IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON INTELLIGENT INFORMATION SERVICES

Received December 20, 2019, accepted January 3, 2020, date of publication January 14, 2020, date of current version January 23, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2966540

Fault Localization With Weighted Test Model in Model Transformations

PENGFEI LI, MINGYUE JIANG, (Member, IEEE), AND ZUOHUA DING, (Member, IEEE)
School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China

Corresponding author: Zuohua Ding (zouhuading@hotmail.com)

This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grant 61210004, Grant 61170015, and Grant 61802349, and in part by the Fundamental Research Funds of Zhejiang Sci-Tech University under Grant 17032184-Y and Grant 2019Q041.

ABSTRACT Model transformations and model-driven engineering (MDE) have been applied widely in service-oriented architecture based information systems. To support the development of such a service-oriented information system, it is necessary to guarantee the quality of model transformations. With the increasing complexity and scale of model transformations, debugging of model transformations becomes more and more time-consuming and difficult, there is an increasing need to rely on efficient and accurate fault localization approaches to help with debugging. Among the existing fault localization approaches, the spectrum-based fault localization (SBFL), as a dynamic analysis method, mainly used the coverage information and execution results of the rules of model transformation to estimate the probability of each rule may be faulty. However, there are many false-positive and false-negative results in the rule coverage information, the accuracy of the SBFL is not ideal, so we consider mining the impact of covered range in different test models to further improve the effectiveness of fault localization. In this paper, we propose an optimized strategy of fault localization based on the impact of the test model, according to the covered range of test models, weight values are assigned to different test models, and the statistical coverage information of rules are weighted and adjusted accordingly. We compare the proposed approach with the SBFL, the experimental results show that under the same techniques for computing the suspiciousness, our approach can help locate around 26% more faulty rules in the ranking Top-1 of the suspicious list than the SBFL, the effectiveness of fault localization techniques can be improved by 50.42% in the best case and 8.9% in the average case.

INDEX TERMS Model transformation, weighted model, spectrum information, fault localization.

I. INTRODUCTION

The emerging of service-oriented architecture (SOA) has revolutionized the development paradigm of information systems, enabling the implementation of enterprise processes and workflows with service compositions. This service orientation trend has brought various benefits, such as service reusability, reliability, scalability, etc. With the increasing complexity of the information service system and the diversification of software evolution, how to develop information systems efficiently has become a common concern of academia and industry. The concept of model-driven engineering (MDE) [1] is widely used in the development of information service systems and is used to solve the above problems.

The associate editor coordinating the review of this manuscript and approving it for publication was Guanjun Liu.

MDE advocates taking models as the core elements, separating the business model of the system from specific implementation details, so as to improve system reusability and portability. MDE abstracts the model describing software systems into the platform-independent model (PIM) and platform-specific model (PSM): PIM a formal description of the business functions of the system, not involving the specific application platform, and PSM describes the technical implementation of PIM on specific platforms. The MDE-based software development process essentially implements a mapping from PIM to PSM and then from PSM to code, where the mapping between different models is automatically implemented through model transformation.

At present, the main applications of model transformation and MDE include the mapping between business processes to services [2], [3], the transformation from models of service network to abstract business processes [3], and even the

MDE based development of information service system [4]. Therefore, model transformations as the key technology in the MDE, the success of building a system with MDE depend on the correctness of model transformations. It needs to provide reliable model transformations to support the design and development of the SOA-based information system.

However, as the scale of the system increases, models involved in MDE become more and more complex, it is difficult to guarantee the correctness of model transformations. At present, most of the quality assurance approaches for model transformations focus on how to find faults, mainly including testing [5], graph theory-based analysis [6], and model checking [7]. In recent years, more researchers begin to study the way of locating faults in model transformations. Current approaches to fault localization include Petri-net analysis [8], trace model [9], static analysis [10], and spectrum-based [11]. Among them, the spectrum-based fault localization (SBFL) [11], is of better accuracy than the static analysis. It analyzes the execution coverage information of the rules under the passing and failing test models, applies the statistical analysis approaches [12], [13], and finally calculates the suspicious score for each rule, providing a list of suspicious rules rankings that reflect the most relevant part of the fault. However, the SBFL only considers the coverage of differentiated rules and ignores the impact of the scope of test models covered rule, eventually, the faulty rule is not always at the top of the suspicious ranking list.

To improve the effectiveness of fault localization, this paper proposes to use the impact of different test models by covering the number of rules in test models, and then adjust the list of suspicious rules of the spectrum accordingly. The general principles are: (1) If there is a failing test model that covers fewer rules, assign more weight to it; (2) The rules associated with a failing test model with more weight may be more suspicious; (3) The rules associated with more suspicious rules may be faulty because inheritance relationships between rules may result in error propagation.

The contributions of this paper are as follows: First, it proposes to consider the impact of test models covered rule range. Secondly, according to the weights of different test models, the spectrum information of test models covered rules is adjusted to improve the fault localization effectiveness of the spectrum approach. Thirdly, we evaluated the feasibility and effectiveness of our approach using three open-source model transformation projects, and the experimental results showed that our approach could maintain better accuracy than the SBFL.

The organization of this paper is as follows. Section 2 uses an open-source model transformation as an example to illustrate the necessity of our approach. Section 3 introduces the overall framework. Section 4 introduces the preliminary of our approach. Section 5 introduces how to apply the test model weights to adjust the suspicious rule list. Section 6 shows the experimental results of the approach. Section 7 introduces and describes some of the work related to us. Section 8 summarizes the paper and proposes some possible work directions.

II. MOTIVATING EXAMPLE

To illustrate the shortcoming of the spectrum approach, we take the *Class2Relational* model transformation as an example, which is in the open-source repository (ATL Transformation Zoo[1]). It consists of six transformation rules that describe the transformation from a class model to a relational data model. We use an OCL assertion specified in [14] as a test oracle to check whether the transformation results satisfy the expected properties.

TABLE 1. An OCL assertion for the Class2Relational transformation.

SrcAttribute.allInstances() $->$ (a | a.type.oclIsKindOf(SrcDataType)) $->$ forAll(a | TrgColumn.allInstances() $->$ exists(clc.name=a.name))

The OCL description is shown in Table 1, it is used to check whether there is a *Column* in the target model for every *Attribute* of type *DataType* in the source model while their name is the same. We introduce an error in the third rule so that some transformation results may cause the OCL assertion to be violated. Then, we randomly generate 6 different input models, and take the cartesian product composed of input models and OCL as a test case: $t_i =< I_i, ocl >$ $(1 \leq i \leq 6)$. When a test case is used, its input model will be provided to the model transformation to produce the output model, and then the input and output model is checked against the OCL. If the OCL is violated, the result of the test case is "F" (failing), otherwise, the result is "P" (passing). The coverage information of a given test case can be collected by checking the trace model of TraceAdder [15].

TABLE 2. Suspiciousness for Class2Relational when OCL fails.

	t_1	t_2	t_3	t_4	t_5	t_6	N_{cf}	N_{uf}	N_{cs}	N_{us}	$Susp$	$Rank$
r_1	•	•	•	•	•	•	2	4	0	0	0.5	4
r_2	•	•	•	•	•	•	2	4	0	0	0.5	4
r_3(bug)	•	•	•	•		•	2	3	0	1	0.57	3
r_4		•				•	1	1	1	3	0.67	1
r_5		•	•				1	1	1	3	0.67	1
r_6		•	•			•	1	2	1	3	0.5	4
Result	F	F	P	P	P	P						

The coverage matrix and execution results of 6 test cases are shown in Table 2, where N_{cf} and N_{cs} are the number of rules executed by failing and passing test cases, respectively; N_{uf} and N_{us} are the number of rules that are not executed by failing and passing test cases, respectively. When a transformation rule is executed by the test case, the corresponding cell is marked with "•". The technique namely Tarantula [13] $(N_{cf}/N_f)/(N_{cf}/N_f + N_{cs}/N_s)$ is used to calculate the suspicious scores for the 6 rules. If a rule has a higher suspicious score, it is more likely to be a faulty rule. The "Susp" and the "Rank" column are the suspicious score of rules

[1]http://www.eclipse.org/atl/atlTransformations

and the ranking of rules, respectively. The sequence to be checked according to the ranking is $\{r_4, r_5, r_3, r_1, r_2, r_6\}$. The debugger has to check three rules before locating the faulty rule. However, when observing the rule coverage information in the spectrum, we find that t_1 and t_2, as failing test cases, its test model has different rules covered ranges: t_1 executes 3 rules, and t_2 executes 6 rules. Usually, when we manually check faulty rules, t_1 has a smaller range to be checked than t_2, so that t_1 can identify faulty rules more easily than t_2. Therefore, we believe that different test models have different abilities to locate faulty rules. If different test models are weighted based on the covered range of the rules, the failing test model with fewer rules has more weights, and the corresponding rules covered by the higher weighted test model are more likely to be the faulty rule. According to the cover range of test models, the spectrum information is weighted, and the suspicious value will be calculated again to obtain an updated list of suspicious rules.

III. FRAMEWORK

The framework of our approach is shown in Fig. 1. In the 1st step, we statically analyze the inheritance relationship between the transformation rules, which is used to distinguish the rules with the same suspicious value, and dynamically collect the execution information of rules in the test models during the model transformation. The 2nd step, according to the collected coverage and inheritance information, establish the connection relationship between the test models and the rules, further construct the connection matrix for iterative calculation, and get the weight value of the rules and the test models from the iterative calculation results. In the 3rd step, we use the calculated weight value to update the spectrum information to generate a list of weighted suspicious rules. The updated rule list shows the new ranking of suspicious rules. Among them, the basic concept of transformation rules in the 1st step and the basic idea of adjusting the weight of rules in the 2nd step, we will make a detailed introduction in the IV section.

IV. PRELIMINARY

In this section, we explain some basic concepts in this paper. It includes the framework language that defines the model transformation: ATL model transformation, and the guiding idea of calculating rule weights based on the impact degree of the test model: PageRank algorithm.

A. ATL MODEL TRANSFORMATION

Model transformations provide an essential mechanism for operating models. The process of model transformations can be described as one or more source(input) models execute the model transformation program to obtain one or more target(output) models, in which the source and target model must conform to the corresponding metamodel. The metamodel defines what types of elements in a specific field and the specifications between them. The basic goal of model transformation is to implement the automatic model mapping by executing the transformation program through the model transformation platform.

Based on this, it needs an appropriate transformation language to implement the model transformation program. At present, there are various model transformation languages. ATL [16] is one of them, has gained wide support in the research of MDE due to its flexibility and ease of integration into the development platform [17], [18]. It is a rule-based model transformation language, each transformation rule matches the elements in the source model and transforms them into the elements of the target model. A model transformation involves mappings between different elements of models. To identify the suspicious transformation rules, the transformation rules should be traceable. ATL has an internal trace mechanism that allows the information of the execution rules in each model transformation process, the source model elements, and the target model elements information generated by the transformations to be stored in the internal trace model. This feature is a prerequisite for us to obtain the coverage information of the rules from the dynamic execution process.

B. PAGERANK ALGORITHM

The idea of calculating the weighted value for rules is inspired by the application of the PageRank [19] algorithm. PageRank is a web page ranking algorithm that was originally applied to improve the quality of web pages in search engines: it abstracts each web page on the internet as a node, and the weight of a web page is evenly distributed to the web page it points to. If a web page is linked by many other web pages, indicating that the web page is more important, its page ranking value is higher; the high-ranking web page is more reliable, the contributing link weight is significant, and the page ranking value it points to is higher. In the end, if a page has the higher the page ranking value, it will be at the top of the website.

PageRank transforms the web page ranking problem into a two-dimensional matrix multiplication problem, it converges to the real value of page ranking by iterative calculation. It assumes that the internet is a directed graph with n nodes to denote n web pages, and the vector $\boldsymbol{b} = [b_1, b_2, \ldots, b_n]^T$ is the ranking value of each web page, assuming that the initial value of all web pages is $1/n$, that is, $\boldsymbol{b_0} = [1/n, 1/n, \ldots, 1/n]^T$. Construct a matrix A of $n \times n$ to indicate the links between web pages: suppose there is an element a_{ij} in matrix A, which means there is a link from node j to node i, its value is $1/N_j$, and N_j is the number of outbound links from node j. Each row of the matrix denotes the probability that all web pages link to this node. Considering that the number of outbound links of a node maybe 0, that is, it has no links to any pages, the case leads to the value of each node is 0 after iterative calculation. To make sure the calculation converges, usually set a constant α and a transition vector $\boldsymbol{v}$ to adjust, it sets $\boldsymbol{v} = [1/n, 1/n, \ldots, 1/n]^T$ to indicate that this page may link to all pages, and its weight is evenly distributed to each page. Assuming that values of the matrix converge

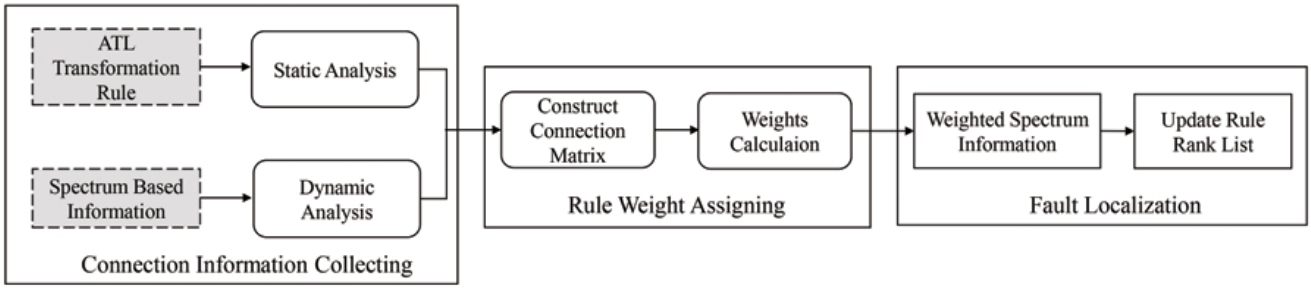

FIGURE 1. The basic framework of our approach.

after k iterations, the iterative formula is shown in eq (1).

$$b_k = \alpha A \cdot b_{k-1} + (1-\alpha)v \tag{1}$$

PageRank is not only used for ranking web pages, but many studies have also applied it to other fields as well. For example, Mirshokraie *et al.* [20] applied it to the mutation test to guide generated mutants on the parts of the code that are more likely to affect the output of the program. Zhang *et al.* [21] applied it to the fault localization of traditional software programs, and different test inputs were considered to have different importance for fault localization. We apply it to the fault localization of the model transformations for the first time.

V. FAULT LOCALIZATION USING WEIGHTED TEST MODELS

In this section, we describe the details of our approach, including the static and dynamic analysis, rules weight value assigning and weighted-based spectrum of fault localization.

A. STATIC INHERITANCE ANALYSIS

Due to the inheritance relationship among ATL transformation rules, a rule with an abstract identifier can be inherited by multiple other rules. If a rule has an error, it will propagate the effects of the error to its descendant rules. Besides, when a rule is executed, rules with inheritance relationships are all recorded in the execution information of the trace model. For example, given three rules r_1, r_2, r_3, which have the following relationship "$r_1 < r_2 < r_3$", where the symbol "$r_a < r_b$" denotes a rule r_a inherits from rule r_b. Suppose that a test model executes r_1, the trace model will not only store the execution of r_1 but will also trigger the execution of the associated rules r_2 and r_3. In the end, the trace model records the execution information of three rules, and the above three rules have the same coverage information in a test case. When calculating the suspicious value of rules, the value of three rules may be the same, making it difficult to distinguish which one is the really faulty rule. Therefore, we statically extract the inheritance relationship between rules by analyzing the textual information of ATL rules, the inheritance diagram of the rules is constructed. Further, generating the connection matrix of rules to distinguish the difference between rules with the same suspicious value. This step obtains the connection between rules.

B. DYNAMIC COVERAGE ANALYSIS

The SBFL considers each OCL assertion as a test case group. If the execution results of a set of test models all satisfy this OCL, it is considered that the transformation satisfies the corresponding properties. Otherwise, when at least one test model in the test suite violates the OCL, the OCL is considered to be not satisfied. During the running of model transformations, we dynamically analyze the passing and failing execution information of test models in each OCL assertion. When the check results of an OCL are all passing, this OCL is ignored. When at least one result is failing, collect the name of the test model and the corresponding coverage rules, to construct a mapping relationship between the test models and the rules. We analyze which rules are covered by each test model and which test models are covered by each rule. This step obtains the connection between test models and rules.

C. RULE WEIGHT ASSIGNING

We first consider each test model and rule as a single node. As we observed during the transformation run, if a rule is not covered by all failing test models, the rule may also be a fault. Similarly, if a rule is not covered by all passing test models, the rule may be correct. Usually, the spectrum approach will calculate the suspicious value of such rule as 0 or 1, but this calculation method may not be accurate. Therefore, when such an isolated rule exists, we assume that all test models may have a certain probability to cover this rule, that is, this rule has a link to all test models. And the parameter α to adjust its proportion. According to the above analysis, we can construct a connection matrix to denotes the connection between different test models and rules. Besides, we set the transition vector $\boldsymbol{v}$ according to the number of rules covered by different test models, calculating the weighting score of rules depends on eq (1). In our experiments, we first set the value of α to 0.1 to evaluate the feasibility and accuracy of the approach. Besides, we also consider the impact of different values of α on the efficiency of fault localization.

1) CONSTRUCTION OF CONNECTION MATRIX

Different connection matrices are constructed based on coverage information of the failing and passing test models. We use "N_r" denotes the number of the rules, "N_t" denotes the number of the test models.

First, construct a matrix $R2R$ of size $N_r \times N_r$ based on inheritance analysis. If the rule has the following inheritance relationship "$r_a < r_b$", we set a link from node r_b to r_a, the matrix element r_{ab} is $1/s$, and s is the total number of rules inherited from r_b. As mentioned above, the inheritance relationship "$r_1 < r_2 < r_3$", we can construct the matrix:

$$R2R = \begin{bmatrix} 0 & 1 & 1/2 \\ 0 & 0 & 1/2 \\ 0 & 0 & 0 \end{bmatrix}$$

Then, we construct two matrices based on the coverage information between rules and test models. Construct a matrix $T2R$ of size $N_r \times N_t$, which denotes the weight that the test models assign to the rules: If a test model t_j covers the rule r_i, we set the matrix element t_{ij} is $1/m$, m is the total number of rules covered by the test model t_i. When the number of rules covered by a test model is smaller, the weight value assigned to the rule is larger. Construct a matrix $R2T$ of size $N_t \times N_r$, which denotes the weight that the rules assign to the test models: if a rule r_j is covered by a test model t_i, a matrix element value r_{ij} is $1/n$, where n is the total number of test models covering this rule. Taking the motivating example in Section II, we can obtain the matrices $T2R$ and $R2T$ under the failing test models respectively:

$$T2R = \begin{bmatrix} 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix}^T$$

$$R2T = \begin{bmatrix} 1/2 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 1/2 & 1/2 & 1 & 1 & 1 \end{bmatrix}$$

In general, the number of test models is much larger than the number of rules, we consider that the impact of our coverage information should be greater than the inheritance relationship. We set a parameter d to adjust the proportion of $R2R$, in the following experiment, we also consider the effect of adjusting different d values. Finally, we construct a block matrix A as the connection matrix for calculating the weight of the rule, matrix A is shown in eq (2).

$$A = \begin{bmatrix} d \cdot R2R & T2R \\ R2T & O \end{bmatrix} \tag{2}$$

where O is the zero matrix, because there is no connection relationship between the test models. The matrix A is a square matrix of size $(N_r + N_t) \times (N_r + N_t)$.

2) CONSTRUCTION OF TRANSITION VECTOR

According to the above analysis, the vector $\boldsymbol{b} = [\boldsymbol{w_r}^T, \boldsymbol{w_t}^T]^T$ should be composed of two sub-vectors, where $\boldsymbol{w_r} = [w_{r1}, w_{r2}, \ldots, w_{rm}]^T$ and $\boldsymbol{w_t} = [w_{t1}, w_{t2}, \ldots, w_{tn}]^T$ denote the weight value of m rules and n test models respectively, each rule and test model initial value is set to $1/(m+n)$.

Construct a transition vector $\boldsymbol{v} = [\boldsymbol{v_r}^T, \boldsymbol{v_t}^T]^T$ to present the impact of different test models. Similarly, the vector $\boldsymbol{v}$ consists of two sub-vectors $\boldsymbol{v_r} = [v_{r1}, v_{r2}, \ldots, v_{rm}]^T$ and $\boldsymbol{v_t} = [v_{t1}, v_{t2}, \ldots, v_{tn}]^T$. The value of the vector $\boldsymbol{v_r}$ is set to $\mathbf{0}$ for both passing and failing test models. But the value of $\boldsymbol{v_t}$ is different, as we have analyzed, a test model with fewer covering rules is more helpful for fault localization and it should be assigned more weight. Therefore, in the failing test models, according to the number of covering rules, the value of each failing test model is assigned as shown in eq (3).

$$v_{tj} = N_j^{-1} / \sum_{j=1}^{n} N_j^{-1} \tag{3}$$

where N_j is the number of rules covered by the jth failing test model, and n is the total number of failing test models. Therefore, the transition vector in the motivating example can be set as $\boldsymbol{v} = [0, 0, \ldots, 0, 2/3, 1/3]^T$, t_1 is assigned a higher value than t_2 because of t_1 covers fewer rules.

However, for a passing test model, the rules it covers are not necessarily the correct rules. If there is a faulty rule in the transformation program, but the target model attribute obtained by the transformation does not violate the OCL assertion, the checking mechanism considers this is a passing test model. In this case, we cannot simply say that the rules covered by the passing test model with higher weights are more likely to be correct, the false-negative situation makes the passing test model not meet the property of covering the number of rules. Therefore, in the passing case, we evenly assign the same weight value to each test model: $1/N_p$, where N_p is the number of passing test models.

3) WEIGHTS CALCULATION

According to eq (1), we can use the calculation of the block matrix to get the weight values of rules and test models shown in eq (4).

$$\begin{aligned} \boldsymbol{w}_r^{(k)} &= \alpha \cdot [dR2R \cdot \boldsymbol{w}_r^{(k-1)} + T2R \cdot \boldsymbol{w}_t^{(k-1)}] \\ \boldsymbol{w}_t^{(k)} &= \alpha R2T \cdot \boldsymbol{w}_r^{(k-1)} + (1-\alpha) \cdot \boldsymbol{v}_t \end{aligned} \tag{4}$$

Finally, we normalize two sub-vectors $\boldsymbol{w_r}$,$\boldsymbol{w_t}$, the weights of the i-th rule and the j-th test model are respectively:

$$w_{ri} = \frac{w_{ri}}{\max(\boldsymbol{w_r})}, w_{tj} = \frac{w_{tj}}{\max(\boldsymbol{w_t})} \tag{5}$$

D. WEIGHTED-BASED SPECTRUM OF FAULT LOCALIZATION

After two iterative calculations under the connection matrix of failing and passing test models, two sets of vector values will be obtained. It denotes the weight of each rule in case of failing and passing, respectively.

We get the weighted-spectrum based on the original spectrum information, the updated approach is shown in eq (6).

$$\begin{aligned} \bar{N}_{cfi} &= wf_{ri} \cdot N_f, \bar{N}_{ufi} = N_f - \bar{N}_{cfi} \\ \bar{N}_{cpi} &= wp_{ri} \cdot N_p, \bar{N}_{upi} = N_p - \bar{N}_{cpi} \end{aligned} \tag{6}$$

where wf_{ri} and wp_{ri} denote the weight value of the i-th rule calculated in the failing and passing respectively, N_f and N_p are the numbers of failing and passing test models respectively, $\bar{N}_{cfi}, \bar{N}_{ufi}, \bar{N}_{cpi}, \bar{N}_{upi}$ are the new metric value of the SBFL technique. We use the same SBFL technique to calculate the weighted-spectrum information to get an updated list of suspicious rule rankings and evaluate the effectiveness of our approach.

We reconsider the example in Section II, based on the constructed connection matrix, after calculation and normalization, we can get the weight of rules in the failing and passing cases: $\boldsymbol{wf_r} = [1, 1, 1, 0.202, 202, 202]^T$, $\boldsymbol{wp_r} = [1, 1, 0.596, 0.163, 0.163, 0.326]^T$. After the weighted calculation of eq (6), it obtains the final suspicious score of the 6 rules are $[0.5, 0.5, 0.627, 0.554, 0.554, 0.383]^T$ under the Tarantula. The rule r_3 ranking is improved and ranked the top. This proves that our approach is effective to some extent. Further, we analyze the feasibility and effectiveness of the approach through specific experiments.

E. COMPLEXITY EVALUATION

In this part, we evaluate the time and space complexity of the approach. Suppose we randomly generate S input models and construct M ocls. It can form $S \times M$ test cases. After the model transformation execution, every ocl judges the failing and passing of the transformation. Suppose that in the $S \times M$ test cases, there are N_f failing test models and N_p passing test models. After collecting the coverage information from the TraceAdder, it assumes that N_f failing test models execute a total of r_f rules and N_p passing test models execute a total of r_p rules. The time complexity of the algorithm mainly comes from the iterative calculation of the matrices, so the time complexity is $O(\max(r_f^2 + r_f \cdot N_f, r_p^2 + r_p \cdot N_p))$. The space complexity mainly use to store the operation results of matrices, it is $O(\max(r_f^2 + r_f \cdot N_f, r_p^2 + r_p \cdot N_p))$. It depends on which part of the failing and passing test models account for the majority in quantity.

VI. EXPERIMENT

In this section, we carried out a series of experiments to verify the accuracy of our approach. Firstly, we introduce our experimental setup, list the detailed experimental data. And then introduce metrics to evaluate experimental results. At last, we analyze the experimental results and consider the impact of different parameter values on the effectiveness of our approach. To distinguish and compare the experimental results, we mark our method of the weighted-spectrum as ''Weight''.

A. EXPERIMENTAL SETUP

In this paper, we use the three open-source model transformation suites for our experimental evaluation. *BibTeX2DocBook* is the transformation in the ATL Transformation Zoo, which is used to transform BibTeXML files to DocBook documents. *UML2ER* is an experimental project in the field of structural modeling [22], which is used to transform the UML class diagrams into the Entity-relationship diagrams. *Ecore2Maude* is used on the specific modeling language tool [23], it is used to transform models that conform to the Ecore metamodel to the models that conform to the Maude metamodel. To compare with the spectrum approach, we use the same test models and OCL assertions, which means that we can have the same test cases and coverage information. We compare the effectiveness of the two methods under the same experimental conditions. Table 3 summarizes this information. Most of them come from the experimental configuration information in the SBFL [11]. Three model transformation artifacts are different in the number of transformation rules, rules size, test cases, OCL assertions, and mutants.

TABLE 3. Details of the three model transformation artifacts.

Transformation Name	Rules	Loc	Mutants	OCL assertions	Assertion violated	Test case
BibTex2DocBook	9	393	4	27	269	2700
UML2ER	8	53	18	14	90	1400
Ecore2Maude	39	1055	50	42	155	4200
Total	56	1501	108	83	514	8300

We select 12 suspicious formulas that are widely used in the field of fault localization. As shown in Table 4, the technical names and calculation formulas are listed. The basis of our assessment is the coverage information generated by the 514 OCL assertions violated shown in Table 3, we analyze coverage information and transformation results and evaluate the result of the updated rules ranking list.

B. EVALUATION METRICS

We use the two metrics $EXAM_{score}$ and *Top-N* to evaluate the effectiveness of the approach, they are also often used to evaluate the fault localization effectiveness of traditional software programs.

$EXAM_{score}$: The metric denotes the ratio of the number of rules to be examined to the total number of rules before the first faulty rule is located. It is usually called the fault localization cost and is described by the eq (7).

$$EXAM_{score} = \frac{number\ of\ rules\ examined}{Total\ number\ of\ rules} \tag{7}$$

As we observe from the eq(7), the fewer rules need to be examined before locates the faulty rule, the $EXAM_{score}$ is smaller, the approach is considered more effective. While the faulty rule has the same suspicious value with the other rules, that is, the ranking is the same, the $EXAM_{score}$ can be divided into three cases: if there is a faulty rule ranking the same as the other *(n-1)* rules, the best-case means that the faulty rule is the first one to be checked and it is in the first position, the worst-case means that the faulty rule is the last one to be checked and it is in the n position and the average-case indicates that the faulty rule is in the middle of the same ranking and it is in the *(n/2)* position. We consider the $EXAM_{score}$ of the average-case to evaluate the effectiveness of our approach, to avoid the impact of special cases.

Top-N: This metric denotes that the number of faulty rules that can be successfully located in the top N position of the

TABLE 4. Techniques applied for suspiciousness calculation.

Technique	Formula	Technique	Formula
Arithmetic Mean	$\frac{2(N_{cf}*N_{us}-N_{uf}*N_{cs})}{(N_{cf}+N_{cs})*(N_{us}*N_{uf})+(N_{cf}+N_{uf})*(N_{cs}+N_{us})}$	Braun-Banquet	$\frac{N_{cf}}{max(N_{cf}+N_{cs},N_{cf}+N_{uf})}$
B-U&Buser	$\frac{\sqrt{N_{cf}*N_{us}}+N_{cf}}{\sqrt{N_{cf}*N_{us}}+N_{cf}+N_{cs}+N_{uf}}$	Cohen	$\frac{2(N_{cf}*N_{us}-N_{uf}*N_{cs})}{(N_{cf}+N_{cs})*(N_{us}*N_{cs})+(N_{cf}+N_{uf})*(N_{uf}+N_{us})}$
Dstar	$\frac{N_{cf}*N_{cf}}{N_{cs}+N_{uf}}$	Kulczynski2	$\frac{1}{2}(\frac{N_{cf}}{N_{cf}+N_{uf}}+\frac{N_{cf}}{N_{cf}+N_{cs}})$
Mountford	$\frac{N_{cf}}{0.5((N_{cf}*N_{cs})+(N_{cf}*N_{uf})+(N_{cs}*N_{uf}))}$	Ochiai	$\frac{N_{cf}}{\sqrt{N_f*(N_{cf}+N_{cs})}}$
Ochiai2	$\frac{N_{cf}*N_{us}}{\sqrt{(N_{cf}+N_{cs})(N_{cf}+N_{cs})(N_{cf}+N_{cs})(N_{cf}+N_{cs})}}$	Op2	$N_{cf}-\frac{N_{cs}}{N_s+1}$
Phi	$\frac{(N_{cf}*N_{us}-N_{uf}*N_{cs})}{\sqrt{(N_{cf}+N_{cs})*(N_{cf}*N_{uf})+(N_{cs}+N_{us})*(N_{uf}+N_{us})}}$	Simple Matching	$\frac{N_{cf}+N_{us}}{N_{cf}+N_{cs}+N_{us}+N_{uf}}$
Tarantula	$\frac{N_{cf}/N_f}{N_{cf}/N_f+N_{cs}/N_s}$	Zoltar	$\frac{N_{cf}}{N_{cf}+N_{uf}+N_{cs}+10000N_{uf}*N_{cs}/N_{cf}}$

suspicious list. Generally, debuggers only check the rules at the top of the suspicious list. Therefore, if an approach has a higher *Top-N* value, it is considered more effective. In the experiment, we only consider the number of faulty rules that can be successfully located in Top-1, Top-2, and Top-3 cases. It means that if the transformation program has a bug, the debugger will only check the top three rules of the suspicious list. If a faulty rule ranking the same as other rules, we consider the average-case, that is, the faulty rule is in the average position.

C. EXPERIMENTAL RESULTS

We first set the parameter values α and d to 0.1 during the experiment, separately evaluate the effectiveness under the three transformation programs. Table 5, Table 6 and Table 7 respectively denote the overall localization results of *Bibtex2DocBook*, *UML2ER* and *Ecore2Maude*.

In the table, each row represents the localization results of each SBFL technique, and each column represents the values of Top-N and $EXAM_{score}$ under two approaches. The "S" column denotes the spectrum-based approach, and the "W" column denotes our approach based on weighted. The value of the "Imp" column shows the improved ratio compared to the spectrum approach in the $EXAM_{score}$. It is calculated by the eq (8).

$$Imp(\%) = \frac{EXAM_S - EXAM_W}{EXAM_S} \times 100 \tag{8}$$

where $EXAM_S$ and $EXAM_W$ denote the $EXAM_{score}$ value of the spectrum and our approach respectively. The last row in the table represents the overall fault localization effectiveness. For the metric of *Top-N*, it indicates the total number of faulty rules can be located at the Top-N. For the metric of $EXAM_{score}$, it indicates the average improvement ratio of all technologies.

As shown in Table 5, the result of *BibTex2DocBook*, we can observe that all techniques have been improved. In general, three of the most obvious improvements are *Dstar*, *B-U&Buser*, and *Ochiai2*. About the improvement of $EXAM_{score}$, they all improved by more than 15%. This means

TABLE 5. The overall effectiveness comparison results of BibTex2DocBook.

	Technique	Top-1		Top-2		Top-3		$EXAM_{score}$		
		S	W	S	W	S	W	S	W	Imp(%)
BibTex2DocBook	Arithmetic Mean	138	156	175	198	202	215	0.284	0.253	10.92
	Braun-Banquet	108	130	143	184	204	236	0.277	0.233	15.9
	B-U&Buser	**87**	**128**	**107**	**151**	**153**	**192**	**0.365**	**0.284**	**22.14**
	Cohen	90	119	109	172	180	212	0.343	0.282	17.87
	Dstar	**127**	**186**	**178**	**231**	**198**	**251**	**0.326**	**0.178**	**45.26**
	Kulcynski2	198	197	222	232	247	252	0.177	0.171	3.39
	Mountford	155	185	209	226	236	247	0.204	0.183	10.1
	Ochiai	182	186	219	229	244	251	0.188	0.179	4.65
	Ochiai2	**72**	**90**	**83**	**118**	**109**	**169**	**0.443**	**0.360**	**18.69**
	Op2	194	195	220	230	246	251	0.182	0.176	3.32
	Tarantula	47	71	68	107	125	153	0.398	0.349	12.3
	Zoltar	197	197	222	232	248	253	0.177	0.171	3.39
	Overall	1595	1840	1955	2310	2392	2682	0.280	0.235	13.994

TABLE 6. The overall effectiveness comparison results of UML2ER.

	Technique	Top-1		Top-2		Top-3		$EXAM_{score}$		
		S	W	S	W	S	W	S	W	Imp(%)
UML2ER	Arithmetic Mean	32	51	34	55	34	64	0.313	0.337	-7.61
	Braun-Banquet	**32**	**51**	**35**	**63**	**35**	**65**	**0.337**	**0.305**	**9.54**
	B-U&Buser	32	35	34	47	34	56	0.336	0.319	4.93
	Cohen	32	51	34	55	34	64	0.313	0.338	-7.83
	Dstar	**4**	**51**	**7**	**70**	**7**	**72**	**0.537**	**0.301**	**43.88**
	Kulcynski2	32	35	34	54	34	56	0.331	0.310	6.43
	Mountford	28	51	29	63	29	72	0.337	0.311	7.68
	Ochiai	32	35	34	54	34	56	0.333	0.313	6.16
	Ochiai2	32	51	34	55	34	64	0.313	0.336	-7.38
	Op2	**32**	**51**	**34**	**70**	**34**	**72**	**0.331**	**0.299**	**9.79**
	Tarantula	9	11	12	15	28	31	0.499	0.476	4.53
	Zoltar	**32**	**51**	**34**	**70**	**34**	**72**	**0.331**	**0.299**	**9.79**
	Overall	329	524	355	671	371	744	0.359	0.329	6.659

TABLE 7. The overall effectiveness comparison results of Ecore2Maude.

	Technique	Top-1		Top-2		Top-3		$EXAM_{score}$		
		S	W	S	W	S	W	S	W	Imp(%)
Ecore2Maude	Arithmetic Mean	4	4	32	22	52	44	0.175	0.177	-0.96
	Braun-Banquet	4	4	22	22	42	44	0.16	0.159	0.49
	B-U&Buser	4	4	32	22	52	44	0.134	0.154	-14.75
	Cohen	4	4	32	22	52	44	0.176	0.177	-0.38
	Dstar	**3**	**4**	**15**	**12**	**16**	**34**	**0.423**	**0.160**	**62.12**
	Kulcynski2	4	4	32	12	52	47	0.133	0.139	-4.54
	Mountford	4	4	40	32	60	54	0.123	0.117	4.85
	Ochiai	4	4	32	22	52	54	0.134	0.131	2.53
	Ochiai2	4	4	32	22	52	54	0.175	0.168	4.15
	Op2	4	4	12	12	32	35	0.245	0.252	-3
	Tarantula	0	0	26	26	26	28	0.211	0.204	3.41
	Zoltar	4	4	22	22	42	45	0.197	0.192	2.63
	Overall	43	44	329	248	530	527	0.191	0.169	4.713

that Dstar as an example, the original debugger needs to check 2.9 of the 9 rules, but now it only needs to check 1.6 rules. About the Top-N, the *B-U&Buser* locates the number of the faulty rule within Top-1 from 87 to 128, which has increased by 47.1%, the *Ochiai2* ranks from 72 to 90, increases by 25%, the *Dstar* ranks the number of Top-1 from 127 to 186, increases by 46.5%. If we see the last row in the table, the overall is that within Top-3, our approach locates nearly 300 faulty rules more than the SBFL. In the *BibTex2DocBook*, the improvement of the $EXAM_{score}$ is basically consistent with Top-N. The overall effectiveness of all techniques is improved to nearly 14% on average. For ease of observation, we draw a histogram to show the number of faulty rules that can be successfully located in Top-3. As shown in Fig.2,

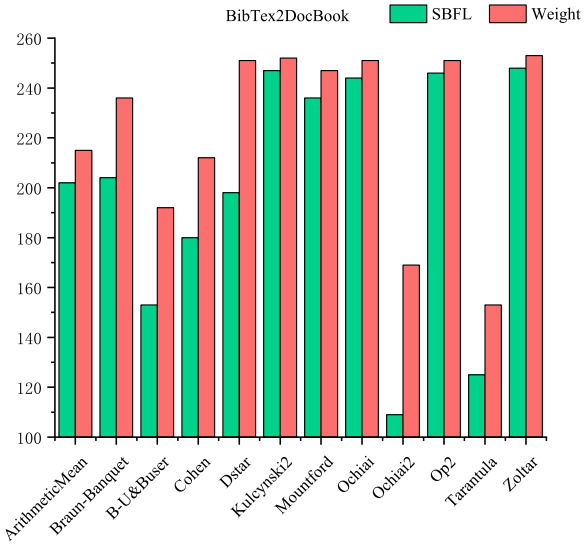

FIGURE 2. The number of faulty rules at Top-3 in Bibtex2DocBook.

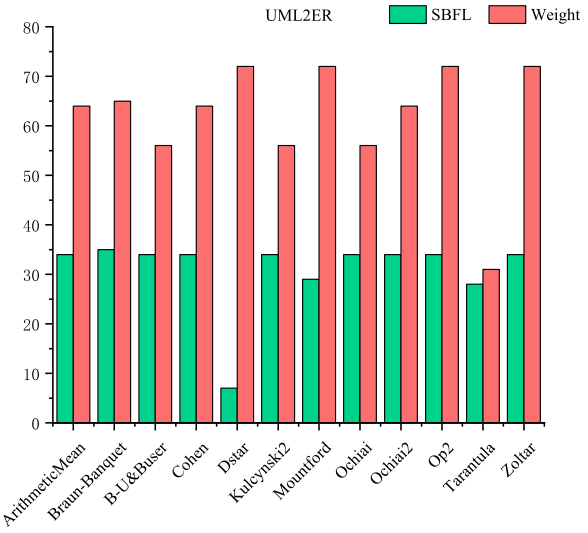

FIGURE 3. The number of faulty rules at Top-3 in UML2ER.

the three most increased are *Ochiai2*, *Dstar*, *B-U&Buser*, they are increased by 60, 53, 39 respectively.

Table 6 shows the result of *UML2ER*, most techniques can locate 50% more the faulty rules in the Top-1 position. Overall, in the Top-3, the number of faulty rules that our approach can locate is increased from 371 to 744. About the improvement of $EXAM_{score}$, three relatively obvious technologies are *Dstar*, *Op2*, and *Zoltar*. The most obvious improvement in the $EXAM_{score}$ is also *Dstar*, it is improved by 43.88%. The overall effectiveness is improved by 6.7%. It can be seen that in different model transformations, the efficiency of each technique improvement is not consistent, but Dstar as one of the methods still obtains good results. On the other hand, it is worth noting that there are a large number of inheritance relationships in the UML2ER model transformation. By static analysis of the inheritance relationships, we can receive good results on the Top-N metrics. On the other hand, we observe the change of the number of faulty rules in Fig. 3,the three most increased are *Dstar*, *Mountford*, *Op2*, they are increased by 65, 43, 38 respectively.

Table 7 is the localization result of *Ecore2Maude*, compared with the other two, its effectiveness did not be improved significantly. The reason is that the EcoreMaude transformation rules include many called helpers, and the calling relationships between rules may lead to error propagation. In our next study, we need to further analyze the calling relationship between the rules in the transformation execution process, it may need to extend the corresponding dynamic analysis technology to analyze rule calls.

But, we should also see that the *Dstar* can be greatly improved in all three model transformation programs. The reason is that we can see in Table 4, the value of N_{cf} in Dstar formula accounts for a large proportion. We adjust the weight value of N_{cf} by the covered range of the failing test model weights and get good results after calculating again. It also proves that our approach has a certain effect.

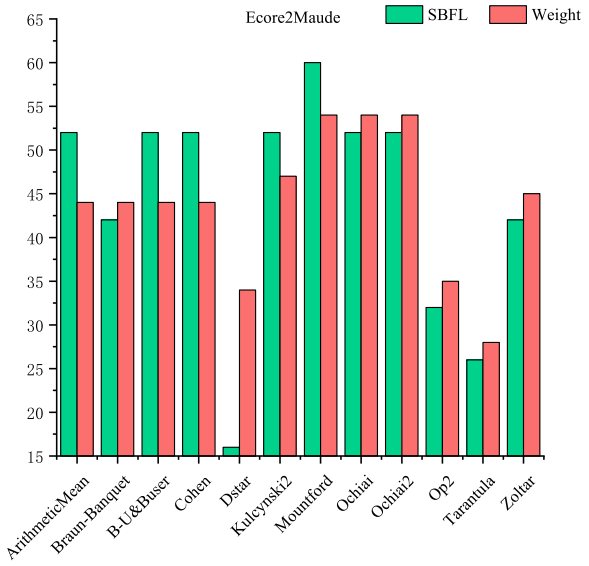

FIGURE 4. The number of faulty rules at Top-3 in Ecore2Maude.

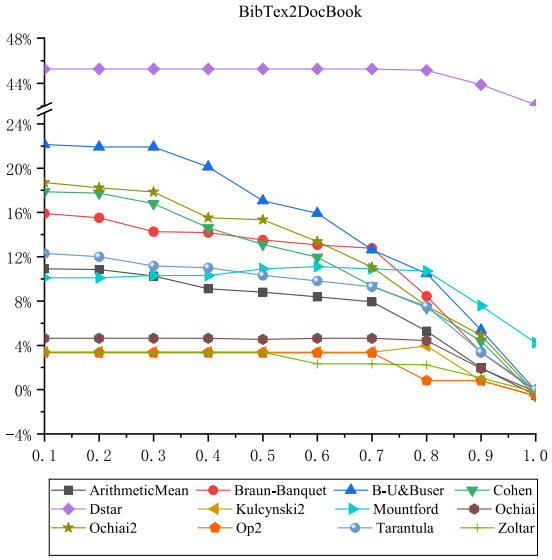

FIGURE 5. The impact of different α value on Bibtex2DocBook.

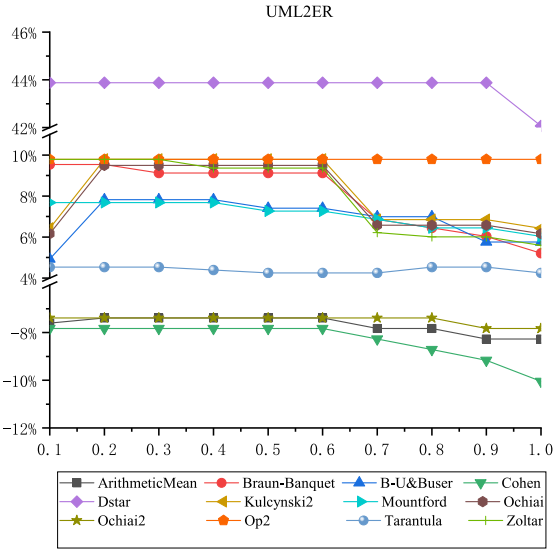

FIGURE 6. The impact of different α value on UML2ER.

According to the above analysis, we consider the average improvement of the $EXAM_{score}$ under three transformation programs in each suspiciousness technique, the most improvement technique is the *Dstar*, which can be improved by about 50.42%. We calculate the average value of all techniques improvement to estimate the overall effectiveness of our approach, BibTex2DocBook, UML2ER, and Ecore2Maude are improved by 13.994%, 5.713%, 6.659%, respectively. The overall effectiveness of the approach can be improved by 8.9%.

On the other hand, we evaluate the average growth rate of the number of faulty rules that can be successfully located at Top-1, Top-2, and Top-3 benchmarks. From the overall result, *BibTex2DocBook* can locate 15.4%, 18.2% and 12.1% more faulty rules than the SBFL. *UML2ER* can locate 59.3%, 89.0% and 100.5% more faulty rules than the SBFL. And these value in *Ecore2Maude* are 2.3%, -24.6%, 0.57%.

Overall, after the calculation, the average growth of the three transformation programs can be derived, we can locate 25.7%, 27.5%, 37.7% more faulty rules than the SBFL at Top-1, Top-2, Top-3, so we believe that our approach is more effective than the SBFL.

D. IMPACT OF PARAMETERS

In this section, we consider the impacts of different parameters on the fault localization effectiveness. It includes different values of α and d.

First, we consider different α values. We set 10 different values in the interval (0,1) to generate different results. The results of three model transformations are shown in the Fig.5, Fig.6 and Fig.7. In the line chart, the X-axis denotes different values of α, and each group of values increases by 0.1. The Y-axis denotes the improvement ratio of the Exam value. To make the figure clearer, we fold the blank part. We use different colors to represent different SBFL techniques. As the value of different α in the X-direction changes, the SBFL techniques have different improvement ratio changes. If we observe these three sets of charts, we can find that when the value of α is closer to 1, the improvement ratio about the Exam value will decrease, which means the lower the effectiveness of localization. This is mainly because the larger the value of α, the smaller the weight of ν in eq (1), and the smaller the weight assigned to the test models. Finally, the covered range of test models plays a small role in the calculation.

Next, we consider the proportion of the inheritance relationships between rules in the connection matrix. Similarly, we set the values of d to 0, 0.01, 0.1 and 1 to analyze the influence of different values. When d = 0, it means that the influence of inheritance relationship is not considered. When d = 1, the proportion of rule inheritance is the same as the

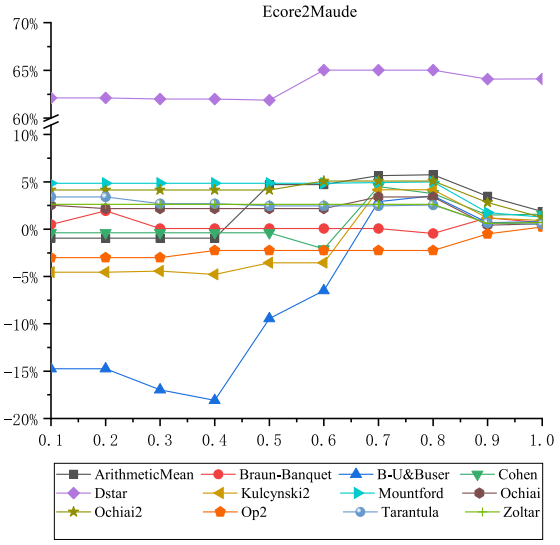

FIGURE 7. The impact of different α value on Ecore2Maude.

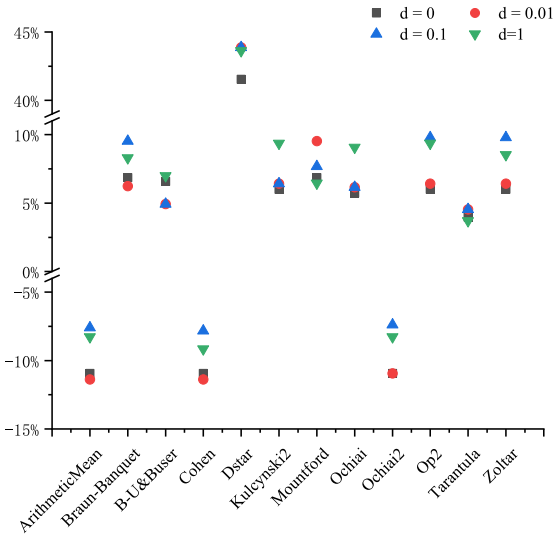

FIGURE 8. The impact of different d value on UML2ER.

test models. As shown in Fig 8, the X-axis denotes the different SBFL techniques, and the Y-axis denotes the different improvement ratio of Exam values. We mark different values of d with four different colors. When d = 0.1, compared with the other three group values, the effectiveness higher than the cases of d = 0 and d = 0.01, but the difference is not obvious compared with d = 1. This is mainly because the number of test models used in this paper is only 100, which is not much larger than the number of rules, but in a production environment, the number of test models should be much larger than the number of rules. So we believe it is necessary to consider the impact of rule inheritance relationship.

VII. RELATED WORK

The current research on model transformation fault localization is mainly divided into two categories: static and dynamic approaches. The static approaches mean that the model transformation program is not executed, the time overhead is small and no input model needs to be generated. The dynamic approach is to execute the model transformation program on one or a set of input models, it is more suitable for small-scale transformation scenarios, and usually can obtain more accurate debugging results.

Hibberd *et al.* [24] propose a static analysis approach to identify the relationship between the source model, the target model, and the transformation rules by analyzing the trace information of the model transformation. Infer the mapping between the generated element types and the rules that may generate these types, ultimately supports fault localization by answering a series of questions related to faults. Wimmer *et al.* [8] provide a model debugging environment, using the Petri-net model transformation language TROPIC to describe the QVT model transformation, and then using its characteristics to implement a debugging environment, providing an interactive platform to support the debugging of model transformation. Burgueno *et al.* [10] use a static analysis approach, extract the types and characteristics of transformation rules and constraints, identify the possible faulty rules by establishing the matching relationship between the rules and the constraints, eventually, when the constraint is violated, it provides a list of suspicious rules related to constraints.

Aranega *et al.* [9] proposes the trace model on the fault localization, by defining and analyzing a trace model describing the execution information of model transformation to determine a series of transformation rules that cause an error. On the basis of static analysis, Troya *et al.* [11] further propose a spectrum-based technique for improving fault localization effectiveness, collecting the information on model transformation runtime in the passing and failing test models and using the idea of the spectrum to identify the model transformation rules that may be faulty. Similar to static analysis, the spectrum-based approach also provides a list of suspicious rules and experimentally demonstrates that the spectrum-based approach is better than the static analysis.

VIII. CONCLUSION AND FUTURE WORK

As the complexity of model transformation increases, the debugging of model transformations becomes more and more difficult. The SBFL can help the debugger locate faults to some extent, but due to the limitations of the spectrum approach, that is, only the rule execution information is considered. We propose an optimized strategy based on the covered range of the test models. The experimental results show that the effectiveness of the fault localization can be improved by using our approach.

However, in our experiments, we also considered the limitations of the approach, that is, it relies on differentiated test models to trigger the execution of different rules, the diversified rule coverage information as a condition for distinguishing the weights of the test models. In further research, we need to consider the impact of different test models

generation methods on the efficiency of fault localization. At the same time, since our study in fault localization is at the rule level, we consider to locate the fault in model transformations at the statement level to meet the needs of more precise fault localization.

REFERENCES

[1] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synth. Lect. Softw. Eng.*, vol. 1, no. 1, pp. 1–182, Sep. 2012.
[2] C. Ouyang, M. Dumas, A. T. Hofstede, and W. Van Der Aalst, "From BPMN process models to BPEL Web services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Chicago, IL, USA, Sep. 2006, pp. 285–292.
[3] T. Hornung, A. Koschmider, and J. Mendling, "Integration of heterogeneous BPM schemas: The case of XPDL and BPEL," in *Proc. CAiSE*, Luxembourg, 2006, pp. 231.
[4] C. Agostinho, H. Bazoun, G. Zacharewicz, Y. Ducq, and H. Boye, "Information models and transformation principles applied to servitization of manufacturing and service systems design," in *Proc. MODELSWARD*, Lisbon, Portugal, 2014, pp. 657–665.
[5] M. Wieber, A. Anjorin, and A. Schürr, "On the usage of TGGs for automated model transformation testing," in *Proc. ICMT*, York, U.K., 2014, pp. 1–16.
[6] E. Denney and B. Fischer, "Generating customized verifiers for automatically generated code," in *Proc. 7th Int. Conf. Generative Program. Compon. Eng. (GPCE)*, Nashville, TN, USA, 2008, pp. 77–88.
[7] L. Lúcio, B. Barroca, and V. Amaral, "A technique for automatic validation of model transformations," in *Proc. MoDELS*, Oslo, Norway, 2010, pp. 136–150.
[8] M. Wimmer, G. Kappel, J. Schoenboeck, A. Kusel, W. Retschitzegger, and W. Schwinger, "A Petri net based debugging environment for QVT relations," in *Proc. IEEE/ACM Int. Conf. Automat. Softw. Eng.*, Auckland, New Zealand, Nov. 2009, pp. 3–14.
[9] V. Aranega, J.-M. Mottu, A. Etien, and J.-L. Dekeyser, "Traceability mechanism for error localization in model transformation," in *Proc. ICSOFT*, Sofia, Bulgaria, 2009, pp. 66–73.
[10] L. Burgueno, J. Troya, M. Wimmer, and A. Vallecillo, "Static fault localization in model transformations," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 490–506, May 2015.
[11] J. Troya, S. Segura, J. A. Parejo, and A. Ruiz-Cortés, "Spectrum-based fault localization in model transformations," *TOSEMACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 1–50, Sep. 2018.
[12] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.
[13] J. A. Jones, M. J. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proc. ICSE*, Orlando, FL, USA. 2002, pp. 467–477.
[14] J. Troya, S. Segura, J. A. Parejo, and A. Ruiz-Cortes, "An approach for debugging model transformations applying spectrum-based fault localization," in *Proc. JISBD*, La Laguna, Spain, 2017.
[15] F. Jouault, "Loosely coupled traceability for ATL," in *Proc. ECMDA*, Nuremberg, Germany, 2005, p. 2.
[16] F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev, "ATL: A model transformation tool," *Sci. Comput. Program.*, vol. 72, nos. 1–2, pp. 31–39, Jun. 2008.
[17] Y. Rhazali, Y. Hadi, and A. Mouloudi, "Model transformation with ATL into MDA from CIM to PIM structured through MVC," in *Proc. ANT/SEIT*, Madrid, Spain, 2016, pp. 1096–1101.
[18] F. Allilaire, J. Bézivin, F. Jouault, and I. Kurtev, "ATL-eclipse support for model transformation," in *Proc. ECOOP*, Nantes, France, 2006, p. 66.
[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the Web," Stanford InfoLab, Stanford, CA, USA, Tech. Rep. SIDL-WP-1999-0120, Nov. 1999.
[20] S. Mirshokraie, A. Mesbah, and K. Pattabiraman, "Guided mutation testing for JavaScript Web applications," *IEEE Trans. Softw. Eng.*, vol. 41, no. 5, pp. 429–444, May 2015.
[21] M. Zhang, X. Li, L. Zhang, and S. Khurshid, "Boosting spectrum-based fault localization using pagerank," in *Proc. ISSTA*, Santa Barbara, CA, USA, 2017, pp. 261–272.
[22] M. Wimmer, S. M. Perez, F. Jouault, and J. Cabot, "A catalogue of refactorings for model-to-model transformations," *J. Object Technol.*, vol. 11, no. 2, pp. 1–40, Apr. 2012.
[23] J. E. Rivera, F. Duran, and A. Vallecillo, "A graphical approach for modeling time-dependent behavior of DSLs," in *Proc. VL/HCC*, Corvallis, OR, USA, 2009, pp. 51–55.
[24] M. Hibberd, M. Lawley, and K. Raymond, "Forensic debugging of model transformations," in *Proc. MoDELS*, Nashville, TN, USA, 2007, pp. 589–604.

PENGFEI LI is currently pursuing the M.S. degree with the School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou, China. His current research interests include system modeling, fault localization, and defect prediction.

MINGYUE JIANG (Member, IEEE) received the B.Sc. degree in computer science and technology from Yunnan Normal University, the M.Eng. degree in computer applied technology from Zhejiang Sci-Tech University, Hangzhou, China, and the Ph.D. degree from the Swinburne University of Technology, Melbourne, VIC, Australia. She is currently a Teacher with the School of Information Science and Technology, Zhejiang Sci-Tech University. Her current research interests include software testing and automated program repair.

ZUOHUA DING (Member, IEEE) received the M.S. degree in computer science and the Ph.D. degree in mathematics from the University of South Florida, Tampa, FL, USA, in 1996 and 1998, respectively.

He is currently a Professor and the Director of the Laboratory of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou, China. He has authored or coauthored more than 70 articles. His current research interests include system modeling, software reliability prediction, intelligent software systems, and service robots.