Using Semantic Similarity in Crawling-based Web Application Testing

Jun-Wei Lin
Dept. of Informatics
University of California, Irvine, USA
junwei.lin@uci.edu

Farn Wang
Dept. of Electrical Engineering
National Taiwan University, Taiwan
farn@ntu.edu.tw

Paul Chu
QNAP Inc.
paulchu@qnap.com

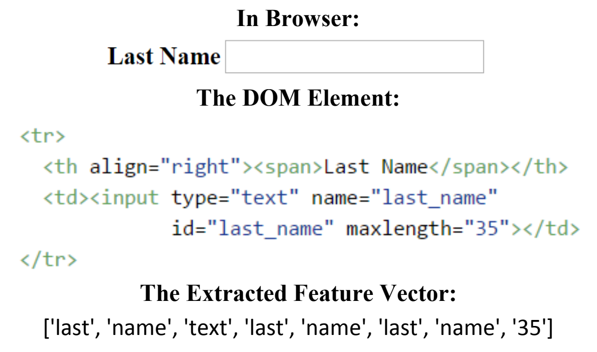
***Abstract*— To automatically test web applications, crawling-based techniques are usually adopted to mine the behavior models, explore the state spaces or detect the violated invariants of the applications. However, their broad use is limited by the required manual configurations for input value selection, GUI state comparison and clickable detection. In existing crawlers, the configurations are usually string-matching based rules looking for tags or attributes of DOM elements, and often application-specific. Moreover, in input topic identification, it can be difficult to determine which rule suggests a better match when several rules match an input field to more than one topic. This paper presents a natural-language approach based on semantic similarity to address the above issues. The proposed approach represents DOM elements as vectors in a vector space formed by the words used in the elements. The topics of encountered input fields during crawling can then be inferred by their similarities with ones in a labeled corpus. Semantic similarity can also be applied to suggest if a GUI state is newly discovered and a DOM element is clickable under an unsupervised learning paradigm. We evaluated the proposed approach in input topic identification with 100 real-world forms and GUI state comparison with real data from industry. Our evaluation shows that the proposed approach has comparable or better performance to the conventional techniques. Experiments in input topic identification also show that the accuracy of the rule-based approach can be improved by up to 22% when integrated with our approach.**

Keywords—Web application testing; semantic similarity; GUI testing

I. Introduction

Web applications nowadays play important roles in our financial, social and other daily activities. Testing modern web applications is challenging because their behaviors are determined by the interactions among programs written in different languages and running concurrently in the front-end and the back-end. To avoid dealing with these complex interactions separately, test engineers treat the application as a black-box and abstract the DOMs (Document Object Models) presented to the end-user in the browser as states. The behaviors of the application can then be modeled as a state transition diagram on which model-based testing can be conducted. Since manual state exploration is often labor-intensive and incomplete, crawling-based techniques [1], [5], [6], [7], [11], [19], [27], [28], [30], [33] are introduced to systematically and automatically explore the state spaces of web applications. Although such techniques automate the testing of complicated web applications to a great extent, their broad use is limited by the required manual configurations for applications under test (AUT). First, many web applications need specific input values to their input fields in order to access the pages and functions behind the current forms. To achieve proper coverage of the state space of the application, a user of existing crawlers has to set the rules for identifying the topics of encountered input fields in advance so as to feed appropriate input values at run time. Typical rules are string-matching based, mapping the DOM representations of input fields to their topics. For example, Fig. 1 illustrates an input field requesting a last name, a value of topic *last_name*. To identify the topic of the input field, the values of its attributes such *id* and *name* have to be compared with a feature string "last_name" and an appropriate value can then be determined by the identified topic. Because input values in different topics such as email, URL and password are necessary for a web page requesting them, the manual configuration has to be repeated.

Second, ability to properly distinguish new GUI states from explored ones is fundamental to a crawler. Web pages may have contents irrelevant to testing such as advertisements or current time. As a result, existing crawlers allow some abstraction mechanisms when determining the equivalence of GUI states by comparing their DOMs. A commonly adopted abstraction technique is DOM content filtering. Users can include or exclude the DOM elements with specific tags, attributes or enclosed text in state comparison. Similarly, the abstraction may not be effective for all applications and usually need to be tailored [30].


In Browser:

Last Name

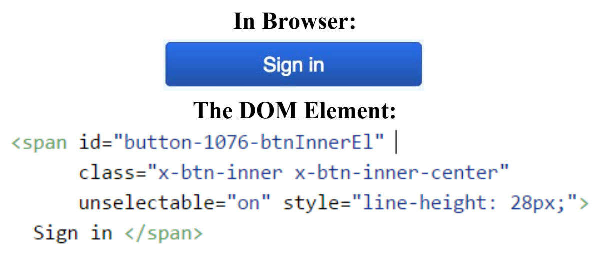
The DOM Element:

```
<tr>
  <th align="right"><span>Last Name</span></th>
  <td><input type="text" name="last_name"
             id="last_name" maxlength="35"></td>
</tr>
```

The Extracted Feature Vector:

['last', 'name', 'text', 'last', 'name', 'last', 'name', '35']

Fig. 1. An example input field requesting a value of the topic *last_name*



```
<span id="button-1076-btnInnerEl"
      class="x-btn-inner x-btn-inner-center"
      unselectable="on" style="line-height: 28px;">
  Sign in </span>
```

Fig. 2. An example sign-in button

Finally, clickables such as links and buttons are DOM elements a crawler interacts with to trigger events on or submit requests to the AUT. While some crawlers have preset rules for common clickables such as <a> or <button>, application-specific rules may be required. For example, Fig. 2 illustrates a sign-in button represented by <span> from a commercial application[1]. Because a rule to click all <span> is too general to be built in existing crawlers, a configuration checking the class attribute of <span> has to be manually set.

Another drawback of adopting string-matching based rules in crawling-based web application testing is that the rules are often application-specific, since the naming conventions for the values of attributes of DOM elements are diverse in web applications. Moreover, in input topic identification, it could be difficult to determine the topic of an encountered input field when it matches multiple rules for different topics. To address these issues, several observations suggest the possibility of using natural-language techniques:

- In markup languages like HTML and XML, the words used for attributes of DOM elements such as *id*, *name*, *type* and *maxlength* are extremely limited. Therefore, unlike traditional natural-language tasks such as sentimental analysis which need large corpora to build the dictionaries, the sizes of representative corpora for the tasks in crawling-based web application testing could be moderate.
- Human interacts with web applications through the text in natural language but not the DOM structures or attributes. Therefore, abundant natural language information in labels or attribute values can be leveraged to help choose appropriate actions for testing.
- While the words and sentences used for input fields of the same topic may be different among web applications, they are often semantically similar. For example, different websites may use "last name", "surname", "family name" or other related words to describe and name the input fields taking the user's last name.

Inspired by above observations, this paper presents a novel and natural-language technique to address the issues of rule-based approach used in crawling-based web application testing. For the interested DOM elements such as input fields, divisions or even whole web pages, we extract their feature vectors consisting of the words used in the attributes, the nearest labels or the enclosed text. An example feature vector is illustrated in Fig. 1. We then apply a series of transformations including Bag-of-words, Tf-idf (Term frequency with inverse document frequency) and LSI (Latent Semantic Indexing) [10] to these vectors to represent them with multi-dimensional, real-number vectors. Later, with a training corpus, for a new DOM element we can figure out its most similar vectors in the corpus by calculating the cosine similarities. This similarity information can then be used for input topic identification, state equivalence checking or clickable detection. In input topic identification, the proposed approach is under a supervised learning paradigm. That is, each feature vector in the corpus has to be labeled with a topic. A running example will be provided in Section III. In GUI state comparison and clickable detection, the proposed approach is under an unsupervised learning paradigm. In other words, we can determine whether a DOM element is equivalent to an existing state by its similarity with the vectors in the corpus without additional labeling. We believe that the proposed method can relieve the burden of constructing rules for unexplored web applications and enhance existing crawling-based techniques.

To evaluate the proposed approach, we conducted experiments on input topic identification and GUI state comparison, respectively. First, for input topic identification, we collected 100 real-world forms, and split them into training and testing data to validate the effectiveness. The experimental results show that when the proportion of training data increases, our approach performs comparably to the rule-based one. In addition, the accuracy of the rule-based approach is significantly improved by up to 22% when integrated with the proposed technique. Second, for GUI state comparison, we used the data from industry to evaluate the effectiveness of different abstraction mechanisms for state equivalence. The proposed approach outperformed other mechanisms in three of the five dataset, and had close performance to the best mechanisms for the rest.

The main contributions of this paper include:

- A novel technique using semantic similarity in crawling-based web application testing to address the limitations of the rule-based approach for input topic identification, GUI state comparison and clickable detection.
- The implementation and evaluation of the proposed approach in input topic identification and GUI state comparison. Experiments with 100 real-world forms and data from industry show promise for our approach. The source code and data of experiments on input topic identification are also publicly available[2] to make them reproducible.
- A discussion of usage scenarios of the proposed technique to help reduce the manual effort or increase the effectiveness of existing crawlers.

II. Background and Motivation

Today's web applications interact intensively with the users by dynamically changing the DOMs using client-side

[1] https://www.qnap.com/solution/qcenter/index.php?lang=en

[2] https://github.com/jwlin/icst2017

JavaScript. To capture the behaviors of such applications, crawling-based technique plays a significant role [34] in automated web application testing [1], [5], [6], [7], [11], [19], [27], [30], [33]. The technique analyzes the data and models collected from dynamic exploration of the applications. Although exhaustive crawling can cause state explosion problem in most industrial web applications that have huge state spaces, the navigational diversity of the crawling is important for deriving a test model with adequate functionality coverage [1], and achieving this diversity is still challenging [20], [28].

While there was a crawling technique ignoring text input [11], most existing crawlers [6], [19], [33] operate with randomly generated or user-specified data. To specify the data used in particular input fields, users have to provide feature strings (i.e., the values of the DOM attributes such as *id* or *name* of the input fields) in rules to identify the topics. For example, if we want "James" for an input field with *id* "lastName", we could add a configuration similar to the following when using a crawler:

```
input.id("lastName").setValue("James")
```

Implicitly, the configuration creates the following rules:

$$DOM.tag = input \land DOM.id.substrings \in \{"lastName"\} \\ \rightarrow last\_name(DOM) \ is \ true$$

$$last\_name(DOM) \ is \ true \rightarrow DOM.value \in \{"James"\}$$

Here {"lastName"} is a set of feature strings for the topic *last_name*. The feature strings in the set would be used to match all the substrings of the *id*, and the set may be manually expanded with more feature strings such as "surname" or "last" when more input fields of the same topic are encountered. {"James"} is the data set we prepare for input fields with the topic.

The string-matching based rules for input topic identification are widely used in existing crawlers. Nevertheless, the rules for one web application may not work for another. As a result, users may have to reconstruct or adjust the rules for new AUT. For instance, Table I shows input fields collected from four real-world forms. The first input field contains two attributes, *id* and *name*, both with values "firstName". To identify the input field and assign values used for it, the rule containing a feature string "first" is created to match the *id* or *name*. However, as illustrated, the rule derived from the first input field does not work for the second one which needs feature string "fn" in the rule. Moreover, both rules fail in identifying the third input field of the same topic, because the *id* and *name* look randomly generated. To address

TABLE I. EXAMPLE INPUT FIELDS AND RULES FOR IDENTIFYING THEIR TOPICS. STRING: FEATURE STRING.

Input Field	Topic	String
<input id="firstName" name="firstName">	first_name	"first"
<input id="aycreatefn" name="aycreatefn">	first_name	"fn"
<input id="textfield-1028-inputEl" name="1023000000003015">	first_name	(*)
<input id="permanenttel" name="permanenttel">	phone	"tel"
<input id="aycreateln" name="aycreateln">	last_name	"ln"

* To be discussed

this issue, our approach takes the nearest labels or descriptions of a DOM element into consideration. The intuition is that the nearest labels are likely the text about the input field for human to read, and if so, the text for the same topics of input fields is often semantically similar even in different websites. In fact, the third input field was successfully identified in our experiments.

Another issue of the rule-based approach for input topic identification is that it is difficult to determine the topic if there are multiple candidates. For example, after setting the rules containing the feature string "tel" for the fourth input field and "ln" for the fifth input field in Table I, the fifth input field will be categorized as *phone* and *last_name* simultaneously because both rules match the *id* "aycreateln". In contrast, the similarity information with the input fields in the corpus could help resolve this ambiguity, and it worked for the above example in our experiments.

Constructing accurate state transition models for testing is difficult because of the dynamism in AJAX web applications nowadays. To focus on only interested parts of the web pages when computing state equivalence, many abstraction techniques are proposed, and most of them are based on DOM content filtering [5], [6], [24], [29], [30]. That is, to eliminate specific nodes of DOM trees such as ones containing time stamps, invisible blocks or tags with particular attributes when examining the equivalence of two documents. These abstraction mechanisms are sometimes called oracle comparators [5], [6], [29] for users of a crawler to choose. For example, in the tool implemented in [5], users can use *DateOracleComparator* to ignore time stamps on web pages. As a result, two identical pages except for the different time stamps would be treated as the same GUI states. However, the state-equivalence notions are often application-specific and may need to be tailored [6], [30]. For instance, a test engineer probably pays attention on a particular window on top and intentionally ignores changes in the background even if they are visible. As a result, two web pages with different DOM structures might be considered the same GUI state. Instead of string-matching or DOM-checking, our approach addresses the issue by similarity with explored web pages (vectors in the corpus). We consider only text on web pages and reduce the effort of picking or implementing different DOM filters. Our evaluation shows the proposed approach has better or comparable performance to previous abstraction techniques in the literature.

Clickables such as links or buttons on web pages are important for crawling-based web application testing, because a crawler has to click them to interact with the AUT. Common clickables such as <a> and <button> can be detected and fired by a crawler following HTML specification [16], but as mentioned in Section I, manual configurations may be required to detect some uncommon clickables. Because the clickables are also DOM elements, our approach may be adopted to detect them.

In unsupervised document analysis, vector transformations such as Tf-idf and LSI are algorithms that project a text corpus to a vector space by examining the word statistical co-occurrence patterns [10]. The concept behind Tf-idf is that the words appearing frequently in a document and infrequently in other documents could be used to uniquely represent the document. Furthermore, LSI is used to reduce the rank of a word-document matrix by applying Singular Value Decomposition [25], a mathematical technique in linear algebra. Each dimension of the rank-reduced vector space hopefully represents a latent concept in the text. In this paper, we apply these transformations to feature vectors extracted from interested DOM elements, and measure how similar two vectors are by calculating the cosine similarity (i.e., the cosine of the angle).

III. Approach

An overview of our approach is depicted in Fig. 3. First, we extract feature vectors from the collected DOM elements to build a training corpus. Each vector in the corpus has to be labeled by its topic if the corpus is for input topic identification. However, we may label a group of vectors as the same topic at a time instead of one by one. The detail will be provided in Section III-C. Afterwards, for encountered DOM elements, we extract and transform them to project them into the vector space constructed by training data, and do inference based on their similarities to vectors in the training corpus. Here we provide a running example and detailed explanation about how to exploit the proposed technique on input topic identification. Once the topics are identified, the corresponding values can then be selected from a pre-established databank [28] or generated by data models such as smart profile [18]. The same concept can be applied to state comparison and clickable detection without significant changes.

A. *Feature Extraction*

A novelty of this paper comparing with other crawling-based techniques for web application testing is that we consider not only the attributes but also the nearby labels or descriptions of DOM elements for input topic identification. Algorithm 1 shows how it is achieved. First, we specify DOM attributes such as *id*, *name*, *placeholder* and *maxlength* which concern input topic identification in an attribute list, and the values of matched attributes of the DOM element will be put into the feature vector (line 2 to 4). Moreover, to find the corresponding descriptions, we search the siblings of the DOM element for tags such as span and label in a tag list and put the texts enclosed by the tags into the feature vector (line 11 to 18). If no such tags are found, the search will continue on the DOM's parent recursively for several times (line 20). In addition, we perform a couple of normalizations such as special character filtering and lowercase conversion to the words in the extracted feature vector.

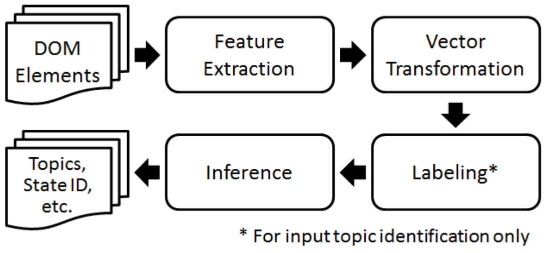


Fig. 3. An overview of the proposed approach

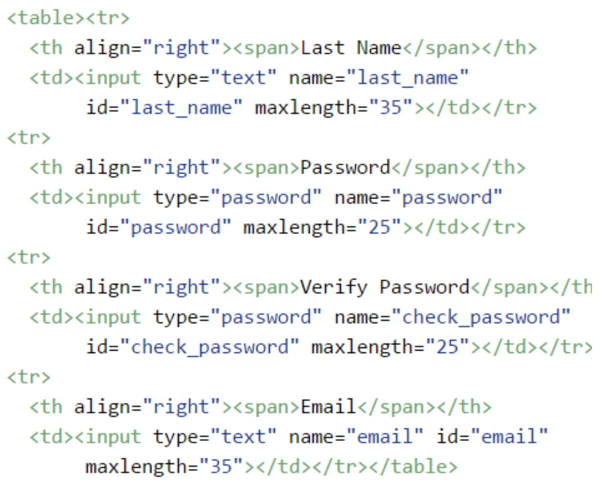
Algorithm 1 Extract Features

```
Input:
    A DOM element $D$;
    An attribute list $L$;
    Maximum number of iterations $MAX$;
    A tag list $T$;
Output:
    A feature vector $F$ extracted from $D$;

 1: $F = \varnothing$;
 2: for each $attribute \in D.attributes$ do
 3:     if $attribute \in L$ then
 4:         $F = F \cup \{D.getValue(attribute)\}$
 5: $F = F \cup findClosestLabels(D)$;

 6: function FINDCLOSESTLABELS(DOM element $D$, $iteration$=0)
 7:     if $iteration = MAX$ then
 8:         return $\varnothing$
 9:     else
10:         $labels = \varnothing$
11:         for each $sibling \in D.siblings$ do
12:             for each $tag\_name \in T$ do
13:                 $tags = sibling.find(tag\_name)$
14:                 if $tags \neq \varnothing$ then
15:                     for each $tag \in tags$ do
16:                         $labels = labels \cup tag.getText()$
17:         if $label \neq \varnothing$ then
18:             return $labels$;
19:         else
20:             return $findClosestLabels(D.parent, iteration + 1)$;
```

A running example is shown in Fig. 4. For the first input field in Fig. 4, the feature vector was first constructed with its values of attributes: "text", "last", "name", "last", "name" and "35" (line 2 to 4). Then in *findClosestLabels()*, because the input element has no siblings, the algorithm searched siblings of its parent (line 20). In the second iteration of the search, a <span> with text "Last Name" under <th> was found, and the words "last" and "name" were put into the feature vector. The feature vectors for the input fields in Fig. 4 are shown in Fig. 5.



```
<table><tr>
  <th align="right"><span>Last Name</span></th>
  <td><input type="text" name="last_name"
       id="last_name" maxlength="35"></td></tr>
<tr>
  <th align="right"><span>Password</span></th>
  <td><input type="password" name="password"
       id="password" maxlength="25"></td></tr>
<tr>
  <th align="right"><span>Verify Password</span></th
  <td><input type="password" name="check_password"
       id="check_password" maxlength="25"></td></tr>
<tr>
  <th align="right"><span>Email</span></th>
  <td><input type="text" name="email" id="email"
       maxlength="35"></td></tr></table>
```

Fig. 4. A running example

```
['last', 'name', 'text', 'last', 'name', 'last', 'name', '35']
['email', 'text', 'email', 'email', '35']
['password', 'password', 'password', 'password', '25']
['verify', 'password', 'password', 'check', 'password',
 'check', '25']
```

Fig. 5. The extracted feature vectors from the example in Fig. 4

B. Vector Transformation

After all DOM elements for training are represented as feature vectors in a corpus, three transformations are applied to the vectors sequentially: Bag-of-words, Tf-idf and LSI [10]. These transformations convert the vectors from words to multi-dimensional and real-number vectors, and each dimension of the vectors hopefully represents a latent concept consisting of the words in the corpus. In the following paragraphs we use document and feature vector interchangeably because they are identical in the context of this section.

First, Bag-of-words transformation is used to represent each document in natural language in a corpus as an integer vector based on its word counts, and the dimension of each vector is the number of distinct words (the "dictionary") in the corpus. For example, Table II shows that given the four documents in Fig. 5, there are nine distinct words in the documents. As a result, the documents can be represented by nine-dimensional vectors as shown in Fig. 6. We can see the third vector is [(5, 1), (6, 4)] because the document is ['password', 'password', 'password', 'password', '25'] in which the word with index 5 ("25") appears once and the word with index 6 ("password") appears four times. We use sparse representation, i.e. only (index, value) pairs for the dimensions with non-zero value, to make the following explanation clearer. Bag-of-words transformation is a simplified representation because it disregards grammar and word order in documents. Fortunately, for DOM elements in web applications we do not care about these two properties either.

Second, Tf-idf transformation converts the integer-value Bag-of-word vectors to real-value ones. Intuitively, if a word appears frequently in a document and infrequently in all other documents, the word could uniquely represent the document. Tf-idf assigns weights to words in documents based on this intuition. A common weighting scheme is: $f_{t,d} \times (\log_2)(N/n_t)$. $f_{t,d}$ is the frequency of the word t in document d, N is the number of all documents and n_t is the number of documents in which t appears. The Tf-idf representations of the documents in Fig. 5 are shown in Fig. 7. Taking the third document as an example, for the word "25" (index 5), the $f_{t,d}$ is 1 and n_t is 2 because it appears once in the document and twice in all four documents, so the Tf-idf is 1. For the word "password" (index 6) appearing four times in the document, the $f_{t,d}$ is 4 and n_t is 2, and the Tf-idf is 4. Therefore, we have [(5, 1), (6, 4)] as the Tf-idf representation of the third document. After normalizing the vector to unit length, we have [(5, 0.2425), (6, 0.9701)] (rounding off after the 4th place and so are the following values in matrices) as shown in Fig. 7.

TABLE II. THE DICTIONARY OF DOCUMENTS IN FIG. 5.

Word	Index
"text"	0
"last"	1
"name"	2
"35"	3
"email"	4
"25"	5
"password"	6
"verify"	7
"check"	8

```
[(0, 1), (1, 3), (2, 3), (3, 1)]
[(0, 1), (3, 1), (4, 3)]
[(5, 1), (6, 4)]
[(5, 1), (6, 4), (7, 1), (8, 2)]
```

Fig. 6. The Bag-of-word representations of the documents in Fig. 5

```
[(0, 0.1162), (1, 0.6975), (2, 0.6975), (3, 0.1162)]
[(0, 0.1622), (3, 0.1622), (4, 0.9733)]
[(5, 0.2425), (6, 0.9701)]
[(5, 0.1644), (6, 0.6576), (7, 0.3288), (8, 0.6576)]
```

Fig. 7. The Tf-idf representations of the documents in Fig. 5

Finally, LSI transformation tries to deal with the problem that different words used in the same context may have similar meanings. The transformation reduces the dimension of the vector space constructed by the words and documents in a corpus using a mathematical technique called Singular Value Decomposition (SVD) [25]. Each dimension of the rank-reduced vector space hopefully represents a latent concept contained in documents. To do LSI transformation, we first describe the documents as a term-document matrix X, and then decompose X by SVD as $X=U\Sigma V^T$. Here U and V^T are orthogonal matrices that could represent latent concepts in the documents and the coordinates of the documents in the latent vector space, respectively. Σ is a diagonal matrix that could state the importance of each latent concept. For example, after Bag-of-word and Tf-idf transformations, the documents in Fig. 5 can be written as a term-document matrix X as:

$$\begin{array}{c|cccc|} & d_1 & d_2 & d_3 & d_4 \\ \hline text & 0.1162 & 0.1622 & 0 & 0 \\ last & 0.6975 & 0 & 0 & 0 \\ name & 0.6975 & 0 & 0 & 0 \\ 35 & 0.1162 & 0.1622 & 0 & 0 \\ email & 0 & 0.9733 & 0 & 0 \\ 25 & 0 & 0 & 0.2425 & 0.1644 \\ passwoda & 0 & 0 & 0.9701 & 0.6576 \\ verify & 0 & 0 & 0 & 0.3288 \\ check & 0 & 0 & 0 & 0.6576 \end{array}$$

After decomposing X as $U\Sigma V^T$, U is

$$\begin{array}{c|cccc|} & c_1 & c_2 & c_3 & c_4 \\ \hline text & 0.0000 & -0.1933 & 0.0331 & 0.0000 \\ last & 0.0000 & -0.4842 & -0.5028 & 0.0000 \\ name & 0.0000 & -0.4842 & -0.5028 & 0.0000 \\ 35 & 0.0000 & -0.1933 & 0.0331 & 0.0000 \\ email & 0.0000 & -0.6756 & 0.7016 & 0.0000 \\ 25 & -0.2221 & 0.0000 & 0.0000 & 0.0973 \\ password & -0.8886 & 0.0000 & 0.0000 & 0.3894 \\ verify & -0.1795 & 0.0000 & 0.0000 & -0.4096 \\ check & -0.3590 & 0.0000 & 0.0000 & -0.8192 \end{array}$$

Each column in U could be interpreted as a latent concept in the documents. For example, the first concept c_1 in the corpus is:

$$-0.2221 \times \text{"25"} - 0.8886 \times \text{"password"} - 0.1795 \times \text{"verify"} - 0.3590 \times \text{"}check\text{"}$$

and so are the rest three concepts. Note that the latent concepts are justified on the mathematical level and probably have no interpretable meaning in natural language. Σ is a diagonal matrix containing singular values that could represent the importance of latent concepts in descending order. V^T is:

$$\begin{array}{c} \\ c_1 \\ c_2 \\ c_3 \\ c_4 \end{array}\begin{array}{c} \begin{array}{cccc} d_1 & d_2 & d_3 & d_4 \end{array} \\ \begin{pmatrix} 0 & 0 & -0.7071 & -0.7071 \\ -0.7071 & -0.7071 & 0 & 0 \\ -0.7071 & 0.7071 & 0 & 0 \\ 0 & 0 & 0.7071 & -0.7071 \end{pmatrix} \end{array}$$

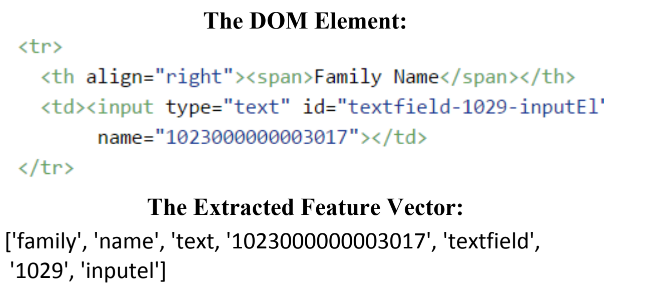
where each column of V^T represents the coordinates of the corresponding document in the vector space formed by the latent concepts. For example, the coordinates of d_3 (['password', 'password', 'password', 'password', '25'] in Fig. 5) is [(0, -0.7071), (3, 0.7071)], which could be interpreted as that the document consists of $-0.7071\times c_1$ and $0.7071\times c_4$. With the same transformations, we can transform a document q into the latent vector space in which its coordinates $\hat{q} = \Sigma^{-1}U^Tq$. The similarity of q to other documents can then be calculated. In practice, we compare documents in a rank-reduced vector space by keep the k largest singular vales in Σ and their corresponding vectors in U and V^T.

C. Labeling

For input topic identification, each input field in the training corpus has to be labeled by its topic, and we may take advantage of the results from previous stages to facilitate the labeling process. First, conceptually similar input fields are expected to be close in the latent vector space. For example, d_3 ([(0, -0.7071), (3, 0.7071)]) and d_4 ([(0, -0.7071), (3, -0.7071)]) look close, and both of them should be labeled as the topic of password. In addition, the most appropriate latent concept for representing d_3 may be c_1 or c_4 since the absolute weights in these dimensions are maximal (0.7071) over other dimensions, and so is d_4. As a result, we developed a quick heuristic to map each input field to a latent concept in which its absolute value is maximal in the transformed vector. For example d_3 and d_4 would be mapped to c_1 or c_4, and d_1 and d_2 would be mapped to c_2 or c_3. Users can label all the input fields belonging to the same latent concept at a time, or label some of them separately. In addition, it should be noticed that when integrating the proposed approach with rule-based method for input topic identification, the labeling effort may be negligible because it is implicitly included in rule construction.

D. Inference

The inference is done by calculating the similarities of an encountered DOM element with the ones in training corpus. For a DOM element, we first extract its feature vector with a dictionary built in previous stages, and apply the same Bag-of-word and Tf-idf transformations to it. The converted vector is then projected into the latent vector space formed by the training corpus with the matrices in previous decomposition. Finally, the similarity is calculated. Cosine similarity, i.e., the cosine of the angle between two vectors is adopted, because it is reported a good measure in information retrieval [8]. For example, if we want to infer the topic of an input field in Fig. 8 with a corpus built from Fig. 5. We first obtain the Tf-idf representation of the input field as [(0, 0.4472), (2, 0.8944)]. Note that only words "text" (index 0) and "name" (index 2) are used in the representation because other words extracted from the input field do not exist in the dictionary. Second, we transform the representation into a rank-three approximation of



The DOM Element:

```
<tr>
  <th align="right"><span>Family Name</span></th>
  <td><input type="text" id="textfield-1029-inputEl'
       name="1023000000003017"></td>
</tr>
```

The Extracted Feature Vector:

['family', 'name', 'text, '1023000000003017', 'textfield',
 '1029', 'inputel']

Fig. 8. An input field to be inferred

the latent vector space built from the corpus, as mentioned in Section III-B. The coordinates of the input field to be inferred would be: $\hat{q}_3 = (\Sigma_3)^{-1}(U_3)^Tq_3$. That is

$$\begin{pmatrix} 1.2953 & 0 & 0 \\ 0 & 1.0187 & 0 \\ 0 & 0 & 0.9810 \end{pmatrix}^{-1} \begin{pmatrix} 0.0000 & -0.1933 & 0.0331 \\ 0.0000 & -0.4842 & -0.5028 \\ 0.0000 & -0.4842 & -0.5028 \\ 0.0000 & -0.1933 & 0.0331 \\ 0.0000 & -0.6756 & 0.7016 \\ -0.2221 & 0.0000 & 0.0000 \\ -0.8886 & 0.0000 & 0.0000 \\ -0.1795 & 0.0000 & 0.0000 \\ -0.3590 & 0.0000 & 0.0000 \end{pmatrix}^T \begin{pmatrix} 0.4472 \\ 0 \\ 0.8944 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

$$= (0.0000 \quad -0.5100 \quad -0.4433)^T$$

The cosine similarity of $\hat{q}_3$ with documents in the corpus is:

$$d_1: Cosine(\hat{q}_3, [0 \quad -0.7071 \quad -0.7071]^T) = 0.9976$$
$$d_2: Cosine(\hat{q}_3, [0 \quad -0.7071 \quad 0.7071]^T) = 0.0697$$
$$d_3: Cosine(\hat{q}_3, [-0.7071 \quad 0 \quad 0]^T) = 0.0000$$
$$d_4: Cosine(\hat{q}_3, [-0.7071 \quad 0 \quad 0]^T) = 0.0000$$

$\hat{q}_3$ is most similar to d_1. Since d_1 should be labeled as the topic of *last_name* in previous steps, we can infer the topic of the input field in Fig. 8 as *last_name*, too. Note that in this example, the input field to be inferred uses randomly generated *id* and *name*, and the description ("Family Name") may contain no keywords users have learned previously (e.g. "Last") for identifying the topic. The idea behind the proposed method is that DOM elements are inferred by their usages of the used words, which string-matching based approaches do not take advantage of.

When applying the proposed approach to input topic identification, several input fields with different topics could be extremely similar with the input field to be inferred. We handle this condition with a voting process. First, the topic of the vector in the latent space most similar to the one to be inferred will be selected. If the difference of the similarities between the top 5 most similar vectors is less than a threshold, the topic will be determined by a voting process within the top 5 vectors. If there are multiple candidates after the vote, a random choice will be made. The voting process provides a chance to correctly infer the topic when there are multiple vectors with close similarity scores. For example, Table III shows that the inferred topic is mistaken when only the most similar vector is considered, but there is a chance to correct the mistake since the voting process may guess the right topic in random choice from *last_name* and *password*.

TABLE III. AN EXAMPLE FEATURE VECTOR AND ITS TOP 5 MOST SIMILAR VECTORS.

Similarity	Vector	Topic
(N/A)	['text', 'psurname', '30'] (To be inferred)	last_name
0.9982 (Most similar)	['text', 'fname', '30']	first_name
0.9978	['text, 'lname', '30']	last_name
0.9963	['familyname', 'familyname', 'text', '30']	last_name
0.9296	['conf', 'pword', password', '30']	password
0.9220	['pword', 'password', '30']	password

E. Integration with the Rule-based Approach

We may address the issues of the rule-based approach mentioned in Section II by integrating it with the proposed approach. Here we describe how to implement it in input topic identification, and the same concept can also be adopted in GUI state comparison and clickable detection in terms of taking suggestions from the our method. First, for the input fields not identified by the rule-based method, we output the answer found by our technique. Second, for the input fields matching multiple rules for different topics, we select the answer with the help of our technique. Specifically, if the natural-language answer appears in the candidates, the answer will then be selected, or a random choice will be made among all candidates including the natural-language one. In Section V, we evaluated the effectiveness of the integration

IV. IMPLEMENTATION

We implemented the proposed method with Python 2.7. A Python library, gensim [23] is used for vector space related operations such as vector transformations and similarity calculation. Interaction with web applications is supported by Selenium Webdriver [31], and BeautifulSoup [9] is used to parse and manipulate DOMs.

V. EVALUATION

To assess the efficacy of the proposed approach, we conducted two experiments on input topic identification and GUI state comparison, respectively. In the first experiment, we collected and labeled input fields from 100 real-world forms, and divided them as training and validation data to evaluate the performances of the proposed and rule-based approaches. In the second experiment, we used GUI states collected from real tests in QNAP, a software company in Taiwan, to evaluate the effectiveness of the proposed technique and different abstraction mechanisms. In general, the research question is:

Q1. What is the effectiveness of the proposed approach in input topic identification and GUI state equivalence computation, comparing with conventional methods used in crawling-based web application testing?

For input topic identification, there are two additional research questions:

Q2. How much training data is required for the proposed approach?

Q3. Can the proposed approach be used to improve the rule-based one?

A. Input Topic Identification

1) Subject Forms

We collected 100 graduate program registration forms across 9 countries in the world. There are totally 958 input fields in the forms, ranging from two to fifty-eight for each form, and 62 input topics such as *password*, *email*, *first_name* and *zipcode* are labeled. Table IV shows some labeled topics and the number of input fields for each topic. Each topic has to be distinguished from each other to pass the forms. For example, several date-related topics with different formats such as *date-mm/dd/yyyy*, *date-mm/yyyy* and *year-yyyy* are labeled for input fields in different forms taking date information. We choose registration forms as subject data for several reasons. First, they usually contain many different topics of input fields such as user profile, date or URL, which is appropriate for our evaluation. Second, the application states behind the forms are important because they take information from the users and then interact with them. However, the states are usually difficult to be reached using existing crawlers with random inputs. In addition, we want to evaluate the effectiveness of the methods on inferring unknown forms with training data in the same category.

2) Experimental Setup

To understand how the proportion of training data affects the performances of the methods under evaluation, we randomly chose 10% to 70% of the subject forms as training data, respectively. We derived labeled corpora, dictionaries and vector space models for the proposed approach, and generated rules (i.e., mappings from feature strings to topics) from the input fields of the training forms for the rule-based approach. All the artifacts were used to infer the input fields in the remaining forms, and the inference accuracies (i.e., number of correctly identified input fields over number of all inferred input fields) were calculated. Experiments with 10% to 70% training data were repeated 1000 times, respectively. Five methods were evaluated in these experiments: (1) NL, the proposed natural- language approach. (2) RB, the rule-based approach. (3) RB+NL-n (no-match), using the NL approach to identify input fields not recognized by RB, as discussed in Section III-E. (4) RB+NL-m (multiple), using the NL approach to help identify input fields from multiple candidates by RB, as discussed in Section III-E. (5) RB+NL-b (both), using both (3) and (4).

TABLE IV. LABELED INPUT TOPICS AND NUMBER OF INPUT FIELDS FOR EACH TOPIC IN THE EXPERIMENT.

Topic	#	…	Topic	#
password	188		validation_action	1
email	151		digit-16	1
last_name	105		ssn-middle	1
first_name	105		secure_q	1
username	48		job_title	1
middle_name	46		ssn-swiss-postfix-2	1
phone	46	…	date-yyyy-mm-dd	1
date-mm/dd/yyyy	43		unknown_hidden	1
zipcode	41		ssn-postfix	1
date-mm/yyyy	28		user_status	1
city	25		visa_number	1
street-line-2	13		ssn-prefix	1
street-line-1	13		**Total**	**985**

3) Results and Discussion

The average accuracies that each method achieved when the considered percentages are used as training data are shown in Table V. First, with less training data, the rule-based approach (RB) performed better than the proposed approach (NL). However, the effectiveness of NL increased with the percentage of training data, while RB performed oppositely. As a result, the proposed approach performs comparably to the rule-based one with 50% or more training data. Second, while RB+NL-n achieved slight improvement in general, using the proposed approach to help pick the correct topic from multiple candidates by the rule-based approach (RB+NL-m) can greatly improve the accuracy. Therefore, the improvement of RB+NL-b over RB ranges from 8.8% (6.63% increase) to 22% (15.95% increase).

To determine whether the improvements we observed in average accuracies are statistically significant, we conducted a *t-test for matched pairs* [15] for the NL, RB and RB+NL-b approaches. That is, the accuracies of these three methods in each trial are considered matched pairs to each other. We assume that there is no difference in the average accuracies of NL and RB, NL and RB+NL-b and RB and RB+NL-b, respectively (the null hypotheses). If the computed *p-value* is less than 0.05 (the significance level), statistical practitioners often infer that the null hypothesis is false. Table VI shows the *p-values* computed for these three methods in our experiments. It indicates that the observed differences between these three methods are statistically significant except for the one between NL and RB using 50% training data.

To further understand the experimental results, for experiments adopting the rule-based approach, we investigated the number of identified topics and created feature strings from the training data. Also, the number of inferred input fields and the percentage of no-matches and multiple-topic ones were recorded. First, as Table VII shows, hundreds of feature strings were created for mappings to tens of topics, which indicates that the rules for input topic identification could be application-

TABLE V. AVERAGE ACCURACIES ACHIEVED BY DIFFERENT METHODS WHEN THE CONSIDERED PERCENTAGES ARE USED AS TRAINING DATA.

% training	Accuracy (%)				
	NL	**RB**	**RB+ NL-n**	**RB+ NL-m**	**RB+ NL-b**
10%	70.42	75.60	75.70	82.13	82.23
20%	72.48	75.81	75.85	85.00	85.04
30%	72.66	75.04	75.05	86.18	86.19
40%	72.67	74.14	74.14	86.86	86.86
50%	73.26	73.50	73.50	87.47	87.47
60%	73.29	72.64	72.64	87.54	87.54
70%	74.05	72.44	72.44	88.39	88.39

TABLE VI. COMPUTED P-VALUES OF T-TEST FOR MATCHED PAIRS.

% training	NL & RB	NL & RB+NL-b	RB & RB+NL-b
10%	0.0000	0.0000	0.0000
20%	0.0000	0.0000	0.0000
30%	0.0000	0.0000	0.0000
40%	0.0000	0.0000	0.0000
50%	0.1677	0.0000	0.0000
60%	0.0004	0.0000	0.0000
70%	0.0000	0.0000	0.0000

TABLE VII. AVERAGE NUMBER OF IDENTIFIED TOPICS AND CREATED FEATURE STRINGS, INFERRED, NO-MATCH AND MULTIPLE-TOPIC INPUT FIELDS IN THE EXPERIMENTS WITH RULE-BASED APPROACH.

% training	# topic	# string	# inferred	% no-matches	% multiple-topic
10%	19.96	196.86	885.56	12.85	16.30
20%	27.71	247.29	789.04	9.37	22.44
30%	33.78	281.59	690.75	7.87	26.52
40%	38.77	306.52	592.93	7.11	29.24
50%	44.10	328.63	491.82	6.51	31.32
60%	47.93	343.88	394.46	6.44	32.36
70%	52.01	362.84	295.25	5.62	33.60

specific, and the manual effort could be significant. Second, with larger proportion of training data, less no-match and more multiple-topic elements were inferred. The observation is reasonable because with more training data introduced, the rule set derived from them is larger, and more input fields are likely to match multiple rules for different topics. For example, with 50% training data, more than 30% of the inferred elements were multiple-topic. We believe that this observation contributes to both the decreased accuracy of RB and the improvement of RB+NL-m on increased training data. In addition, the improvement of RB+NL-n comparing with RB is not significant, which could result from two reasons. First, the average number of no-match elements is not high. Second, from Table IV we can see that many topics of the input fields appear only a few times. In fact, 12.4% of total topics appear less than 10 times, and identifying these topics could be difficult even with assistance of our method because the input fields are not included in the training data.

Recommendation. If reducing manual effort in crawling-based web application testing is the primary concern, our method may be a good choice to adopt in input topic identification because the number of topics is far less than the number of rules (feature strings) required for mapping the topics, and the effort of labeling could be less than the effort of creating rules. In addition, the proposed method performs better with more training data. On the other hand, if the identification accuracy is the primary concern, the proposed approach can be integrated with the rule-based approach with little effort, because users have to go through input fields to construct rules, and the labeling required for the proposed method could be done simultaneously.

B. GUI State Comparison

1) Subject GUI States

QNAP Q'Center[3] is an AJAX web application for monitoring and managing multiple QNAP NAS (Network Attached Storage). It contains 1276 files and 324346 lines of code in v1.3.503, including 971 Python files (173258 LOC) and 133 JavaScript files (95917 LOC). From the test suites of the automatic GUI regression test on its daily build, we instrumented the test scripts to collect the GUI states in HTML after the execution of each test step. Also, a test engineer in the development team of Q'Center manually clustered the GUI states by examining the corresponding screenshots. Table VIII shows the characteristics of the collected data. The value of the

[3] https://www.qnap.com/solution/qcenter/index.php?lang=en

TABLE VIII. THE COLLECTED GUI STATES AND NUMBER OF DISTINCT CLUSTERS

Test Suite	Description	# Test Cases	# GUI States	# Clusters
install_wiz	Installation wizard	6	60	22
rule	NAS Rule management	3	237	80
server_add	NAS addition and removal	6	200	88
server_app	App management and config backup	7	207	42
settings	Account and server settings management	8	451	84

subject data is that it comes from real testing scenarios in industry. The GUI state equivalence information is also valuable because it can be used to know the effectiveness among different techniques for state equivalence and the gaps between the results produced by algorithms and human.

2) *Experimental Setup*

To understand the effectiveness of the proposed approach in computing GUI state equivalence, we clustered the GUI states with the proposed approach and other two abstraction mechanisms, respectively. The F-measure (the harmonic mean of precision and recall) of these three methods was calculated using the correct answers from previous subsection.

To apply the proposed approach (NL) in state comparison, for GUI states in HTML, we extracted their feature vectors from the enclosed text of the visible DOM elements. Fig. 9 shows an example of the extracted vector. Afterwards, we transformed the feature vectors as described in Section III (without the labeling stage) to calculate their similarities. If the similarity of an encountered state s_i with an existing state s_j exceeds a threshold (0.99999 in this experiment), s_i is considered equivalent to s_j. Otherwise, s_i would belong to a new cluster of itself. If there are multiple candidate clusters for a new state, a voting process similar to the one in Section III-D would be adopted to choose one cluster from the candidates. After inferred, a state would be put into the training corpus for future comparisons.

One of the abstraction mechanisms used in this experiment is WhiteSpace (WS). It replaces all line breaks and tabs with white space and then collapses white space in the HTML before computing state equivalence. This is a common oracle comparator because browsers also do so, and additional white space may not affect user's interpretation on a web page. The other abstraction mechanism used in this experiment includes three steps. First it keeps only the tag names and the corresponding important attributes of a HTML, and then applies WS abstraction to remove the line breaks and extra white space. Finally it removes timestamps shown in the HTML. This abstraction mechanism (TagAttrWD) is reported an oracle comparator with the highest mean effectiveness in fault detection [29]. We considered the *required* attributes from HTML specification [16] and *name*, *type* and *value* for input tags as important attributes, as designated in [29].

In clustering result evaluation, true positive is the number of pairs of data points belonging to the same cluster and classified as the same group. False positive is the number of pairs of data points belonging to different clusters and classified as the same group. True negative and false negative are defined similarly. By counting the positives and negatives, we can calculate the precision, recall and F-measure of the clustering results.

The HTML:

```
<!DOCTYPE html>
<html>
<head><script src="test1.js"></script><head>
<body>
<h1>The demo form</h1>
<p>Please fill the form:</p>
<form action="demo_form.asp">
  <label for="fname">Family Name</label>
  <input type="text" id="fname" name="fname">
  <input type="submit" value="Submit">
</form>
<script src="test2.js"></script>
</body>
< /html>
```

The Extracted Feature Vector:

```
['the', 'demo', 'form', 'please', 'fill', 'the', 'form', 'family',
'name']
```

Fig. 9. An GUI state in HTML and its extracted feature vector

3) *Results and Discussion*

Table IX shows the F-measure of the clustering results with different methods. First, WS had the worst effectiveness in the experiments. Second, TagAttrWD had better F-measure than WS, which is consistent with the experimental results in fault detection effectiveness conducted in [29]. Finally, the proposed method outperformed other two abstraction techniques in three of the five test suites, which means it can better distinguish encountered GUI states when executing the three test suites. In the other two test suites, the effectiveness of the proposed approach is still better than WS and close to the best (TagAttrWD).

The proposed approach may better distinguish GUI states in crawling-based web application testing, because it considers semantic similarity among states when computing their equivalence, which is analogous to what human does in the same situation. Human doesn't rely on the DOM structures but the text on web pages to determine the equivalence. Therefore, it is possible that two web pages with totally different DOM structures are considered identical for testing purpose. We believe that it is one of the reasons the proposed approach outperformed other two conventional abstraction mechanisms. Nevertheless, most of the clustering results are far from the correct ones labeled by human (over two-thirds of the F-

TABLE IX. F-MEASURE OF THE CLUSTERING RESULTS

Test Suite	F-measure		
	WS	TagAttrWD	NL
install_wiz	0.7817	**0.8194**	0.7826
rule	0.3241	0.3599	**0.4443**
server_add	0.4281	0.4751	**0.6776**
server_app	0.1532	0.1809	**0.3559**
settings	0.1180	**0.4512**	0.4156

measure are smaller than 0.5). Two observations might contribute to this issue. First, a test engineer may consider consecutive GUI states as the same when doing a series of input actions (e.g. filling a form). Second, as describe in Section II, a test engineer may ignore a changed visible DOM element, because the element is irrelevant to what he cares about under current testing scenario.

Recommendation. Experimental results showed promise for adopting semantic similarity in computing state equivalence. Also, it is possible to integrate the proposed approach with conventional abstraction mechanisms to improve the effectiveness of GUI state equivalence checking.

C. Threats to Validity

The implementation of the proposed approach could affect the validity of results. To ensure the correctness, we adopted mature and open-sourced libraries such as gensim [23], BeautifulSoup [9] and Selenium Webdriver [31] in key steps of our implementation. In input topic identification, a single category of forms (graduate program registration) used and the setup of the evaluation such as the labeled topics and derived rules might affect generality of the results. However, the topics (e.g. *email*, *password* and *last_name*) in the subject forms are common in other categories of forms. We also open our source code and data to the public for review and replication. In GUI state comparison, the results on data from a commercial application could not be generalize to web applications in other categories, but the experiments on data from real-world scenarios are still valuable.

VI. Related Work

Crawling-based techniques for modern web applications have been studied [3], [6], [11], [19], [28], [33] and adopted [1], [5], [7], [20], [27], [30] in automated web application testing. Raghavan and Garcia-Molina [28] presented an operational model to crawl the content hidden behind search forms. Their work is highly related to ours in terms of analyzing labels to understand the topics of form elements. However, they neither considered semantic similarity nor addressed GUI state comparison. Duda et al. [11] proposed algorithms to crawl AJAX-based web applications and index the states. Similarly, a tool developed by Mesbah et al. [6] called Crawjax tries to infer a finite state machine of an AJAX-based web application through dynamically analyzing the changes of the DOM contents. The tool is also used for detecting and reporting violated invariants [5] and cross-browser incompatibilities [27] in web applications. Schur et al. [19] presented a crawler called ProCrawl to extract abstract behavior models from multi-user web applications, focusing on building a model close to business logic. A crawler developed by Dallmeier et al. [33], WebMate, can autonomously explore a web application, and use existing test cases as an exploration base. Marchetto et al. [3] extracts a finite state machine from existing execution traces of web applications, and generates test cases consisting of dependent event sequences. In addition, Fard et al. [7] combined the knowledge inferred from manual test suites with automatic crawling in test case generation for web applications. Thummalapenta et al. [30] presented a technique to confine the number of a web application's GUI states explored by a crawler with existing business rules. Nevertheless, none of these studies considers leveraging semantic similarity in input value handling or GUI state comparison as our work does.

About the measurement of crawling diversity or effectiveness, Alshahwan et al. [20] proposed crawlability metrics to quantify the extent to which a crawler is able to explore the web pages or forms. These metrics combine dynamic measurements such as statement coverage, with static information such as lines of code. Moreover, Fard and Mesbah [1] presented a couple of metrics such as JavaScript code coverage, path diversity and DOM diversity to assess the test model derived by a crawler. These measurements may be tailored to evaluate the effectiveness of our techniques.

To verify that the executed test cases produce expected results in web application testing, Sprenkle et al. [29] proposed a suite of automated oracle comparators. The comparators were applied to expose faults by reporting the difference between the actual and expected HTML output. The fault detection effectiveness of individual comparators and selected comparator combinations were analyzed by calculating their precision (i.e., if the reported differences were due to faults) and recall (i.e., if all faults were revealed by reported differences). While one of their comparator combinations, TagAttrWD, was used in our experiment, our work focuses on the ability of different abstraction mechanisms to correctly distinguish new GUI states during crawling.

Studies on GUI ripping for testing purpose [4], [12] and automatic black-box testing on mobile applications [2], [32] are also related to our work in terms of how they explore the interfaces of the applications and derive test models with dynamic analysis. As a result, the proposed technique could be applied in these contexts.

With respect to using latent topic models in software testing and debugging, Maletic and Marcus [17] adopted LSI to cluster similar files of source code for software maintenance or reengineering. Andrzejewski et al. [13] approached debugging using a variant of LDA (Latent Dirichlet Allocation) [14] to identify weak bug topics from strong interference. LDA was also adopted by Lukins et al. [26] on a developer's input such as a bug report to localize faults statistically. Later, DiGiuseppe and Jones [21], [22] adopted natural-language techniques such as feature extraction and Tf-idf in fault description and clustering. To our knowledge, this paper is the first to apply latent topic models in crawling-based web application testing.

VII. Conclusion

In this paper, we proposed a natural-language technique to improve the effectiveness of crawling-based web application testing. By considering semantic similarities between a training corpus and a DOM element to be inferred, input topic identification, GUI state comparison and clickable detection can be performed with the proposed approach. In the future, we plan to evaluate how the proposed techniques impact overall crawling efficacy with more data and other topic model alternatives such as LDA. Moreover, the proposed feature extraction algorithm could be improved with more information about DOM elements such as comments.

REFERENCES

[1] A. M. Fard and A. Mesbah, “Feedback-directed exploration of web applications to derive test models,” in 2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE), 2013, pp. 278–287.

[2] A. Machiry, R. Tahiliani, and M. Naik, “Dynodroid: An Input Generation System for Android Apps,” in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, New York, NY, USA, 2013, pp. 224–234.

[3] A. Marchetto, P. Tonella, and F. Ricca, “State-Based Testing of Ajax Web Applications.pdf,” in 2008 1st International Conference on Software Testing, Verification, and Validation, 2008, pp. 121–130.

[4] A. Memon, I. Banerjee, and A. Nagarajan, “GUI ripping: reverse engineering of graphical user interfaces for testing,” in 10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings, 2003, pp. 260–269.

[5] A. Mesbah, A. van Deursen, and D. Roest, “Invariant-Based Automatic Testing of Modern Web Applications,” IEEE Transactions on Software Engineering, vol. 38, no. 1, pp. 35–53, Jan. 2012.

[6] A. Mesbah, A. van Deursen, and S. Lenselink, “Crawling Ajax-Based Web Applications Through Dynamic Analysis of User Interface State Changes,” ACM Trans. Web, vol. 6, no. 1, p. 3:1–3:30, Mar. 2012.

[7] A. Milani Fard, M. Mirzaaghaei, and A. Mesbah, “Leveraging Existing Tests in Automated Test Generation for Web Applications,” in Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, New York, NY, USA, 2014, pp. 67–78.

[8] A. Singhal, “Modern Information Retrieval: A Brief Overview,” IEEE Data Eng. Bull., vol. 24, no. 4, pp. 35–43, 192001 2007.

[9] BeautifulSoup. https://pypi.python.org/pypi/beautifulsoup4

[10] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. New York, NY, USA: Cambridge University Press, 2008.

[11] C. Duda, G. Frey, D. Kossmann, R. Matter, and C. Zhou, “AJAX Crawl: Making AJAX Applications Searchable,” in IEEE 25th International Conference on Data Engineering, 2009. ICDE ’09, 2009, pp. 78–89.

[12] D. Amalfitano, A. R. Fasolino, P. Tramontana, S. De Carmine, and A. M. Memon, “Using GUI Ripping for Automated Testing of Android Applications,” in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, New York, NY, USA, 2012, pp. 258–261.

[13] D. Andrzejewski, A. Mulhern, B. Liblit, and X. Zhu, “Statistical Debugging Using Latent Topic Models,” in Proceedings of the 18th European Conference on Machine Learning, Berlin, Heidelberg, 2007, pp. 6–17.

[14] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent Dirichlet Allocation,” J. Mach. Learn. Res., vol. 3, pp. 993–1022, Mar. 2003.

[15] G. Keller and B. Warrack, Statistics for Management and Economics, 6th ed. Pacific Grove, CA: Thomson/Brooks/Cole. 2003

[16] HTML 4.01 Specification. http://www.w3.org/TR/html4/, Dec. 1999.

[17] J. I. Maletic and A. Marcus, “Using latent semantic analysis to identify similarities in source code to support program understanding,” in 12th IEEE International Conference on Tools with Artificial Intelligence, 2000 (ICTAI 2000), pp. 46–53.

[18] M. Benedikt, J. Freire, and P. Godefroid, “VeriWeb: Automatically Testing Dynamic Web Sites,” in International World Wide Web Conference (WWW), Honolulu, HI, 2002, pp. 654–668.

[19] M. Schur, A. Roth, and A. Zeller, “Mining Behavior Models from Enterprise Web Applications,” in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, New York, NY, USA, 2013, pp. 422–432.

[20] N. Alshahwan, M. Harman, A. Marchetto, R. Tiella, and P. Tonella, “Crawlability Metrics for Web Applications,” in Verification and Validation 2012 IEEE Fifth International Conference on Software Testing, 2012, pp. 151–160.

[21] N. DiGiuseppe and J. A. Jones, “Concept-based Failure Clustering,” in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, New York, NY, USA, 2012, p. 29:1–29:4.

[22] N. DiGiuseppe and J. A. Jones, “Semantic Fault Diagnosis: Automatic Natural-language Fault Descriptions,” in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, New York, NY, USA, 2012, p. 23:1–23:4.

[23] R. Ř. uřek and P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, 2010, pp. 45–50.

[24] S. Choudhary, M. E. Dincturk, G. V. Bochmann, G.-V. Jourdan, I.-V. Onut, and P. Ionescu, “Solving Some Modeling Challenges when Testing Rich Internet Applications for Security,” in 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012, pp. 850–857.

[25] S. H. Friedberg, A. J. Insel, and L. E. Spence, Linear Algebra, 4th Edition, 4th edition. Upper Saddle River, N.J: Pearson, 2002.

[26] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, “Bug localization using latent Dirichlet allocation,” Information and Software Technology, vol. 52, no. 9, pp. 972–990, Sep. 2010.

[27] S. R. Choudhary, M. R. Prasad, and A. Orso, “CrossCheck: Combining Crawling and Differencing to Better Detect Cross-browser Incompatibilities in Web Applications,” in 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST), 2012, pp. 171–180.

[28] S. Raghavan and H. Garcia-Molina, “Crawling the Hidden Web,” in Proceedings of the 27th International Conference on Very Large Data Bases, San Francisco, CA, USA, 2001, pp. 129–138.

[29] S. Sprenkle, L. Pollock, H. Esquivel, B. Hazelwood, and S. Ecott, “Automated Oracle Comparators for Testing Web Applications,” in The 18th IEEE International Symposium on Software Reliability, 2007. ISSRE ’07, 2007, pp. 117–126.

[30] S. Thummalapenta, K. V. Lakshmi, S. Sinha, N. Sinha, and S. Chandra, “Guided Test Generation for Web Applications,” in Proceedings of the 2013 International Conference on Software Engineering, Piscataway, NJ, USA, 2013, pp. 162–171.

[31] Selenium HQ. http://seleniumhq.org/.

[32] UI/Application Exerciser Monkey. http://developer.android.com/tools/help/monkey.html

[33] V. Dallmeier, B. Pohl, M. Burger, M. Mirold, and A. Zeller, “WebMate: Web Application Test Generation in the Real World,” in 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 2014, pp. 413–418.

[34] V. Garousi, A. Mesbah, A. Betin-Can, and S. Mirshokraie, “A systematic mapping study of web application testing,” Information and Software Technology, vol. 55, no. 8, pp. 1374–1396, Aug. 2013.