# A Document-based Parameter Correlation Metric for Test Design

Hiroyuki Nakagawa
Graduate School of Information Science and Technology
Osaka University
Osaka, Japan
Email: nakagawa@ist.osaka-u.ac.jp

Nobukazu Ishii
Information and Computer Sciences
School of Engineering Science
Osaka University
Osaka, Japan
Email: n-ishii@ist.osaka-u.ac.jp

Tatsuhiro Tsuchiya
Graduate School of Information Science and Technology
Osaka University
Osaka, Japan
Email: t-tutiya@osaka-u.ac.jp

***Abstract*—Efficient software testing requires precise test space definition. To determine the test space, constraint elicitation is one of the important processes in a test design; however, the process usually requires manual capturing and precise definition of constraints. We have developed a constraint elicitation process that helps to define constraints from documents relevant to the test model. In this paper, we propose a refined metric that finds parameter combinations to be extracted more precisely. This metric determines the parameter correlation on the basis of word co-occurrences in the specification document. We conduct experiments on some test models and demonstrate that our metric allows us to find parameter combinations that form constraints with a high recall rate.**

## I. Introduction

Software testing is an essential activity in the software development process. In order to conduct the activity correctly and efficiently, the testing technique requires precise test space definition to perform testing. We consider a test space that is modeled by a set of parameters, their values, and constraints on the value combinations [1] [2]. The constraints define the value combinations that are prohibited and should be excluded from test cases. Constraint elicitation and handling is crucial in test design [3] [4]; however, the constraint elicitation process has not been well studied.

Our objective is to construct a constraint elicitation process that helps us define constraints. In general, constraint elicitation processes have to solve two problems, that is, *"How do we find which parameter combinations form constraints?"*, and *"How do we find which value combinations on the given parameters define constraints?"*.

We have designed the overview of a constraint elicitation process [5]. To solve the first problem, we proposed a metric for calculating the correlation between parameters, which uses the distances between words in a document of a System Under Test (SUT) to estimate parameter correlations, in the previous work [5]. In this paper, we introduce an enhanced metric to estimate the correlation between parameters more precisely. Since constraints are defined on the relationships between relevant parameters, such a metric helps to find which parameter combinations should be considered to define constraints. We previously proposed a metric for calculating the correlation between parameters in [5], which sums up the distances between words of two parameters in a specification document of a System Under Test (SUT). We conduct experiments on two real world applications, a web application and a Unix command, to evaluate the validity of our metric. This empirical evaluation indicates that our new metric allows us to find parameter combinations that form constraints from a specification document of SUT with a higher precision and recall rate than the previous metric.

The rest of the paper is organized as follows: Section 2 gives the background of this study by providing the explanation of constraints; Section 3 gives the overview of our approach; Section 4 describes how we find parameter combinations that probably cause constraints using our metric; Section 5 presents the results of two experiments on real world testing examples, and Section 6 explains how we evaluate our approach with the experimental results; Section 7 discusses related work, and Section 8 concludes the paper.

## II. Constraints

As an example of a System Under Test (SUT), consider a web application which may be influenced by various factors including operating systems, browsers and memory size. Suppose that the three factors are chosen as test parameters. We list the possible values of parameters in Table I. This test model has three parameters, i.e., OS, Browser, and RAM (memory), with their parameter values. A test case is a vector of parameter values, such as (Windows, Chrome, 4GB).

Formally the test space is modeled by a set of parameters, their values, and constraints on the value combinations. In particular, constraints define the combinations that never happen and must be excluded from test cases. For example, when we choose Mac for the parameter OS, we should choose as the

TABLE I: A cross-browser test model.

| Parameter | Values |
|---|---|
| OS | Windows, Mac, Linux |
| Browser | IE, Safari, Chrome |
| RAM | 512MB, 1GB, 2GB, 4GB |

DOI: 10.18293/SEKE2018-069

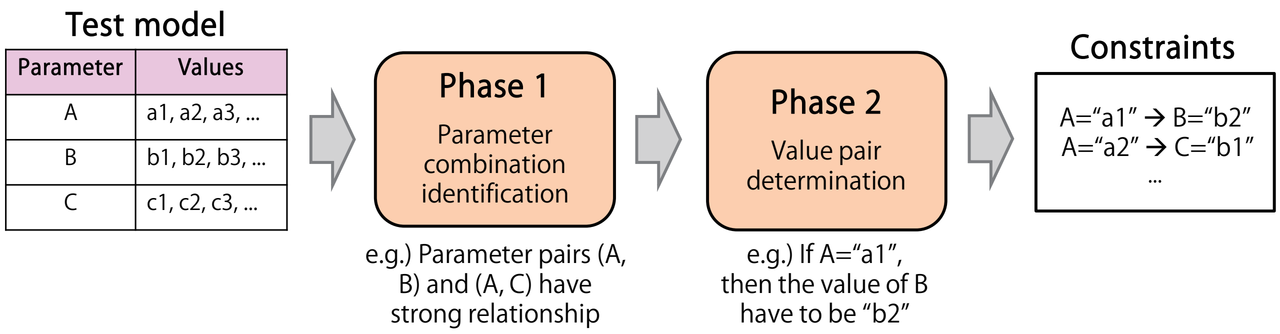


Fig. 1: An overview of the constraint elicitation process. This paper focuses on Phase 1. We have proposed a method [5] for the activity in Phase 2.

browser Safari or Chrome, because Mac OS does not support IE (Internet Explorer). This constraint is defined as follows: OS = "Mac" ⇒ Browser = "Safari" || "Chrome".

In the presence of constraints it is necessary to design a test suite such that all test cases satisfy the constraints. Otherwise, some test cases would not be executable because of constraint violation, resulting in redoing the test design process. A test suite constructed *with* considering constraints is different with one constructed *without* considering constraints because these test spaces are not the same.

## III. Constraint Elicitation Process

The objective of our study is to support identifying which combinations of values define constraints. There are two difficulties in the constraint elicitation: one exists in *parameter combination identification* and the other exists in *value combination identification*. In our example explained in Section II, we first have to identify which parameter combinations, such as parameters "OS" and "Browser", form constraints. Next, after identifying parameter combinations that cause constraints, we also have to determine which value combinations cause constraints. In the above example, it corresponds to the identification of the value combinations, such as "Mac" and "Safari" on the parameter combination "OS" and "Browser".

Figure 1 illustrates an overview of our constraint elicitation process. This process consists of the following two steps:

- **Phase 1: Parameter combination identification.** We identify which parameter combinations form constraints. We find such combinations by analyzing a specification document for the SUT. For this analysis, we use a metric that we propose in this paper.
- **Phase 2: Value pair determination.** We determine which value combinations cause constraints.

The goal of this paper is to help find which parameter combinations have strong relationships in Phase 1. Although Phase 2 is beyond the scope of our paper, we have proposed a method of determining value combinations that define constraints using a web search engine [5]. We use hits of search results as a metric. We support that excessively higher/lower hits indicate that the corresponding value pairs are bound/uncommon pairs and therefore they probably define constraints.

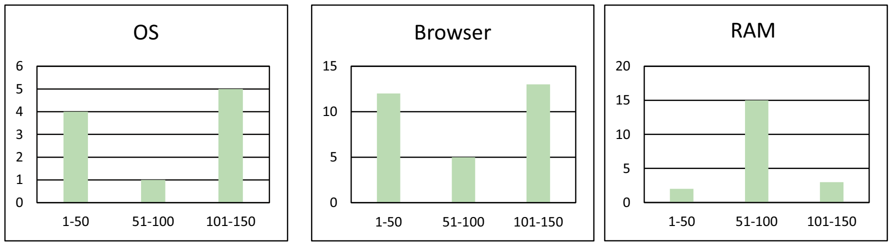
## IV. Parameter Combination Identification

The objective of Phase 1 is to identify which parameter combinations have strong relationships. Most constraints are caused by strong relationships between parameters. In order to find such strong relationships, we use a specification document as a definitive source and a metric for calculating correlation between parameters. The correlation $\rho(f, g)$ between parameters $f$ and $g$ represents how strong the relationship between two parameters is. The value of the metric is calculated by using a $\mathit{diff}(f, g)$ value, which represents how differently relevant words of these two parameters appear in the document.

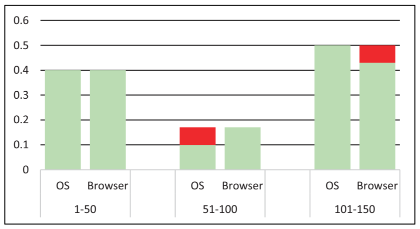
Our parameter combination identification process consists of the following steps. A specification document for the test model is given to the process as an input data. We assume that the document is given as a sequence of English words.

- **Step 1:** select a set of words for each parameter. The members of the set are the parameter name and values of the parameter. If a value of the parameter is a common word, such as "on" and "off", the value is excluded from the set. Instead, representative words that can explain the parameter, such as the full name of the parameter, can also be the members, if they exist. We call this set *word group*.
- **Step 2:** split the given document into multiple small parts (bins). In this paper, we construct bins all of which have the same size, that is, all of the bins contain the same number of words.
- **Step 3:** count the occurrences of words in every word group within each bin.
- **Step 4:** construct relative frequency tables for each word group using the results of Step 3.
- **Step 5:** sum up the difference of bin values between every two word groups. These values provide the *diff* values.
- **Step 6:** output the reciprocals of the *diff* values as *correlation* $\rho$ values ($\rho(f, g) \propto 1/\mathit{diff}(f, g)$).

We briefly explain our identification process using our cross-browser testing example described in Section II. First, we define word groups (Step 1). The word group is defined as a set whose members are the parameter, the values of the parameter, and relevant words. In our example, the word group of OS contains the words such as "OS" and "Windows". Next,



(a)



(b)

Fig. 2: (a) Histograms for word groups in the cross-browser testing example. (b) The difference of bin values between relative frequency tables for parameters OS and Browser. The sum of red bars corresponds to the *diff(OS, Browser)* value.

TABLE II: Frequency and relative frequency tables. The table unifies three frequency tables (denoted by "Freq.") and three relative frequency tables ("Relative").

| Group | Bin | 0-50 | 51-100 | 101-150 | Total |
|---|---|---|---|---|---|
| OS | Freq. | 4 | 1 | 5 | 10 |
| | Relative | 0.4 | 0.1 | 0.5 | 1 |
| Browser | Freq. | 12 | 5 | 13 | 30 |
| | Relative | 0.4 | 0.17 | 0.43 | 1 |
| RAM | Freq. | 2 | 15 | 3 | 20 |
| | Relative. | 0.1 | 0.75 | 0.15 | 1 |

we split the document into small parts (Step 2). We assume that a given document for the system consists of 150 words. In this example, we split it into three parts, i.e., words from the first to the 50th word as the first part, words from the 51st to the 100th word as the second part, and so on. Third, we count the occurrences of the words in every word group within each bin (Step 3). Figure 2(a) illustrates the counting result of the example. After counting the words within each bin for every word group, we construct relative frequency tables (Step 4). Algorithm 1 explains the details of Step 4. Table II lists the results of Step 4 in our example. Next, we sum up the difference of bin values between every pair of two word groups to calculate the *diff*$(f, g)$ values (Step 5). Figure 2(b) represents the difference of bin values between relative frequency tables for parameters OS and Browser. We sum up these differences (red bars) and regard the total value as the *diff* value. This value becomes low when the occurrences of a word group is similar to those of the other word group. Finally we calculate parameter correlations as the reciprocals of the results of Step 5 (Step 6). Algorithm 2 explains the details of Steps 5 and 6. Table III lists the correlation $\rho$ values of our example. This result indicates that the parameter combination (OS, Browser) has strong relationship, and therefore, the combination may cause constraints, such as OS = "Mac" ⇒ Browser = "Safari" || "Chrome".

## V. Experiments

In order to evaluate the validity of our metric, we conducted experiments on some applications. We applied our process to two applications: a real world web application (Exp 1) and a Unix command (Exp 2).

**Algorithm 1** Construct relative frequency tables from a document.

```
[Input] doc: given specification document
[Input] wordgroup[][]: word groups
    // wordgroup[n]: word group for the n th parameter
[Input] BINS: the number of bins
[Output] relFreqDist[][]: relative frequency tables
    // relFreqDist[n]: a relative frequency table for the n th parameter (word group)
    // relFreqDist[n][i]: the relative frequency of i th bin for the n th parameter (word group)
 1: // split given document into a word list
 2: words ← split(doc);
 3:
 4: // count frequency
 5: freqDist[][] ← initialized by 0;
 6: currentBin ← 0;
 7: binSize ← words.size() / BINS; // the number of words for each bin
 8: for i = 0 to words.size() −1 do
 9:     for j = 0 to wordgroup.size() −1 do
10:         if wordgroup[j].contains(words[i]) then
11:             freqDist[j][currentBin]++;
12:         end if
13:     end for
14:     if (i + 1)%binSize== 0 then
15:         currentBin++; // move to the next bin
16:     end if
17: end for
18:
19: // construct relative frequency tables
20: for i = 0 to wordgroup.size() −1 do
21:     for j = 0 to BINS−1 do
22:         relFreqDist[i][j]
                    ← freqDist[i][j] / sum(freqDist[i])
23:     end for
24: end for
```

### A. Exp 1: Testing of web application

First, we conducted an experiment on a web application testing. The target application is Confluence [6], which is a web application designed for team collaboration. The SUT model for the Confluence testing has seven parameters, i.e., database, server, client, browser, add-ons, and attachment file type. We derived seven word groups corresponding to the parameters for this experiment as illustrated in Table IV. We

TABLE III: Parameter Correlations ($\rho$ values).

| | OS | Browser | RAM |
|---|---|---|---|
| OS | - | 7.14 | 0.77 |
| Browser | - | - | 0.86 |
| RAM | - | - | - |

**Algorithm 2** Calculate parameter correlation $\rho(f, g)$ using relative frequency tables.

```
[Input] $f[]$, $g[]$: relative frequency tables for word groups $f$ and $g$, respectively
    // $f[i]$: relative frequency of $i$ th bin
[Input] $BINS$: the number of bins
[Output] correlation between parameters (word groups) $f$ and $g$
1: for $i = 0$ to $BINS-1$ do
2:     $diff \leftarrow diff + |f[i] - g[i]|$
3: end for
4:
5: return $1/diff$; //parameter correlation
```

used the document [7] as an input document. We split the document into 960 parts (330 words per bin) based on the number of pages in this document. We determined the bin size to let each bin roughly correspond to a page. In reality, constraints such as the following ones exist in this test model:

- Constraint 1-a: Database = "Microsoft SQL Server" $\Rightarrow$ Server = "Windows Server".
- Constraint 1-b: Client = "Mac" $\Rightarrow$ Browser $\neq$ "IE".
- Constraint 1-c: Macro = "Multimedia Macro" $\Rightarrow$ File Type = "audio" || "video" || "animation".

We regarded the following six combinations, (Database, Server), (Browser, Client), (Browser, File type), (Browser, Add-on), (File type, Add-on), and (File Type, Macro), as the combinations to be extracted.

Table V lists the results of the correlation calculation. We extracted the top 30% combinations, whose values are shaded in Table V, from the calculation results. Observing the results of this experiment, although we extracted the combination (Server, Client), which is not involved in any constraints, the metric allowed us to extract most of the correct combinations that should be extracted.

We also observed the correctness of the proposed method. We evaluated the proposed metric by comparing it with the previous metric proposed in our preliminary work [5] as the baseline metric. Briefly explained, our previous study determines parameter correlation using the distance metric, which sums up the distances between words of two parameters in the document.

We use the following definition of the precision and recall: *Precision* = $|Correct \cap Extracted|/|Extracted|$; and *Recall* = $|Correct \cap Extracted|/|Correct|$ , where *Correct* is the set of correct pairs to be extracted and *Extracted* is the set of pairs that the baseline or proposed method actually extracts from the document. Figure 3 shows the precision and recall rates of the two methods. From the figure, both precision and recall rates of the proposed method are higher than the rates of the baseline method.

## B. Exp 2: Testing of a Unix command

Next, in order to evaluate the scalability of our approach, we applied our metric to a Unix command, `mount` (*Exp 2*). The SUT model for the testing on mount command has parameters for options, such as `-a`, `-o`, and `-t`, and arguments, such as ones for specifying devices, directories, and volume labels. We constructed 29 word groups corresponding to the parameters for this experiment. We used a manual of *mount* command [8] as an input document[1]. We split the document into 400 parts based on the number of lines in this document (32 words per a bin). In reality, some of constraints exist in the test model as follows:

- Constraint 2-a: "`-O`"= "ON" $\Rightarrow$ "`-a`"= "ON".
- Constraint 2-b: "`-r`"= "ON" $\Rightarrow$ "`-w`"= "OFF".
- Constraint 2-c: "`-t`"= "ON" $\Rightarrow$ "vfstype" $\neq$ "NULL".

In this experiment, we changed the extraction rate (top 5%, 10%, or 15%) and observed each precision and recall rates. From the results listed in Table VII, it was found that we were able to extract most of the parameter combinations to be extracted even if we extracted only top 5% combinations.

## VI. Discussion

We now discuss our correlation metric in the light of our experiments.

The experimental results in Exp 1 and Exp 2 demonstrate that our metric could extract most of the parameter combinations that form constraints. In Exp 1, both precision and recall rates of the proposed method are higher than the rates of our previous metric. While our previous metric evaluate the strength of parameter relationship using the distance between relevant words, the new metric uses co-occurrence of relevant words. If the word frequency is largely different among word groups, the latter metric works more properly than the previous one.

The fact that recall rates in Exp 2 are better than precision rates indicates that our metric extracts parameters to be extracted with low false negative rates. It means that the process using the metric allows developers to find almost all of the parameter combinations to be extracted by continuously relaxing the threshold, i.e., extraction rate in Exp 2. This process is equivalent to acquiring parameter combinations one by one from the parameter combination list sorted by the correlation metric in descending order. We can define the end condition of extraction using the number of a series of wrong extractions, which mean the extracted combinations are not involved any constraints.

Precision, on the other hand, was not high. The main reason is that the strong relationships do not always cause constraints. While this fact decreases the precision rate, we can still improve the precision rate. For example, we can improve the construction of bins. The current process constructs bins by dividing an input document equally so that the bins contain the same number of words. The mapping of bins to

[1]This document can be shown by specifying the keyword "mount" with "8- Maintenance Commands" option.

TABLE IV: Word groups for the Confluence testing (Exp 1). Group names correspond to the parameter names.

| **Group** | **Words in word group** |
|---|---|
| Database | database, PostgreSQL, MySQL, Oracle, Microsoft SQL Server, H2 |
| Browser | browser, IE, Internet Explorer, Firefox, Chrome, Safari, Mobile Safari |
| Server | server, Windows Server, Linux, Unix, Mac |
| Client | client, Windows, Linux, iOS, Android, Mac |
| File type | file type, file extension, upload file, image, Office, PDF, video, audio, animation, docx, doc, pptx, ppt, xlsx, xls, txt |
| Add-on | add-on, plugin, Scroll versions, copy space, scroll PDF Exporter, Gliffy, Lucidchart, Balsamiq |
| Macro | macro, Multimedia Macro, Space Attachments Macro, Office PowerPoint Macro, Gallery Macro, PDF Macro, View File Macro |

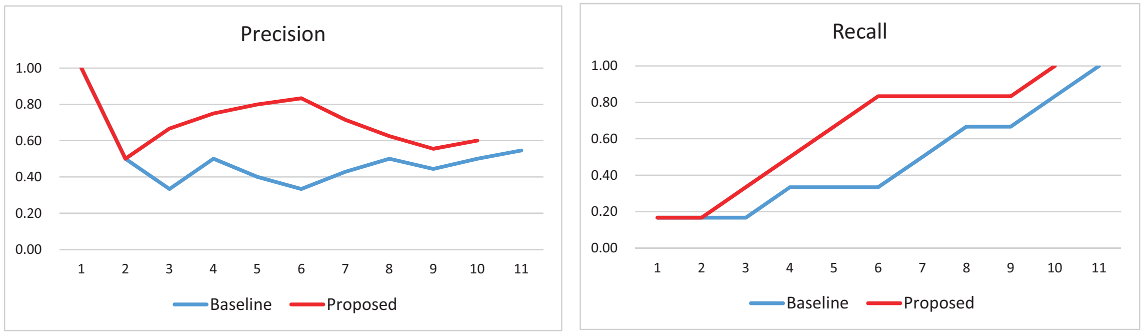


Fig. 3: The transition of precision and recall rates in Exp 1. X axis represents the number of extracted pairs in descending order.

TABLE V: Correlation values in Exp 1. Bold values represent correct combinations to be extracted. Shaded values represents combinations that our method extracted.

| | Browser | Server | Client | File type | Add-on | Macro |
|---|---|---|---|---|---|---|
| Database | 0.575 | **0.716** | 0.568 | 0.512 | 0.548 | 0.508 |
| Browser | | 0.578 | **0.608** | **0.602** | **0.588** | 0.525 |
| Server | | | 0.698 | 0.528 | 0.58 | 0.522 |
| Client | | | | 0.561 | 0.572 | 0.58 |
| File type | | | | | **0.575** | **0.603** |
| Add-on | | | | | | 0.567 |

sentimental blocks, such as paragraphs or sentences, will make the extraction more precise. The main objective of this paper is to elicit constraints; however, we can use the metric for other activities in the test design, such as test suite reduction [9] or the identification of a subset of parameters that require higher coverage strength in variable strength interaction testing [10].

The effectiveness of our parameter combination identification also depends on how we select relative words of parameters in their word groups. As demonstrated in Exp 1 and Exp 2, we added some characteristic words related to the parameters, such as synonyms and full names of options, into word groups. In order to choose more adequate word groups, we need a mechanism of collecting relevant words. We could use a query augmentation method, such as [11], or word2vec [12][13] for this purpose.

## VII. Related Work

Currently there are few studies on the constraint elicitation. Blue et al. [14] presents a test suite reduction method, which excludes test cases that contain prohibited value combinations. Their method uses existing test cases and minimizes the test suite with covering all combinations that are included in the existing test cases. Their method is useful to define a small set of test cases without violating constraints; however, the main focus is on the extraction of minimized test cases from the existing test cases.

Our approach, including our previous metric [5] explained in Section V, uses word frequencies in a document to identify the correlation between parameters. TF-IDF [15] is also known as a method of reasoning the relationship between words. Gabrilovich et al. [16] proposed a method called *Explicit Semantic Analysis (ESA)*, in which a word is represented as a vector whose attributes represent the relevance of individual concepts calculated using TF-IDF by using Wikipedia as an input document. While these co-occurrence-based approaches require a large number of document, our approach uses only one document. This feature is useful because documents related to the target systems are often limited.

The literature in the requirements engineering field has dealt with linguistic techniques. Falessi et al. [17] evaluate the performance of a large number of natural language processing techniques. They define seven principles for evaluating the performance of these techniques. Some of them could be useful to improve the correlation metric. Query augmentation techniques, such as one in [11], may improve the correctness of extracted parameter combinations by enhancing word groups.

## VIII. Conclusions

We defined a metric for estimating a parameter correlation for the test design. This metric allows us to identify

TABLE VI: Correlation values in the `mount` command testing (Exp2). Bold values represent correct combinations to be extracted. Shaded values represent the top 5% combinations that our method extracted.

| | -V | -v | -a | -F | ... | -r | -w | -L | -U | -t | -O | -o | -B | -R | -M | vfstype | device | dir | uuid | label | num | dirs | opts |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -h | 1000 | 1.5 | 0.583 | 0.5 | ... | 0.5 | 0.5 | 0.5 | 0.5 | 0.528 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 |
| -V | | 1.5 | 0.583 | 0.5 | ... | 0.5 | 0.5 | 0.5 | 0.5 | 0.528 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 |
| -v | | | 0.583 | 0.5 | ... | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.505 |
| -a | | | | **0.538** | ... | 0.5 | 0.5 | 0.5 | 0.5 | 0.679 | **1** | 0.519 | 0.5 | 0.5 | 0.5 | 0.538 | 0.524 | 0.536 | 0.5 | 0.5 | 0.5 | 0.5 | 0.526 |
| -F | | | | | ... | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.517 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.505 |
| ... | | | | | | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| -r | | | | | | | **0.667** | 0.5 | 0.5 | 0.5 | 0.5 | 0.538 | 0.5 | 0.5 | 0.5 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.511 |
| -w | | | | | | | | 0.5 | 0.5 | 0.5 | 0.5 | 0.519 | 0.5 | 0.5 | 0.5 | 0.5 | 0.51 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.509 |
| -L | | | | | | | | | 1000 | 0.5 | 0.5 | 0.519 | 0.5 | 0.5 | 0.5 | 0.5 | 0.51 | 0.5 | 0.917 | **0.667** | 0.5 | 0.5 | 0.503 |
| -U | | | | | | | | | | 0.5 | 0.5 | 0.519 | 0.5 | 0.5 | 0.5 | 0.5 | 0.51 | 0.5 | **0.917** | 0.667 | 0.5 | 0.5 | 0.503 |
| -t | | | | | | | | | | | 0.633 | 0.583 | 0.5 | 0.5 | 0.5 | **0.679** | 0.547 | 0.594 | 0.5 | 0.5 | 0.5 | 0.5 | 0.537 |
| -O | | | | | | | | | | | | 0.519 | 0.5 | 0.5 | 0.5 | 0.583 | 0.51 | 0.536 | 0.5 | 0.5 | 0.5 | 0.5 | 0.517 |
| -o | | | | | | | | | | | | | 0.538 | 0.538 | 0.5 | 0.538 | 0.556 | 0.605 | 0.519 | 0.547 | 0.5 | 0.56 | **0.57** |
| -B | | | | | | | | | | | | | | 0.625 | 0.5 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | **0.857** | 0.509 |
| -R | | | | | | | | | | | | | | | 0.833 | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | **0.706** | 0.507 |
| -M | | | | | | | | | | | | | | | | 0.5 | 0.503 | 0.5 | 0.5 | 0.5 | 0.5 | **0.6** | 0.501 |
| vfstype | | | | | | | | | | | | | | | | | 0.517 | 0.577 | 0.5 | 0.5 | 0.5 | 0.5 | 0.509 |
| device | | | | | | | | | | | | | | | | | | 0.6 | 0.524 | 0.547 | 0.503 | 0.503 | 0.567 |
| dir | | | | | | | | | | | | | | | | | | | 0.55 | 0.526 | 0.5 | 0.5 | 0.519 |
| uuid | | | | | | | | | | | | | | | | | | | | 0.909 | 0.5 | 0.5 | 0.507 |
| label | | | | | | | | | | | | | | | | | | | | | 0.5 | 0.5 | 0.524 |
| num | | | | | | | | | | | | | | | | | | | | | | 0.5 | 0.501 |
| dirs | | | | | | | | | | | | | | | | | | | | | | | 0.509 |

TABLE VII: Experimental results in Exp 2.

| Extraction rate | Top 5% | Top 10% | Top 15% |
|---|---|---|---|
| Extracted combinations | 23 | 44 | 69 |
| Precision | 0.55 | 0.23 | 0.15 |
| Recall | 0.73 | 0.91 | 1.00 |

parameter combinations that probably cause constraints. Our experimental results demonstrated that our metric helps us extract valid parameter combinations when we analyze specification documents for the SUT. Such parameter combination extraction also helps other activities in the test design, such as test suite reduction or the identification of a subset of parameters that require higher coverage strength in variable strength interaction testing.

The results presented in this paper indicate some possible directions of further work and improvements. There are two major directions for improving the elicitation mechanism to identify more precise parameter combinations. First one is to develop an additional mechanism to improve the precision rate. The second direction is to define guidelines for applying the metric, which include the criteria for defining thresholds and word groups.

## Acknowledgments

This work was supported by JSPS Grants-in-Aid for Scientific Research (Grant Numbers 15K00097, 15K00098).

## References

[1] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in *Proc. of the 21st International Conference on Software Engineering (ICSE'99)*, ser. ICSE '99. ACM, 1999, pp. 285–294.

[2] C. Nie and H. Leung, "A survey of combinatorial testing," *ACM Computing Surveys (CSUR)*, vol. 43, no. 2, pp. 11:1–11:29, Feb. 2011.

[3] M. Grindal, J. Offutt, and J. Mellin, "Managing conflicts when using combination strategies to test software," in *Proc. of the 18th Australian Software Engineering Conference 2007 (ASWEC'07)*, April 2007, pp. 255–264.

[4] M. B. Cohen, M. B. Dwyer, and J. Shi, "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach," *IEEE Transactions on Software Engineering*, vol. 34, no. 5, pp. 633–650, Sep. 2008.

[5] H. Nakagawa and T. Tsuchiya, "A search-based constraint elicitation in test design," *IEICE Transactions on Information and Systems*, vol. E99-D, no. 9, pp. 2229–2238, Sep. 2016. [Online]. Available: https://doi.org/10.1587/transinf.2015KBP0010

[6] Atlassian, "Confluence," https://www.atlassian.com/software/confluence/.

[7] ——, "Documentation for confluence 5.9," https://confluence.atlassian.com/alldoc/confluuence-documentation-directory-12877996.html.

[8] T. F. Foundation, "FreeBSD Man Pages," https://www.freebsd.org/cgi/man.cgi.

[9] P. J. Schroeder and B. Korel, "Black-box test reduction using input-output analysis," in *Proc. of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '00)*. ACM, 2000, pp. 173–177.

[10] M. Cohen, P. Gibbons, W. Mugridge, C. Colbourn, and J. Collofello, "A variable strength interaction testing of components," in *Proc. of the 27th Annual International Computer Software and Applications Conference (COMPSAC 2003)*, Nov 2003, pp. 413–418.

[11] M. Gibiec, A. Czauderna, and J. Cleland-Huang, "Towards mining replacement queries for hard-to-retrieve traces," in *Proc. of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10)*. ACM, 2010, pp. 245–254.

[12] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. of the 26th International Conference on Neural Information Processing Systems (NIPS'13) Volume 2*. Curran Associates Inc., 2013, pp. 3111–3119.

[13] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *CoRR*, 2013, https://arxiv.org/abs/1301.3781.

[14] D. Blue, I. Segall, R. Tzoref-Brill, and A. Zlotnick, "Interaction-based test-suite minimization," in *Proc. of the 2013 International Conference on Software Engineering (ICSE 2013)*. IEEE Press, 2013, pp. 182–191.

[15] G. Salton and M. J. McGill, *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1983.

[16] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis," in *Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI'07)*. Morgan Kaufmann Publishers Inc., 2007, pp. 1606–1611.

[17] D. Falessi, G. Cantone, and G. Canfora, "Empirical principles and an industrial case study in retrieving equivalent requirements via natural language processing techniques," *IEEE Transactions on Software Engineering*, vol. 39, no. 1, pp. 18–44, Jan. 2013.