2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-6654-3786-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/SANER53432.2022.00045

BCL-FL: A Data Augmentation Approach with Between-Class Learning for Fault Localization

Yan Lei[1,2*], Chunyan Liu[1], Huan Xie[1], Sheng Huang[1,2], Meng Yan[1,2], Zhou Xu[1,2]
[1]School of Big Data and Software Engineering, Chongqing University, Chongqing, China
[2]Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, China
Email:{yanlei, chunyanliu, huanxie, huangsheng, mengy, zhouxullx}@cqu.edu.cn

***Abstract*—Automated fault localization (FL) techniques collect runtime information as input data and then analyze input data to identify the relationship between program statements and failures. They usually take advantages of the statistics of the input data to develop a suspiciousness evaluation methodology (e.g., spectrum-based formulas and deep neural network models) by exploring the underlying correlation rooted in the input data. Thus, the quality of input data is critical for FL. In the actual process of development, developers seek to generate adequate test cases for testing the function or the robustness of a subject program. However, regarding a fault, most test cases are passed test cases and a very few ones are failed test cases since a very small portion of inputs in input domain will lead to a program failure. It means that FL usually faces a problem of imbalanced data, and this problem has been proven to pose an adverse effect on FL effectiveness.**

To address this problem, we propose BCL-FL: a data augmentation approach based on between-class learning, which produces new synthesized failed test samples by mixing two classes of real test cases (i.e., a passed test case and a failed one) with a random ratio. Specifically, BCL-FL uses the characteristics of real failed test cases to design a data synthesis formula suitable for failed test samples, which can make the synthesized failed test samples closer to real test cases. Since the synthesized data is different from real data, we ingeniously assign a continuous value between 0 and 1 to label the synthesized sample according to the mixing ratio of original labels. We take the synthesized failed test samples and the original test cases as the balanced input data for FL techniques to address the imbalanced data problem. To evaluate the effectiveness of BCL-FL, we conduct large-scale experiments on 287 faulty versions of eight large-sized programs (from ManyBugs and Defects4J) using six state-of-the-art FL approaches. The experimental results show that BCL-FL significantly improves the effectiveness of existing FL techniques, e.g., BCL-FL improves the CNN-FL approach in Top-1, Top-5, and Top-10 by 150%, 136.36%, and 193.1%, respectively.

***Index Terms*—fault localization, imbalanced data, between-class learning, data augmentation**

I. Introduction

Software debugging is an important but resource-intensive task in software engineering. At the same time, the ubiquity of bug fixes in software development and maintenance has led to high debugging costs. Automated Fault Localization (FL) techniques can reduce the time and energy of software developers in this process [1]. Therefore, FL techniques have gained popularity in the past few years [2]–[4].

*Corresponding author.

Fig. 1 shows a typical workflow of FL. As shown in Fig. 1, FL collects the coverage information and test results by running each test case in the test suite, and then denotes coverage information and test results as coverage matrix and labels, respectively. The coverage matrix and labels are **raw data** for FL. In the raw data, each row of the coverage matrix represents a test case, and each column corresponds to the coverage information of a statement in all test cases of a test suite. Each cell indicates whether a statement is executed by a test case or not. If the statement is executed, the value of this cell is 1; otherwise, its value is 0. For instance, y_i represents the label corresponding to the *i*-th test case, and x_{ij} represents the execution information of the *i*-th test case at *j*-th statement. Specifically, y_i=1, indicating that *i*-th test case is a failed test case, and y_i=0, indicating that *i*-th test case is a passed test case. x_{ij}=1 means that the *i*-th test case executes the *j*-th statement, and x_{ij}=0 means that the *i*-th test case does not execute the *j*-th statement. After acquiring the raw data, many FL techniques use them as the input for analyzing and evaluating the suspiciousness of each statement of being faulty, e.g., the two of the most popular FL techniques: spectrum-based fault localization (SBFL) [4], [5] and deep learning-based fault localization (DLFL) [6]–[8]).

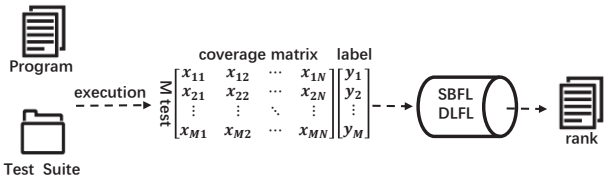

Fig. 1. **The overall workflow of FL.**

SBFL uses raw data as input, assigns a suspiciousness score to each statement through a well-designed formula [9], and then ranks all statements in terms of suspiciousness scores. Unlike the SBFL, DLFL uses a neural network model to learn the relationship between statements and program failures. After the training, the model also outputs a suspiciousness score to each statement, and also rank them in descending order with a ranking list of suspicious statements.

However, for SBFL and DLFL techniques, their effectiveness is affected by the quality of the data sets. Studies

978-1-6654-3786-8/22/$31.00 ©2022 IEEE
DOI 10.1109/SANER53432.2022.00045

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

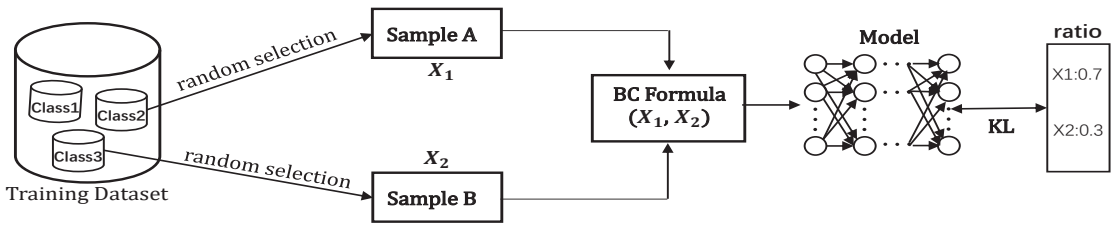

Fig. 2. **The overview architecture of BC Learning.**

[10]–[12] have shown that most of the existing data sets have various potential threats, such as data imbalance, data incompleteness, etc. As one of the most common problems, data imbalance may cause the FL approaches to be ineffective or even invalid. The problem of data imbalance means that a test suite in the real world usually includes a large number of passed test cases (i.e., a majority class) and a small number of failed test cases (i.e., a minority class). Zhang et al. [11] find that when the number of failed test cases is approximately equal to the number of pass test cases, fault localization can always achieve the best effectiveness.

Researches usually use two methods to solve the problem of imbalanced classes in the raw data, namely, reducing the number of passed test cases and increasing the number of failed test cases. Generally speaking, we adopt the method of increasing the number of failed test cases rather than reducing the number of passed test cases, because the latter method will lose some essential information that may be beneficial to FL. However, increasing the number of failed test cases in the real world is a difficult task even impossible because just a very small portion of inputs will cause a program failure [13]–[16]. Inspired by the widely used data augmentation in deep learning [11], [17]–[19], we may produce new synthesized failed samples, rather than generate new real failed test cases, to address the data imbalance problem.

When we consider the use of data augmentation to solve the imbalance of categories in the raw data, we find that it fits the characteristics of the two different categories in *between-class learning* (BC Learning). We take each test case in the raw data as a sample and its corresponding test result as a label. In other words, each vector of the coverage matrix is a sample, each sample is composed of features (i.e., statements), and the test result corresponding to each vector is the label of the sample. It should be noted that if we use the idea of data augmentation to address the imbalanced data problem, the synthesized failed test samples added to the raw data are not the real failed test cases, but the synthesized test samples with the feature of the failed test cases, i.e., synthesized samples with a failed label.

BC Learning is one of the widely-used deep learning approaches (as shown in Fig. 2). Its powerful learning ability and data augmentation ability are applied to various fields [20]–[24]. For example, in speech recognition, Yuji et al. [20] use BC Learning to learn the discriminative feature space by recognizing the sounds between classes. Specifically, they create the most suitable mix formula for speech recognition and use BC Learning to train the deep learning model. Their experimental results clearly show that BC Learning improves the performance of various speech recognition networks, data sets, and data augmentation approaches, and even surpasses human performance in speech classification tasks. Later in [21], they proved that BC Learning also performed well on images. These two studies show that the power of BC Learning is not limited to a certain data modality. And more importantly, BC Learning can mix two classes of data and output the mixing ratio of two different classes of data. We might as well regard BC Learning as an effective data augmentation approach.

Therefore, we propose BCL-FL: an effective data augmentation approach based on BC Learning. Our approach uses existing test cases to synthesize comprehensive test samples, thereby improving the data imbalance in fault localization. We conduct large-scale experiments on eight large-sized programs from the real world, and compare BCL-FL with six state-of-the-art FL techniques. Experimental results show that BC Learning improves the performance of six FL techniques. We have demonstrated that BCL-FL is expected to play a role in various FL techniques. For instance, under the top-1, top-5, and top-10 metrics, it can increase the baseline of CNN-FL by 150%, 136.36%, and 193.10%.

The main contributions of this paper can be summarized as:

- To the best of our knowledge, this is the first study to apply BC Learning-based architecture for data augmentation in fault localization.
- We propose a data augmentation approach that can effectively solve the imbalance problem. Our approach is designed to the widely used raw data in many FL techniques and can serve as a universal data augmentation for a wide spectrum of FL techniques.
- We conduct large-scale experiments on six state-of-the-art FL techniques, showing that BCL-FL significantly improves fault localization effectiveness.

The remainder of this paper is organized as follows. Section II introduces background information. Section III presents our approach BCL-FL. Section IV and Section V show the experimental results and discussion. Section VI discusses related work and Section VII draws the conclusion.

II. Background

In Section I, we focus on the imbalance of the test suite and its adverse effects on the fault localization approaches. We

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

have found that in the existing studies, there have been a large number of approaches that have tried to solve the problem of class imbalance in raw data and have achieved effective results. For example, researchers use general approaches of flip, rotation, scale, crop, and translation [25]–[28], or use deep learning for data augmentation, such as variational autoencoders, and convolutional neural networks [11], [18], [19], [29], [30].

We focus on data augmentation in deep learning. Inspired by [20], [21], we find that the preliminary work of BC Learning can be regarded as an approach for data augmentation. Fig. 2 shows the process of BC Learning. For instance, suppose a data set containing multiple sounds, randomly extracted two sounds from the data set, mix the data in a unique BC Learning approach. BC Learning has the easiest way to mix such as Eq. 1, it is the original formula for BC Learning. The mixed speech is used as the input of the BC model. The BC model learns the characteristics of each sound through input samples and outputs the mixing ratio of each sound.

What caught our attention is that when BC Learning performs well on speech recognition, it simulates the mixed speech stage in the real scenario (as shown in Fig. 2). We can abstract the mixed mode of BC Learning into the following steps: first, randomly select two samples from the training set of the BC Learning model, sample A, and sample B, and then construct the mixing formula (A, B) according to the characteristics of the sound (such as amplitude, frequency, etc.). If you don't want to design formula, you can also use the origin mixed formula in BC Learning (such as Eq.1). Substituting A and B into the formula to obtain a synthesis category, the synthesis category has the characteristics of a mixed category in the real world. Research [20] [21] separately used BC Learning to mix sounds and images. In [21], Tokozume et al. believe that BC Learning is equally effective in image cognization. They first design an image mixing formula for mixing two types of images, then use BC Learning to learn the mixing ratio of the output image, and finally verify the effectiveness of BC Learning on the image.

Observing the above two studies, we find that BC Learning can retain mixed data with real-world characteristics, which servers a potential solution to data augmentation. Thus, we observe that the data synthesis approach used in BC Learning is also a data augmentation approach. We try to use the BC formula to synthesize failing test samples with failed test features to enhance the weaker part of the FL raw data set and improve the imbalance of the test suite.

In summary, data synthesis approach used in BC Learning is also a data augmentation approach. We try to use the BC formula to enhance the unbalanced fault localization feature of the weaker part of the data set. On the other hand, the output of the BC Learning model is the mixing ratio of each class, which has nothing to do with our purpose. We only use mixed data in the first half of the BC Learning process. Therefore, BC Learning cannot be directly used for fault localization. We use the core idea of BC Learning to construct a data synthesis formula that matches the data requirements of the fault localization approach. Section III depicts our approach in detail.

Algorithm 1: BC data synthesis algorithm

Input:
collected M*N matrix:X
the label corresponding to each vector m in the matrix: Y
Output:
K*N synthesis matrix: X'

```
for i=0;i ≤ N;i++ do
    calculate the number of y=1:Count=∑_{i=1}^{N} y_i;
end
S=M;
while Count ≠ (the number of y=0 vectors) do
    j=random.randint (0,S);
    if y_j=1 then
        if ( h=random.randint (0,M)) && (h ≠ j ) then
            f=random.randint (0,N);
            X[M+1]=synthesis (X[j],X[h]));
            Y[M+1]=BC Formula (Y[j],Y[h]));
            M++;
            Count++;
        end
    end
end
Function synthesis(X[j],X[h]):
    F ⟵ X[j]; S ⟵ X[h];
    n=random.rand(0,N);
    for i=1;i ≤ N;i++ do
        if i ≤ n then
            H[i]=F[i];
        else
            H[i]=S[i]
        end
    end
    return H;
End Function
Function BC Formula(Y[j],Y[h]):
    F ⟵ Y[j]; S ⟵ Y[h];
    α=random.rand(0,1);
    β=random.rand(0,1);
    H=(αsumF)/(αsumF+βsumS)
    return H;
End Function
```

III. Approach

A. Overview

Given a faulty program *P* with *N* statements, it is executed by the test suite *T* with *M* test cases, which contains at least one failed test case. We collect the execution information of all test cases and obtain the $M \times N$ coverage matrix. As shown in Fig. 3, We use *X* to represent the coverage matrix

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

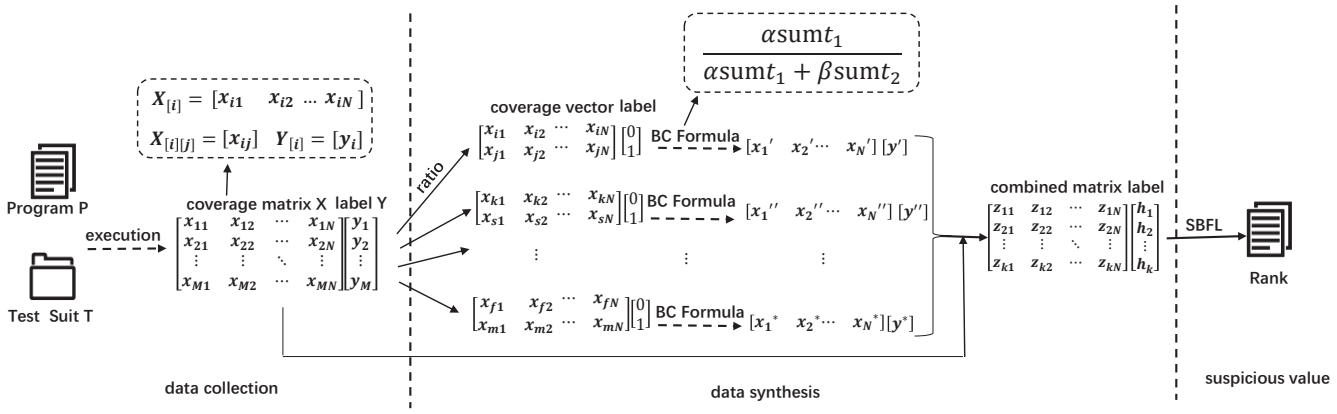

Fig. 3. **The pipeline of BCL-FL.**

of $M \times N$. Each row of coverage matrix *X* is represented by vector $X_{[i]}$, which corresponds to the execution information of the *i*-th test case, $i \in \{1, 2, ..., M\}$. Each column of the coverage matrix X represents a statement of the program. For example, $X_{[i][j]}$ represents the *j*-th statement of the *i*-th vector, $i \in \{1, 2, ..., M\}$, $j \in \{1, 2, ..., N\}$. For instance, $X_{[i][j]}$ = 1, indicating that vector $X_{[i]}$ executes statement *j*. $X_{[i][j]}$ = 0, indicating that vector $X_{[i]}$ does not execute statement *j*. We record the results of the vector (i.e., pass and fail) as the label corresponding to the vector. The label $Y_{[i]}$ of the vector $X_{[i]}$ corresponds to the label of the *i*-th test case, $i \in [0, 1]$. $Y_{[i]}$ = 0, indicating that the execution of vector $X_{[i]}$ passed. $Y_{[i]}$ = 1 indicates that the execution of vector $X_{[i]}$ failed. This $M \times N$ matrix *X* and the label corresponding to each test case is the input of our approach.

Fig. 3 shows the pipeline of our approach, which is divided into three stages. The first stage is data collection, collecting information about test cases executed in the program, including coverage matrix and execution results. In the following description, we also refer to test cases as coverage vectors. The second stage is the main stage of our approach. At this stage, we randomly select two coverage vectors from the collected coverage vectors. In particular, at least one of the two selected cover vectors should be a cover vector with the label of 1. Subsequently, we will use the data synthesis formula to synthesize a new fault coverage vector from the two extracted coverage vectors. The synthesis approach will be described in detail in Section III-B. When the class is balanced, the integration process ends. These synthesized fault coverage vectors contain the characteristics of real fault coverage vectors. Finally, we obtain a special coverage matrix by combining the synthesized coverage vectors with the initially collected matrix. The third stage is the calculation of the suspiciousness value. In this stage, we take the new coverage matrix obtained in the second stage as input, use the suspiciousness score calculation formula to calculate the suspiciousness score of each statement and obtain a list of suspicious statements. Next, we will describe the above steps in detail.

B. BC synthesis formula

We formally show the steps of BCL-FL in Fig. 3. Algorithm 1 describes the BC Learning process of data synthesis in detail. We analyzed the research of Tokozume et al. [20], [21]. We believe that the best aspect of BC Learning is the designed data synthesis formula, not the model used for training. Excellent data synthesis formula can make the synthesized data have real data characteristics, which is more conducive to the training of the model. We concentrate on how to design a data synthesis formula, which we call the BC Formula, which comes from the origin data synthesis formula in BC Learning (such as formula 1).The BC Formula is mainly used for the synthesis of label data.

$$mix = rt_1 + (1 - r)t_2 \tag{1}$$

In Eq. 1, t1 and t2 are two different types of labels, and r is a random ratio automatically generated by the program $r \in$ (0,1). These two labels can be mixed simply by Eq. 1. However, according to research [20], using only Eq. 1 does not guarantee the best synthesized data. Tokozume et al. chose 0 as the absolute center and substituted sound energy with distance. Created a well-thought-out sound synthesis formula in [21]. From the perspective of fault localization, we have improved the Eq. 1 and obtained a data synthesis formula that is more suitable for our approach, we call it BC Formula, As shown in Eq. 2:

$$BCFormula : Y_{[s]} = \frac{\alpha sumX_{[i]}}{\alpha sumX_{[i]} + \beta sumX_{[j]}} \tag{2}$$

The variables α and β in Eq. 2 are generated by the program and are random floating-point values, α,β,$\in$[0, 1]. $X_{[i]}$ refers to the execution information of the i-th test case randomly selected. $Y_{[s]}$ is the label corresponding to the synthesis vector. The specific vector synthesis steps will be described in detail in the following summary. sum$X_{[i]}$ is the sum of the partial coverage statements of the original vector intercepted for synthesis

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

In order to evaluate the effectiveness of the BC Formula, we designed another set of experiments through the following formula,we call it BCS Formula:

$$BCSFormula : Y_{[s]} = rsumX_{[i]} + (1-r)sumX_{[j]} \quad (3)$$

The experimental results will be explained in detail in Section IV.

C. Synthesizing new test samples using BC Formula

This section explains how to use the BC Formula to synthesize two vectors in detail, as shown in Fig. 4. Note that what we synthesize here is not real-world test cases, but synthesized test samples with features of real-world test cases. In the data synthesis stage, a total of two parts of data need to be synthesized: the synthesis of two randomly selected vectors $X_{[s]}$ and $X_{[k]}$, and the synthesis of labels $Y_{[s]}$ and $Y_{[k]}$ corresponding to $X_{[s]}$ and $X_{[k]}$ respectively. The vector synthesis steps are as follows: a) Calculate the count of the number of effectively executed statements in the vector $X_{[s]}$ and vector $X_{[k]}$. The weight of each statement that causes the program to failure is represented by ratio, ratio=$\frac{1}{count}$. Among them, because vector $X_{[k]}$ is a failure vector, statements executed in vector $X_{[k]}$ but not executed in vector $X_{[s]}$ are more likely to cause program failure than other statements. We count the number of statements above and mark it as count', and the corresponding statement ratio=$\frac{1}{count'}$. Then we update the ratio of other statements in the vector to $\frac{1}{count-count'}$. Statements not executed in the vector ratio=0. b) We get a random number n, $n \in$ (1, N) where N is the total number of statements in each vector. Our composite vector is composed of the (1, n) part of $X_{[s]}$ and the (n, N) part of $X_{[k]}$, and then we calculate the label value corresponding to the composite vector according to the BC Formula. c) The parameter values we need to apply the BC Formula are randomly generated floating point numbers α and β, (α, β)$\in$ (0,1). The sum parameters sumvector$X_{[s]}$ and sumvector$X_{[k]}$ by:

$$sumvectorX_{[m]} = \sum_{i=a}^{b}(ratio) \quad (4)$$

In Eq. 4, a is the starting point of the intercepting position of the vector$X_{[m]}$, and b is the ending point of the intercepting position of the vector$X_{[m]}$. Substituting the prepared parameters into the BC Formula, the synthesis label value of the vector$X_{[d]}$ is obtained. d) Put the synthesized vector and label into the original test suites to complete the synthesis of a test case.

Take Fig. 4 as an example to demonstrate the steps of synthesizing vectors. The vector $X_{[s]}$ (1,1,1,1,0,1,0,1,0,1), the vector $X_{[k]}$ (1,1,1,1,1,1,1,1,1,1) corresponds to the label $Y_{[s]}$ =[0], $Y_{[k]}$=[1]. We assign weights to each statement in the two vectors. $ratio_{[s]}$=$\frac{1}{7}$=0.14. $ratio_{[k]}$'=$\frac{1}{3}$=0.33,$ratio_{[k]}$=$\frac{1}{7}$=0.14. b) Suppose we get a random value n=6, we synthesize a new vector$X_{[d]}$ (1,1,1,1,0,1,1,1,1,1). c) Suppose we randomly generate floating-point numbers α=0.4, β b=0.3. We get $Y_{[d]}$ (the label of the vector$X_{[d]}$), $Y_{[d]}$ =$\frac{0.4\times(0.14+0.14+0.14+0.14+0.14)}{0.4\times(0.14+0.14+0.14+0.14+0.14)+0.3\times(0.33+0.14+0.33+0.14)}$ = 0.498. d) We put the composite vector$X_{[d]}$ (1,1,1,1,0,1,1,1, 1,1)$Y_{[d]}$[0.498] into the raw test suites.

Finally, we summarize the steps of BCL-FL as follows:

(1) Use the coverage data and execution results of the test cases in Fig. 4 as samples to synthesize a new sample. BCL-FL randomly selects two test cases with inconsistent labels (also called two vectors) from the sample set.
(2) Input the selected two test cases and their corresponding labels into the BC Formula to obtain a synthesized test sample and a synthesized label. Add the synthesized test sample to the end of the original matrix. Repeat (1) and (2) until the label categories of the test suite are balanced.
(3) The new matrix that has dealt with the problem of data imbalance is used as the input of the fault localization approach. The fault localization approaches here include fault localization approaches based on spectrum and fault localization approaches based on machine learning. Sort the statements from high to low and output the rank.

IV. Empirical Evaluation

TABLE I
THE CHARACTERISTICS OF SUBJECT PROGRAMS

Program	Description	Versions	LoC (k)	Test
gzip	Data compression	5	491	12
libtiff	Image processing	12	77	78
python	General-purpose language	8	407	355
Chart	Java chart library	26	96	2205
Lang	Apache commons-lang	65	22	2245
Math	Apache commons-math	106	85	3602
Time	Standard date and time library	27	28	4130
Mokito	Mocking framework for Java	23	67	1075
total	-	287	1273	13702

To evaluate the effectiveness of our approach, we implemented BCL-FL into a pipeline that can rank statements based on their suspiciousness. We use six state-of-the-art FL techniques as the baselines to be compared with BCL-FL. Since BCL-FL presents a BC formula according to FL scenario, it is different from the pure original BC Learning formula (i.e., BCS Formula in Eq. 3). Some may argue that if we directly use the BCS formula, BCL-FL is whether still better than the one using BCS formula. Thus, we further propose and implement BCLS-FL using BCS formula and compare BCL-FL with BCLS-FL. To explain BCL-FL more clearly, we divide the experiments into the following three parts.

A. Datasets

We evaluate BCL-FL on real-world programs. The gzip, libtiff, and python are from ManyBugs[1]. The Chart, Lang,

[1]ManyBugs, http://repairbenchmarks.cs.umass.edu/manybugs/

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

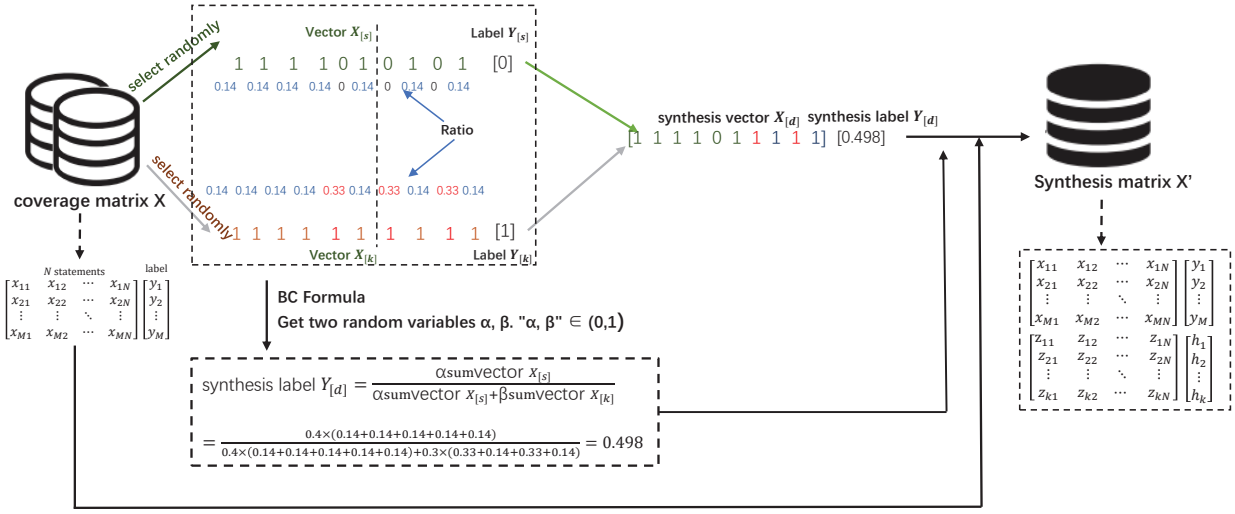

Fig. 4. **The process of data synthesis.**

Math, Time, and Mokito are from Defects4J[2]. The summary of experimental programs are shown in Table I.

B. Evaluation Metrics

For the evaluation process, we use the following widely used metrics:

- **Top-K:** This metric is also called Acc@k, Hit@k, etc. It represents the number of faulty versions of at least one fault statement in the first k (k=1, 5, 10,...) program modules. In previous studies, many studies took K=1, 5, and 10 [31]–[33]. Based on previous works, we assign values 1, 5, 10 to K for evaluation.
- **Mean Average Rank (MAR):** For a faulty version, it is the average position of all fault statements in the ranking list.
- **Mean First Rank (MFR):** This metric first calculates the rank of the first faulty statement of each version and then calculates the average of the ranks in the project.
- **Relative Improvement (RImp):** This metric can see the improvement of one fault localization approach relative to another fault localization approach. It is defined as follows:

$$RImp = \frac{MAR_{improved}}{MAR_{Origin}} \tag{5}$$

In our study, $MAR_{improved}$ represents the MAR value of approach after using BC Formula or BCS Formula.

C. Research Questions and Results

We evaluate the effectiveness of our approach through the following three research questions. We use six FL techniques that currently perform best in real faults [7], [34]. Including three SBFL approaches (i.e., Dstar, Ochiai, and Barinel) and three DLFL approaches (i.e., MLP-FL, CNN-FL, and RNN-FL). To evaluate the effectiveness of our proposed BC Formula, we compared it with BCS Formula (Eq. 3). The BCS Formula comes from the original formula by BC Learning. We call the data augmentation approach using BC Formula as BCL-FL, and the data augmentation approach using BCS Formula as BCLS-FL. We list the six FL techniques that we used as follows:

- The SBFL approaches we use **Dstar [35]**,**Ochiai [3]** and **Barinel [36]**. For more details, please refer to **[9]**.
- The DLFL approaches we use **MLP-FL [8]**, **CNN-FL [6]**, and **RNN-FL [7]**. MLP-FL, CNN-FL, and RNN-FL have no public source code, so we reimplement the model based on their papers.

RQ1. How effective is BCL-FL compared with the state-of-the-art SBFL?

As shown in Table II, we list three SBFL approaches (i.e., Dstar, Ochiai, and Barinel) under two scenarios: Origin (baseline) and BCL-FL. We use Top-K, MAR, and MFR metrics to compare the performance of BCL-FL and baseline. Many studies use K=1, 5, 10 when using the Top-K metric [31]–[33]. As a result, this paper use the Top-1, Top-5, and Top-10 evaluation metrics. For the convenience of reading, we bold the better experimental results in the tables. From Table II, we can find that BCL-FL performance is better than the baseline. Taking the libtiff data set as an example, when using the Top-1, Top-5, and Top-10 metrics of BCL-FL, BCL-FL has increased by 100%, 50%, and 33.33% respectively compared with the baseline of Dstar. Experimental results of Top-K show that BCL-FL can effectively improve the positioning accuracy of the baseline.

Fig. 6 shows the MAR metric of the six FL approaches in three scenarios (i.e., Origin, BCLS-FL, and BCL-FL), and Fig. 5 shows the MFR metric of the six FL approaches in

[2]https://github.com/rjust/defects4j

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

TABLE II
THE RESULTS OF TOP-1,TOP-5,TOP10 BY COMPARISON OF SBFL APPROACH AND BCL-FL.

program	Scenario	Top-1			Top-5			Top-10			MAR			MFR		
		Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel
Chart	origin	**3**	**3**	**2**	**9**	**9**	**9**	**12**	**14**	13	827.80	696.96	655.24	322.88	209.24	141.04
	BCL-FL	**3**	2	**2**	**9**	**9**	**9**	**12**	**14**	**14**	**815.12**	**659.57**	**654.77**	**245.68**	**138.28**	**116.36**
Lang	origin	5	5	**5**	24	24	24	38	38	**38**	81.45	60.25	61.31	29.14	30.77	33.17
	BCL-FL	**7**	**6**	4	**25**	**25**	**25**	**39**	**40**	**38**	**62.92**	**58.90**	**60.68**	**25.62**	**22.51**	**29.32**
Time	origin	**2**	**2**	**2**	**10**	**10**	**10**	**11**	11	**11**	**675.00**	874.30	875.33	398.63	598.96	600.78
	BCL-FL	**2**	**2**	**3**	**10**	**10**	8	**11**	**12**	9	683.61	**788.43**	**843.61**	**252.67**	**453.20**	**600.54**
Mokito	origin	**5**	**6**	**4**	13	13	11	15	15	15	439.06	398.61	401.95	254.39	**201.87**	211.89
	BCL-FL	3	5	**4**	**15**	**15**	**14**	**16**	**17**	**17**	**336.18**	**397.44**	**398.05**	**259.83**	233.66	**207.33**
Math	origin	**17**	14	17	38	34	39	**47**	42	**48**	315.70	209.44	68.24	59.84	72.63	212.71
	BCL-FL	16	**16**	**18**	**39**	**38**	**41**	45	**44**	47	**257.24**	**202.96**	**63.86**	**57.4**	**67.53**	**204.37**
gzip	origin	**0**	0	**0**	**1**	**1**	**1**	**2**	**1**	**1**	85.20	133.60	133.60	66.60	103.80	103.80
	BCL-FL	**0**	**1**	**0**	**1**	**1**	**1**	**2**	**1**	**1**	**45.50**	**98.50**	**101.25**	**21.90**	**71.00**	**97.00**
libtiff	origin	1	**1**	**1**	2	2	2	3	3	**3**	175.44	199.12	206.72	136.67	140.36	147.96
	BCL-FL	**2**	**1**	**1**	**3**	**3**	**3**	**4**	**4**	**3**	**123.76**	**145.64**	**195.28**	**57.28**	**69.00**	**142.00**
python	origin	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1243.63**	1456.01	1457.25	668.75	860.25	860.25
	BCL-FL	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	1306.53	**1448.38**	**1455.61**	**722.75**	**860.25**	**860.25**
total	origin	**33**	31	31	97	93	96	128	124	**129**	480.41	503.54	476.43	244.61	277.24	288.95
	BCL-FL	**33**	**33**	**32**	**102**	**101**	**101**	**129**	**132**	**129**	**453.85**	**474.97**	**471.63**	**205.39**	**239.43**	**282.15**

TABLE III
THE RESULTS OF TOP-1,TOP-5,TOP10 BY COMPARISON OF DLFL APPROACH AND BCL-FL.

program	Scenario	Top-1			Top-5			Top-10			MAR			MFR		
		MLP	CNN	RNN	MLP	CNN	RNN	MLP	CNN	RNN	MLP	CNN	RNN	MLP	CNN	RNN
Chart	origin	**2**	1	1	**7**	2	4	7	2	6	1015.64	1380.12	**826.36**	558.48	1001.24	351.84
	BCL-FL	**2**	**2**	**2**	**7**	**8**	**10**	**8**	**10**	**12**	**813.38**	**789.00**	827.54	**176.00**	**457.12**	**242.2**
Lang	origin	1	0	4	13	7	17	26	10	27	96.12	305.73	92.54	42.60	259.47	41.39
	BCL-FL	**6**	**1**	**8**	**25**	**16**	**25**	**39**	**24**	**42**	**53.84**	**121.02**	**71.17**	**17.68**	**66.46**	**17.32**
Time	origin	**0**	0	2	**0**	1	7	**1**	2	8	1265.89	2493.85	994.19	1031.49	2188.67	697.70
	BCL-FL	**0**	**1**	**3**	**0**	**4**	**9**	**1**	**7**	**11**	**1087.00**	**974.78**	**876.56**	**931.52**	**600.75**	**328.74**
Mokito	origin	**0**	**1**	2	4	4	5	4	4	5	607.56	717.64	**514.53**	378.08	499.44	369.81
	BCL-FL	**0**	**1**	**4**	**5**	**7**	**10**	**11**	**11**	**13**	**446.11**	**446.11**	593.64	**211.43**	**387.14**	**274.33**
Math	origin	**6**	4	**13**	23	7	29	27	8	35	337.21	554.62	399.08	319.52	410.82	155.87
	BCL-FL	4	**10**	12	**28**	**15**	**35**	**31**	**32**	**42**	**188.02**	**256.74**	**356.41**	**134.43**	**178.32**	**102.35**
gzip	origin	**0**	**0**	0	**0**	**0**	**1**	**1**	**0**	**2**	798.00	1029.00	673.00	116.60	145.60	100.00
	BCL-FL	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**2**	**514.00**	**668.00**	**553.00**	**102.00**	**112.00**	**86.30**
libtiff	origin	**0**	**0**	**1**	**1**	1	**2**	**2**	**3**	**3**	1264.20	1281.00	1143.80	**93.20**	120.80	83.80
	BCL-FL	**0**	**0**	**1**	**1**	**2**	**2**	1	1	**3**	**752.25**	**896.80**	**987.00**	117.40	**79.30**	**78.60**
python	origin	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**828.00**	867.25	668.75	**828.00**	867.25	668.75
	BCL-FL	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	1147.41	**811.30**	**657.31**	1060.25	**753.00**	**668.75**
total	origin	9	6	23	48	22	65	68	29	86	776.58	1078.65	664.03	421.00	686.66	308.64
	BCL-FL	**12**	**15**	**31**	**66**	**52**	**92**	**92**	**85**	**125**	**625.20**	**620.43**	**615.32**	**343.84**	**329.26**	**224.82**

three scenarios (i.e., Origin, BCLS-FL, and BCL-FL). In Fig. 6 and Fig. 5, the ordinate represents the FL approach, and the abscissa represents the MAR/MFR index under this approach.

Compare the MAR metric of the three SBFL approaches in the BCL-FL and Origin scenarios. From Fig. 6, we can find that in either approach, BCL-FL improves the baseline. Although BCL-FL has an improvement in some data sets, it is slight to improve. Perhaps it is because Dstar, Ochiai, and Barinel are already at the forefront of many bug versions of Defects4J. Considering the MFR metric, the result of BCL-FL is also better than the baseline of the six FL approaches. It shows that we can find all the faulty versions the fastest with the least resources when we use BCL-FL.

Summary for RQ1: *In RQ1, we discuss the effectiveness of the BCL-FL technique compared with the original SBFL approaches. Experimental results show that the FL approaches perform better in the BCL-FL scenario. It shows that BCL-FL is effective to improve the imbalance of a test suite.*

RQ2. How effective is BCL-FL compared with the state-of-the-art DLFL?

To answer RQ2, we use five metrics (Top-1, Top-5, Top10, MAR, and MFR) to compare BCL-FL with three state-of-the-art DLFL techniques (i.e., MLP-FL, CNN-FL, and RNN-FL).

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

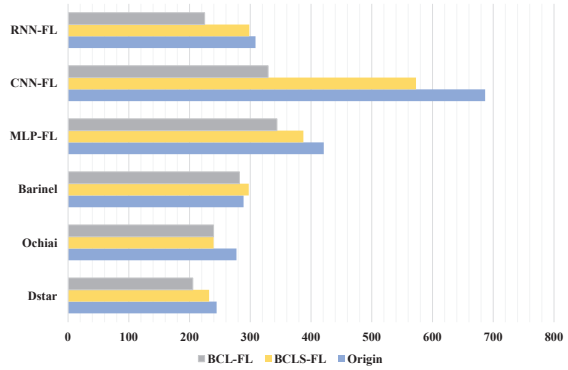

Fig. 5. **MFR values of Origin,BCLS-FL and BCL-FL approaches.**

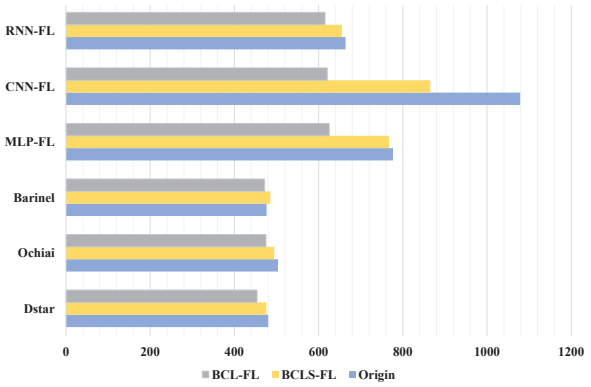

Fig. 6. **MAR values of Origin,BCLS-FL and BCL-FL approaches.**

As shown in Table III, MLP, CNN, and RNN in Table III represent MLP-FL, CNN-FL, and RNN-FL, respectively. From the table, we can find that the performance of CNN-FL is the most outstanding. CNN-FL under Top-1, Top-5 and Top-10 metrics were 15, 52, 85, with an increase of 150%, 136.36% and 193.1%. The other two DLFL approaches have also improved in the BCL-FL scenario. Compared with the baseline, RNN-FL increased by 34.78%, 41.54%, and 45.35%. MLP-FL increased by 33.33%, 37.5% and 35.29%. In comparison to SBFL approaches, we find that BCL-FL can achieve better performance in DLFL. We think this is because BCL-FL solves the problem of data imbalance from the perspective of deep learning, so BCL-FL may be more suitable for DLFL.

We analyzed the improvement of MAR and MFR. From Fig. 6, we can find that in either approach, BCL-FL performs better than the baseline. Take the CNN-FL approach as an example. At the origin, the MAR value of this approach is 1078.65, which means that on average, 1078.65 lines of code need to be checked to find all faulty versions. After using the BCL-FL approach for data augmentation, it is find that all the faulty versions only need to check 620.43 lines of code on average. An increase of 42.48% compared to the origin. From Fig. 5 we can find that the improvement of BCL-FL is relatively small in only one approach. The reason why BCL-FL performs well in DLFL may be because BCL-FL can synthesize robust negative samples, enabling the machine learning model to learn more reliable correlations, which is conducive to fault localization.

Summary for RQ2: *In RQ2, we discuss the effectiveness of BCL-FL compared with the original state-of-the-art DLFL techniques. Based on the experimental results of metrics on Top-K, MAR, MFR, we can think that BCL-FL is an effective DLFL data augmentation approach. BCL-FL can provide reliable failed test samples for DLFL. In addition, we find that BCL-FL is more effective for DLFL compared to SBFL.*

RQ3. How effective is BCL-FL compared with the pure original BC Learning approach (i.e., BCLS-FL)?

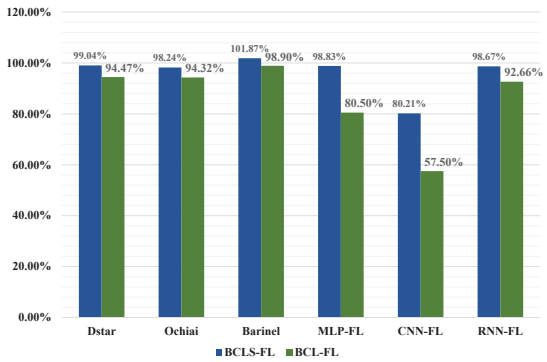

Fig. 7. **The comparison of BCLS and BCL-FL over original approach under RImp metric.**

TABLE IV
THE RESULTS OF TOP-1, TOP-5, TOP-10, MAR, AND MFR BY COMPARISON OF BCLS-FL AND BCL-FL.

Metric	Scenario	Dstar	Ochiai	Barinel	MLP-FL	CNN-FL	RNN-FL
Top-1	Origin	**33**	31	31	9	6	23
	BCLS-FL	32	32	31	9	**17**	29
	BCL-FL	**33**	**33**	**32**	**12**	15	**31**
Top-5	OriginL	97	93	96	48	22	65
	BCLS-FL	95	94	98	54	33	62
	BCL-FL	**102**	**101**	**101**	**66**	**52**	**92**
Top-10	Origin	128	124	**129**	68	29	86
	BCLS-FL	**129**	128	**129**	74	55	84
	BCL-FL	**129**	**132**	**129**	**92**	**85**	**125**
MAR	Origin	480.41	503.54	476.43	776.58	1078.65	664.03
	BCLS-FL	475.80	494.70	485.34	767.52	865.16	655.23
	BCL-FL	**453.85**	**474.97**	**471.63**	**625.20**	**620.43**	**615.32**
MFR	Origin	244.61	277.24	288.95	421.00	686.66	308.64
	BCLS-FL	232.16	239.58	297.42	387.42	572.82	298.31
	BCL-FL	**205.39**	**239.43**	**282.15**	**343.84**	**329.26**	**224.82**

To prove the validity of the BC formula, we proposed the BCLS-FL approach to compare BCL-FL. BCLS-FL uses the

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

origin formula (Eq. 3) in BC Learning to synthesize data. *RQ3* is proposed to verify that BC Formula is more suitable for data augmentation of fault localization than the pure original BC Learning approach (i.e., BCS Formula).

First, we experiment with the original approach in the BCLS-FL scenario, then we compare BCL-FL with BCLS-FL and the baseline. BCLS-FL still uses six FL approaches (i.e., Datar, Ochiai, Barinel, MLP-FL, CNN-FL, RNN-FL) to experiment on eight programs in the real world. Table IV shows the experimental results of BCLS-FL, BCL-FL, and baseline under Top-1, Top-5, Top-10, MAR, and MFR metrics. In particular, we counted the overall metrics of all faulty versions (including ManyBugs and Defects4J, a total of 287 faulty versions). Looking at Table IV, we find that BCLS-FL is slightly better than the baseline overall. The BCLS-FL seems to have a good performance is because the BCS formula has certain data synthesis capabilities. For example, from the MFR and Top-K metrics, the six approaches in the BCLS-FL scene are better than the baseline. In the CNN-FL approach under the Top-1 metric, BCLS-FL is 200% better than the baseline. However, the performance of BCLS-FL under the MAR metric is not satisfactory. It may be because the origin BC Learning approach requires long-term training of the model [21], and our parameters may need to be improved.

Although BCLS-FL has improved over the baseline under the Top-K index, it is still inferior to BCL-FL. As seen from Table IV, all the results of BCL-FL under Top-1, Top-5, and Top-10 are better than BCLS-FL. It means that under the same number of code lines, BCL-FL can locate more bugs than BCLS-FL. Although BCLS-FL has improved over the baseline under the Top-K index, it is still inferior to BCL-FL.

We also analyzed the distribution of BCLS-FL and BCL-FL under the MAR and MFR metrics. Table IV shows that BCL-FL is better than BCLS-FL in all approaches. Especially under the CNN-FL approach, BCL-FL needs to check 620.43 lines on average to find all the faulty versions, and BCLS-FL needs to check 865.16 lines on average. BCL-FL increases by 38.32% compared with BCLS-FL.

To evaluate the effectiveness of BCLS-FL and BCL-FL more fairly, we also calculated the RImp value of BCLS-FL and BCL-FL (as shown in Fig. 7). The abscissa of Fig. 7 represents the RImp value, and the ordinate represents the approach. In Fig. 7, we compare the comparison results of Dstar, Ochiai, Barinel, CNN-FL, RNN-FL, and MLP-FL in the BCLS-FL and BCL-FL scenes. Observing this graph, we can conclude that BCL-FL has the best performance. Taking the CNN-FL approach as an example, the RImp value of BCLS-FL is 80.21%, and the RImp value of BCL-FL is 57.50%. Compared with BCLS-FL, BCL-FL increases CNN-FL by 22.71%. In addition, BCL-FL increased 18.33% on MLP-FL. Therefore, a good data synthesis formula is more conducive to locating bugs than the origin BC mixing approach.

Summary for RQ3: *In RQ3, we compare BCL-FL with BCLS-FL technique using different data mixing formulas. The design of the BC Formula comes from the characteristics of the coverage matrix in fault localization. Considering that in the coverage matrix, statements executed only in failed test cases are more suspicious, we use the BC Formula to assign them higher weights whereas this is not possible with the BCS Formula. Based on all the experimental results, BCL-FL performs better than BCLS-FL. This shows the validity of the BC Formula we designed.*

V. Discussion

A. Threats to Validity

The randomness of BC Learning in BCL-FL. Although our proposed BCL-FL approach has achieved much improvement, there still exist some threats. BC Learning itself has some randomness. To ensure the fairness of the selected data synthesis vector, we obtain two vectors by random selection. When designing the BC Formula, to make the synthesized test case closer to the test case generated in the real world, we also randomly generated the values of α and β. However, this is an inevitable threat to BC Learning itself, the combination of multiple random factors may lead to random and unrepeatable experimental results. To deal with this situation, we follow the convention strategy by conducting 10 experiments on each faulty version. The final recorded experimental results are the average of 10 experiments.

Implementation of baselines and our approach. Our implementation of the baseline may not be entirely correct. For the Dstar, Ochiai, and Barinel approaches, we reimplemented them according to the formulas in their papers and manually verified the correctness of the experimental results. But for the deep learning-based fault localization approach, we cannot guarantee that the results are in full compliance with the results in their paper. Due to the use of deep learning-based fault localization approaches, the results may involve a degree of randomness. On the other hand, we cannot guarantee that our code is completely correct because building a deep learning model requires many parameters, but we cannot get the parameter values used by the authors of the paper.

The validity of the generalizability. The data sets used in our experiments come from the publicly available (ManyBugs and Defects4J). Although the subject programs selected in our experiments are all from the real world and our approach performs well on these programs, it may not apply to other programs. This is because no data set can contain all types of errors. In addition, our approach is to solve the problem of data imbalance. Thus, for balanced data sets, our approach may not work and may even be counterproductive. In addition, we only synthesizes data from the same program, and does not synthesize data from different programs.

B. Reasons for BCL-FL is effective

The reasons why BC Formula (Eq. 2) is superior to the BCS Formula (Eq. 3) are as follows: a) Each covered statement in every failed test case has different contributions to a program

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

failure, BC Formula assigns different weights for each covered statement. b) We consider the influence of the label value. Each synthesized test sample does not necessarily pass or fail. If we make the label 0 or 1, the performance of the approach will deteriorate. We use continuous values instead of discrete values to better reflect the characteristics of the synthesized test samples, so we also calculated the value of the label. For example, when the label of a synthesized test sample is 0.78, it has a 78% chance of causing the program to fail.

We have studied the choice of the raw data of synthesized test cases. Here, we found that for the synthesized test samples with the best experimental result, at least one of its raw test cases sources should be a vector with a label of 1. The combination of the two passing vectors has almost no effect on performance. The features that are more meaningful to the experimental results are those statements that are more likely to cause the program to fail. We focus on how to highlight these fault-relevant characteristics.

VI. Related Work

A. Spectrum-based Fault Localization

Spectrum-based fault localization (SBFL) [2], which some people call coverage-based fault localization (CBFL), has been intensively studied due to its effectiveness and lightweight. Typical SBFL techniques (e.g., Tarantula [13], Ochiai [3], DStar [35], Jaccard [5]) first dynamically execute test cases and collect coverage information and execution results. Then perform statistical analysis on the coverage data, calculate the suspicious scores of the code elements, and finally sort the code elements in descending order of the suspicious scores. The higher the ranking, the more likely the element is a bug. Although SBFL has been already successful, it is also not sufficient. Many researchers have enhanced the effectiveness of the SBFL approaches by changing the way the program spectrum is constructed [37], [38] and improving the quality of the test suites [39], [40].

B. Deep-Learning-based Fault Localization

Deep-Learning-Based Fault Localization is an approach based on a deep neural network model. From the perspective of deep learning, this approach uses the advantages of deep neural network to solve unresolved problems in software fault localization. Wang et al. proposed a fault localization approach BPNN-FL [41] based on BP neural network, using BP neural network model as a pipeline for learning input and output relationships, inputting the execution information of test cases, and outputting the sorting of suspicious statements. Then, Wong *et al.* improved the effectiveness of the BPNN-FL approach through program slicing technology [40]. Similar to Wong's idea, using raw data directly as training data, Zhang *et al.* proposed CNN-FL [7] and MLP-FL [7], and Lou *et al.* proposed Grace [42].Because the performance of the machine learning model is affected by the quality of the training data, some research work is different from BPNN-FL. They preprocess the training data before entering the model, such as resampling the unbalanced data set [40], and weighting the test case information [43] to synthesized test cases [39] to generate failed test case [44].

C. Between-Class Learning

BC Learning is widely used in various fields [20]–[22]. In standard BC Learning in sounds recognition, two different sounds are first selected randomly, and then the two sounds are mixed using an origin formula from BC Learning. Then the mixed sound is input to the BC Learning model, and the training model outputs the mixing ratio of the two sounds. BC Learning's strength is in its ability to recognize the features of mixed data, thus it's been widely applied in various industries. In [45], BC learner is quite powerful. As an abstract model for learning computable functions, the BC learner accepts increasing quantities of unknown function data and provides a sequence of hypotheses to the program for the function at hand [45]. Inspired by high performance of BC Learning in the field of voice recognition, Tokozume et al. successfully applied BC Learning in the field of image recognition [21].

VII. Conclusion

In this paper, we propose BCL-FL, which is a data augmentation approach that uses the BC Learning to solve the imbalance problem of the test suite and improve the effectiveness of FL techniques. Our main ideas include: (1) treating the coverage matrix and the result of test cases as samples and labels; (2) using data augmentation approaches to solve the problem of class imbalance and enhance the features of minority class; (3) using BC Learning to design the formula suitable for the synthesis of failed test samples on fault localization; (4) changing the label from discrete values to continuous values.

Specifically, we use the BC Formula to synthesize new failing test samples that meet the characteristics of the real scenarios. To make the label of the synthesized test sample better reflect the possibility of the test sample causing the program to failure, we increase the scope of the label. We have implemented our approach and integrated it into the FL pipeline. In addition, we evaluated BCL-FL on ManyBugs and Defects4J, and experimental results show that our approach is statistically superior to six FL baselines.

In future work, we intend to use more thematic procedures and replace the BC Formula with more powerful data augmentation approaches.

Acknowledgment

This work is partially supported by supported by the National Key Research and Development Project of China (No. 2020YFB1711900), the National Defense Basic Scientific Research Project (No. WDZC20205500308), the Chongqing Science and Technology Plan Projects (No. cstc2018jszx-cyztzxX0037), the Key Project of Technology Innovation and Application Development of Chongqing (No. cstc2019jscx-mbdxX0020), the National Natural Science Foundation of China (No. 62102054), and the Open Foundation of Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education of China (CPSDSC202004).

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

REFERENCES

[1] J. S. Collofello and S. N. Woodfield, "Evaluating the effectiveness of reliability-assurance techniques," *Journal of systems and software*, vol. 9, no. 3, pp. 191–195, 1989.

[2] B. Liblit, "Cooperative bug isolation, doctor of philosophy dissertation," *University of California, Berkeley*, 2004.

[3] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "An evaluation of similarity coefficients for software fault localization," in *2006 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06)*. IEEE, 2006, pp. 39–46.

[4] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on software engineering and methodology (TOSEM)*, vol. 20, no. 3, pp. 1–32, 2011.

[5] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "On the accuracy of spectrum-based fault localization," in *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*. IEEE, 2007, pp. 89–98.

[6] Z. Zhang, Y. Lei, X. Mao, and P. Li, "Cnn-fl: An effective approach for localizing faults using convolutional neural networks," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 445–455.

[7] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and X. Zhang, "A study of effectiveness of deep learning in locating real faults," *Information and Software Technology*, vol. 131, p. 106486, 2021.

[8] W. Zheng, D. Hu, and J. Wang, "Fault localization analysis based on deep neural network," *Mathematical Problems in Engineering*, vol. 2016, 2016.

[9] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[10] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?" in *Proceedings of the 2011 international symposium on software testing and analysis*, 2011, pp. 199–209.

[11] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and J. Wen, "Improving deep-learning-based fault localization with resampling," *Journal of Software: Evolution and Process*, vol. 33, no. 3, p. e2312, 2021.

[12] L. Zhang, L. Yan, Z. Zhang, J. Zhang, W. Chan, and Z. Zheng, "A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization," *Journal of Systems and Software*, vol. 129, pp. 35–57, 2017.

[13] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 273–282.

[14] T.-D. B. Le and D. Lo, "Will fault localization work for these failures? an automated approach to predict effectiveness of fault localization tools," in *2013 IEEE International Conference on Software Maintenance*. IEEE, 2013, pp. 310–319.

[15] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 273–282.

[16] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *Journal of Systems and Software*, vol. 83, no. 2, pp. 188–208, 2010.

[17] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[18] K. Lata, M. Dave, and N. KN, "Data augmentation using generative adversarial network," in *Proceedings of 2nd International Conference on Advanced Computing and Software Engineering (ICACSE)*, 2019.

[19] K. Wang, A. Sullivan, D. Marinov, and S. Khurshid, "Fault localization for declarative models in alloy," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 391–402.

[20] Y. Tokozume, Y. Ushiku, and T. Harada, "Learning from between-class examples for deep sound recognition," *arXiv preprint arXiv:1711.10282*, 2017.

[21] Y. Tokozume, Y. Ushiku, and T. Harada, "Between-Class Learning for Image Classification," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008.

[22] J. Case, S. Jain, and F. Stephan, "Vacillatory and bc learning on noisy data," in *International Workshop on Algorithmic Learning Theory*. Springer, 1996, pp. 285–298.

[23] N. Japkowicz, "Concept-learning in the presence of between-class and within-class imbalances," in *Conference of the Canadian society for computational studies of intelligence*. Springer, 2001, pp. 67–77.

[24] Y. Wu, W. Jin, Y. Li, and D. Wang, "A novel method for simultaneous-fault diagnosis based on between-class learning," *Measurement*, vol. 172, p. 108839, 2021.

[25] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[27] I. Sato, H. Nishimura, and K. Yokoi, "Apac: Augmented pattern classification with neural networks," *arXiv preprint arXiv:1505.03229*, 2015.

[28] L. Wan, M. Zeiler, S. Zhang, Y. Le Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *International conference on machine learning*. PMLR, 2013, pp. 1058–1066.

[29] S. Mun, S. Park, D. K. Han, and H. Ko, "Generative adversarial network based acoustic scene training set augmentation and selection using svm hyper-plane," *Proc. DCASE*, pp. 93–97, 2017.

[30] F. Zhou, S. Huang, and Y. Xing, "Deep semantic dictionary learning for multi-label image classification," *arXiv preprint arXiv:2012.12509*, 2020.

[31] Y. Küçük, T. A. Henderson, and A. Podgurski, "Improving fault localization by integrating value and predicate based causal inference techniques," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 649–660.

[32] F. Feyzi, "Cgt-fl: using cooperative game theory to effective fault localization in presence of coincidental correctness," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3873–3927, 2020.

[33] S. R. Heris and M. Keyvanpour, "Effectiveness of weighted neural network on accuracy of software fault localization," in *2019 5th International Conference on Web Research (ICWR)*. IEEE, 2019, pp. 100–104.

[34] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating and improving fault localization," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, 2017, pp. 609–620.

[35] W. E. Wong, V. Debroy, Y. Li, and R. Gao, "Software fault localization using dstar (d*)," in *2012 IEEE Sixth International Conference on Software Security and Reliability*. IEEE, 2012, pp. 21–30.

[36] R. Abreu, P. Zoeteweij, and A. J. Van Gemund, "Spectrum-based multiple fault localization," in *2009 IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2009, pp. 88–99.

[37] V. Dallmeier, C. Lindig, and A. Zeller, "Lightweight defect localization for java," in *European conference on object-oriented programming*. Springer, 2005, pp. 528–550.

[38] C. Yilmaz, A. Paradkar, and C. Williams, "Time will tell," in *2008 ACM/IEEE 30th International Conference on Software Engineering*. IEEE, 2008, pp. 81–90.

[39] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 169–180.

[40] W. E. Wong and Y. Qi, "Bp neural network-based effective fault localization," *International Journal of Software Engineering and Knowledge Engineering*, vol. 19, no. 04, pp. 573–597, 2009.

[41] W. E. Wong, L. Zhao, Y. Qi, K.-Y. Cai, and J. Dong, "Effective fault localization using bp neural networks." in *SEKE*. Citeseer, 2007, pp. 374–379.

[42] Y. Lou, Q. Zhu, J. Dong, X. Li, Z. Sun, D. Hao, L. Zhang, and L. Zhang, "Boosting coverage-based fault localization via graph-based representation learning," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2021, pp. 664–676.

[43] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 273–282.

[44] J. Röβler, G. Fraser, A. Zeller, and A. Orso, "Isolating failure causes through test case generation," in *Proceedings of the 2012 international symposium on software testing and analysis*, 2012, pp. 309–319.

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.

[45] F. Stephan and S. A. Terwijn, "Counting extensional differences in bc-learning," in *International Colloquium on Grammatical Inference*. Springer, 2000, pp. 256–269.

Authorized licensed use limited to: CHONGQING UNIVERSITY. Downloaded on December 16,2022 at 03:13:08 UTC from IEEE Xplore. Restrictions apply.