A Test Case Recommendation Method Based on Morphological Analysis, Clustering and the Mahalanobis-Taguchi Method

Hirohisa Aman
Center for Information Technology
Ehime University
Matsuyama, Ehime 790-8577, Japan
Email: aman@ehime-u.ac.jp

Takashi Nakano
Hideto Ogasawara
Toshiba Corporation
Kawasaki, Kanagawa 212-8582, Japan

Minoru Kawahara
Center for Information Technology
Ehime University
Matsuyama, Ehime 790-8577, Japan

***Abstract*—This paper focuses on the content of test cases, and categorizes test cases into clusters using the similarity between test cases; their degree of similarity is obtained through a morphological analysis. If there are two similar test cases, they would test the same or similar functionalities in similar but different conditions. Thus, when one of them is run for a regression testing, the remaining one should be run as well, in order to reduce a risk of overlooking regressions. Once a test engineer decides to run a set of test cases, the method proposed in this paper can recommend adding similar test cases to their candidate set. The proposed method also considers the priorities of recommended test cases by using the Mahalanobis-Taguchi method. This paper reports on an empirical study with an industrial software product. The results show that the proposed method is useful to prevent overlooking regressions.**

I. Introduction

The successful release of a software product requires well-performed testing. Since a product is usually evolved through various upgrades, regression testing plays a significant role at each release of the product. Regression testing is also a costly activity [1] in the software development and maintenance phase—it is especially common in software products embedded in equipment: we have to install an upgraded version of the software into a piece of equipment, and test its behaviors by hand. Thus, to enhance the efficiency of regression testing is a major challenge for all test engineers.

When a software product is upgraded, it is not sufficient to test only the modified functions because a code modification may cause an unexpected failure. In other words, any upgrade has a risk of creating a regression. Therefore, it would be useful to test not only the modified functions but also other functions which might be affected by the modification. This means that we need to single out possible test cases from our test case pool, and rerun those test cases as well [1], [2]. In this paper, we propose a novel method for recommending related test cases during a regression testing, and examine its usefulness through an empirical study.

The key contributions of this paper are as follows: (1) we leverage a morphological analysis method for a clustering of test cases; (2) we propose a test case recommendation method using the cluster information, in order to reduce the risk of overlooking regressions; (3) the usefulness of the proposed method is examined through an empirical study with an industrial software product.

The remainder of this paper is organized as follows: Section II describes studies related to this paper. Section III explains our proposal and Section IV reports on an empirical study. Finally, Section V presents the conclusion of this paper and our future work.

II. Related Work

When test engineers run test cases as a regression test, there may be the risk of overlooking latent failures. In order to reduce such a risk, it is important to rerun as many test cases as possible. However, there is a limit to our available effort for testing. In this context, the prioritization of test cases is one of the most popular topics in the software testing [1], [2].

Jeffrey et al. [3] and Mirarab et al. [4] focused on the relationship between a test case execution and its corresponding program, and proposed ways of prioritizing test cases through the program slicing analysis or the code coverage analysis. Kim et al. [5] focused on the testing history data rather than the source program, and prioritized test cases by using the notion of the exponentially smoothed moving average. Aman et al. [6], [7] evaluated test cases by using their failure rates in the past and the length of the non-tested version's sequence, and proposed considering a prioritization of test cases as a 0-1 programming problem.

There have also been studies leveraging a clustering algorithm. Sherrif et al. [8] analyzed the change history of source files in order to classify related test cases into a cluster, and proposed prioritizing test cases using that cluster information. Carlson et al. [9] and Leon et al. [10] categorized test cases by using the code coverage data or the execution profiles, and prioritized test cases based on those clusters.

While the above studies seem to be useful proposals or reports, they require a linking of test cases to corresponding programs or an accumulation of testing data; if we have little data on test runs, those methods might not work successfully.

Thus, we believe that it would also be important to analyze test cases from another perspective other than the test case running. Arafeen et al. [11] proposed performing a clustering of test cases by focusing on the requirement specification of the software product. Although it would be a promising way, we have to make a link between a specification and a test case, and such linking is not easy work. There have also been studies focusing on the text of test cases. Ledru et al. [12] proposed a test case prioritization method using string distance—it incrementally builds the set of test cases to be run by selecting the farthest test case from the set of preceding test cases. While it is a remarkable work to prioritize test cases and may also be utilized for clustering test cases, their method is based on character-level distances, so it evaluates relationships among test cases from a lexicographical perspective. That is to say, it is better to take into account semantical aspects of test cases as well. Thomas et al. [13] leveraged the topic modeling technique to quantify similarities (or distances) between test cases. They extracted "topics" from the set of test cases, and represented the features of test cases as vectors of membership degrees to extracted topics. While the work by Thomas et al. is a noteworthy preceding study, it is a different approach than a test case clustering. From a similar perspective, we will propose another, easier, way of test case clustering by focusing on words of test cases in the following section.

III. Test Case Recommendation

A. Our Current Condition and Goal

At first, we will explain the current condition of our regression testing.

For a software product, we have n test cases, t_i (for $i = 1, \ldots, n$). The product has been evolved through minor upgrades, and the latest version is the m-th version. For the sake of convenience, we denote the j-th version by v_j (for $j = 1, \ldots, m$); that is to say, the latest version is v_m.

Now, we are about to test v_m of the product. In other words, the tests for v_1, v_2, $\cdots$, v_{m-1} are already finished, and we have the results (the test history) like those shown in Table I. In the table, symbols "P" and "F" show that the result of the corresponding test case's run at the corresponding version was "Pass" and "Fail," respectively; the symbol "—" signifies the corresponding test case was not executed at the corresponding version. Since we are about to test at v_m, the cells in the column of v_m are not decided: they are denoted by "?."

Next, we describe our goal in this study.

TABLE I
Format of our test history data.

	v_1	v_2	v_3	$\cdots$	v_{m-2}	v_{m-1}	v_m
t_1	P	—	—	$\cdots$	—	—	?
t_2	—	F	P	$\cdots$	F	P	?
t_3	F	P	—	$\cdots$	P	—	?
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
t_n	—	—	P	$\cdots$	—	F	?

(P: Pass; F: Fail; —: No run; ?: to be decided)

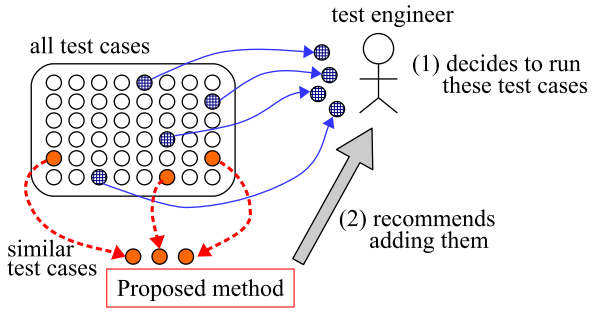

Fig. 1. Image structure of our proposal: blue hatched circles are manually-selected test cases, and orange ones are automatically-recommended test cases which are similar to manually-selected ones.

Test engineers usually decide which test cases they rerun, based on their knowledge in regard to the upgraded functions and the past records of testing. We do not criticize such a decision for regression testing. Since they would have specific knowledge and experience in testing their software products, it would be a practical way of making a decision. However, there is a risk of overlooking something because test engineers are also human beings. Thus, our aim is to automatically recommend appropriate test cases to reduce such a risk. That is to say, once a test engineer has decided to run a set of test cases, our method recommends adding other test cases to their candidate set if there are any that are appropriate (see Fig. 1).

B. Similarity between Test Cases

In general, test engineers prepare many test cases and run them to test the functionalities of their software product. In those test cases, there may be ones that are similar to each other. For example, suppose we want to test two different functions, A and B, under a certain condition, and have two test cases t_{AB} and t_{BA}: while t_{AB} executes in the order (A, B), t_{BA} does in the opposite order (B, A). In such a case, the contents of t_{AB} and that of t_{BA} would be similar. However, since they are not identical, we should not skip over either t_{AB} or t_{BA}; we have to run both of them. Some test engineers might run only t_{AB}, and overlook a failure which would have been detected by t_{BA}.

In this paper, we focus on the similarity of the contents between test cases. Using a morphological analysis, we evaluate their similarity. Then, when a test case is run, we consider whether to recommend running similar test cases as well.

We describe a quantification of the similarity between test cases in the remainder of this subsection.

1) A morphological analysis of a test case:
For each test case, we prepare a plain text file describing the content of the test case. Then, we apply a morphological analysis tool to the file, and extract words written in each test case and their parts of speech.
Since our (the authors') test cases are written in Japanese, we use MeCab[1] which is the most popular morphological analysis tool for Japanese, as the tool.

[1]http://taku910.github.io/mecab/

2) A set of words of interest:
In accordance with many cases of natural language processing, we focus only on a few parts of speech in this study—nouns, adjectives and verbs. Then, we form the set of words (nouns, adjectives and verbs) appearing in each test case. For the sake of convenience, let W_i be the set of words of test case t_i.
3) A similarity computation:
We define the similarity between test cases t_i and t_j by the following Jaccard index [14], $J(t_i, t_j)$:

$$J(t_i, t_j) = \frac{|W_i \cap W_j|}{|W_i \cup W_j|} \ . \quad (1)$$

The Jaccard index shows how many words are common to two documents (test cases). If their word sets are the same, the Jaccard index is 1; if their word sets have no intersection, the index is 0.

□

Let us consider a simple example. Suppose we have three test cases t_1, t_2 and t_3: their word sets W_1, W_2 and W_3 are shown in Table II.

Then, we obtain the followings:

- $W_1 \cup W_2 =$ {account, button, chronological, click, configuration, date, display, download, file, log, order, password, reset, user} and $W_1 \cap W_2 =$ {button, click, file}; $|W_1 \cup W_2| = 14$ and $|W_1 \cap W_2| = 3$.
- $W_1 \cup W_3 =$ {archive, button, click, check, chronological, date, display, download, file, order, log} and $W_1 \cap W_3 =$ {button, click, chronological, date, download, file, order}; $|W_1 \cup W_3| = 11$ and $|W_1 \cap W_3| = 7$.
- $W_2 \cup W_3 =$ {account, archive, button, click, chronological, configuration, date, download, file, order, password, reset, user} and $W_2 \cap W_3 =$ {button, click, file}; $|W_2 \cup W_3| = 13$ and $|W_2 \cap W_3| = 3$.

Therefore, their similarities are as follows:

- $J(t_1, t_2) = 3/14 \simeq 0.214$;
- $J(t_1, t_3) = 7/11 \simeq 0.636$;
- $J(t_2, t_3) = 3/13 \simeq 0.231$.

C. Clustering Test Cases

Next, we categorize test cases into clusters by using the similarity between test cases discussed above. By clustering similar test cases, we can narrow down a set of candidate test cases to be recommended for regression testing; when a test case t is rerun as a regression test, we may recommend running similar test cases as well, from the cluster which t belongs to.

TABLE II
EXAMPLE OF WORD SETS.

word set	words (only nouns, adjectives and verbs)			
W_1	button, display, order	click, download,	chronological, file,	date, log,
W_2	account, file,	button, password,	click, reset,	configuration, user
W_3	archive, date,	button, download,	click, file,	chronological, order

In order to perform a clustering algorithm for test cases, we have to define a "distance" between test cases. In this paper, we will consider a more similar pair of test cases to be a closer pair. Thus, we define the distance between two test cases t_i and t_j, $d(t_i, t_j)$, as:

$$d(t_i, t_j) = 1 - J(t_i, t_j) \ . \quad (2)$$

$d(t_i, t_j)$ is called Jaccard distance. While there are also other distance functions (other than Jaccard distance), we will use it because it is easy-to-implement and widely known in the natural language processing world.

Once a distance between test cases is defined, we can apply a clustering algorithm to a set of test cases. In this study, we adopt the complete linkage method as our clustering method, which is one of the most popular clustering methods, because of its ease of implementation. A popular open source statistical computing environment R[2] provides function `hclust` which performs a hierarchical cluster analysis with the complete linkage method. We can obtain a tree diagram (dendrogram) like one shown in Fig. 2. Then, we can get cluster data by setting its "cut level," which is the threshold of distance between test cases, to decide if those test cases belong to the same cluster or not. We empirically set 0.3 as the cut level in this study.

D. Prioritizing Test Cases by Mahalanobis-Taguchi Method

While the above clustering method can form a set of test cases that are recommended to be rerun, we also have to decide the order of their rerunning.

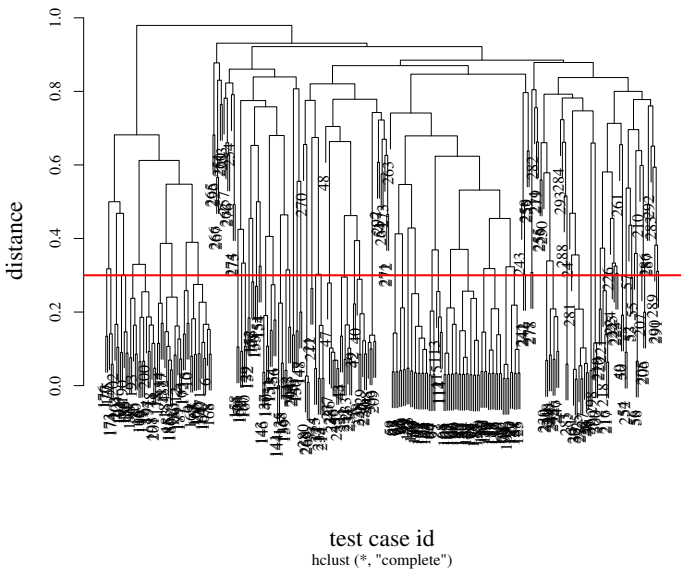

Fig. 2. Example of dendrogram with the cut level 0.3.

[2]https://www.r-project.org/

We have used the following two criteria for evaluating test cases in the past [7]:

- Gap between the Last-run version and the Current version (GLC): GLC(t) is the number of consecutive versions in which test case t was not run until the current version. A higher GLC value means that the test case has not been tested for a longer time. Thus, there is a higher risk of overlooking a regression.
- Failure Rate (FR): FR(t) is the failure rate of test case t. That is to say, it is the number of versions failed by t, divided by the number of versions at which t was run. A test case with a higher FR value has a better track record of detecting failures. Thus, it seems to have a higher worth to be rerun.

In order to consistently integrate different evaluations of a test case by the above criteria, we have leveraged the Mahalanobis-Taguchi method [7], [15].

In the remaining part of this subsection, we briefly describe how to prioritize test cases by using the Mahalanobis-Taguchi method (see [7] for more detail).

Suppose a feature of an object is expressed by n-dimensional vector $\boldsymbol{x}$, and similar objects form a set (group) in the vector space. Let $\overline{\boldsymbol{x}}$ be the mean vector of the feature vectors belonging to a group, and let S be the variance-covariance matrix of them. Then, the following $d_M(\boldsymbol{x}, \overline{\boldsymbol{x}})$ is known as the "Mahalanobis distance" between $\boldsymbol{x}$ and $\overline{\boldsymbol{x}}$:

$$d_M(\boldsymbol{x}, \overline{\boldsymbol{x}}) = \sqrt{(\boldsymbol{x} - \overline{\boldsymbol{x}})^T S^{-1} (\boldsymbol{x} - \overline{\boldsymbol{x}})} \ . \tag{3}$$

In the simple one-dimensional case, it is:

$$d_M(x, \overline{x}) = \sqrt{(x - \overline{x}) \frac{1}{\sigma^2} (x - \overline{x})} = \frac{|x - \overline{x}|}{\sigma} \ , \tag{4}$$

where σ is the standard deviation; σ^2 is the variance.

That is to say, the Mahalanobis distance can be interpreted as the "Euclidean distance" normalized by the data dispersion. A larger Mahalanobis distance means that $\boldsymbol{x}$ has loss of a membership in the group. The Mahalanobis-Taguchi method computes the Mahalanobis distance from the center of objects which are working normally, and uses the distance as the evaluation of the abnormality; it is a well-known quality control method (see Fig. 3).

Based on the notion of the Mahalanobis-Taguchi method, we have proposed to evaluate the rerunning value of a test case

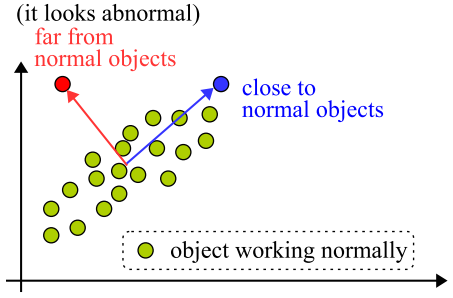

Fig. 3. Example of the Mahalanobis-Taguchi method.

TABLE III
SIMPLE EXAMPLE OF TEST HISTORY DATA, GLC, FR AND RERUNNING VALUES OF TEST CASES.

	v_1	v_2	v_3	v_4	v_5	v_6	GLC	FR	value
t_1	P	—	—	—	—	?	5	0.00	9.16
t_2	—	F	P	F	P	?	1	0.50	8.46
t_3	F	P	—	P	—	?	2	0.33	8.03
t_4	F	P	F	P	—	?	2	0.50	10.25

(P: Pass; F: Fail; —: No run; ?: to be decided)

by using the Mahalanobis distance from the least valuable test case such that both of its GLC value and FR value are zero [7].

Table III shows a simple example of 4 test cases $t_1, \cdots, t_4$ and 6 versions $v_1, \cdots, v_6$: we are able to test the product at v_6. The GLC values and FR values of $t_1, \cdots, t_4$ are also given in the table. For example, since the last-run version of t_1 is v_1, its GLC value is $5 (= 6 - 1)$; t_3 was run 3 times and it failed once (at v_1), thus its FR value is $1/3 \simeq 0.33$. Since the covariance of GLC and FR is -0.389, and the variances of GLC and FR are 3 and 0.056, respectively, we have the following variance-covariance matrix S:

$$S = \begin{bmatrix} 3 & -0.389 \\ -0.389 & 0.056 \end{bmatrix} .$$

Then, we can obtain its inverse matrix S^{-1}:

$$S^{-1} = \begin{bmatrix} 3.36 & 23.32 \\ 23.32 & 179.87 \end{bmatrix} .$$

Therefore, we can compute the rerunning value of t_1 with Eq.(3) as

$$\sqrt{\begin{bmatrix} 5-0 & 0-0 \end{bmatrix} \begin{bmatrix} 3.36 & 23.32 \\ 23.32 & 179.87 \end{bmatrix} \begin{bmatrix} 5-0 \\ 0-0 \end{bmatrix}} \simeq 9.16 \ .$$

Similarly, we can get the rerunning values of t_2, t_3 and t_4. The computation results are shown in "value" column of Table III.

E. Recommendation Procedure

Now, we suppose that a test engineer decides to rerun some test cases as a regression test on a version of a software product. Let R_0 be the set of those test cases.

Then, we propose the following method for recommending other test cases in order to prevent an overlooking of regressions; let R_1 be the set of those recommended test cases ($R_0 \cap R_1 = \emptyset$):

1) We classify test cases into clusters by using the similarity of test case descriptions (see Sections III-B, III-C).
2) Let $R_1 = \emptyset$. Then, we iterate the following operation for each $t \in R_0$:
   $R_1 \leftarrow R_1 \cup (C(t) \backslash R_0)$, where $C(t)$ is the cluster to which t belongs, and "\" signifies the operator to obtain the difference set[3].

[3] $x \in A \backslash B \iff x \in A \wedge x \notin B$.

3) For each $t \in R_1$, we collect its GLC value and its FR value from its test history, then compute its rerunning value (the Mahalanobis distance from the least valuable case: GLC = FR = 0) (see Section III-D).
4) We recommend $t \in R_1$ in descending order of the rerunning value, except for the ones whose values are zero.

□

IV. Empirical Study

A. Aim and Dataset

The aim of this empirical study is to show the usefulness of the proposed method, by using the regression testing data on an actual software product.

In this study, our experimental subjects are $300 (= n)$ test cases for a software product (a Web-based information system) that has been maintained by two of the authors. All test cases are written in Japanese (natural language)[4]. Thirteen $(13 (= m))$ versions of the product have been tested by those 300 test cases—for the sake of convenience, we denote these versions by v_1, v_2, $\cdots$, v_{13} in chronological order.

Since it is past test data, a follow-up investigation of regressions is already finished. According to the results of the follow-up investigation, there were regressions when the product was upgraded from v_6 to v_7. However, regression testing (rerunning test cases) missed those failures. We try to detect those overlooked failures with the test cases which are recommended by the proposed method.

B. Procedure

This empirical study is conducted in accordance with the following procedure:

1) We perform a morphological analysis on each of the 300 test cases. Then, for each combination of test cases, we compute the distance between them based on their similarity, with Eqs. (1), (2). Using those distances, we classify the test cases into clusters. In the clustering algorithm, we empirically set 0.3 as the cut level.
2) We iterate the following for each version (v_j for $j = 2, \ldots, 13$):
   a) Let R_0 be the set of test cases which were run at v_j.
   b) We create the set of recommended test cases, R_1, according to the procedure described in Section III-E.
   c) We recommend test cases belong to R_1 in descending order of their rerunning values which are computed by the Mahalanobis-Taguchi method.
   d) We examine if the recommended test cases can detect a regression.

C. Results and Discussions

Table IV shows the results of test case recommendation at each version:

[4]Since the contents of test cases are confidential, we will omit details of them and avoid giving examples in this paper.

TABLE IV
Results of Recommendations and Regression Detections.

Version	#run ($\lvert R_0 \rvert$)	#recommended ($\lvert R_1 \rvert$)	#defects detected
v_2	160	13	0
v_3	17	16	0
v_4	27	0	0
v_5	10	1	0
v_6	5	5	0
v_7	7	15	6
v_8	2	0	0
v_9	5	1	0
v_{10}	9	4	2
v_{11}	13	1	0
v_{12}	6	5	2
v_{13}	3	1	0

- #run column: the number of test cases that were run at the version (past records). That is $|R_0|$.
- #recommended column: the number of test cases that are recommended by the proposed method. That is $|R_1|$.
- #defects detected column: the number of test cases that are included in R_1 and can detect a regression. That is to say, at a version whose #defects detected > 0, if we also rerun those test cases, we would have detected a regression which was in fact overlooked.
  Since there were regressions when the product was upgraded from v_6 to v_7, #defects detected $= 0$ for v_2, $\cdots$, v_6.

□

Table IV shows interesting results. At v_7 which is soon after the introduction of latent faults (regressions), only 7 test cases were rerun (see v_7 row, #run column in Table IV). However, the proposed method recommends 15 test cases to rerun in addition to those 7, and 6 out of the newly recommended 15 test cases succeed in detecting regressions. It is a remarkable point that the proposed method recommends more than two times as many test cases as the number actually run (15 vs. 7). That is to say, the original regression testing plan at v_7 seemed to be insufficient, and it caused an overlooking of regressions. We consider the effectiveness of the proposed method to detect regressions at v_7 as well. The proposed method recommended 15 test cases at v_7, and 6 out of them found regressions. If we randomly selected 15 test cases (except for those 7 test cases which were already selected) at v_7, the expected number of test cases which succeed to detect regressions is about 1.13. Thus, the proposed method could be about six times more effective than the random selection.

Figure 4 shows the ratio of the recommended test cases to the run test cases ($|R_1|/|R_0|$). After the occurrence of latent regressions, the proposed method succeeds in finding a regression when the ratio is relatively high.

From these results, we can expect that our recommendation method based on the similarity between test cases' descriptions is useful to prevent overlooking regressions. When a test case's description is similar to another's description, those two test cases would test similar functions or related functions

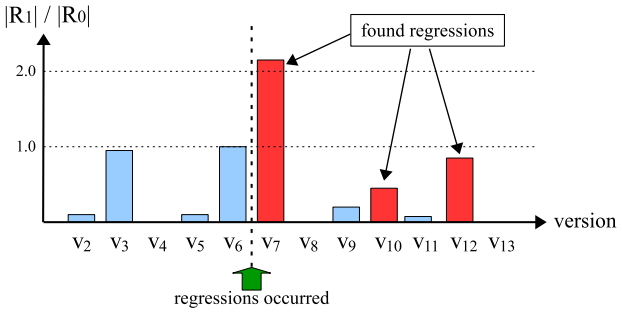

Fig. 4. Ratio of the recommended test cases to the run test cases ($|R_1|/|R_0|$).

in partially different conditions. In such a case, it would be insufficient to run only one of them; by skipping a run of the remaining test case, we might overlook a regression. This empirical study would show such a case of oversight.

Next, we consider the usefulness of prioritizing test cases within the recommended test case set (R_1). As shown in Table IV, 15 test cases are recommended at v_7. Table V shows the results of prioritizing those 15 test cases in descending order of the rerunning value (the Mahalanobis distance from the least valuable case). In the table, test cases' names are relabeled by the order; the "detecting defect" column shows if the corresponding test case succeeds in detecting a regression. Since all of the successful test cases are ranked in the upper half, our prioritization leveraging the Mahalanobis-Taguchi method seems to work successfully.

Meanwhile, there might be a case that the proposed method recommends too many test cases to run. In such a situation, we have to single some test cases out from the recommendation set. The simplest way is the greedy method in which we continue to select test cases in descending order of priority until we exhaust our testing effort. However, the greedy method cannot present the optimal solution; there is a better, more cost-effective way in which we can formulate the test case selection problem as a 0-1 programming problem and use its solution [7]—let r_i and c_i be the rerunning value of test case t_i and the cost to rerun t_i, respectively (for $i = 1, \ldots, n$). Then, our test case selection problem can be formulated as:

$$\begin{aligned} \text{Maximize} \quad & \sum_{i=1}^{n} r_i \cdot x_i, \\ \text{subject to} \quad & \sum_{i=1}^{n} c_i \cdot x_i \leq c_L, \\ & x_i \in \{0, 1\} \ (i = 1, \ldots, n), \end{aligned}$$

where x_i expresses whether we select t_i ($x_i = 1$) or not ($x_i = 0$), and c_L is the upper limit of cost which we cannot exceed. The solution of the above problem provides the best set of test cases to be rerun under conditions that the rerunning values of all test cases and the upper limit of cost are given.

Finally, we discuss the "cut level" at our clustering in this empirical work. While we set 0.3 as the cut level based on experience, another cut level would produce another result. For version v_7, we examined all cut levels from 0.1 to 0.9 with 0.1 increases. Table VI shows the results: the number of recommended test cases (#recommended), the number of recommended test cases which can detect a defect (#defects) and the detection rate (= #defects / #recommended) for each cut level. Needless to say, a higher cut level produce larger clusters, thus more test cases are recommended, and more regressions (defects) are also detected. While the cut level that we empirically used (0.3) showed the best detection rate in these candidates, it does not succeed to detect all overlooked regressions. Thus, another cut level may be the best in another context. A further analysis of cut level is our significant future work.

D. Threats to Validity

Our findings may be subject to the following concerns.

Since our study covers a part of regression testing for a single product, we cannot say our results are generalizable. However, we believe this study could contribute to stir up a utilization of the morphological analysis in the regression testing world.

Since the test cases that we used in our empirical work are written in a natural language, there might be a variety of vocabulary. That is to say, different engineers might use different words to describe the same thing in their test cases, and they could cause incorrect evaluations of similarities

TABLE V
MAHALANOBIS DISTANCES OF RECOMMENDED TEST CASES AT v_7.

Test case	rerunning value (Mahalanobis distance)	detecting defect
t_{x1}	8.21	Yes
t_{x2}	8.21	No
t_{x3}	8.21	Yes
t_{x4}	8.21	No
t_{x5}	8.21	Yes
t_{x6}	8.21	Yes
t_{x7}	8.21	Yes
t_{x8}	8.21	Yes
t_{x9}	8.21	No
t_{x10}	6.57	No
t_{x11}	6.57	No
t_{x12}	6.57	No
t_{x13}	6.57	No
t_{x14}	6.57	No
t_{x15}	6.57	No

TABLE VI
RESULTS OF RECOMMENDATIONS AND REGRESSION DETECTIONS AT v_7 FOR DIFFERENT CUT LEVELS.

cut level	#recommended	#defects	detection rate
0.1	4	0	0
0.2	6	0	0
0.3	15	6	0.400
0.4	28	8	0.286
0.5	27	8	0.296
0.6	40	10	0.250
0.7	59	12	0.203
0.8	66	12	0.182
0.9	116	22	0.190

among test cases. In order to avoid such a variety, we need to prepare a common dictionary of terms (words). Alternatively, it would be better to perform a data preprocessing to link a word with another word which has the same meaning. A further analysis of vocabulary in test cases is our future work.

The cut level, a parameter of test case clustering, plays an important role in our proposal. As mentioned above, another cut level can produce another performance in recommending test cases. While we set 0.3 as the cut level based on experience, our experience might not work well for another data set: if we use another set of test cases which are prepared by other engineers of another organization, we would have to reexamine the cut level. We plan to perform a further analysis of cut level and seek a better way of deciding an appropriate cut level in the future.

V. Conclusion

We focused on the content of test cases, and evaluated the similarity between test cases through a morphological analysis. Based on those similarities, we categorized test cases into clusters. Then, we proposed a cluster-based method with which to recommend test cases for the regression testing—once a test case is selected for a regression testing, the proposed method recommends other test cases which should also be run in order to reduce the risk of overlooking regressions (failures). The proposed method also leveraged the Mahalanobis-Taguchi method for prioritizing test cases within the recommendation set of test cases.

We conducted an empirical study with a real product, and the results showed that the proposed method successfully recommended test cases which found latent regressions. The Mahalanobis-Taguchi method also worked well for prioritizing test cases. Moreover, we saw the following finding: if more test cases are recommended by the proposed method at a version, there may be a higher risk of overlooking regressions at that version. Thus, the proposed method would be helpful in evaluating a test plan as well.

Our future work includes the following: (1) to apply the proposed method to other products for the purpose of examining its generality; (2) to perform further analyses on the parameter of the clustering algorithm and on the features of test cases from the perspective of the natural language analysis.

References

[1] T. L. Graves, M. J. Harrold, J.-M. Kim, A. Porter, and G. Rothermel, "An empirical study of regression test selection techniques," *ACM Transactions on Software Engineering and Methodology*, vol. 10, no. 2, pp. 184–208, Apr. 2001.

[2] S. Yoo and M. Harman, "Regression testing minimization, selection and prioritization: A survey," *Software Testing, Verification and Reliability*, vol. 22, no. 2, pp. 67–120, Mar. 2012.

[3] D. Jeffrey and N. Gupta, "Test case prioritization using relevant slices," in *Proc. 30th Annual International Computer Software and Applications Conference*, vol. 1, 2006, pp. 411–420.

[4] S. Mirarab and L. Tahvildari, "A prioritization approach for software test cases based on bayesian networks," in *Proc. 10th International Conference on Fundamental Approaches to Software Engineering*, 2007, pp. 276–290.

[5] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proc. 24th International Conference on Software Engineering*, 2002, pp. 119–129.

[6] H. Aman, M. Sasaki, K. Kureishi, and H. Ogasawara, "Application of the 0-1 programming model for cost-effective regression test," in *Proc. 37th Annual International Computer Software and Applications Conference*, July 2013.

[7] H. Aman, Y. Tanaka, T. Nakano, H. Ogasawara, and M. Kawahara, "Application of Mahalanobis-Taguchi method and 0-1 programming method to cost-effective regression testing," in *Proc. 2016 42nd Euromicro Conference on Software Engineering and Advanced Applications*, Aug. 2-16, pp. 240–244.

[8] M. Sherriff, M. Lake, and L. Williams, "Prioritization of regression tests using singular value decomposition with empirical change records," in *Proc. 18th IEEE International Symposium on Software Reliability Engineering*, 2007, pp. 81–90.

[9] R. Carlson, H. Do, and A. Denton, "A clustering approach to improving test case prioritization: An industrial case study," in *Proc. 27th IEEE International Conference on Software Maintenance*, 2011, pp. 382–391.

[10] D. Leon and A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases," in *Proc. 14th International Symposium on Software Reliability Engineering*, 2003, pp. 442–453.

[11] M. J. Arafeen and H. Do, "Test case prioritization using requirements-based clustering," in *Proc. IEEE 6th International Conference on Software Testing, Verification and Validation*, 2013, pp. 312–321.

[12] Y. Ledru, A. Petrenko, S. Boroday, and N. Mandran, "Prioritizing test cases with string distances," *Automated Software Engineering*, vol. 19, no. 1, pp. 65–95, Mar. 2012.

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

[14] G. Salton, *Automatic text processing: the transformation, analysis, and retrieval of information by computer*. Boston, MA: Addison-Wesley Longman Publishing, 1989.

[15] G. Taguchi, S. Chowdhury, and Y. Wu, *The Mahalanobis-Taguchi System*. New York, NY: Mc, 2001.