A Comparative Study of Vectorization-Based Static Test Case Prioritization Methods

Hirohisa Aman
Center for Information Technology
Ehime University
Matsuyama, Ehime, Japan
aman@ehime-u.ac.jp

Sousuke Amasaki
Tomoyuki Yokogawa
Faculty of Computer Sc. & Systems Eng.
Okayama Prefectural University
Soja, Okayama, Japan

Minoru Kawahara
Center for Information Technology
Ehime University
Matsuyama, Ehime, Japan

***Abstract*—To enhance the efficiency of software testing, researchers have studied various test case prioritization (TCP) methods. A topic model-based TCP is one of the promising methods, which expresses test cases by topic vectors and prioritizes them in the order such that the set of already-prioritized test cases have the maximum dispersion in the vector space. However, the topic model is not the only option available for vectorizing test cases. Moreover, the distance metric in the vector space and the scheme to prioritize test cases (the way to find the test case that is the farthest from the set of already-prioritized ones) also have some available options. Because the combinations of the above options have not been well-discussed in the past, this paper conducts a comparative study of 36 TCP methods, which are the combinations of (1) three vectorization methods, (2) three distance metrics, and (3) four prioritization schemes (36=3x3x4). The empirical results show the following findings. The choice of the vectorization method has a significant impact on the testing efficiency: a promising option is Doc2Vec (PV-DBoW). The combination with the distance metric may also be impactful: a useful combination is Doc2Vec (PV-DBoW) and Euclidean distance. The third aspect, i.e., the choice of the scheme to find the farthest test case, is not always influential.**

***Index Terms*—testing, test case prioritization, vectorization**

I. Introduction

Software testing is an essential activity to ensure the quality of the software system under development and maintenance. Developers prepare various test cases and execute them to minimize the risk of missing faults that lie latent in their software system, *the system under test* (SUT) [1]. Although it is ideal for executing a great variety of test cases, a testing activity sometimes becomes expensive and time-consuming [2], [3]. To overcome such a challenge, researchers have studied various approaches for promoting the testing efficiency [4]. *Test case prioritization* (TCP) [5]–[7] is one of the most useful testing approaches.

TCP approaches seek to find the ideal execution order of test cases to detect more faults as early as possible. The prioritization aims at maximizing the expected number of detected faults even if the testing is halted at an arbitrary point. Various TCP approaches have been proposed in the past. For example, Jeffrey and Gupta [8] proposed to utilizes the program slicing technique to link a modified part with the related test cases and prioritize test cases by using the slice-based data. Carlso et al. [9] leveraged a clustering approach together with code coverage information and code complexity metric data to prioritize test cases. Kim and Porter [10] focused on the test history (the results of prior test executions) and proposed to quantify the expectation that a test case causes a failure at the next test. Moreover, there have been studies utilizing the genetic algorithm (GA) that seeks the best execution order of test cases in which their fitness functions (evaluation functions) use the above test-related data [11].

Although the above TCP approaches are useful, they require the test case execution history or advanced program analysis (e.g., the program slicing), which may not always be easy to collect or perform [12]. If we add new test cases to the test suite, we cannot obtain the test history of them. When SUT is a large-scale system, it is highly expensive and time-consuming to perform an advanced program analysis for all programs included in the SUT. Therefore, researchers have recently studied light-weight static TCP approaches as well, which require neither the execution of test cases nor an advanced program analysis like the program slicing [13]–[16]; A recent approach focuses on the textual data of test cases [13], which evaluated the similarity between test cases by using the string edit distance (Manhattan distance) and prioritized test cases through maximizing the diversity of test cases [14], [15]. Thomas et al. [16] proposed a more sophisticated approach that is based on the topic model [17], [18]. In [16], Thomas et al. performed a topic analysis for all test cases and expressed test cases by the corresponding topic vectors. Then, they proposed to prioritize test cases through maximizing the diversity of test cases in the topic vector space.

Although Thomas et al. empirically proved that the topic vector-based approach is useful for prioritizing test cases effectively, we have a simple question "How does it work when we replace the vectorization method with another one?" In the world of natural language processing, the Doc2Vec [19]–[21] has become a promising method for producing a vector representation of a document in recent years. That is, a Doc2Vec-based approach may be a better option for the vectorization-based static TCP methods.

Moreover, we have some options for the distance metric as well. To quantify the distance between test cases, we need to define the distance metric in the vector space. Although the previous work [16] used the Manhattan distance, the use of

different distance metrics may have an impact on the TCP performance. Furthermore, we also need to focus on the way of maximizing the diversity of test cases. During the test case prioritization process, the following two steps are iterated: (a) find the test case that is the farthest from the set of already-prioritized test cases, and (b) append the farthest test case to the set of already-prioritized ones. Here, to decide the farthest test case, we have to define the distance between a single test case and a set of test cases. The clustering methods [22] provide some options for such distance computation.

Hence, we can consider the following three aspects that compose a vectorization-based static TCP method: (1) the vectorization method, (2) the distance metric, and (3) the prioritization scheme (the way to find the test case that is the farthest from the set of already-prioritized ones). To the best of our knowledge, the combinations of the above three aspects have not been well-discussed. Thus, toward a better static TCP method, we conduct a comparative study and report the results in this paper. The key contributions of this paper include:

- We report the performances of 36 vectorization-based static TCP methods, which are combinations of
  - Three vectorization methods: topic model, Doc2Vec (PV-DM), and Doc2Vec (PV-DBoW),
  - Three distance metrics: Manhattan distance, Euclidean distance, and angular distance, and
  - Four prioritization schemes: single linkage-based scheme, complete linkage-based scheme, average linkage-based scheme, and Ward's method-based scheme.
- We empirically proved that the choice of the vectorization method has a significant impact on the testing efficiency, and the combination with the distance metric may also be impactful: promising options are Doc2Vec (PV-DBoW) and Euclidean distance. The third aspect, i.e., the choice of the scheme to find the farthest test case, did not always become influential.

The remainder of this paper is organized as follows. Section II describes the TCP method of interest and the details of the above three aspects that compose a TCP method. Section III presents the results of our comparative study of vectorization-based TCP methods. Finally, Section IV concludes the paper and outlines directions for future work.

II. Vectorization-Based Static Test Case Prioritization

In this section, we briefly explain the TCP method of interest (Sects. II-A, II-B) and describe the details of three aspects that compose a TCP method: (1) the vectorization method (Sect. II-C), (2) the distance metric (Sect. II-D), and (3) the prioritization scheme (Sect. II-E). Then, we summarize the combinations of their options that are the subjects of our comparative study (Sect. II-F).

A. Static TCP and Its Evaluation Criterion

We present a formal definition of the static TCP.

TABLE I
Simple example of relationships among test cases and detectable faults (fault matrix)

		Test Case				
		t_1	t_2	t_3	t_4	t_5
Detectable Fault	F_1		✓		✓	
	F_2					✓
	F_3		✓			
	F_4	✓				✓

Def. 1 (Static test case prioritization problem): Given a test suite T, which consists of $n(>0)$ test cases t_i (for $i = 1, 2, \ldots, n$). Let $\mathcal{PT}$ be the set of all permutations of T. The static test case prioritization seeks to find the permutation $PT_o \in \mathcal{PT}$ such that

$$\forall PT(\in \mathcal{PT})\ f(PT_o) \geq f(PT)\ , \tag{1}$$

where $f(\cdot)$ is the evaluation function of a permutation; The higher the value, the better the permutation, in terms of the test case prioritization. □

For example, when we have test suite $T = \{\ t_1, t_2, t_3\ \}$, $\mathcal{PT}$ consists of six permutations: $\mathcal{PT} = \{\ (t_1, t_2, t_3), (t_1, t_3, t_2), (t_2, t_1, t_3), (t_2, t_3, t_1), (t_3, t_1, t_2), (t_3, t_2, t_1)\ \}$. The static TCP is to find the best permutation in these six candidates, which has the highest evaluation value computed by $f(\cdot)$.

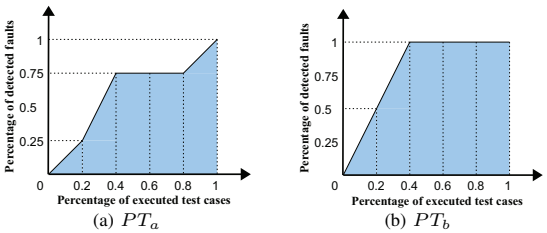
According to previous studies of TCP methods [4], [16], we adopt the *Average Percentage of Fault-Detection* (APFD) [7] as our evaluation function in Eq.(1).

Def. 2 (Average Percentage of Fault-Detection (APFD)): Given a permutation of the test suite, PT. Suppose $m(>0)$ different faults are detectable by executing all test cases in PT. The Average Percentage of Fault-Detection (APFD) of PT is defined as

$$\mathrm{APFD}(PT) = 1 - \frac{1}{nm}\sum_{i=1}^{m} TF_i + \frac{1}{2n}\ , \tag{2}$$

where $TF_i = k$ if and only if the i-th fault is first detected by executing the k-th test case in PT. □

To demonstrate a TCP evaluation using the APFD, we consider an example shown in Table I, where we have five test cases $T = \{t_1, t_2, t_3, t_4, t_5\}$ and four detectable faults $\{F_1, F_2, F_3, F_4\}$. The checkmarks (✓) indicate that the corresponding fault is detectable by the corresponding test case(s)[1];



(a) PT_a (b) PT_b

Fig. 1. Comparison of fault detection processes between PT_a and PT_b.

[1]A test case can find at most one "failure." After a fail of a test, we analyze the SUT and detect the fault causing the failure. In the analysis, we may detect two or more faults. Test cases t_2 and t_5 in Table I correspond to such cases.

For example, F_1 is detected when we execute t_2 or t_4.

Now we consider two permutations PT_a and PT_b: $PT_a = (t_1, t_2, t_3, t_4, t_5)$, and $PT_b = (t_2, t_5, t_1, t_3, t_4)$. Fig. 1 shows the differences between the fault detection processes of PT_a and PT_b. In the figure, the horizontal axis signifies the percentage of executed test cases. Because we have five test cases in this example, one execution of a test case corresponds to a 0.2 increase on the axis. The vertical axis indicates the percentage of detected faults, and it increases by 0.25 with every detection because we have four detectable faults in total. The earlier the curve grows, the better the TCP is. Hence, TP_b is better than TP_a in terms of early fault detection.

The APFD is the area under the curve shown in Fig. 1, and can be a reasonable evaluation value of the corresponding TCP method. By Eq.(2), we obtain the following evaluations:

- $\mathrm{APFD}(PT_a) = 1 - \frac{1}{20}(2+5+2+1) + \frac{1}{10} = 0.6$, and
- $\mathrm{APFD}(PT_b) = 1 - \frac{1}{20}(1+2+1+2) + \frac{1}{10} = 0.8$,

i.e., we get the relationship $\mathrm{APFD}(PT_a) < \mathrm{APFD}(PT_b)$, which indicates that TP_b is better than TP_a.

B. Steps of Vectorization-Based Static TCP

We can perform a vectorization-based static TCP in the following steps.

1) *Vectorization of Test Cases*:
   As the first step, we vectorize our test cases by using a natural language processing (NLP) technique (Fig. 2). We describe the options for the vectorization techniques in Sect. II-C. Hereafter, we denote the vector corresponding to the test case t_i by $\boldsymbol{v}_i$ (for $i = 1, \ldots, n$).
2) *Computation of Distances between Test Cases*:
   For each pair of test cases $(t_i,\ t_j)$, we compute the distance between them in the vector space, $d(\boldsymbol{v}_i, \boldsymbol{v}_j)$. Because there are some options for the distance metric, we elaborate them in Sect. II-D.
3) *Prioritization of Test Cases*:
   a) At first, we find the test case t_x that is the farthest from the others $\{t_i | i \neq x\}$, and then we assign the highest priority to t_x. In this paper, we denote the set of vectors corresponding to the already-prioritized test cases by A. That is, right after the above selection, we have $A = \{\boldsymbol{v}_x\}$.
   To determine the farthest test case, we need to define the distance between a single vector $\boldsymbol{v}_i$ and a set of vectors A; We denote the distance between them by $D(\boldsymbol{v}_i, A)$. Because there are some options for D, we elaborate them in Sect. II-E.

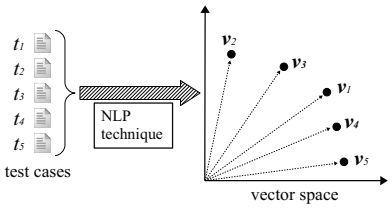


Fig. 2. Vectorization of test cases.

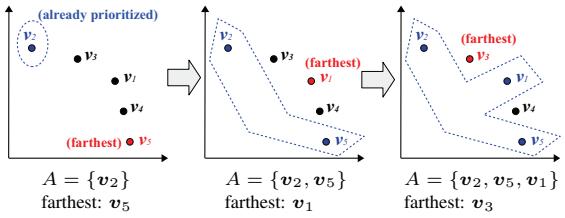


Fig. 3. Example of the TCP process.

   b) After that, we single out the test case t_y, which is the farthest from A, s.t. $\forall i\ D(\boldsymbol{v}_y, A) \geq D(\boldsymbol{v}_i, A)$. Then, we append $\boldsymbol{v}_y$ to A, i.e., $A \leftarrow A \cup \{\boldsymbol{v}_y\}$.
   c) By iterating the above step b), i.e., a selection of the farthest test case and an update of A, we obtain the prioritized set of test cases. □

Fig. 3 illustrates an example of the prioritization process. Suppose $\boldsymbol{v}_2$ is the farthest from the others, and we start the TCP with $A = \{\boldsymbol{v}_2\}$. Next, we find that $\boldsymbol{v}_5$ is the farthest from A, and then we update A as $A = \{\boldsymbol{v}_2, \boldsymbol{v}_5\}$. After that, we find $\boldsymbol{v}_2$ is the farthest from A, and we obtain $A = \{\boldsymbol{v}_2, \boldsymbol{v}_5, \boldsymbol{v}_1\}$. By iterating the similar steps, we prioritize the test cases.

C. Vectorization Method

We describe two primary NLP techniques to obtain a vector representation of a document. In the context of this study, a document corresponds to a test case.

1) Topic Model: The topic model [18] is a model for analyzing the latent semantic of a document. The model considers some *topics* and estimates the probability distributions which link the terms (words) with the topics. In the topic model, one word may belong to two or more topics. Hence, one document may have two or more topics, i.e., a mix-up of topics. Fig. 4 presents a simple image structure of a topic model in which we have four documents, eight words, and three topics. For example, *document*-1 contains three words w_1, w_2, and w_3. Because these words belong to *topic*-1 or *topic*-2, we regard that *document*-1 is a mix-up of *topic*-1 and *topic*-2.

Since we cannot observe the topics directly, we estimate the relationship between each word and each latent topic by using a probability distribution. One of the most popular methods is the latent Dirichlet allocation (LDA) [23]–[25], which uses the Dirichlet distribution; the LDA performs the maximum likelihood estimation to associate a word with a topic. By using

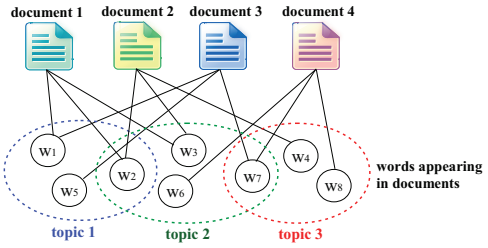


Fig. 4. Simple image of the topic model.

the probabilistic model of topics, we can quantify the weight of each topic in each document. In other words, "topics" correspond to "soft clusters," and the topic modeling is a soft clustering of documents. The weight of a topic in a document expresses the degree that the document belongs to the topic.

When we consider $K(> 0)$ topics, we can obtain K-dimensional vector for a document in which the i-th element indicates the weight of the i-th topic in the document (for $i = 1, \ldots, K$). We use such a vector as a vector representation of a document, i.e., a test case in our context. In the previous work [16], the topic model (built by the LDA) was used to vectorize test cases.

2) Doc2Vec: The Doc2Vec [19] is one of the most promising NLP techniques to obtain a vector representation of a document [20]. For a pair of similar documents, a well-trained Doc2Vec produces a pair of vectors that are close to each other in the vector space. The Doc2Vec is an extension of the Word2Vec [26] technique that vectorizes a word. Both of these techniques utilize a neural network that learns words.

Fig. 5 presents a simple image structure of a neural network used in a Word2Vec model. Suppose we have a training sentence "I play tennis every weekend," and we focus on the word "tennis." As a training of the neural network, we input one word which appears around "tennis" (e.g., "play") and expect the neural network to output "tennis." Through the training using various sentences, we update the weights between layers iteratively. After the training, we look at the weight between the i-th unit of the input layer and the j-th unit of the hidden layer, w_{ij}. These weights form a map of the word onto a multidimensional space whose number of dimensions is the unit count in the hidden layer. We use the vector of weights for the i-th unit, $\boldsymbol{w}_i = (w_{i1}, w_{i2}, \cdots, w_{ij}, \cdots)^T$, as a vector expression of the i-th word[2]. This vector is referred to as a "distributed representation" of the word. Although it is hard to interpret the meaning of each element of the vector, the combination of the elements can form a well-produced map of the semantic relationship among words. If two words semantically similar to each other, the trained neural network would make a similar output for either of these two words. That is, the weight vectors for these two words are also close to each other. Notice that the above example of the learning is called *Continuous Bag-of-Words (CBoW)* model; There is another learning model called *Skip-gram* in which the focused word ("tennis") is used as the input, and its neighborhood word (e.g., "play") is expected as the output of the neural network.

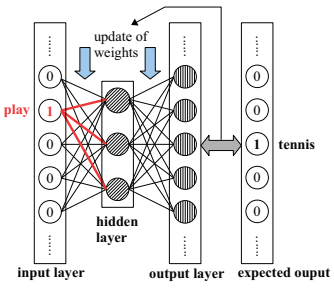


Fig. 5. Simple image of the neural network learning a word.

[2]Symbol T signifies the transpose of a vector.

The Doc2Vec uses not only the words in a document but also the document ID to extend the Word2Vec. There are two different learning models for Doc2Vec.

- *Paragraph Vector Distributed Memory* (*PV-DM*): The PV-DM is similar to the CBoW model of the Word2Vec. The inputs of the neural network are the document ID and the words appearing in the document.
- *Paragraph Vector Distributed Bag-of-Words* (*PV-DBoW*): PV-DBoW is similar to the Skip-gram model of the Word2Vec. It uses the document ID as the input. Notice that PV-DBoW does not take into account the order of words in the sentence.

In this paper, we denote the Doc2Vec models trained by PV-DM and PV-DBoW by *Doc2Vec-DM* and *Doc2Vec-DBoW*, respectively. These models can become our options for the vectorization method of test cases as well as the topic model.

D. Distance Metric

Next, we describe metrics to quantify the distance between two vectors, i.e., two test cases. In this subsection, let K be the dimension number of vector, and $\boldsymbol{v}_i = (v_{i1}, v_{i2}, \ldots, v_{iK})^T$ be the corresponding vector of test case t_i (for $i = 1, 2, \ldots, n$).

1) Manhattan Distance: In the previous work [16], Thomas et al. used the Manhattan distance:

$$d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sum_{k=1}^{K} |v_{ik} - v_{jk}| \ . \tag{3}$$

The reason why they used the Manhattan distance is due to the comparison conducted in their empirical study. They compared the topic model-based TCP method with the conventional string similarity-based method [13]. Because the string-based method had used the Manhattan distance and it is also applicable to the topic model-based TCP method, Thomas et al. adopted it as the distance metric.

2) Euclidean Distance: As we explained in Sect. II-B, we express test cases by multidimensional vectors and prioritize them while considering the distances among them. The prioritization procedure has a similar characteristic[3] to a clustering of data [22]. Because the Euclidean distance is widely used in clustering methods, it is natural that we consider the Euclidean distance as an option for the distance metric in this study:

$$d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \sqrt{(\boldsymbol{v}_i - \boldsymbol{v}_j)^T (\boldsymbol{v}_i - \boldsymbol{v}_j)} \ . \tag{4}$$

3) Angular Distance: In the NLP studies, the cosine similarity is a well-known measure of the closeness between two feature vectors [27]:

$$\mathrm{CosSim}(\boldsymbol{v}_i, \boldsymbol{v}_j) = \frac{\boldsymbol{v}_i^T \boldsymbol{v}_j}{\sqrt{\boldsymbol{v}_i^T \boldsymbol{v}_i} \sqrt{\boldsymbol{v}_j^T \boldsymbol{v}_j}} \ . \tag{5}$$

[3]Although a clustering method usually seeks to find the *nearest* data point, we try to find the *farthest* one.

Since our vectorization methods for test cases are NLP techniques, we may consider an *inverted* cosine similarity to be an option for the distance metric. Although the inverted measure "$1 - \mathrm{CosSim}(\boldsymbol{v}_i, \boldsymbol{v}_j)$" evaluates a dissimilarity between two vectors, we cannot use it as a distance metric because it does not satisfy the triangle inequality[4], one of the necessary conditions of a distance metric. Thus, we use the following angular distance instead:

$$d(\boldsymbol{v}_i, \boldsymbol{v}_j) = \frac{1}{\pi} \arccos(\mathrm{CosSim}(\boldsymbol{v}_i, \boldsymbol{v}_j)) \ . \tag{6}$$

This metric focuses on the *angle* between the vectors rather than the cosine value because $\arccos(\cos\theta) = \theta$.

E. Scheme to Find the Farthest Test Case

When one or more test cases are already prioritized, a static TCP method seeks to find the test case that is the farthest from the set of the already-prioritized ones (see step 3 in Sect. II-B and Fig. 3). To this end, we need to define the distance between a single vector $\boldsymbol{v}_i$ and a set of vectors A, i.e., $D(\boldsymbol{v}_i, A)$; Using $D(\boldsymbol{v}_i, A)$, we can find the farthest vector $\boldsymbol{v}_y$ such that $\forall i \ D(\boldsymbol{v}_y, A) \geq D(\boldsymbol{v}_i, A)$. Because the process of finding the farthest vector is essentially identical[5] to a clustering of data, we leverage the following representative methods in this study.

1) Single Linkage Method: For a vector $\boldsymbol{v}_i$ ($\notin A$), the single linkage method focuses on the vector $\boldsymbol{v}_c$ ($\in A$) that is the *closest* to $\boldsymbol{v}_i$ and considers $d(\boldsymbol{v}_i, \boldsymbol{v}_c)$ to be the distance between $\boldsymbol{v}_i$ and A. That is, we have

$$D(\boldsymbol{v}_i, A) = \min_{\boldsymbol{v}_c \in A} [\ d(\boldsymbol{v}_i, \boldsymbol{v}_c)\ ] \ , \tag{7}$$

and find the farthest test case using Eq.(7). For example, when we have $\boldsymbol{v}_i$ and $A = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\}$ shown in Fig. 6 (1), $D(\boldsymbol{v}_i, A)$ is the distance between $\boldsymbol{v}_i$ and its closest one within A, i.e., $d(\boldsymbol{v}_i, \boldsymbol{v}_2)$.

In this paper, we call the above scheme as the *single linkage-based scheme*. Thomas et al. used this scheme in their empirical work [16].

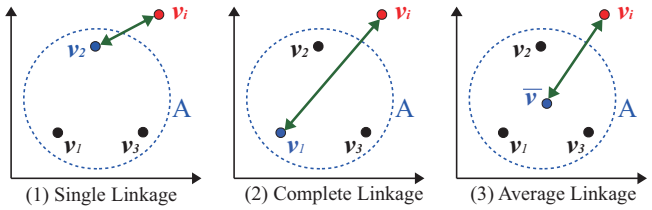


Fig. 6. Simple examples of $D(\boldsymbol{v}_i, A)$.

[4]Suppose we have three vectors $\boldsymbol{v}_1, \boldsymbol{v}_2$, and $\boldsymbol{v}_3$, and $\mathrm{CosSim}(\boldsymbol{v}_1, \boldsymbol{v}_2) = \cos(\pi/3) = 1/2$ and $\mathrm{CosSim}(\boldsymbol{v}_1, \boldsymbol{v}_3) = \mathrm{CosSim}(\boldsymbol{v}_3, \boldsymbol{v}_2) = \cos(\pi/6) = \sqrt{3}/2$. If we use $d'(\boldsymbol{v}_i, \boldsymbol{v}_j) = 1 - \mathrm{CosSim}(\boldsymbol{v}_i, \boldsymbol{v}_j)$, we obtain $d'(\boldsymbol{v}_1, \boldsymbol{v}_2) = 1 - 1/2 = 1/2$ and $d'(\boldsymbol{v}_1, \boldsymbol{v}_3) = d'(\boldsymbol{v}_3, \boldsymbol{v}_2) = 1 - \sqrt{3}/2 = (2 - \sqrt{3})/2$. Here, $d'(\boldsymbol{v}_1, \boldsymbol{v}_3) + d'(\boldsymbol{v}_3, \boldsymbol{v}_2) = 2 - \sqrt{3} > 1/2$. That is, the triangle inequality $d'(\boldsymbol{v}_1, \boldsymbol{v}_2) \leq d'(\boldsymbol{v}_1, \boldsymbol{v}_3) + d'(\boldsymbol{v}_3, \boldsymbol{v}_2)$ is not satisfied.

[5]In the case of clustering, we seek to find the nearest vector rather than the farthest one.

2) Complete Linkage Method: In contrast to the single linkage method, the complete-link method looks at the *farthest* vector $\boldsymbol{v}_f$ ($\in A$) and defines the distance as:

$$D(\boldsymbol{v}_i, A) = \max_{\boldsymbol{v}_f \in A} [\ d(\boldsymbol{v}_i, \boldsymbol{v}_f)\ ] \ . \tag{8}$$

For example, when we have $\boldsymbol{v}_i$ and A shown in Fig. 6 (2), $D(\boldsymbol{v}_i, A)$ is the distance between $\boldsymbol{v}_i$ and its farthest one within A, i.e., $d(\boldsymbol{v}_i, \boldsymbol{v}_1)$. Hereafter, we refer to the prioritization scheme using Eq.(8) as the *complete linkage-based scheme*.

3) Avarage Linkage Method: The average linkage method uses the following definition of D, which is the average of distances of all pairs:

$$D(\boldsymbol{v}_i, A) = \frac{1}{|A|} \sum_{\boldsymbol{v}_a \in A} d(\boldsymbol{v}_i, \boldsymbol{v}_a) \ . \tag{9}$$

For example, when we have $\boldsymbol{v}_i$ and A shown in Fig. 6 (3), $D(\boldsymbol{v}_i, A)$ is the average of $d(\boldsymbol{v}_i, \boldsymbol{v}_1)$, $d(\boldsymbol{v}_i, \boldsymbol{v}_2)$, and $d(\boldsymbol{v}_i, \boldsymbol{v}_3)$. It is also the distance between $\boldsymbol{v}_i$ and the mean vector of $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3\}$ ($\overline{\boldsymbol{v}}$ in Fig. 6 (3)). We call the prioritization scheme using Eq.(9) as the *average linkage-based scheme*.

4) Ward's Method: Ward's method has a different perspective from the above three methods, and it focuses on the variance within the cluster. Let $S(A)$ be the sum of squared deviation within the set A:

$$S(A) = \sum_{\boldsymbol{v}_s \in A} d(\boldsymbol{v}_s, \overline{\boldsymbol{v}})^2 \ , \tag{10}$$

where

$$\overline{\boldsymbol{v}} = \frac{1}{|A|} \sum_{\boldsymbol{v}_s \in A} \boldsymbol{v}_s \ .$$

If we append a vector $\boldsymbol{v}_i$ to A, the above sum of squared deviation increases. Ward's method considers the increment to be the distance between $\boldsymbol{v}_i$ and A:

$$D(\boldsymbol{v}_i, A) = S(A \cup \{\boldsymbol{v}_i\}) - S(A) \ . \tag{11}$$

We refer to the prioritization scheme using Eq.(11) as *Ward's method-based scheme*.

F. Available TCP methods: Combinations of Aspects

In this section, we have described three aspects of the vectorization-based static TCP methods. Here, we have 36 combinations of these aspects because

1) the vectorization method has three options: *Topic model*, *Doc2Vec-DM*, and *Doc2Vec-DBoW*;
2) the distance metric has three options: *Manhattan distance*, *Euclidean one*, and *the angular one*;
3) the prioritization scheme has four options: *the single linkage-based scheme*, *the complete linkage-based one*, *the average linkage-based one*, and *Ward's method-based one*.

Table II shows these combinations. The previous work [16] used the combination "Topic model / Manhattan distance / single linkage-based scheme" (T-m-S). We compare it to the remaining 35 combinations in our comparative study.

TABLE II
COMBINATIONS OF THREE ASPECTS FORMING TCP METHODS

Distance Metric	Scheme to Find the Farthest	Vectorization Method: [T] Topic Model	[Dm] Doc2Vec-DM	[Db] Doc2Vec-DBoW
[m] Manhattan	[S] single	T-m-S*	Dm-m-S	Db-m-S
	[C] complete	T-m-C	Dm-m-C	Db-m-C
	[A] average	T-m-A	Dm-m-A	Db-m-A
	[W] Ward	T-m-W	Dm-m-W	Db-m-W
[e] Euclidean	[S] single	T-e-S	Dm-e-S	Db-e-S
	[C] complete	T-e-C	Dm-e-C	Db-e-C
	[A] average	T-e-A	Dm-e-A	Db-e-A
	[W] Ward	T-e-W	Dm-e-W	Db-e-W
[a] angular	[S] single	T-a-S	Dm-a-S	Db-a-S
	[C] complete	T-a-C	Dm-a-C	Db-a-C
	[A] average	T-a-A	Dm-a-A	Db-a-A
	[W] Ward	T-a-W	Dm-a-W	Db-a-W

(* Combination "T-m-S" is the conventional method used in the previous work [16].)

III. COMPARATIVE STUDY

We conducted a comparative study of 36 vectorization-based static TCP methods shown in Table II. We report on our comparative study and discuss the results in this section.

A. Aim and Dataset

Toward a better static TCP, this study aims to reveal the effects of the vectorization method, the distance metric, the scheme to find the farthest test case in the vector space, and their combinations through the comparisons in terms of testing efficiency (APFD value). To this end, we tackle the following two research questions (RQs).

RQ1: Which TCP method would work well?

RQ2: Which options or combinations of them would be useful in the static TCP method?

RQ1 is a straightforward question about our comparison. If some TCP methods work better than the conventional method ("T-m-S" in Table II), this comparative study is worth doing for the promotion of the static TCP method. RQ2 seeks to gain a more in-depth insight into the components (options) of the vectorization-based static TCP methods. By analyzing the impacts of the options and their combinations, we can advise a promising TCP method for future testing.

Our dataset consists of five sets of test cases shown in Table III, which is the same as the dataset used in [16]. They are available at the Software-artifact Infrastructure Repository (SIR) [28] and contain the source files of SUT, the test cases (test programs), and the fault matrices that present the links between a test case and the detectable faults.

All of the test programs included in the dataset are Java programs. The test programs for Ant are JUnit[6] test cases that are descendants of junit.framework.TestCase. On the other hand, although the test cases for Derby are also Java programs, they adopt another style of testing other than the JUnit framework. The test cases provide Java code to run some tasks on Derby, and the test harness org.apache.derbyTesting.functionTests.harness.RunTest executes each test case and checks its result.

[6]https://junit.org/

TABLE III
SYSTEMS UNDER TEST, NUMBER OF TEST CASES AND NUMBER OF FAULTS TO BE DETECTED

Software	Version	# of Test Cases	# of Faults
Ant	1.5.3	105	6
Derby	10.1.2.1	98	7
Derby	10.1.3.1	106	9
Derby	10.2.1.6	120	16
Derby	10.3.1.4	53	26

B. Procedure

We conducted our comparative study in the following steps[7].

1) *Preprocessing of test cases*: As with the previous work [16], we perform the following tasks for each test case.
   a) Remove non-alphabetical characters and the keywords of the programming language (Java).
   b) Split the identifiers by the camel case rule and the snake case rule.
   c) Stem all words and remove English stop words[8].
2) *Vectorization of test cases*: We build the following three types of vectorization models using the preprocessed test cases: Topic model, Doc2Vec-DM, and Doc2Vec-DBoW. We use LdaModel class and Doc2Vec class provided in gensim library of Python [25] for the model construction. To mitigate a threat to the construct validity caused by the random seed setting during the model construction, we iterate the model constructions 30 times[9] while changing the random seed, i.e., we build different 30 instances for each model.
   Notice that we need to specify the dimension number of the vector when we build the above models. According to [16], we use $n/2.5$ as the dimension number where n is the number of test cases; when $n/2.5$ is not an integer, we round it to the nearest integer.
3) *Evaluation of TCP methods*: For each of the 36 TCP methods (Table II), we compute the APFD as the evaluation value. Because we have 30 instances for each vectorization model, we obtain 30 APFD values for each method. To answer RQ1 and RQ2, we analyze the results by checking the mean APFD and by performing the analysis of variance (ANOVA).

C. Results

Table IV presents the mean APFD of each static TCP method for each SUT. In the table, we emphasize the highest value within each SUT by boldface and asterisk(*).

As a result, the conventional TCP method used in [16], i.e., the combination "T-m-S," did not show the best performance for any SUT; the best TCP method having the highest mean APFD varies among different SUT. Nonetheless, for all SUT, the best TCP methods use Doc2Vec-DBoW or Doc2Vec-DM as their vectorization methods. From the perspectives of the

[7]We developed tools to perform the preprocessing and the vectorization. Our tools are available at http://se.cite.ehime-u.ac.jp/tool/SEAA2020/.

[8]We used the Natural Language Toolkit (https://www.nltk.org/).

[9]The number of iterations (30) is decided according to [16].

TABLE IV
EVALUATION VALUES: MEAN APFD VALUE FOR EACH COMBINATION

		(a) Ant 1.5.3			(b) Derby 10.1.2.1			(c) Derby 10.1.3.1			(d) Derby 10.2.1.6			(e) Derby 10.3.1.4		
		[T]	[Dm]	[Db]	[T]	[Dm]	[Db]	[T]	[Dm]	[Db]	[T]	[Dm]	[Db]	[T]	[Dm]	[Db]
[m]	[S]	0.875	0.876	0.878	0.977	0.974	0.977	0.885	0.871	0.851	0.967	0.968	**0.996***	0.925	0.917	0.908
	[C]	0.872	0.877	**0.884***	0.974	0.974	0.978	0.866	0.871	0.865	0.961	0.956	0.964	0.926	0.927	0.909
	[A]	0.872	0.871	0.877	0.976	0.974	0.977	0.880	0.870	0.851	0.967	0.960	0.994	0.928	0.923	0.909
	[W]	0.872	0.870	0.877	0.976	0.974	0.976	0.878	0.869	0.851	0.965	0.962	0.985	0.918	0.921	0.908
[e]	[S]	0.873	0.876	0.878	0.976	0.974	0.977	0.870	0.871	0.851	0.966	0.966	**0.996***	0.923	0.914	0.908
	[C]	0.872	0.876	**0.884***	0.978	0.974	**0.979***	0.887	0.871	0.870	0.966	0.956	0.972	0.919	0.927	0.909
	[A]	0.871	0.870	0.878	0.977	0.974	0.977	0.885	0.871	0.851	0.960	0.957	**0.996***	0.917	0.923	0.909
	[W]	0.872	0.868	0.878	0.977	0.974	0.977	0.888	0.871	0.851	0.961	0.957	**0.996***	0.917	0.922	0.909
[a]	[S]	0.874	0.878	0.876	0.978	0.974	0.975	0.880	0.870	0.857	0.968	**0.996***	0.962	0.923	0.925	0.906
	[C]	0.873	0.878	0.883	0.975	0.974	0.975	0.876	0.891	0.978	0.964	0.962	0.962	0.918	**0.967***	0.909
	[A]	0.873	0.876	0.871	0.976	0.974	0.974	0.894	0.870	0.928	0.970	0.956	0.965	0.926	0.925	0.910
	[W]	0.873	0.874	0.867	0.976	0.974	0.974	0.894	0.863	**0.979***	0.970	0.956	0.990	0.928	0.925	0.910

TABLE V
NORMALIZED EVALUATION VALUES

Distance Metric	Scheme to Find the Farthest	Vectorization Method: [T] Topic Model	[Dm] Doc2Vec-DM	[Db] Doc2Vec-DBoW
[m] Manhattan	[S] single	$+0.148$	-0.110	**$+0.237$**
	[C] complete	-0.251	-0.120	**$+0.232$**
	[A] average	-0.045	-0.311	**$+0.185$**
	[W] Ward	-0.134	-0.340	-0.010
[e] Euclidean	[S] single	-0.071	-0.145	**$+0.245$**
	[C] complete	$+0.061$	-0.133	**$+0.382$**
	[A] average	-0.147	-0.373	**$+0.252$**
	[W] Ward	-0.065	-0.433	**$+0.228$**
[a] angular	[S] single	$+0.123$	**$+0.315$**	-0.327
	[C] complete	-0.206	**$+0.483$**	**$+0.425$**
	[A] average	$+0.100$	-0.175	-0.143
	[W] Ward	$+0.123$	-0.238	**$+0.236$**

distance metric and the scheme to find the farthest test case, there does not seem to be a common trend.

To compare an average performance of the TCP methods across SUT, we normalize the raw APFD values to z scores[10] within each SUT and compute the average of the z scores for each TCP method. Table V shows the results. In the table, we emphasize the z scores that are higher than the conventional method ($= 0.148$) by boldface. As a result, most of the methods using Doc2Vec-DBoW show better performances than the conventional one (see the column of [Db]).

Next, to see the impacts caused by each of the three factors (the vectorization method, the distance metric, and the scheme to find the farthest test case) and the interactions among them, we performed a three-way ANOVA. We present the results in Table VI. In the table, the columns *Df*, *SS*, *MS*, *F*, *p*, and η^2 signify the degree of freedom, the sum of squares, the mean square, the *F* statistic, the *p*-value, and the effect size, respectively; we emphasize the effect size (η^2) corresponding to a *large* effect by boldface and an asterisk(*), and the one corresponding to a *medium* effect by a dagger(†), respectively[11].

As a result, the vectorization methods had large or medium impacts on the testing efficiency for four out of five SUT; Nonetheless, for the remaining SUT (c), the interaction between the vectorization method and the distance metric had the highest effect that is a medium-level impact. On the other hand, the distance metric showed a medium-level effect only for the SUT (c), and the prioritization scheme did only for the SUT (a) and (d). As a consequence, although most of the factors and their interactions showed statistically-significant ($\alpha = 0.001$) impacts on the testing efficiency, only a limited number of them showed large or medium impacts.

D. Discussion

1) [RQ1] Which TCP method would work well?: In this study, we compared 36 static TCP methods that are the combinations of the vectorization methods, the distance metric, and the scheme to find the farthest test case. As a result, although the best TCP method differs among SUT (Table IV), the conventional topic model-based method was not the best TCP method for any SUT, and Doc2Vec-based methods showed better performances (Table V).

In Table V, although "Dm-a-C" has the highest z score, there are also many negative scores in the row of [Dm]. On the other hand, most of the z scores in the row of [Db] are positive and better than the conventional one, including the second-highest z score. Hence, it would be better to choose "Db", i.e., to use Doc2Vec-DBoW as the vectorization method rather than the topic model.

Although the method using "Db" did not show the top performance in Table V, the main reason would be that those methods showed relatively-low APFD values for only the SUT (e) (see Table IV). Because the SUT (e) has the fewest test cases and the most faults among all SUT (Table III), it might have a different trend from the other SUT.

Now, we answer to the RQ1 as follows. The TCP methods using Doc2Vec-DBoW tend to work well. Although the performance of the conventional topic model-based method is not low, TCP methods using Doc2Vec-DBoW can outperform the conventional one in many cases.

2) [RQ2] Which options or combinations of them would be useful in the static TCP method?: We conducted a three-way ANOVA for the APFD values to examine which options or combinations of them have significant impacts on the testing efficiency. Although the results varied among SUT, as shown

[10]For a raw score x, the corresponding z score is $(x - \overline{x})/s$ where $\overline{x}$ and s are the mean of all scores and the standard deviation of them, respectively.

[11]Cohen [29] presented the rules on magnitudes of effect size: the thresholds of the large effect and the medium effect are 0.14 and 0.06, respectively.

TABLE VI
RESULTS OF THE THREE-WAY ANOVA

Source	*Df*	*SS*	*MS*	*F*	*p*	η^2
(a) Ant 1.5.3						
Vector (V)	2	0.00434	0.00217	96.984	< 0.001	$0.108^{\dagger}$
Metric (M)	2	0.00002	0.00001	0.427	0.652	0.000
Scheme (S)	3	0.00498	0.00166	74.087	< 0.001	$0.123^{\dagger}$
V × M	4	0.00316	0.00079	35.338	< 0.001	$0.078^{\dagger}$
V × S	6	0.00312	0.00052	23.222	< 0.001	$0.077^{\dagger}$
M × S	6	0.00009	0.00001	0.634	0.703	0.002
V × M × S	12	0.00130	0.00011	4.83	< 0.001	0.032
Error	1044	0.02337	0.00002			
(b) Derby 10.1.2.1						
Vector (V)	2	0.00082	0.00041	45.101	< 0.001	$0.071^{\dagger}$
Metric (M)	2	0.00033	0.00017	18.286	< 0.001	0.029
Scheme (S)	3	0.00003	0.00001	1.003	0.391	0.002
V × M	4	0.00041	0.00010	11.249	< 0.001	0.036
V × S	6	0.00012	0.00002	2.262	0.036	0.011
M × S	6	0.00013	0.00002	2.339	0.030	0.011
V × M × S	12	0.00017	0.00001	1.51	0.114	0.014
Error	1044	0.00951	0.00001			
(c) Derby 10.1.3.1						
Vector (V)	2	0.02680	0.01339	9.131	< 0.001	0.011
Metric (M)	2	0.21720	0.10858	74.054	< 0.001	$0.089^{\dagger}$
Scheme (S)	3	0.05480	0.01827	12.457	< 0.001	0.023
V × M	4	0.31110	0.07778	53.047	< 0.001	$0.128^{\dagger}$
V × S	6	0.09490	0.01582	10.791	< 0.001	0.039
M × S	6	0.07120	0.01186	8.092	< 0.001	0.029
V × M × S	12	0.12340	0.01029	7.016	< 0.001	0.051
Error	1044	1.53070	0.00147			
(d) Derby 10.2.1.6						
Vector (V)	2	0.07569	0.03784	201.571	< 0.001	$\mathbf{0.193}^{*}$
Metric (M)	2	0.00112	0.00056	2.977	0.051	0.003
Scheme (S)	3	0.02577	0.00859	45.746	< 0.001	$0.066^{\dagger}$
V × M	4	0.02968	0.00742	39.525	< 0.001	$0.076^{\dagger}$
V × S	6	0.02949	0.00491	26.179	< 0.001	$0.075^{\dagger}$
M × S	6	0.00491	0.00082	4.36	< 0.001	0.013
V × M × S	12	0.02902	0.00242	12.882	< 0.001	$0.074^{\dagger}$
Error	1044	0.19600	0.00019			
(e) Derby 10.3.1.4						
Vector (V)	2	0.06251	0.03125	104.028	< 0.001	$\mathbf{0.142}^{*}$
Metric (M)	2	0.00745	0.00372	12.395	< 0.001	0.017
Scheme (S)	3	0.00674	0.00225	7.475	< 0.001	0.015
V × M	4	0.01009	0.00252	8.396	< 0.001	0.023
V × S	6	0.01787	0.00298	9.915	< 0.001	0.040
M × S	6	0.00434	0.00072	2.409	0.026	0.010
V × M × S	12	0.01886	0.00157	5.232	< 0.001	0.043
Error	1044	0.31367	0.00030			

in Table VI, it seems to be a common trend that the vectorization method is the most influential on the performance of static TCP method. By combing with the results regarding RQ1, we can say that Doc2Vec-DBoW would be a useful option.

On the other hand, the choice of the remaining two aspects cannot always be impactful for enhancing the static TCP method. However, the interaction between the vectorization method and the distance metric may have a medium-level impact (Table VI). Because all options using both Doc2Vec-DBoW and Euclidean distance showed better performance values in Table V, the combination of them may also be useful in the static TCP method.

From the above results, we answer to the RQ2 as follows. The choice of vectorization method plays the most important role in the static TCP method, and the better option is Doc2Vec-DBoW rather than the topic model. Moreover, the interaction with the distance metric may also become influential, and the promising combination is Doc2Vec-DBoW and Euclidean distance.

3) Impact of Vectorization Method: The above analysis showed that the vectorization method has the most significant impact on testing efficiency. In this study, we have three options for vectorization methods, Topic model, Doc2Vec-DM, and Doc2Vec-DBoW. Although the dimension number of vectors representing a test case is common to all methods, the compositions of vectors differ among the vectorization methods, and such differences would cause the above variations in the observed results.

The topic model focuses on a co-occurrence relation among words (tokens) in test cases and performs a soft clustering of words. Each element of the test case vector presents the likelihood that the test case belongs to the corresponding soft cluster. On the other hand, the Doc2Vec models are based on the notion of word embedding, and the produced vectors map semantic features of words on the vector space. That is, the vector elements are not independent of each other, and their combinations represent some semantics of words and documents. Thus, a Doc2Vec model and a topic model would produce different vectors for the same test case. Moreover, Doc2Vec-DM and Doc2Vec-DBoW become different models due to the difference in learning algorithm even if they learned the same dataset: Doc2Vec-DBoW does not take into account the order of words in test cases. These differences in the vectorization methods would cause the variation of observed data (testing efficiency values).

As a result, Doc2Vec-DBoW seems to be useful for prioritizing test cases. Although the Doc2Vec-DBoW model is insensitive to the order of words, it would be better to evaluate the similarity among test cases because some test cases run some of the same functions in different orders; The Doc2Vec-DM may be sensitive to such a difference of word orders in test cases, and consequently, it might be less appropriate to evaluate the test case similarity.

E. Threats to Validity

Construct validity: The setting of parameters has an impact on the training of vectorization models. If we train the topic models and Doc2Vec models using different parameters, we may obtain different empirical results. However, to conduct a fair comparison of TCP methods, we used the default parameters of gensim library, except for the dimension number of the vector; As the default dimension number differs between LdaModel and Doc2Vec, we set the same number to them.

Internal validity: Both the topic model and the Doc2Vec model use the words (terms) appearing in the test cases. When there are compound terms that disobey the camel case rule or the snake case rule (e.g., "filename"), the vectorization methods may not treat the semantic features of such terms properly because we fail to split them appropriately. Although we adopted the same preprocessing method as the previous work [16], it would be better to leverage the advanced identifier-splitting studies [30], and this is our future work.

Our TCP scheme described in Sect. II-B (Step 3) can be another threat to the internal validity. Although we select

test cases to maximize the diversity of selected (already-prioritized) test cases, the selection algorithm is the greedy algorithm. That is, it might not provide the optimal solution in terms of the diversity of test cases. We may mitigate this threat by adopting another sophisticated algorithm, such as the genetic algorithm [11]. It is also our significant future work.

External validity: Because our dataset is limited to five sets obtained from the SIR [28] and all of them are Java programs, our findings may not work well for the testing of another system. That is, we would not be able to generalize our empirical results to any software systems. To mitigate this threat to validity, we would need to perform a further comparative study using various systems, including other open source products and commercial ones, as our future work. Furthermore, we also have to use other systems written in programming languages other than Java to examine the impact of the programming language on the results.

IV. Conclusion and Future Work

In this paper, we conducted a comparative study of 36 vectorization-based static TCP methods focusing on three aspects (1) the vectorization method, (2) the distance metric, and (3) the scheme to find the test case that is the farthest from the set of already-prioritized test cases. As a result, we empirically proved that the TCP methods using Doc2Vec (PV-DBoW) tend to work well in terms of the testing efficiency. Moreover, the combination of Doc2Vec (PV-DBoW) and Euclidean distance may be useful in the static TCP. On the other hand, the choice of the scheme (the above third aspect) may not be influential. The empirical results would become a useful baseline toward further development of vectorization-based static TCP methods.

From the results, the vectorization method seems to be the most promising factor in enhancing the static TCP. Because a vectorization method such as Doc2Vec has many adjustable parameters, a further study of the parameter turning approach for test cases is our significant future work. Other of our future work includes: (1) a real-world validation using test suites for commercial software products, (2) applying the vectorization methods to the dynamic TCP, and (3) enhancing the TCP scheme by adopting a sophisticated algorithm such as the genetic algorithm.

Acknowledgment

This work was supported by JSPS KAKENHI #18K11246 and #20H04184.

References

[1] A. P. Mathur, *Foundations of Software Testing*. Delhi: Pearson Education, 2008.

[2] J. Hartmann and D. J. Robson, "Techniques for selective revalidation," *IEEE Softw.*, vol. 7, no. 1, pp. 31–36, Jan 1990.

[3] I. Sommerville, *Software Engineering*, 10th ed. Essex, UK: Pearson Education, 2016.

[4] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization:a survey," *Softw. Test. Verif. Reliab.*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[5] W. Wong, J. R. Horgan, S. London, and A. P. Mathur, "Effect of test set minimization on fault detection effectiveness," in *Proc. 17th Int. Con. Softw. Eng.*, Apr. 1995, pp. 41–50.

[6] G. Rothermel, R. H. Untch, Chengyun Chu, and M. J. Harrold, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001.

[7] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Test case prioritization: a family of empirical studies," *IEEE Trans. Softw. Eng.*, vol. 28, no. 2, pp. 159–182, Feb. 2002.

[8] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," in *Proc. 30th Int. Comp. Softw. App. Conf.*, Sept. 2006, pp. 411–420.

[9] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *Proc. 27th Int. Conf. Softw. Maintenance*, Sep. 2011, pp. 382–391.

[10] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proc. 24th Int. Conf. Softw. Eng.*, May 2002, pp. 119–129.

[11] A. Bajaj and O. P. Sangwan, "A systematic literature review of test case prioritization using genetic algorithms," *IEEE Access*, vol. 7, pp. 126 355–126 375, 2019.

[12] L. Zhang, J. Zhou, D. Hao, L. Zhang, and H. Mei, "Prioritizing junit test cases in absence of coverage information," in *Proc. Int. Conf. Softw. Maintenance*, Sep. 2009, pp. 19–28.

[13] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Auto. Softw. Eng.*, vol. 19, no. 1, pp. 65–95, Mar. 2012.

[14] H. Hemmati, L. C. Briand, A. Arcuri, and S. Ali, "An enhanced test case selection approach for model-based testing: an industrial case study," in *Proc. 18th Int. Symp. Foundations Softw. Eng.*, Nov. 2010, pp. 267–276.

[15] H. Hemmati, A. Arcuri, and L. Briand, "Empirical investigation of the effects of test suite properties on similarity-based test case selection," in *Proc. 4th Int. Conf. Softw. Test. V. & V.*, Mar. 2011, pp. 327–336.

[16] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empir. Softw. Eng.*, vol. 19, no. 1, pp. 182–212, Feb. 2014.

[17] T. Hofmann, "Probabilistic latent semantic indexing," in *Proc. 22nd Int. Conf. Research & Dev. Inf. Retrieval*, Aug. 1999, pp. 50–57.

[18] D. M. Blei, "Probabilistic topic models," *Commun. ACM*, vol. 55, no. 4, pp. 77–84, Apr. 2012.

[19] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. 31st Int. Conf. Machine Learning*, vol. 32, no. 2, Jun. 2014, pp. 1188–1196.

[20] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," in *Proc. 1st Workshop Representation Learning for NLP*, Aug. 2016, pp. 78–86.

[21] R. Řehůřek, "gensim: models.word2vec—deep learning with word2vec," https://radimrehurek.com/gensim/models/word2vec.html, Mar. 2018.

[22] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: a review," *ACM Computing Surveys*, vol. 31, no. 3, pp. 264–323, Sep. 1999.

[23] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *J. Machine Learning Research*, vol. 3, pp. 993–1022, Jan. 2003.

[24] D. M. Blei and M. I. Jordan, "Modeling annotated data," in *Proc. 26th Int. Conf. Research & Dev. Inf. Retrieval*, Jul. 2003, pp. 127–134.

[25] R. Řehůřek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proc. LREC 2010 Workshop New Challenges for NLP Frameworks*, May 2010, pp. 45–50.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances Neural Inf. Processing Syst.*, vol. 26, Dec. 2013, pp. 3111–3119.

[27] A. Singhal, "Modern information retrieval: A brief overview," *Bulletin IEEE Comp. Society Tech. Committee Data Eng.*, vol. 24, no. 4, pp. 35–43, Dec. 2001.

[28] H. Do, S. Elbaum, and G. Rothermel, "Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact," *Empir. Softw. Eng.*, vol. 10, no. 4, pp. 405–435, Oct. 2005.

[29] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*, 2nd ed. London: Routledge, 1988.

[30] E. Hill, D. Binkley, D. Lawrie, L. Pollock, and K. Vijay-Shanker, "An empirical study of identifier splitting techniques," *Empir. Softw. Eng.*, vol. 19, no. 6, pp. 1754–1780, Dec. 2014.