A Test Prioritization Method for Configurable Software Systems based on Variability Mining

TOMOJI KISHI[†1] TAIKI KOYAMA[†2] NATSUKO NODA[†3]
KEISUKE HORIUCHI[†1] KENSHO LI[†1] CHAOQUN ZHANG[†1]

Abstract: Testing configurable software systems such as IoT systems is challenging since these systems have variability and could have combinatorial number of configurations. To reduce the testing costs, test prioritization is a promising approach. There are proposed test prioritization methods for variability-intensive systems. In such methods variability model such as FM (Feature Model) and FTS (Featured Transition System) are given. However, if systems are configured with devices from other manufactures, the variability model is not always known in advance. In this paper we propose a test prioritization method based on variability mining. In our method, variability information and statistical usage information are extracted from BT (Bluetooth) communication logs between the target system and the connected device and prioritize tests utilizing them. Our experiment shows that the method effectively prioritizes test cases derived from the STM (state transition model) of the target system.

Keywords: Testing, Test Prioritization, Variability, Variability Mining

1. Introduction

Variability referrers to functional and non-functional characteristics that may differ among products[12]. This concept is important for not only software product lines (SPLs), but also configurable software systems such as operating systems[1] and IoT systems[5]. For example, if we consider an audio system that can be connected to BT (Bluetooth) speakers from several manufacturers, the capabilities of each BT speaker can be different and therefore used in different ways depending on how they are configured. Namely, there are variabilities among possible configurations.

Testing configurable software systems is challenging since these systems could have combinatorial number of configurations in the number of variable characteristics. To reduce testing costs, several testing methods have been proposed[16]. One of these methods is statistical test prioritization method based on usage model[7]. Usage models are typically given as Markov chains to show how often/rare system behaviors occur. Based on usage models, priorities of tests are determined. In these methods, variability models such as FM (feature model) and FTS (featured transition system) are used to avoid generating test cases that are not possible in terms of variability[7][19].

In this paper, we propose a test prioritization method based on variability mining. Variability mining is a technique to extract variability information from some artifacts or data. In our method, we capture BT communication logs between the system and connected devices, extract events (each of them is the reception of BT command defined in a BT profile), and then determine transition sequences. We obtain multiple transition sequences from different system-device configurations, and extract variability information using Haslinger's algorithm[10]. Extracted variability information includes common transitions that appear in all configurations, exclusive transitions that do not appear in the same configurations, and so on. We also obtain statistical information from communication logs, and prioritize tests based utilizing mined information.

In previous prioritization methods[7][17][19], variability information is assumed to be given; typically, developers construct variability model such as FM and FTS prior to the testing. However, if we consider systems that can be connected with devices from various manufacturers, it may be difficult to determine variabilities among various configurations in advance. In such situation, mining variability information from actual usage data would be preferable.

Note that it is difficult to mine the similar variability information as those defined by developers. Variability information obtained by variability mining tends to be at lower abstraction level. Nevertheless, our experiment shows that we can effectively prioritize tests using mined variability information.

In section 2, we introduce related works, and in chapter 3 explains important concepts. In section 4, we denote the problem setting and research questions. In section 5, we propose a variability mining method for test prioritization, and .in section 6, we explain how to extract statistical information. In section 7, we evaluate the effectiveness of test prioritization based on mined variability information. In section 8, we make technical discussions, and section 9 concludes the paper.

2. Related Works

In this section, we introduce related works.

2.1 Testing Variability-intensive Systems

Testing variability-intensive systems such as configurable software systems is challenging because they could have combinatorial number of configurations in the number of variable characteristics. As it is unrealistic to test all possible configurations, various testing methods are proposed to reduce the costs[11][16][17]. One approach is to select a subset of products (configurations) as representative products. SPL

†1 Waseda University.
†2 NS Solutions Corporation
†3 Shibaura Institute of Technology

pairwise testing is a such method in which representative products are selected so as to all possible feature pairs are included[15]. Other approach is test prioritization, i.e., to determine the order of tests to increase the early faut detection rate. For example, Al-Hajjaji et al. propose a test prioritization method utilizing similarity among products [9].

Some testing methods use usage models for prioritization. Usage models are typically given as Markov chains to show how often/rare system behaviors occur. Devroey et al. propose a method in which usage model is used along with FM and FTS to statistically select test sequences on FTS, and then, determine representative products and their priorities[7]. Samih et al. combine MBT (Model Based Testing) tools with OVM (orthogonal variability model)[18] to generate usage model variants from given set of features[19].

Though most previous methods assume that variability models are given, our method extract variability information from usage data and prioritize test utilizing the extracted information. We believe that our approach is suitable for the situation in which variability among configurations is not known in advance.

2.2 Variability Mining

Variability mining is a technique to extract variability information from some artifacts or data. In SPL field, variability mining is typically used to construct core assets (reusable assets) from existing set of similar software artifact and is also referred as asset mining[2]. There are proposed various techniques and tools that extract variability information from artifacts such as source code[13][20] and OS configuration file[8].

Haslinger et al. propose a reverse engineering algorithm that construct FM from given Feature Set Table [10]. Feature Set Table includes a set of valid feature configurations. This algorithm identifies relations among features such as implication and mutex, and construct FM based on these relations. In our method, we utilize this algorithm to extract variability information of transitions among multiple system-device configurations. Namely, we apply the algorithm to identify variability among different behaviors instead of static feature configurations.

As for variability mining for behavioral aspects, Seidl et al. propose a method to construct state transition model that expresses all possible behaviors of family of products from set of cloned model variants[21]. Though this work extract variability information from model variants, our method extracts information from communication logs.

3. Background

3.1 Feature Model (FM)

Features are any prominent and distinctive concepts or characteristics that are visible to various stakeholders, and FM depicts features in a SPL and constraints among them[14]. There are several notations, but constraints typically include mandatory, optional, OR, XOR, require (imply) and exclude (mutex). A feature configuration is called valid, if features in the feature configuration satisfy the constraints defined in the FM. A valid configuration corresponds to a product in the SPL.

Fig. 1 shows an example of FM. Alphabets below feature names are added to facilitate the description of FTS explained shortly. This FM has multiple valid configurations such as (v, p, s, c) and (v, f).

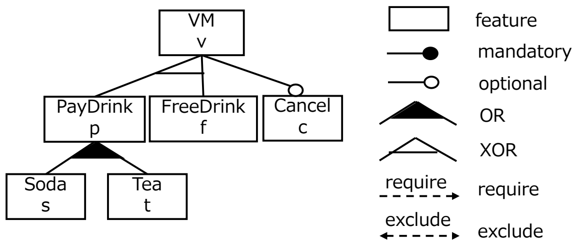

Fig. 1 An Example of FM (modified from [6])

3.2 Feature Set Table (FS Table)

FS Table is a table to indicate a set of products and their feature configurations. Each row corresponds to a product, and each column corresponds to a feature. Selected features are shown as '1' and unselected features are shown as '0'. FS Table includes a subset of all possible feature configurations, i.e., it shows a sample. Table. 1 is an example of FS Table that indicates some feature configurations derived from the FM (Fig. 1).

Table. 1 An Example of FS Table

.	v	p	s	t	f	c
p1	1	1	1	0	0	0
p2	1	1	1	1	0	1
p3	1	0	0	0	1	0
p4	1	1	0	1	0	1

3.3 Featured Transition System (FTS)

FTSs represent the behavior of possible instances of variability-intensive systems[6]. An FTS defines a set of states and transitions among them. Each transition has an action associated with it, and one of states is specified as the initial state. As an FTS represents the behavior of possible instances, each transition defined in the FTS must appear as the behavior of at least one possible instance of the system.

A feature expression, i.e., a Boolean expression over the feature defined in the FM may be attached to a transition. Given the feature configuration of an instance, features included in the configuration are interpreted as true, otherwise false. Only transitions whose feature expression is evaluated as true are enabled for the configuration (instance). Transitions without feature expressions are always enabled. In this manner, the behavior of a system instance is determined.

Fig. 2 is an example of FTS. Feature expressions are given after actions with '/' as delimiter. This FTS is based on the FM (Fig. 1) and feature expressions refers features defined in the FM. For example, for a valid feature configuration (v, p, s, c), transitions with trigger "pay", "cancel". "soda", and "serve" are enabled (Fig. 3 (a)). For a valid configuration (v, f), transitions with trigger "free", "water" and "serve" are enabled (Fig. 3 (b)).

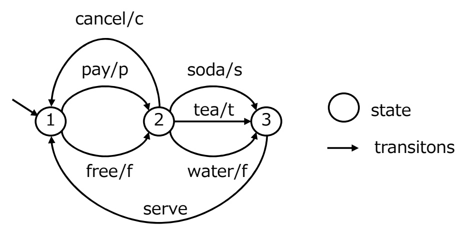

Fig. 2 An Example of FTS (modified from [6])

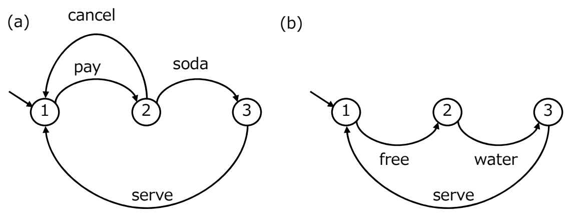

Fig. 3 Examples of System Instance's Behavior

4. Problem Statement

In this section, we introduce an example system, explain our problem settings, and then present our research questions.

4.1 Music Player Example

We use a music player that can be connected with BT speakers as an example system. There are three types of BT speakers, from different manufacturer, say A, B and C. The music player can be connected to one device at a time. We call the music player as the target system, and BT speakers as devices.

BT communication between the target system and devices uses AVRCP[3] that is a BT profile for audio/video remote control. In the profile, commands for audio/video remote control are defined. Among various commands, we focus on four commands "play", "pause", "forward" and "backward".

Fig. 4 shows the behavior of the target system defined as state transition model (STM) with states, transitions, an initial state, and triggers. Triggers correspond to receptions of AVRCP command mentioned above.

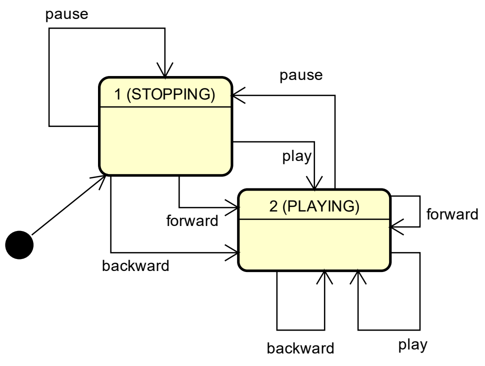

Fig. 4 STM of the target system

Each device has different functionality and may communicate with the target system using different set of commands in different sequences. Consequently, there could be common transitions that appear in all configurations and exclusive transitions that do not appear in the same configurations. Namely, there is variability among possible configurations.

4.2 Test Prioritization

We consider statistical prioritization of test for the target system. Concretely speaking, we generate (abstract) test cases from the STM and prioritize them based on the usage model.

Test prioritization is to determine the best order in which these test cases are executed. In case testing resources are limited, it would be beneficial to order test cases to make high-priority test cases appear earlier. Devroey determines priorities of paths (transition sequences) based on usage model, in which the priority of each path is given as the product of probabilities of transitions included in the path[7]. In this paper, we also use the same priority.

When we test variability-intensive system, we need to take variability information into account. For example, the vending machine introduced in section 3 does not have paths such as (free, soda) and (pay, water), because feature "free" (f) and "pay" (p) are exclusive. If we simply give priorities to paths using the method explained above, we may give higher priority to paths that are not actually exist. To avoid such inadequate prioritization, we need to exclude paths that are not possible in terms of variability.

4.3 Variability Mining for Prioritization

In previous prioritization methods, variability information is assumed to be given; Typically, developers construct variability model such as FM and FTS prior to the testing [5][6][7]. However, in our settings, the target system can be connected with devices from various manufacturers, and it is difficult to determine variabilities among various configurations in advance. Furthermore, even if developers suppose typical usage in advance, it does not necessarily mean that the system will actually be used in that way.

We assume the situation in which the STM of the target system is known but variability information is not known. We collect actual usage data and extract variability information using variability mining technique. Concretely speaking, we extract variability information from BT communication logs between the target system and connected devices.

4.4 Research Questions

In this paper, we address the following research questions:

- RQ1: How to mine variability information from communication logs between the target system and devices?
- RQ2: Whether the mined variability information can be used for effective test prioritization?

For RQ1, we propose a variability mining method (section 5), and for RQ2, we evaluate the effectiveness of test prioritization based on mined variability information (section 7).

5. Variability Mining Method.

In this section we propose a variability mining method that construct FM and FTS from BT communication logs.

5.1 Preliminaries

We assume that STM of the target system is known, and the STM is finite and deterministic. Triggers defined in the STM are events of receiving commands from the connected device. In our example, commands are four AVCRP commands (play, pause, forward and backward).

Prior to variability mining, we need to prepare communication logs between the target system and connected device. We use Wireshark[23] to capture BT packets. Each log corresponds to a specific system-device configuration. There may be multiple logs that correspond to the same configuration. At the start of logging, the target system should be in its initial state.

From each communication log, we construct an Event List (EL), that is a list of AVRCP commands received by the target system sorted by the time they were received. Though communication logs include various AVRCP commands, we only extract commands that appear as triggers in the STM. This process simply extracts specific rows and columns from the Wireshark's data, so we will not go into details. Each EL has a label to indicates the corresponding configuration.

5.2 Overview

Input of our method is STM of the target system and a set of ELs explained above. In our example, each EL is constructed from a communication log between the target system and one of devices, i.e., device A, device B or device C.

Output of our variability mining method is FM and FTS for the target system.

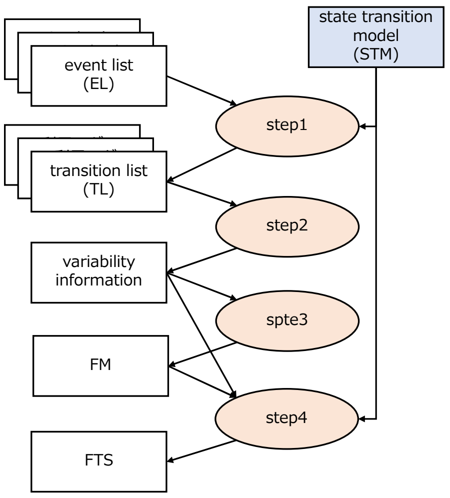

Fig. 5 Method Overview

Fig. 5 shows the overview of our method. The method consists of the following steps:

Step1: Construct Transition Lists
Step2: Extract Variability Information
Step3: Construct FM
Step4: Construct FTS

We will explain these steps in the following subsections.

5.3 Step1: Construct Transition List

For each EL, construct a Transition List (TL). Since events in EL are triggers of STM, the event sequence in an EL causes a transition sequence of the STM. TL is an ordered list of transitions. Each transition is specified by a pair of a state and a trigger where the state is the transition's source state, and the event is an event (trigger) attached to the transition.

As mentioned above, we assume that the target system is in its initial state at the start of logging. Therefore, the first transition of TL can be obtained by pairing the initial state with the first event in EL. The next transition of TL is obtained by pairing the destination state of the first transition and the next event in EL. TL can be constructed by repeating this operation until the end of EL. For example, in the example system, from a EL (play, play, pause), we can obtain a TL (<1, play>, <2, play>, <2, pause>).

To indicate the corresponding configuration, the TL is labeled with the same label as the corresponding EL.

5.4 Step2: Extract Variability Information

We extract variability information using the following procedure.

5.4.1 Construct FS Table

From TLs constructed in Step 1, construct a FS Table. Though FS Table is originally used to indicate the features included in each product, it is used to indicate the transitions appeared in each system-device configuration. Each row of FS Table corresponds to a system-device configuration, and each column corresponds to a transition defined in the STM of the target system. If a transition appears in a configuration, the corresponding cell is marked as '1', otherwise '0'.

As each TL has a label that indicates the corresponding configuration, the FS Table can be constructed by checking each TL to identify transitions included in it. In case there are two or more TLs for the same configuration, we simply accumulate them, i.e., we assume that a transition is used in the configuration if the transition appears at least one of these TLs. Table. 2 shows an example of FS Table for the Target System.

Table. 2 FS Table for the Target System

	1 play	1 pau	1 fwd	1 bwd	2 play	2 pau	2 fwd	2 bwd
A	1	1	0	0	0	1	0	0
B	0	0	0	1	1	0	1	1
C	1	0	0	0	0	1	1	1

(pau: pause, fwd: forward, bwd: backward)

5.4.2 Apply Haslinger's Algorithm [10]

Haslinger et al. propose a reverse engineering algorithm that construct FM from a Feature Set Table. In the algorithm, various variability information are extracted from the FS Table. We apply the algorithm to the FT Table constructed in 5.4.1 to extract variability information among different configurations in terms of transitions. Namely, we assume that a system-device configuration and a transition as a product and feature respectively.

Firstly, we extract the following information by applying the algorithm.

- Common Features: a set of features that appear in all products.
- Atomic Sets: An atomic set is a set of features that always appear together in all products.

For the FS Table shown in Table. 2, we can obtain Common Features = {}, and Atomic Set = {(<1, play>, <2, pause>), (<1, backward>, <2, play>), (<2, forward>, <2, backward>)}.

Secondly, we extract the following relationships among features, also applying the algorithm.

- Implication Relationships: An implication relationship is a binary relation between two features f1 and f2 where the appearance of f1 implies the appearance of f2 in all products.
- Mutex Relationships: A mutex relationship is a binary relation between two features f1 and f2 where f1 and f2 do not appear together in all products.

If a feature in an atomic set has a relationship with other feature f, other features in the same atomic set also have relationship with f. To avoid extracting such duplicate relationships, select a representative feature for each atomic set, and then extract the relationship.

Suppose that we select <1, play>, <1, backward> and <2, forward> as representative features for atomic sets. Then, we can extract one implication relationship (<1, pause>, <1, play>), and two mutex relationships (<1, pause>, <2, forward)>) and (<1, play>, <1, backward>).

5.5 Step3: Construct FM

Using the variability information extracted in Step 2, construct a FM. Generally, multiple FMs can be constructed from the same variability information. For example, implication relationship can be expressed using either an optional relationship or a require relationship.

Hence, we apply a simple strategy, as follows:

1. Put a root feature.
2. Put a feature representing common features as mandatory child of the root feature.
3. Put features each of which representing to an atomic set as optional children of the root feature.
4. Put other features as optional children of the root feature.
5. Define require relationships representing implication relationships.
6. Define exclude relationships representing mutex relationships.

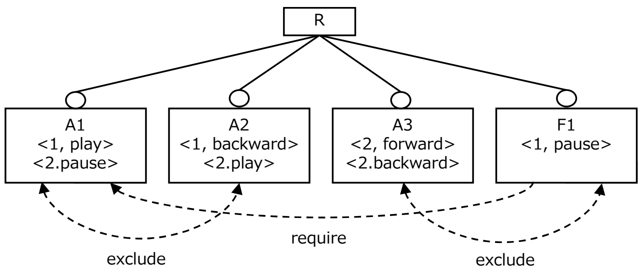

Fig. 6 FM for the Target System

Fig. 6 shows FM obtained by this procedure. Feature name R, A1, A2, A3 and F1 are mechanically given. In the Figure, corresponding transitions are written under feature names for convenience.

5.6 Step4: Construct FTS

Based on the FM constructed in Step 3, construct the FTS for the target system. This is done by attaching feature expressions to the STM of the target system. For each feature in the FM, identify corresponding transitions, and put the feature name as feature expressions of the transitions. If a transition has no corresponding feature, put "false" as expressions because the transition does not appear in any configurations.

Fig. 7 shows the FTS constructed from the STM (Fig. 4) and FM (Fig. 6).

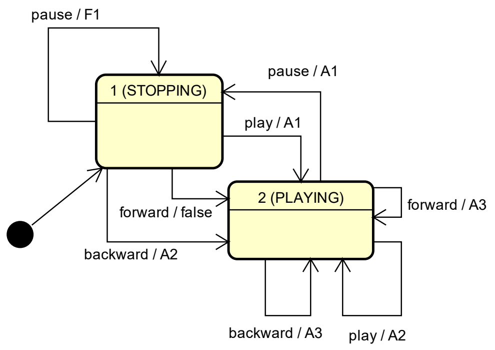

Fig. 7 FTS for the Target System

6. Extracting Statistical Information

For statistical prioritization, we need the usage model for the target system. We construct usage model from TLs constructed in Step 1. of our variability mining method. We count the number of occurrences of each transition in TLs, and calculate the probabilities based on that.

Suppose that the numbers of occurrence of transitions are as the second column of the Table. 3. Probability of each transition can be calculated by dividing the number of occurrences by the total number of occurrences of transitions that originate from the same state. For example, the probability of <1, play> can be calculated by 11/(11+1+1+0) since there are four transitions originated from the sate 1. Likewise, other probabilities can be calculated. The third column shows probabilities of the transition to two decimal places.

Table. 3 Probability of Each Transition

transition	N of occurrences	probability
<1, play>	11	0.85
<1, pause>	1	0.08
<1, forward>	1	0.08
<1, backward>	0	0.00
<2, play>	10	0.29
<2, pause>	10	0.29
<2, forward>	6	0.17
<2, backward>	9	0.26

Fig. 8 shows the usage model of the target system obtained by attaching probabilities to transitions of the STM. The probability of each transition is described after the trigger, enclosed in '[' and ']'.

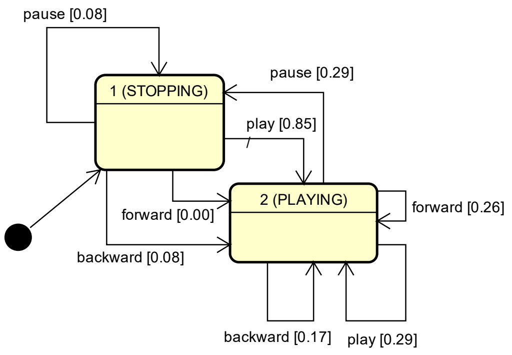

Fig. 8 Usage Model of the Target System

7. An Experiment: Test Prioritization

In this section, we introduce an experiment of test prioritization based on mined variability information. Though constructed FTS, FM and usage model could be used in various testing methods such as prioritization of representative products[7], we use them for prioritization for n-switch testing. Namely, we generate test cases from STM based on n-switch coverage criterion, and then prioritize them.

7.1 Objective

The objective of the experiment is to evaluate the effectiveness of test prioritization based on extracted variability information (section 5) and statistical information (section 6).

7.2 Method

7.2.1 construct FM, FTS and usage model

We applied our variability mining method to the example system to generate FM, FTS and usage model. We implement the program for Step 1 and Step 2 of our variability mining method explained in section 5. Step 3 is done manually, and for Step 4, we did similar process on spread sheet. We also make program to calculate probabilities.

7.2.2 test case generation

We generate (abstract) test cases from the STM using n-switch coverage criterion [4]. Each test case corresponds to a sequence of transitions defined in the STM. To satisfy the criterion, we generate test cases to cover n-switch set that consists of all transition sequences of length n+1. For the target system's STM (Fig. 4), 1-switch set includes test cases such as (<1, play>, <2, play>) and (<2, pause, 1, pause>), and 2-switch set includes test cases such as (<1, play>, <2, play>, <2, forward>) and (<2, pause>, <1, pause>, <1, play>). For this STM, 1-switch set contains 32 test cases, and 2-swithc set contains 128 test cases.

7.2.3 test case prioritization

We determine the order of test cases to important test cases appear earlier. We determine the weight (importance) of each test case (transition sequence) using the same method as [7], in which the weights are calculated as the product of probabilities of transition included in the test case.

However, as explained in 4.2, for variability-intensive systems, it is important to consider variability information. Here, we make the weight of a test case to be 0, if the test case contains mutually exclusive transitions, because such transition sequence is not possible. After that, we order test cases to make test cases with higher weights come earlier.

7.2.4 comparison

To evaluate the effectiveness of our prioritization method, we compare the following four prioritization method.

- PM1 (Proposed): This method is our method explained in 7.2.3 that uses both variability information and statistical information defined in usage model.
- PM2 (Usage only): This method is to order test cases to make higher weights come earlier without considering variability information.
- PM3 (Variability only): This method removes test cases that include mutually exclusive transitions, and then randomly order other test cases (removed test cases are attached at the end.
- PM4 (Random): This method determines the order randomly.

7.2.5 metrics

Accumulate the weights of the test cases according to the determined order. The firster the accumulated weights increase, the order is better, because this means important test cases appear earlier. We can examine it by plotting accumulated weights. Fig. 9 shows an example of accumulation graph. Here, the horizontal axis shows the number of test cases, and the vertical axis shows the accumulated weights. The weights are normalized to the sum of all weights becomes 1.

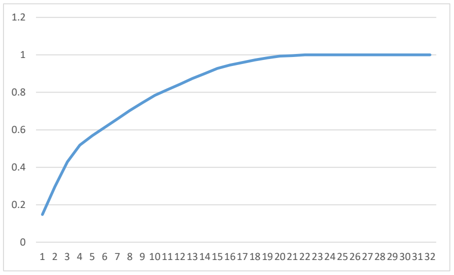

Fig. 9 An Accumulation Graph

A similar comparison can be made with a quantitative measure called exponential decay [22]. Fig. 10 shows a decay graph corresponding to Fig. 9. Here the vertical axis shows sum of weights of untested test cases in logarithm scale. A dotted line is exponential trendline expressed as $e^{-\lambda t}$. The slope of exponential decay is indicated by λ; the larger number indicates the faster decay, i.e., better prioritization. R^2 is coefficient of determination: higher value shows good correlation between actual values and regression model.

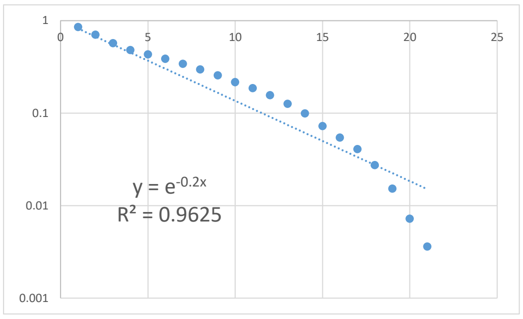

Fig. 10 A Decay Graph

We use accumulation graphs and exponential decay as metrics.

7.3 Comparison

We generate test cases from the STM of example system based on 1-switch criterion and 2-switch criterion using a commercial tool. Then, prioritize test cases using the four methods explained in 7.2.4. and compare four methods in terms of accumulation graphs and exponential decay. Since PM3 and PM4 includes randomness, we use the average of 10 times to make comparison.

7.3.1 1-switch Coverage

We prioritize 32 test cases generated by 1-switch coverage criterion by four methods. Fig. 11 shows the comparison using accumulation graphs. Also, Table. 4 shows the comparison using exponential decay.

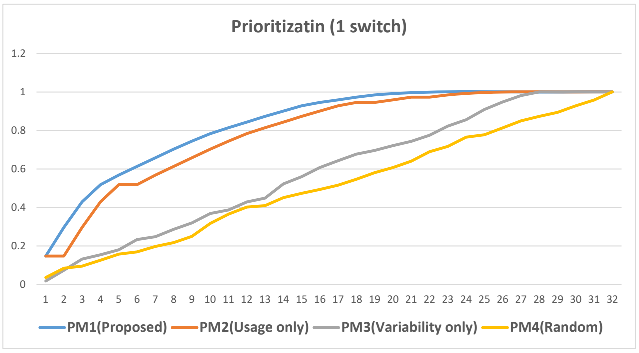

Fig. 11 Results for 1-switch Coverage (Accumulation Graphs)

Table. 4 Results for 1-switch Coverage (Exponential Decay)

	PM1	PM2	PM3	PM4
λ	0.2	0.194	0.078	0.073
R^2	0.9625	0.9229	0.921	0.8961

Both comparisons show that the PM1 is the best, followed by PM2, PM3 and PM4.

7.3.2 2-switch Coverage

We prioritize 128 test cases generated by 2-switch coverage criterion by four methods. Fig. 12 shows the comparison using accumulation graphs. Also, Table. 5Table. 4 shows the comparison using exponential decay.

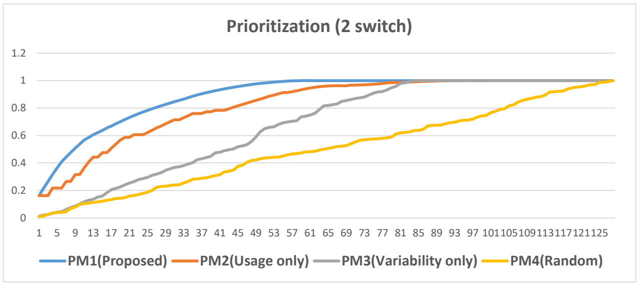

Fig. 12 Results for 2-switch Coverage (Accumulation Graphs)

Table. 5 Results for 2-switch Coverage (Exponential Decay)

	PM1	PM2	PM3	PM4
λ	0.092	0.066	0.025	0.018
R^2	0.8669	0.9102	0.8946	0.8668

As same as 1-switch Coverage, both comparisons show that the PM1 is the best, followed by PM2, PM3 and PM4.

7.4 Results

In this experiment, our proposed method PM1 was found to be the best prioritization method. In both cases (1-switch criterion and 2-switch criterion), PM1 shows the best result.

The second is PM2, that order test cases using statistical information without considering variability information. For 2-switch coverage, difference between PM1 and PM2 are bigger than that for 1-swithc coverage. This is because, the longer the test cases, more chance to include mutually exclusive transitions, and effect of removing them becomes larger.

Compared with PM1 and PM2, PM3 and PM4 are less effective. However, for 2-switch coverage, difference between PM3 and PM4 becomes larger because of the same reason explained above. Therefore, when the test case is long, considering only the variability information is also expected to have some effectiveness.

8. Discussion

In this paper, we proposed a variability mining method to extract the variability information among multiple device-system configurations. Our contributions are two folds, to propose a variability mining method from BT communication logs and demonstrate the effectiveness of test prioritization based on mined variability information.

Since we extract variability information from data, the validity of the data is the essential issue. There are various factors that affect the extracted variability information such as functionalities of devices, user's operation. Also, it is difficult to determine if the data is sufficient or not. How to prepare the data is an issue for the future.

It is not new to obtain probabilities from event logs to construct usage models. However, it should be noted that there are two aspects to variability-intensive systems, one is the probability of user operation, and the other is the probability of the occurrence of the product (configuration). The latter is specific to variability-intensive systems. As there could be combinatorial number of configurations, we cannot use all of them, and pick up some configurations from them. Namely, the probabilities of the product occurrence become a matter. How to handle these two aspects of probability is also the future issue.

We evaluated the effectiveness of our method by an experiment. Though this is one of typical experiment methods for test prioritization, it is necessary to further examine the validity of the experiment. As described above, the validity of the data is an issue, and we need to evaluate using different data. It is also desirable to apply methods other than the n-switch testing.

9. Concluding Remarks

In this paper, we proposed a test prioritization method for configurable systems. We believe that testing based on variability mining is effective in situations where variability is not known in advance. We would like to develop the method for more diverse situations in the future.

Reference

[1] T. Berger, S. She, R. Lotufo, A Wąsowski and K. Czarnecki: Variability Modeling in the Real: A Perspective form the Operating System Domain, in Proc. of Automated Software Engineering 2010 (ASE'10), pp.73-82, (2010).

[2] J. Bergey, L. O'Brian and D. Smith: Mining Existing Assets for Software Product Lines, Technical Report CMU/SEI-2000-TN-008, Software Eng. Inst., (2000).

[3] Bluetooth SIG: Audio/Video Remote Control, Bluetooth Profile, Specification, v1.6.2. (2019).

[4] T.S. Chow: Testing Software Design Modeled by Finite-State Machines, IEEE Transactions on Software Engineering, Vol.SE-4, No.3, pp.178-187, (1978).

[5] L. Chumpitaz, A. Furda and S. Loke: Evolving Variability Requirements of IoT Systems, Chapter 14, Software Engineering for Variability Intensive Systems, CRC Press, (2019).

[6] A. Classen, M. Cordy, Pierre-Yves Schobbens, P. Heymans, A. Legay and Jean-Francois Raskin: Featured Transition Systems: Foundations for verifying Variability-Intensive Systems and Their Application to LTL Model Checking, IEEE Trans. on Software Engineering, Vol.39, No.8, pp.1069-1089, (2013).

[7] X. Devroey, G. Perrouin., M. Cordy, H. Samih, A. Legay, P. Schobbens and P. Heymans: Statistical prioritization for software product line testing: an experience report, Software and Systems Modelling, Volume 16, Issue 1, pp.153-171, (2017).

[8] D. Fernandez-Amoros, R. Heradio, C. Mayr-Dorn and A. Egyed: A Kconfig Traslation to Logic with One-Way Validation System, in Proc. of the 23rd System and Software Product-Line Conference (SPLC'19), pp.303-308, (2019).

[9] M. Al-Hajjaji, T. Thun, J. Meinicke, M. Lochau and G. Saake: Similarity-Based Prioritization in Software Product Line Testing, in Proc. of Software Product Line Conference (SPLC), pp.197-206, (2014).

[10] E. N. Haslinger, R. Erick L. Herrejon and A. Egyed: On Extracting Feature Models from Sets of Valid Feature Combinations, in Proc. of the 16th International Conference on Fundamental Approaches to Software Engineering (FASE 2013), pp.53-67, (2013).

[11] A. Y. Hassan: A Survey on Software Product-Line Testing, International Journal of Advanced Research, 9(01), pp.90-96, (2020).

[12] ISO/IEC: ISO/IEC 26550, Reference Model for Product Line Engineering and Management, (2015).

[13] C. Kaestner, A. Dreiling and K. Ostermann: Variability Mining: Consistent Semi-automatic Detection of Product-Line Features, IEEE Transaction on Software Engineering, Vol.40, No.1, (2014).

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson: Feature-Oriented Domain Analysis (FODA) Feasibility Study, CMU/SEI-90-TR-21, ESD-90-TR-222, (1990).

[15] B. P. Lamancha and M. P. Usola: Testing Product Generation in Software Product Lines Using Pairwise for Features Coverage, in Proc. of the 22nd IFIP WG 6.1 International Conference on Testing Software and Systems (ICTSS'10), pp.111-125, (2010).

[16] I. do C. Machado, J. D. McGregor, Y. C. Cavalcanti and E. S. de Almeida: On Strategies for testing software product lines: A systematic literature review, Information and Software Technology, 56, pp1183-1199, (2014).

[17] K. L. Petry, E. OliveiraJr and A. F. Zorzo: Model-based testing of software product lines: Mapping study and research roadmap, The Journal of Systems and Software, vol.167, (2020).

[18] K. Pohl, G. Boeckle and F. van der Linden: Software Product Line Engineering - Foundation, Principles, and Techniques, Sprinter, (2005).

[19] H. Samih, M. Acher, R. Bogusch, H. Le Guen and B. Baudry: Deriving Usage Model Variants for Model-based Testing: An Industrial Case Study, in Proc. of 19th International Conference on Engineering of Complex Computer Systems (ICECCS), pp.77-80, (2014).

[20] T. Savage, M. Revelle and D. Poshyvanyk: FLAT3: Feature Location and Textual Tracing Tool, in Proc. of International Conference of Software Engineering (ICSE), pp. 255-258, (2010).

[21] C. Seidl, D. Wille and I. Schaefer: Software Reuse: From Cloned Variants to Managed Software Product Lines, in Automotive Systems and Software Engineering State of the Art and Future Trends (ed. Y. Dajsuren, M. van den Brand), Springer, (2019).

[22] I. Segall: Repeated Combinatorial Test Design – Unleashing the Potential in Multiple Testing Iterations, Proc. of International Conference on Software Testing, Verification and Validation, pp.12-21, (2016).

[23] Wireshark: https://www.wireshark.org/ (access 2021, Oct. 1st).