Poster: Efficient GUI Test Generation by Learning from Tests of Other Apps

Andreas Rau
Saarland University
Saarbrücken, Germany
andreas.rau@cispa.saarland

Jenny Hotzkow
Saarland University
Saarbrücken, Germany
jenny.hotzkow@cispa.saarland

Andreas Zeller
Saarland University
Saarbrücken, Germany
andreas.zeller@cispa.saarland

ABSTRACT

Generating GUI tests for complex Web applications is hard. There is lots of functionality to explore: The eBay home page, for instance, sports more than 2,000 individual GUI elements that a crawler has to trigger in order to discover the core functionality. We show how to leverage *tests of other applications* to guide test generation for a new application: Given a test for payments on Amazon, for instance, we can guide test generation on eBay towards payment functionality, exploiting the *semantic similarity* between UI elements across both applications. Evaluated on three domains, our approach allows to discover "deep" functionality in a few steps, which otherwise would require thousands to millions of crawling interactions.

ACM Reference Format:
Andreas Rau, Jenny Hotzkow, and Andreas Zeller. 2018. Poster: Efficient GUI Test Generation by Learning from Tests of Other Apps. In *ICSE '18 Companion: 40th International Conference on Software Engineering Companion, May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3183440.3195014

1 INTRODUCTION

Generating tests for Graphical User Interfaces (GUIs) is hard. Not only can it be difficult to determine which actions would be allowed for a specific user interface; the large number of possible interactions on modern user interfaces can make automated test generation practically impossible already. In fact, tasks that are seemingly straightforward for human users can be incredibly difficult for a computer. For us as humans, such activities are much easier. That is because we can interact with GUIs based on the *semantics* of user interfaces—we *know* what terms like "feedback", "pay", "cancel", "special offer" mean, and we choose the ones that are closest to our goals. We also have *experience* with similar sites; if we have shopped once on, say, Amazon.com, we know how to repeat the process on other shopping sites, where we can re-identify familiar concepts like "cart", "checkout", or "payment".

Now assume that for one application A, *we already have a test* that selects a product, adds it to the cart, proceeds to check out, and finally pays for the product. *Would it be possible to leverage the knowledge from this test to generate a test for another application B?* By extracting the general sequence of actions (selecting, adding to a cart, checking out, paying) from the existing test case, we *guide* test generation for the target B towards those actions already found on the original test A.

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195014

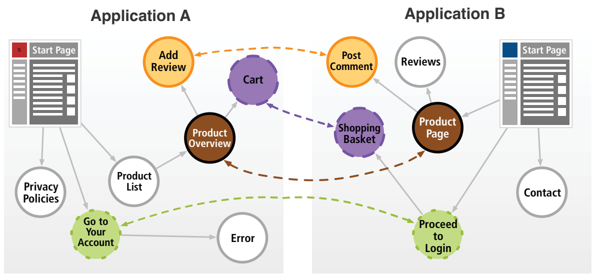

Figure 1: Applied feature mapping across applications. The natural language content of two apps is matched to the closest semantic match using semantic text similarity.

This *semantic mapping* is what we do in this paper. We start with an existing test for a *source* Web application (say, Amazon). Each step in the test induces a state in the source application. From these states, we extract the *semantic features*—that is, the user interface elements the test interacts with together with their textual labels. To generate a test for a new target Web application (say, eBay), we attempt to *match* these features while exploring the application, determining the *semantic similarity* between the labels captured in the source and the labels found in the target. We then *guide* exploration towards the best matches, quickly discovering new and deep functionality, which is then made available for test generation. The final result is a *mapping* of the features found in the source to the features found in the target, allowing for easy (and often even automatic) adaptation of the tests created for the source application onto the new target application.

(1) *semantic mapping* maps features between source and target applications with a recall of 75% and a precision of 83%.
(2) If the searched functionality is deeper in the application, this speedup multiplies across states, yielding a total speedup along the path that can be exponential.

2 SEMANTIC MAPPING

The graphical user interface abstracts complex technical events and is designed for human understanding. The natural language content helps users to achieve their goal by understandably exposing the functionality, i.e. the *features* of the application, by presenting a describing label next to input fields or by filling fields with expressive default data. Human understanding allows to grasp the underlying semantic meaning of such descriptions.

The core of our presented technique consists of two steps. In the *feature identification phase*, we identify essential features given an existing Selenium test suite. The UI elements, on which the

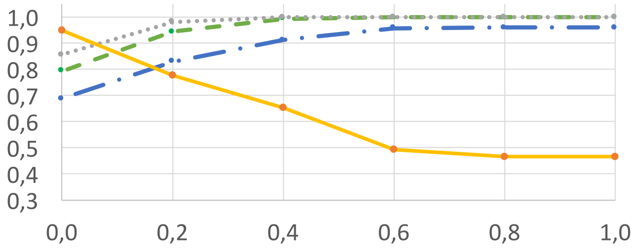

Figure 2: Averaged results of feature mapping. The x-axis shows the threshold applied on $\equiv_{sem}$, the y-axis displays the values for $P@1$ (blue dash-dot), $P@3$ (green-dashed), $P@10$ (grey-dotted), and *Recall* (solid yellow line).

test suite executes commands, together with their describing labels form a feature. Using a visual page segmentation algorithm [1], we group together the interactive elements with their describing labels. After extracting their textual content, we *match features across applications* by matching the textual content to the labels in the target application using semantic text similarity as a metric in the next step. The matched features then guide test generation.

In the *feature mapping phase*, we calculate for every feature, identified in phase one, the closest semantically matching labels in the target application. *Semantic similarities* between words [2] are obtained by training a word vector model (short `word2vec`) on a set of documents. Words expressing similar concepts are mapped to nearby points. These *word vectors* capture meaningful semantic regularities, i.e. within the given documents one can observe constant vector offsets between pairs of words sharing a particular relationship. This methodology follows the assumptions that the inflectional form introduced by the grammar of a language and even the order of words are of minor importance for human understanding (bag-of words assumptions). Instead of training on a large set of pre-labeled documents, we use the googleNewsVector model [4] off-the-shelf. The non-domain-specific model has the advantage to be applicable to arbitrary application domains, while specialized models would require retraining.

All labels are processed to sanitize the given input. Illegal characters such as invalid/incomplete html-tags (e.g. '<',/) or special symbols (e.g. special Unicode characters) are removed. After tokenizing the labels into individual words, we reduce each word to a common base form by morphological stemming and lemmatization. We filter out the most common words of the language (stopword removal) and remove unknown or invalid words, which are not in the corpus. To calculate the semantic similarity of two labels, we compute the cosine similarity between all possible word pairs expressed in an $n \times m$ matrix, where n and m are the respective lengths of the labels. The best matching word pairs (ignoring the order of words) under the conditions that every word is only matched once are summed up and the resulting value is normalized. The computed value $\equiv_{sem}$ is in the interval $[-1, 1]$. A value close to -1 indicates highly dissimilar concepts, while values close to 1 indicate a semantic match.

To match features across applications, we take the features that were previously learned and grouped by executing the test cases of application A, and find in each target state (DOM-tree) of application B the labels with the closest semantic matches. By re-running the VIPS algorithm on the mapped labels, we identify which UI elements are relevant for the feature in the target application. The more of the describing labels we can find in a single DOM, the higher the certainty that the correct features are identified. The algorithm returns the list of matched UI elements altogether with the calculated matching index $\equiv_{sem}$ which serves as a certainty factor that the UI elements in the target application match the wanted feature. The list of potential matches is sorted in descending order, ranking the most likely matches at the top.

3 EVALUATION

We selected the top 12 industry-sized real world applications (more than 30k interactive elements) from the domains *eCommerce*, *Search Engines* and *Knowledge Bases* according to the Alexa 500 index. We developed custom SELENIUM test suites for each, which execute a total of 551 features. The test setup runs for instance the test suite for AMAZON, extracts the tested features with their describing labels and identifies the features in EBAY based on the DOM data. To verify the correctness, we manually labeled each of the 551 features and test if the designated XPATHs match the predicted ones.

Each application is tested against all other applications within the same domain. Figure 2 presents the averaged results of our evaluation. The *precision at k* [3] denotes how many good results are among the top k predicted ones. $P@1$ for instance indicates the precision of a perfect match (the top predicted element is the correct feature), while $P@10$ indicates that the correct result was in the top 10. $P@1$ shows a precision of 83% at 75% recall. $P@10$ shows a precision/recall rate of about 90%.

4 CONCLUSION

Modern applications offer so many interaction opportunities that automated exploration and testing is practically impossible without guidance towards relevant functionality. We propose a method that reuses existing tests from other applications to effectively guide exploration towards semantically similar functionality. This method is highly effective: our guidance allows to discover deep functionality in only a few steps. In the long run, our new technology *semantic mapping* allows to write test cases for only one representative application in a particular domain ("select a product", "choose a seat", "book a flight", etc.) and automatically reuse and adapt these test cases for any other application in the domain, leveraging and reusing the domain experience. To facilitate replication and comparison, the data referred to in this paper is available for download. The package comprises exploration graphs for all evaluated applications, the mappings as determined by our approach, as well as our established ground truth. For details, see:

https://www.st.cs.uni-saarland.de/projects/tdt

REFERENCES

[1] M. Elgin Akpinar and Yeliz Yesilada. 2013. Vision Based Page Segmentation Algorithm: Extended and Perceived Success. In *Revised Selected Papers of the ICWE 2013 International Workshops on Current Trends in Web Engineering - Volume 8295*. Springer-Verlag New York, Inc., New York, NY, USA, 238–252.

[2] Aminul Islam and Diana Inkpen. 2008. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Transactions on Knowledge Discovery from Data* 2, 2 (2008), 1–25. https://doi.org/10.1145/1376815.1376819

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA. 161–163 pages.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. GoogleNews-vectors-negative300.bin.gz - Efficient estimation of word representations in vector space. (2013). https://code.google.com/archive/p/word2vec/