Diversity-based Test Case Prioritization Technique to Improve Faults Detection Rate

Jamal Abdullahi Nuh[1], Tieng Wei Koh[2], Salmi Baharom[3], Mohd Hafeez Osman[4], Lawal Babangida[5], Sukumar Letchmunan[6], Si Na Kew[7]

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia (UPM), 43400 Serdang, Malaysia[1, 2, 3, 4, 5]
School of Computer Sciences, Universiti Sains Malaysia (USM), 11800 Pulau Pinang, Malaysia[6]
Faculty of Social Sciences and Humanities, Universiti Teknologi Malaysia, 80130 Skudai, Malaysia[7]

***Abstract*—Regression testing is an important task in software development, but it is often associated with high costs and increased project expenses. To address this challenge, prioritizing test cases during test execution is essential as it aims to swiftly identify the hidden faults in the software. In the literature, several techniques for test case prioritization (TCP) have been proposed and evaluated. However, existing weight-based TCP techniques often overlook the true diversity coverage of test cases, resulting in the use of average-based weighting practices and a lack of systematic calculation for test case weights. Our research revolves around prioritizing test cases by considering multiple code coverage criteria. The study presents a novel diversity technique that calculates a diversity coverage score for each test case. This score serves as a weight to effectively rank the test cases. To evaluate the proposed technique, an experiment was conducted using five open-source programs and measured its performance in terms of the average percentage of fault detection (APFD). A comparison was made against an existing technique. The results revealed that the proposed technique significantly improved the fault detection rate compared to the existing approach. It is worth noting that this study is the first of its kind to incorporate the true diversity score of test cases into the TCP process. The findings of our research make valuable contributions to the field of regression testing by enhancing the effectiveness of the testing process through the utilization of diversity-based weighting techniques.**

Keywords*—Regression testing; fault detection; test case prioritization; test case diversity; test case coverage; species diversity

I. Introduction

Despite the importance of regression testing, it has been described as an expensive operation, and various approaches have been developed to overcome this challenge. One of these effective approaches is test case prioritization (TCP), which aims to prioritize the execution of the most critical test cases to detect hidden faults more rapidly during the test execution process. TCP employs several techniques such as algorithms and metrics to reorder test cases. The primary objective of these techniques is to determine the optimal ordering combination of the test suite based on specific criteria.

The most crucial aspect of TCP is the detection of faults, which can only be known after executing the test cases. Therefore, it becomes necessary to estimate or predict the test cases that can achieve this goal before the test execution process begins. During the estimation phase, TCP techniques rely on various code coverage measures as surrogate indicators. These measures directly examine the code and describe how multiple test cases cover the code under test [1]. These coverage measures include line, branch, method, and so on [2], also referred to as criteria. Moreover, relying solely on a single criterion may not be sufficient and can be misleading, as it only represents a portion of the code structure. Thus, considering multiple criteria has been recognized as effective and can potentially identify the test cases that will ultimately enhance the fault detection rate [3]-[6].

Researchers have proposed several weight-based techniques to address TCP problems by incorporating code coverage information [2], [7]-[10]. These techniques integrate different code coverage criteria such as line, branch, method, and more. The objective is to assign weights to each criterion and calculate the final priority value for each test case. However, some of these techniques derive weights based on averages [3], [5], [11], while others assign fixed weight scores to the considered criteria based on specific factors [12], [13]. Such informal practices of weight assignment to test cases and criteria can be deemed unfair [14], as test cases cover distinct code structures and are executed with diverse test contents and different input values [4]. However, test cases with these characteristics can unveil hidden faults within the covered structures. Nonetheless, previous weight-based techniques may not effectively maximize the fault detection rate due to ineffective weightage. This inefficiency primarily stems from the informal characterization of code coverage information, leading to ambiguous and unjust weight scores for the test cases. Consequently, these practices ultimately result in the selection of ineffective test cases that fail to expedite fault detection during the testing process [14]. As the nature of faults is diverse, identification of all fault locations within a program poses challenges due to the distribution of faults [15]. Therefore, the source code information (e.g., code-coverage data) are the best available resource to use as surrogate measure in order to identify the best capable test cases that can execute the fault code and eventually reveal such faults [16]. Hence, the future strategies will depend on the level of understanding and representation, as well as the reformulation of the current code coverage data. These factors will determine the extent to which fault detection can be maximized.

Within the specialized literature, there has been a suggestion regarding the need to develop a new TCP strategy that enhances the fault detection rates of test cases [17], [18].

This motivation has prompted our current study, which aims to explore new research directions by specifically examining the role of diversity within test case coverage data to improve the fault detection rate. While some previous techniques drew inspiration from concepts in Biology or related domains, certain biodiversity concepts, such as species diversity and its associated metrics, have remained unexplored in TCP research. Thus, the concepts of species diversity have served as the inspiration for this study and applying these concepts during the weighting practices is crucial.

In this study, assumption made is that whenever a test case covers a diverse code structure and that program structure is executed correctly, any faults in that area can be revealed. Our goal is to transform the previous test case problem into a species diversity problem and adopt species diversity metrics [30] to measure the true diversities of the test cases based on their coverage counts. Each line of code is treated as unique, and it is assumed that every test case is susceptible to faults. Therefore, the prioritization of test cases is based on their diversity scores, with higher scores indicating coverage of diverse code structures. It is believed that a test case that maintains diverse coverage is better in terms of fault detection compared to a test case that covers few code structures or receives a lower diversity score.

The contributions of this work can be summarized as follows:

- This study represents a pioneering approach as it is the first of its kind to utilize species diversity concepts to weigh test cases based on their true diversities in terms of code coverage counts.
- Through rigorous comparative experimental studies, this study provides empirical validation of the proposed metric's effectiveness in terms of the average percentage of fault detection rate (APFD).

The paper is structured as follows: Section II provides an overview of related works in the field. Section III introduces the proposed technique, while Section IV presents a motivational example. The study experiment is presented in Section V, followed by the presentation of results and analysis in Section VI. Finally, Section VII summarizes the study and suggests potential future research directions.

II. Related Work

TCP is a vibrant area of research, with numerous algorithms and metrics being proposed and evaluated [2], [19], [20]. This section provides an overview of existing studies on weight-based techniques and their research directions.

Earlier studies delved into TCP techniques, aiming to address cost constraints and improve fault detection in regression testing [7], [8]. These studies introduced a general-purpose TCP technique that incorporated statement and branch coverage information. The primary focus was on two classical greedy algorithms: a total greedy algorithm and an additional greedy algorithm.

The literature also explored combinations of various TCP techniques [4], [21]. These studies integrated strategies from the total and additional greedy algorithms and incorporated probabilistic techniques by assigning probability scores to the relevant criteria. These probability values were adjusted based on the specific technique employed.

Criticism has been directed at existing practices that revolve around code isolation units or single-criteria techniques [22]. Such isolations may lead to a loss of valuable coverage information crucial for identifying program faults. Other studies ranked test cases based on estimated coverage information derived from static code structures [23], while some employed multi-criteria decision-making (MCDM) techniques for test case ranking [14].

TCP techniques have also been applied to Object-Oriented Programs (OOP). For instance, in [24], nine coverage criteria were considered, and fixed weights were assigned to each criterion to rank the test cases. In [12], a coupling measure was employed to rank test cases based on a constructed dependency graph. Another study [13] incorporated dependence-based analysis, selecting test cases based on their dependency scores.

Iyad and Khalid [11] introduced the average weight-based technique for TCP, utilizing line of code (LOC) and method coverage as criteria. Test cases with higher scores were assigned higher priority rankings. They evaluated their technique using a small experimental program.

Initial research on multi-criteria weight-based TCP techniques proposed a method incorporating ten factors grouped into four categories: Requirement, time, defect, and complexity [25]. Each factor was assigned a weighted score, and the evaluation focused on factors such as defect severity, prioritization time, and acceptable test case size.

Another weighted technique was proposed by Ahmad et al. [26], where test case execution decisions were based on final weight scores derived from three criteria: pairwise event, frequency pairwise, and fault matrix. These weights were used to prioritize the execution of test cases.

A TCP technique considering multiple coverage criteria was proposed by Prakash and Rangaswamy [3]. The researchers criticized the use of single criteria and existing techniques for being time-consuming and costly. They introduced a multi-coverage Average Weight-based Technique (AWT) and empirically demonstrated its superiority in terms of modified APFD. Building upon this work, Ammar et al. [5] proposed an Enhanced AWT (EAWT) technique that assigned different weights to test cases with similar weight scores, assuming they covered the same code segment and revealed similar faults. However, it is worth noting that this assumption may not hold true in practice, as test cases can vary significantly and possess dissimilar input values.

Several TCP techniques have been presented to enhance fault detection rates, with multi-coverage weighting techniques exhibiting promising performance. However, there has been relatively less focus on TCP techniques for OOP, particularly in Java [27] to [29].

Moreover, single coverage criteria have shown lower effectiveness compared to lightweight metrics, particularly multi-coverage weight-based metrics [3], [22]. However,

existing techniques often treat criteria and test cases, similarly, disregarding their inherent differences in nature and lacking formal weight calculations. Additionally, diversity weighting practices have been largely overlooked in the TCP literature.

Other additional notable gap in the existing literature is that their consideration of prioritizing test cases based on greedy approaches, where only the test case with the highest coverage count is executed. However, this greedy approach fails to assess whether the test case covers multiple criteria or not, as it primarily focuses on maximizing coverage for a single criterion. When dealing with multiple criteria, this approach proves to be inadequate and might even lead to some test cases to receive similar weights. This study argues that test cases that cover multiple criteria in a diverse manner have a higher potential for detecting hidden faults. Since the faults are distributed over the different code structures and the exact locations of faults are unknown, considering diverse coverage becomes crucial in enhancing fault detection capabilities.

A recent survey on TCP and several other studies have emphasized the need for novel techniques, including diversity measures, to improve the current state of TCP techniques [17] [31]. This paper aims to address this need by investigating diversity weighting strategies in test cases, an aspect that has not been previously explored.

III. Proposed Diversity Weighted based Technique

In this section, the proposed technique that considers multiple coverage criteria is discussed including instruction coverage, branch coverage, line coverage, and method coverage. These criteria provide insights into the underlying structure of the source code being tested. Adequate testing of a program necessitates a sufficient number of diverse test cases that target different areas within the code structure. While various TCP techniques with different motivations have been explored in the existing literature to prioritize test cases, our study introduces a unique approach utilizing the species diversity metric, originally employed in ecology to measure species diversity [30]. Therefore, these measures were selected for the following reasons:

- These metrics address the identified gap and serve the purpose of calculating the true diversity score of test cases across multiple coverage criteria.
- In contrast to previous informal weight practices, the selected metric assigns formal and unique weights to the test cases, describing their diverse code coverage characteristics.
- By employing these measurements, each test case can be identified as a unique entity, and their unique diversity scores can be used as tie-breaking strategy and ranking priorities. This means that test cases with higher diversity are ranked higher than those with lower diversity.

On the other hand, the proposed approach encompasses two stages: (1) the adoption of four dynamic code coverages, which are treated as species-based problems, and (2) the introduction of a novel diversity weighting technique that computes the Final Priority Value (FPV) for each test case.

A. Adopt and Charecterize Code Coverage as a Species based Approach

The code coverage criteria adopted in this study encompass instruction coverage, branch coverage, line coverage, and method coverage. This section provides a description of how the TCP problem was formulated, utilizing a species-based approach. This characterization aims to enhance the effectiveness of the test execution process.

Code coverage information consists of multiple test cases, each associated with their respective coverage counts for various code structures, also known as criteria. In this study these entities were treated as a species-based problem and introduce the following terms, definitions, and symbols to facilitate our discussion and analysis:

Definition 1: Test cases are analogous to species(s), and it is important to emphasize that each species is unique. Test cases are purposefully designed with different test inputs and the ability to target various parts of the system under test (SUT).

Definition 2: A test suite, consisting of a group of unique species, represents a community(s). S is defined as S = {s1, s2…. si}, where i represents the ith species in S. s1 denotes the first species, while si denotes the last species in the set.

Definition 3: Code structures or criteria are considered as species habitat (HB): In this case, the criteria include branch coverage (br), instruction coverage (in), line coverage (li), and method coverage (me). HB is defined as HB = {Hb1, Hb2…Hbj}, where j represents the jth habitat in HB (e.g., in, br, li, and me). A species can be found in any of these habitats, for example, s1 can be present in Hb1, Hb2, Hb3 or Hb4, and their presence is recorded in terms of coverage counts.

Definition 4: The coverage counts are denoted as n, representing the number of ith species (si) found in jth habitat (Hbj), which is also written as nij. It is important to note that the coverage counts, n, can range from 0 to any non-negative number. A count of 0 indicates that the habitat is not represented by that species, indicating a loss of species.

Definition 5: The total coverage count of each species across all habitats is denoted as N.

B. Diversity-Aware TCP Technique

In this paper, a species diversity metric that measures the diversity of each species in relation to its habitats was used. This metric allows us to identify species with higher diversity, indicating their effectiveness and prioritization during the test execution process.

The proposed Diversity-aware TCP technique integrates multiple code coverage metrics to calculate a unique diversity weight for each test case. This diversity weight is determined by the extent of diversity exhibited by the species covering multiple code structures, also known as criteria. Our assumption is that species with higher coverage diversity scores are more likely to uncover hidden faults.

To illustrate the methodology employed in the proposed Diversity-based TCP technique, the following steps are outlined:

a) Step one: Collecting Coverage Values. In this step, the test case coverage needs to be collected for various criteria using tools such as JaCoCo within the JUnit framework.

b) Step two: Calculating Diversity Score. In this step, the species diversity score is calculated using the Gini-Simpson's index, which is a well-known diversity measure. This metric takes into account the dominance of species and assigns higher weight to abundant species. The proposed Gini-Simpson index (1-D) is derived from the original Simpson's index (D) [30]. The formulas for calculating the Gini-Simpson index are as follows:

$$1 - D(s_i) = 1 - \left(\frac{\sum_{i=1}^{j} n_{ij}(n_{ij}-1)}{N_i(N_i-1)} \right) \quad (1)$$

where,

1-D(s_i) denoted as the Gini–Simpson index (1-D) of the *i*th species;

$n_{i,j}$ represents the count of species from the *i*th species under the *j*th habitat;

N_i represents the total count of coverage for the *j*th species across all habitats; and

Σ denotes the sum of multiple terms.

c) Step three: Ranking the Species. In this step, the prioritization of species occurs after calculating the diversity scores using the specified metric. The species are ranked based on these scores in descending order, from highest to lowest.

IV. Motivational Example

To illustrate the functionality of the proposed technique, a demonstration is provided in this section using the information presented in Table I. The table includes eight species (T1 to T8) and four habitats (*in*, *br*, *li*, and *me*). Based on this coverage information, the test cases were assigned weights according to their diversity. The demonstration involves the following phases:

1) Collect the coverage counts for each species in relation to the selected habitats.

2) Calculate the diversity score for each species based on their habitats. This step may include a subset step specific to the calculation process of the chosen metric.

3) Rank the test cases in descending order from highest to lowest value.

1) Phase one: Collecting Coverage Counts

During this phase, the coverage counts of species need to be collected with respect to the selected habitats. Tools such as JaCoCo can be utilized to facilitate this process. The data provided below represents a subset of species obtained from the CruiseControl program, along with the habitats they cover, as shown in Table I. It is important to note that the program consists of 299 species, but for the purpose of demonstration, only eight species was selected from the program.

TABLE I. Species Coverage Counts of Cruisecontrol Program

Species	IN	BR	LI	ME
T1	74	2	27	8
T2	97	8	32	7
T3	87	5	32	6
T4	31	1	12	4
T5	34	1	14	3
T6	41	2	17	5
T7	127	10	31	5
T8	111	2	40	9

2) Phase two: Calculate the Diversity Score

To calculate the diversity associated with each species across the selected habitats, the diversity score is used as a weight for the species. The calculation process can be outlined as follows.

The following example provides guidelines on how to calculate the diversity score using the given data, specifically focusing on the Gini-Simpson index as described in equation (1). Please note that the calculation process for other metrics can be carried out using a similar approach. The following steps are involved in this process:

- Step 1: calculate total coverage count of ith species across all habitats, denoted as Ni where Ni = Σ ni,j. In this case, NT8 = (111+2+40+9) = 162, NT7= (127+10+31+5) = 173, and the remaining values are listed in column two in Table II.
- Step 2: calculate Ni (Ni -1) for each species e.g., NT8 (NT8 -1). In this case, T8 = 162*(162-1) = 26082, T7 = 173*(173-1) = 29756, and the remaining values are listed in column three in Table II.
- Step 3: calculate ni,j (ni,j -1) of ith species in jth habitat e.g. nT8me(nT8me -1). In this case, T8 = 9*(9-1) = 72, T7 = 5*(5-1) = 20, and the remaining values are listed in columns four and five in Table II.
- Step 4: calculate Σ ni,j (ni,j -1) of ith species in jth habitat. In this case, the results of this calculation are listed in column two in Table III.
- Step 5: calculate D where D = Σ ni,j (ni,j -1)/(Ni (Ni -1)) (diversity weight of Simpson). In this case, the results of this calculation are listed in column three in Table III.
- Step 6: calculate 1-D (diversity weight of Gini–Simpson metric). This is the result needed to rank test cases. The results of this calculation are listed in column four in Table III.

TABLE II. Steps 1-3: Coverage Values and Test Cases of CruiseControl Program

	Step 1	Step 2	Step 3: $n_{i,j}(n_{i,j}-1)$			
Species	N_i	$N_i (N_i -1)$	*IN*	*BR*	*LI*	*ME*
T1	111	12210	5402	2	702	56
T2	144	20592	9312	56	992	42
T3	130	16770	7482	20	992	30

	Step 1	Step 2	Step 3: $n_{i,j}(n_{i,j}-1)$			
Species	N_i	$N_i (N_i -1)$	*IN*	*BR*	*LI*	*ME*
T4	48	2256	930	0	132	12
T5	52	2652	1122	0	182	6
T6	65	4160	1640	2	272	20
T7	173	29756	16002	90	930	20
T8	162	26082	12210	2	1560	72

TABLE III. STEP 4-6: COVERAGE VALUES AND TEST CASES OF CRUISECONTROL PROGRAM

	Step 4	Step 5: D	Step 6
Species	$\Sigma\, n_{i,j}(n_{i,j}-1)$	$\Sigma\, n_{i,j}(n_{i,j}-1)/ N_i (N_i -1)$	*1-D*
T1	6162	0.5047	0.4953
T2	10402	0.5051	0.4949
T3	8524	0.5083	0.4917
T4	1074	0.4761	0.5239
T5	1310	0.494	0.506
T6	1934	0.4649	0.5351
T7	17042	0.5727	0.4273
T8	13844	0.5308	0.4692

3) Phase three: Rank the Test Cases

In the second phase, the diversity score of each species is computed using the suggested metric mentioned earlier. This score serves as a priority value, allowing the ranking of species from highest to lowest based on this value. The resulting ranked species are as follows:

$1\text{-}D(s_i)$ = T6, T4, T5, T1, T2, T3, T8, T7.

V. EXPERIMENTS

The objective of this experimentation is to evaluate the fault detection rate of the diversity weighted technique. In this section, an overview of the steps and tools used to assess the effectiveness of the proposed weighted technique in TCP is provided. A comparative analysis is conducted between the proposed weighted technique and an existing weighted technique, including [5].

A. Experimental Goal

The main focus of this paper is to prioritize test cases that can effectively detect hidden faults in a program during the early stages of the execution process. The objective is to determine which technique, between the proposed diversity weighted technique and an existing weight-based technique, exhibits a higher fault detection rate. The research question being investigated is whether the proposed technique outperforms the existing technique in terms of fault detection rate.

B. Study Objects

This study utilized five object-oriented programs, namely CruiseControl (A), DisjointSets (B), AccountSubType (C), Losenotify (D), and Odset (E). These programs were obtained from the Software Artifact Infrastructure Repository and have been previously utilized in other TCP studies [27], [13]. In this study, the entire programs were analyzed without dividing them into different versions.

The characteristics of the programs were measured, including lines of code (LOC), number of classes (NOC), number of species (NOS), and number of mutants (NOM). Unlike previous studies [13], certain characteristics, specifically the program sizes in terms of NOC and LOC, were calculated differently in this study. The measurements were obtained after generating all the program's test cases using an automated tool called o3smeasures, which is an Eclipse plugin. Table IV illustrates that the program sizes varied from 684 to 4989 LOC, while the number of species ranged from 15 to 299 NOS.

The implementation of the study object was done using the Java programming language, and the test cases (species) were written using the JUnit-5 framework. The JUnit species were generated using a tool called Randoop, which automatically generates unit tests for Java classes. The test coverage for these species was calculated using the JaCoCo agent, which is also an Eclipse plugin, taking into account all the desired coverage criteria. Furthermore, program faults were intentionally introduced using popular mutant generation tools called MuJava (μJava) [27], [13].

TABLE IV. STUDY OBJECT CHARACTERISTICS

ID	Objects	LOC	NOC	NOS	NOM
A	CruiseControl	4958	6	299	9
B	DisjointSets	1809	5	15	2
C	AccountSubType	684	8	53	27
D	Losenotify	1463	6	132	6
E	Odset	4989	4	167	5

C. Performance Measures

To compare the effectiveness of different techniques, the Average Percentage of Faults Detected (APFD) is commonly used as a standard metric. This metric facilitates the comparison of fault detection rates achieved by different techniques, aiding in the determination of the most effective approach. The objective is to maximize the fault detection rate by executing the test cases. APFD is well-suited for this task as it provides test engineers with prompt feedback, enabling the early identification and resolution of faults.

Let T represent the test case community, m represent the total number of faults detected in a specific object, n represent the total number of test cases (species), and TF_i denote the position of the first test case that detects the ith fault. The APFD formula is as follows:

$$APFD = 1- [(TF1+TF2+...+TFm)/(m*n)] +1/(2*n) \quad (2)$$

A higher APFD rate indicated better performance, and the results were reported as a percentage to quantify the differences.

VI. RESULT

This section presents the experimental results for all five Java programs when applying the proposed technique using Equation 1. The results were carefully organized, summarized, and presented.

To prioritize species from multiple programs that cover different habitats, the proposed technique was compared to an existing TCP technique. Since these techniques rank the species differently, they can yield distinct results. The evaluation of these techniques was performed using the APFD metric (see Equation 2). Each program received an APFD score (in percentages) from both techniques.

The experiment's findings are summarized and illustrated using a bar chart (refer to Fig. 1) for visual representation of the data. The horizontal axis (x-axis) of the chart represents the five employed programs, identified by their respective ID labels. The vertical axis (y-axis) represents the APFD scores obtained by the different TCP techniques after applying them to the object programs. The APFD score, ranging from 0 to 100 percent, serves as a performance indicator, where higher scores indicate better results.

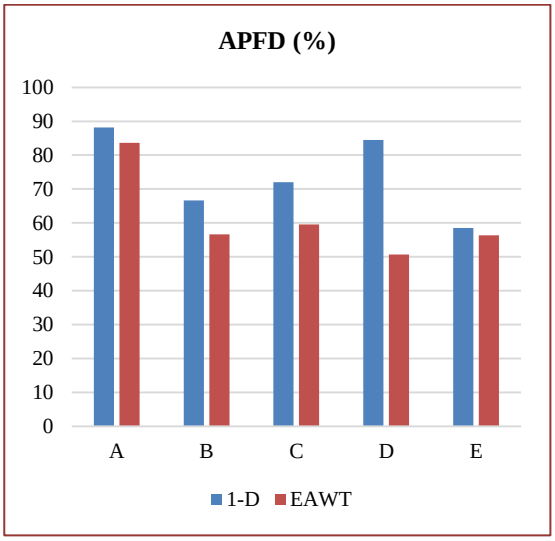

Fig. 1. APFD values of 1-D and EAWT.

Fig. 1 presents a comparison between the diversity-weighted technique utilizing the Gini-Simpson index and the EAWT technique. The data clearly indicates that the newly proposed technique exhibits strong performance across all the programs.

The EAWM technique exhibited the highest and lowest APFD scores among all the programs, achieving 83.67% for program A and 50.63% for program D, respectively. In contrast, the proposed technique consistently achieved the highest or second highest APFD scores across the programs. Specifically, it scored 88.13% and 84.47% for programs A and D, respectively, outperforming the EAWM technique. It is noteworthy that the EAWM technique demonstrated weaker performance in multiple programs, particularly programs D, E, and B, where APFD scores ranged from 50.63% to 56.67%. In comparison, the proposed technique consistently delivered higher APFD scores in those programs. The lowest APFD score obtained by the proposed technique was 58.50% for program E, which still surpassed the corresponding technique's score of 56.35% for the same program. These findings indicate the superior performance of the proposed technique across various programs, even in scenarios with lower APFD scores.

After comparing the APFD results for each object under different techniques, it was evident that the proposed technique based on the Gini-Simpson index (1-D) outperformed the existing weight-based technique (EAWT). The 1-D technique exhibited a substantial improvement in APFD scores, with a mean difference of 12.62% higher than the EAWT technique.

Although the objects varied in size and the techniques produced different rankings for the species, there were indications that prioritizing test cases based on their true diversity score had the potential to achieve higher APFD scores. While certain programs received lower APFD scores, this could be attributed to the nature and distribution of faults within those programs. Another possible reason for the poorer performance of the existing technique could be its feature of postponing certain fault-revealing species. In contrast, the proposed technique avoided species postponement, especially when they received similar final weight values. Conversely, the existing technique delayed some species assuming their similarity in fault identification, which could lead to slower detection of certain faults.

VII. Analysis and Discussions

The results obtained from the experiment conducted clearly stated that the proposed diversity-weighted technique was effective when compared with the existing weight-based technique. This means the proposed technique prioritizes test cases based on their true diversity score, and therefore achieved higher fault detection rates, as depicted by the APFD scores. The obtained results were consistent across all the Java programs considered in this study.

Fig. 1 presents a comparison between the diversity-weighted technique using the Gini-Simpson index (1-D) and the existing technique (EAWT) in terms of APFD scores across all programs. It is evident that the proposed technique consistently outperforms the existing technique. The EAWT technique shows varying levels of performance, with the highest and lowest APFD scores achieved in programs A and D, respectively. On the other hand, the proposed technique consistently achieves the highest APFD scores across all programs.

The APFD results strongly support the effectiveness of the proposed technique. The proposed technique (1-D) consistently outperforms the EAWT technique, with a mean difference reaching up to 12.62%. These findings indicate that prioritizing test cases based on their true diversity scores can significantly improve the fault detection rate.

The higher performance reported in the proposed diversity weighted technique of the Gini-Simpson index can be attributed to its ability of considering the test case's true diversity based on their multiple coverage counts. Such features include treating each test cases and each line of code as unique entity and formally assigning diversity scores accordingly, the proposed technique ensures that test cases covering diverse code structures with higher diversity score are prioritized first. This strategy is different from those in the existing technique, which relies on average-based weight assignment and thus might eventually result in unfair and ambiguous weight scores for test cases.

The variations in APFD scores among the programs can also be attributed to the nature and distribution of faults within the programs. Furthermore, the decreased fault detection performance of the existing technique may be attributed to its feature of postponing certain fault-revealing test cases, especially when they receive similar weight scores. In contrast, the proposed technique avoids postponing strategies and only aim to prioritize test cases based on their true diversity scores, leading to more effective fault detection.

Overall, the findings of this study highlight the significance of incorporating true diversity scoring into the test case prioritization process. Employing diversity-based metrics during the test case weighting enhances the effectiveness of regression testing by considering only those test cases that cover diverse code structures and executing them first during the test execution cycles. This approach improves the fault detection rate and contributes to more efficient and cost-effective software development.

VIII. Threat to Validity

This section describes the validity threats that might arise during the experiments. In this study, programs from the SIR repository were adopted, and their corresponding test cases and mutants were generated using automated tools employed in previous related studies. However, it is important to note that these programs might be outdated, and the tools used may have limitations in generating effective test cases or diverse mutants. This threat (internal validity) was mitigated by addressed by adopting recent tools that are widely utilized in the literature or continuously updated tools were selected.

On the other hand, external validity threats related to the generalizability of the results were also considered. These threats pertain to the subject programs, their test cases, and their mutants, which may affect the external validity. To mitigate these concerns, six Java programs with over six hundred test cases were selected from a reputable open-source repository. However, it is important to acknowledge that there may still be limitations within this context, which can be addressed in future research.

IX. Future Work

In future research, it is recommended to explore the following directions.

In future research, it is recommended to conduct a comparison study between the proposed approach, and the existing ones by applying them to a diverse set of object programs. Such a comparison might provide further insights into the effectiveness of these different approaches in fault detection.

While this study is the first of its kind to investigate the effectiveness of species diversity metrics, the focus has been on the Gini-Simpson index (1-D). However, there are several other species diversity metrics that are worth investigating in the future. This will provide additional insights into the effectiveness of different metrics and their impact on the analysis of diversity in various domains.

The study also suggests improving the informal practices of assigning weights to different criteria of interest. Therefore, investigating the role of the species diversity metrics in formalizing the weighting practices for these criteria before calculating the final priority value of the test cases is recommended.

X. Conclusion

In the context of regression testing, prioritizing test cases is a critical task aimed at optimizing their execution order based on specific criteria to enhance the effectiveness of the testing process.

In this study, a novel diversity-based TCP technique that incorporates multiple code coverage criteria and assigns weights to individual test cases was proposed. Each test case was treated as a unique species, while the coverage criteria were considered as habitats. To quantify the diversity within each test case across its covered habitats, a new diversity metric was introduced. The diversity scores obtained were then used as a basis for ranking the test cases, with higher scores indicating a higher potential for fault detection.

To evaluate the effectiveness of the proposed TCP technique in terms of fault detection, an experiment was conducted using five open-source programs and compared the results with an existing technique. Our proposed diversity-based technique consistently outperformed the existing technique, achieving higher scores in terms of the APFD across all tested programs.

The results obtained from our proposed technique highlight its ability to improve the fault detection rate.

Acknowledgment

This work received partial support from the Fundamental Research Grant Scheme (FRGS) under grant number FRGS/1/2019/SS06/UPM/02/6, project code 05-01-19-2199FR, and vote number 5540324, funded by the Ministry of Education Malaysia.

References

[1] M. Khatibsyarbini, M. Isa, D. N. A. Jawawi, and R. Tumeng, “Test case prioritization approaches in regression testing: A systematic literature review,” Information and Software Technology, vol. 93, pp. 74–93, 2018.

[2] Z. Li, M. Harman, and R. M. Hierons, “Search algorithms for regression test case prioritization,” IEEE Transactions on Software Engineering, vol. 33, no. 4, pp. 225–237, 2007.

[3] N. Prakash and T. R. Rangaswamy, “Weighted method for coverage-based test case prioritization,” Journal of computational Information systems, 2013.

[4] D. Hao, L. Zhang, L. Zhang, G. Rothermel, and H. Mei, “A unified test case prioritization approach,” ACM Transactions on Software Engineering and Methodology, vol. 24, no. 2, p. 10, Dec. 2014.

[5] A. Ammar, S. Baharom, A. A. A. Ghani, and J. Din, “Enhanced weighted method for test case prioritization in regression testing using unique priority value,” 2016 International Conference on Information Science and Security (ICISS), pp. 1–6, 2016.

[6] J. Ahmad, and S. Baharom, (2017). “A systematic literature review of the test case prioritization technique for sequence of events,” International Journal of Applied Engineering Research, 12(7), 1389-1395.

[7] G. Rothermel, R. Untch, C. Chengyun, and M. J. Harrold, "Prioritizing test cases for regression testing," IEEE Transactions on Software Engineering, vol. 27, no. 10, pp. 929–948, 2001.

[8] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: A family of empirical studies," IEEE transactions on software engineering, vol. 28, no. 2, pp. 159–182, 2002.

[9] A. Arrieta, S. Wang, U. Markiegi, G. Sagardui, and L. Etxeberria, "Employing multi-objective search to enhance reactive test case generation and prioritization for testing industrial cyber-physical systems," IEEE Transactions on Industrial Informatics, vol. 14, no. 3, pp. 1055–1066, March 2018.

[10] C. Fang, Z. Chen, K. Wu, and Z. Zhao, "Similarity-based test case prioritization using ordered sequences of program entities," Software Quality Journal, vol. 22, no. 2, pp. 335–361, Jun. 2014.

[11] A. Iyad and M. O. N. Khalid, "Combined source code approach for test case prioritization," ICISS '18: Proceedings of the 2018 International Conference on Information Science and System, pp. 12–15, Apr. 2018.

[12] S. Panda, D. Munja, and D. P. Mohapatra, "A slice-based change impact analysis for regression test case prioritization of object-oriented programs," Advances in Software Engineering, vol. 2016, pp. 1–20, May 2016.

[13] C. R. Panigrahi and R. Mall, "A heuristic-based regression test case prioritization approach for object-oriented programs," Innovations in Systems and Software Engineering, vol. 10, no. 3, pp. 155–163, 2014.

[14] A. D. Shrivathsan, R. Krishankumar, A. R. Mishra, K. S. Ravichandran, S. Kar, and V. Badrinath, "An integrated decision approach with probabilistic linguistic information for test case prioritization," Mathematics, vol. 8, no. 11, 2020.

[15] Y. Wang, X. Chen, W. Zhou, X. Liu, J. Li, and G. Lu, "Using Algebra Graph Representation to Detect Pairwise-Constraint Software Faults," in IEEE Access, vol. 8, pp. 184550–184559, 2020.

[16] T. B. Noor and H. Hemmati, "A similarity-based approach for test case prioritization using historical failure data," 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), pp. 58–68, 2015.

[17] R. Mukherjee and K.S. Patnaik, "A survey on different approaches for software test case prioritization," Journal of King Saud University - Computer and Information Sciences, vol. 33, no. 9, pp. 1041–1054, 2021.

[18] H. Hemmati, "Advances in techniques for test prioritization," Advances in Computers, vol. 112, pp. 185–221, 2019.

[19] K. R. Walcott, M. lou Soffa, G. M. Kapfhammer, and R. S. Roos, "Timeaware test suite prioritization," in International symposium on Software testing and analysis, 2006, pp. 1–12.

[20] S. Wang, S. Ali, A. Gotlieb, D. Pradhan, D. Buchmann, and M. Liaaen, "Multi-objective test prioritization in software product line testing: an industrial case study," SPLC '14: Proceedings of the 18th International Software Product Line Conference, vol. 1, pp. 32–41, Sep. 2014.

[21] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei, "Bridging the gap between the total and additional test-case prioritization strategies," In 2013 35th International Conference on Software Engineering (ICSE), pp. 192–201, 2013.

[22] R. Huang, Q. Zhang, D. Towey, W. Sun, and J. Chen, "Regression test case prioritization by code combinations coverage," Journal of Systems and Software, vol. 169, p. 110712, 2020.

[23] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and G. Rothermel, "A static approach to prioritizing JUnit test cases," IEEE Transactions on Software Engineering, vol. 38, no. 6, pp. 1258–1275, 2012.

[24] N. Chauhan, "A Multi-Factor Coverage Based Test Case Prioritization Technique for Object Oriented Software". International Journal of System & Software Engineering, 3(1), pp. 18-23, 2015

[25] T. Muthusamy and K. Seetharaman, "Efficiency of test case prioritization technique based on practical priority factor," International Journal of Soft Computing, vol. 10, no. 2, pp. 183-188, 2015.

[26] J. Ahmad, S. Baharom, A. A. Abd Ghani, H. Zulzalil, and J. Din, "Measuring the Effectiveness of TCP Technique for Event Sequence Test Cases," In Science and Information Conference, Springer, Cham, 2018, vol 857, pp. 881-897.

[27] R. Mukherjee and K. S. Patnaik, "Prioritizing JUnit Test Cases Without Coverage Information: An Optimization Heuristics Based Approach," IEEE Access, vol. 7, pp. 78092–78107, 2019.

[28] H. Do, G. Rothermel, and A. Kinneer, "Empirical studies of test case prioritization in a JUnit testing environment," Proceedings - International Symposium on Software Reliability Engineering, ISSRE, pp. 113–124, 2004.

[29] V. KS and S. Mathew, "Test case prioritization and distributed testing of object-oriented program," Turkish Journal of Electrical Engineering & Computer Sciences, vol. 27, no. 5, pp. 3582–3598, 2019.

[30] E. H. Simpson, "Measurement of diversity," Nature, vol. 163, pp. 688–688, 1949.

[31] R. Feldt, S. Poulding, D. Clark, and S. Yoo, "Test set diameter: Quantifying the diversity of sets of test cases," 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST), pp. 223–233, 2016.