the classifier modeling. After a classification process, data samples can be categorized into four groups as denoted in the confusion matrix presented in Table 3.

<table>
<thead>
<tr>
<th>Actually Defective</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Positive (TP)</td>
<td>False Negative (FN)</td>
</tr>
<tr>
<td>False Positive (FP)</td>
<td>True Negative (TN)</td>
</tr>
</tbody>
</table>

And several measures can be derived from the confusion matrix. And they are presented in Table 4.

<table>
<thead>
<tr>
<th>Table 3. Confusion matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy = ( \frac{TP \times TN}{TP \times TN + FP + FN} \times 100% )</td>
</tr>
<tr>
<td>Recall = ( \frac{TP}{TP + FN} \times 100% )</td>
</tr>
<tr>
<td>Precision = ( \frac{TP}{TP + FP} \times 100% )</td>
</tr>
<tr>
<td>( F - Measure = 2 \times \frac{Recall \times Precision}{Recall + Precision} \times 100% )</td>
</tr>
<tr>
<td>( pd = Recall = \frac{TP}{TP + FN} \times 100% )</td>
</tr>
<tr>
<td>( pf = \frac{FP}{FP + TN} \times 100% )</td>
</tr>
</tbody>
</table>
Traditionally, accuracy is the most commonly used measure for these purposes. For classification with the class imbalance problem, accuracy is no longer a proper measure since the rare class has very little impact on accuracy as compared to the prevalent class [24]. F-Measure represents a harmonic mean between recall and precision. A high F-Measure value ensures that both recall and precision are reasonable high. According to the results of above, we choose the F-Measure with confusion matrix as our performance measure on the test data.

5. Comparisons and Analysis

In this section, we investigate the results employed COCA for classification. Our experimental environment was Pentium (R) 3.2G CPU, 1G DDR memory, Windows XP operating system and so on.

In this experiment, we split the data set into training data sets and testing data sets, respectively, 80% and 20%, firstly. In order to avoid bias, we run the experiment 100 times and calculated its average.

Naïve Bays (NB) classifiers use statistical combinations of features to predict for class value. Such classifiers assume all the features are statistically independent. Nevertheless, a repeated empirical result is that, on average, seemingly Naïve Bays classifiers perform as well as other seemingly more sophisticated schemes.

Random Forest (RF) [25] is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the class’s output by individual trees. It is one of the most accurate learning algorithms available. For many data sets, it produces a high accuracy classifier.

Radial Basis Function Network (RBFNet) is an artificial neural network that uses radial basis functions as activation functions. It is a linear combination of radial basis functions.

The parameters for each of the compared methods were initialized with the default setting of the WEKA toolkit. And results presented on Figure 1.

![Figure 1. Compared with Other Methods on Five Data Sets](image)

From Figure 1, it is observed that COCA out performs all the compared three methods in F-measure. The experimental results show that the competition strategy is effective for evolutionary algorithm based on organization. The competitive strategy guide the evolution of species. Also it is effective to avoid the blindness of population evolution, and to reduce the algorithm operation time and improve the accuracy of prediction for software defect in F-Measure measure.

Meantime, we analyze the advantage of COCA in this section. There are three advantages. First, in the evolutionary process of the COCA algorithm, the strategy of competition coming from the biological is introduced for promoting the evolution of population. Second, the calculation of fitness function includes performance individual and their competition. Third, three evolutionary operators
are designed for league. One is reduced operator, which is mainly used for reducing of dimension of attributes. Next one is allied operator that is mainly used to merge the similar leagues and reduce the number of leagues. The last one is disturbed operator. It is used for further promoting the diversity of population and avoiding falling into the local optimum. Under the role of above advantage, the diversity of population is increased and the prediction performance of proposed algorithm for software defect data is better than others.

6. Conclusions

In order to improve the accuracy of software defect prediction, a coevolutionary algorithm based on the competitive organization is put forward for software defect prediction. During this algorithm, firstly, competition mechanism is introduced to organization coevolutionary algorithm. Then, three evolution operators which are reduced operator, allied operators and disturbed operators are developed for evolution of population. And competition is considered for calculate the fitness function. When the algorithm applied into software defect prediction, it improves the accuracy of software prediction through increases the diversity of population. Finally, experiment based on the five datasets from NASA is used to validate the method. The experimental results show that the proposed method is effective. It should be noted that this study has examined only on five datasets, which are not large dataset. Hence, Future work is that we should pay more attention to extension of this proposed method is to improve the efficiency of the large number of software defect data sets.

7. Acknowledgement

The authors would like thank the various members of author’s Laboratory, Xi’an Research Inst. of Hi-Tech, for their helpful comments.

8. References


Software Defect Prediction Based on Competitive Organization CoEvolutionary Algorithm
Xiao-dong Mu, Rui-hua Chang, Li Zhang


