Detecting Near-duplicate States in Web Application Model Inference: a Tree Kernel-based Approach

Luigi Libero Lucio Starace
luigiliberolucio.starace@unina.it
Università degli Studi di Napoli Federico II
Naples, Italy

ABSTRACT

In the context of End-to-End testing of web applications, automated exploration techniques (a.k.a. crawling) are widely used to infer state-based models of the application under test. These models, in which states represent dynamic web pages and transitions represent reachability relationships, can be used for several analysis and testing tasks, such as test case or test artifact generation. However, current crawling techniques often lead to models affected by near-duplicates, i.e., multiple states representing slightly different pages that are in fact instances of the same functionality. This has a negative impact on the subsequent model-based testing tasks, adversely affecting, for example, size, running time, and achieved coverage of generated test suites.

In my research, my goal is to improve the model inference of web applications by devising novel near-duplicate detection techniques. My vision is to leverage Tree Kernel (TK) functions, which have been largely investigated and applied, thanks to their flexibility, in the Natural Language Processing domain to compute similarity between tree-structured objects. I envision to design specifically-suited TK functions, meant to consider the peculiarities of the Document Object Model (DOM) tree-structured representation of web pages, to detect near-duplicate web pages, thus improving the quality of the inferred models.

CCS CONCEPTS

• Software and its engineering → Abstraction, modeling and modularity; **• Information systems → Web applications**.

KEYWORDS

Near-duplicate detection, Model inference, Web Application Testing, Tree kernels, Reverse engineering, Model-based testing

ACM Reference Format:
Luigi Libero Lucio Starace. 2021. Detecting Near-duplicate States in Web Application Model Inference: a Tree Kernel-based Approach . In *ISSTA '21: ACM SIGSOFT International Symposium on Software Testing and Analysis, Doctoral Symposium Track, July 11–17, 2021, Aarhus, Denmark.* ACM, New York, NY, USA, 4 pages.

1 PROBLEM STATEMENT AND RELATED WORKS

Web applications have become pervasive and are involved in many aspects of our daily lives. From home banking to public transit trip planning, from e-commerce to social networks, from online news agencies and newspapers to video streaming services, society relies on web applications to an ever-growing extent for a multitude of economic, social, and recreational activities. The impact of failures in a web application may range from simple inconveniences to end-users up to complete business interruption, and can potentially cause significant damages. Hence, ensuring the quality and correctness of web applications is of undeniable importance [21].

End-to-end (E2E) web testing is one of the main approaches to ensure the quality of web applications. In this kind of activity, testers exercise the Application Under Test (AUT) as a whole and from the perspective of an end-user interacting with the Graphical User Interface (GUI), i.e., the web pages, of the application. The goal is to verify that the web application behaves as intended in response to user-generated events and interactions with the GUI (e.g., clicks, scrolls, forms filling and submissions, etc.), corresponding to usage scenarios of interest. To do so, testers typically develop test scripts that, leveraging test automation libraries such as Selenium [6], automate the set of manual operations that the end-user would perform on the GUI of the web application. Developing such test scripts manually, however, is a time consuming and expensive task, often neglected in web projects because of resource constraints [7].

To support E2E web testing activities, automated reverse-engineering techniques are widely used to infer state-based models of the AUTs [26]. These models can be used for several analysis and testing tasks, such as test case generation [3–5, 17, 22] or test artifact generation [24, 25].

A popular approach to model construction for modern web applications is automated state exploration [26], also referred to as web application crawling [15]. In such models, a state represents a dynamic web page of the application, and transitions between states represent the fact that the target state is reachable from the source one under particular conditions (e.g., when a particular event is fired). Crawling-based techniques dynamically and systematically analyze the AUT starting at an initial page, and then automatically explore the application by generating events and checking the web page for changes. When, as a result of a fired event, a state change is detected, the model is updated to reflect the event causing the new state.

From a testing view-point, these inferred models should contain a minimal set of significantly different states, yet adequately cover all the functionalities of the AUT. In practice, however, models inferred automatically through state exploration are affected by

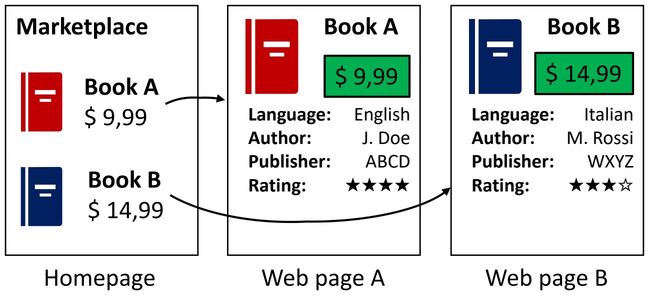

Figure 1: Example of near-duplicate web pages

near-duplicates [10, 11, 13, 16], i.e., replicas of the same functional web page differing only by small, insignificant changes [26].

As an example, let us consider Figure 1, in which three web pages from an imaginary bookstore web application are depicted. The homepage of the application shows a catalog of available books. After clicking on one of the books, the user is redirected to a detail web page with additional information, from which it is possible to add the book to the cart and finalize the purchase. The detail pages for the two books in the example are of course different, but from a functional testing view-point they are conceptually the same, as both are an instance of the "Show book details" functionality. Similarly, a catalog page containing three books would be conceptually the same of the one depicted in the example, since they would both represent the same "Show catalog" functionality.

From an E2E testing perspective, near-duplicate states in web application models have a negative impact on the accuracy, completeness and effectiveness of the models, hindering the application of model-based testing techniques. For instance, test suites generated from models with many near-duplicate states can be noticeably worse in terms of size, running time, and achieved coverage [26].

To detect and discard such near-duplicate web pages during their exploratory process, crawlers have adopted state abstraction functions as a proxy for the similarity of web pages [26]. The way these state abstraction functions are defined heavily impacts the quality of the obtained models. State abstractions that are too strict and mark as distinct any two web pages that are not exactly identical will likely result in models containing many near-duplicate states. Conversely, state abstractions that are too loose and flag two web pages with noticeable differences as same-state are likely to result in incomplete models, in which many functionalities are not represented by a dedicated state.

Many techniques have been developed to detect near-duplicate web pages *across* different web applications. For instance, the problem of detecting duplicate and near-duplicate web pages arises naturally in the web indexing process of search engines. In this field, the concept of duplication and near-duplication is mainly related to the content of the web page, and hence Information Retrieval techniques such as `simhash` [8] have been found to be quite effective [13].

Detecting near-duplicate pages is also a challenge for automatic phishing detection. In this case, malicious websites are often designed to look as similar as possible to the original website they try to impersonate, while maintaining an entirely different HTML structure to avoid detection. Hence, Computer Vision techniques such as [27] have often been applied with good results [1].

However, despite being a crucial step for effective model inference activities, less work has been directed towards detecting near-duplicates *within* the same web application. A first study in this direction was presented at ICSE2020 [26]. In this study, 10 near-duplicate detection techniques from the different domains of information retrieval, computer vision, and web testing are applied and compared in the context of web application model inference. That study highlighted that there is a need for further research in devising near-duplicate detection techniques geared specifically towards model inference.

We plan to fill this research gap by investigating novel near-duplicate detection techniques specifically designed for supporting model inference for web applications. In particular, we intend to leverage Tree Kernel (TK) functions, a class of kernel functions largely investigated in the Natural Language Processing domain to evaluate similarity between tree-structured objects [20]. We envision that TKs, thanks to their flexibility, might be effective tools to capture different types of near-duplicate web pages, improving the overall detection performance.

The remainder of this paper is organized as follows. In Section 2 we give some preliminary notions on TK functions and then, in Section 3, we sketch the tree kernel-based approach we are currently investigating to detect near-duplicates and hence improve model inference for web applications. In Section 4 we present some preliminary results we obtained, and in Section 5, we provide a road-map detailing future research efforts.

2 TREE KERNEL FUNCTIONS

Tree Kernel (TK) functions are a particular family of kernel functions which specifically evaluate similarity between two tree-structured objects. These functions have been extensively studied in Natural Language Processing [20], and have also been applied with promising results in the Software Engineering domain. In particular, TKs have been applied on Abstract Syntax Tree representations of source code for clone detection [9], and their usage is also being investigated for test case prioritization tasks [2]. More recently, [14, 23] presented an effective approach to fake website detection, which leveraged TK functions.

To compute the similarity between two trees T_1 and T_2, TK functions consider, for each tree, a set of *tree fragments*. A tree fragment is a subset of nodes and edges of the original tree. Then, the similarity between the tree fragments of the two trees is evaluated, and the overall similarity of the two trees is computed by aggregating, in some meaningful way, the similarities of the single fragments. Depending on how the set of fragments to consider is defined, it is possible to characterize different classes of tree kernel functions. Widely-used classes include [19]:

- *Subtree Kernels*, which consider only proper subtrees of the orginal trees, i.e., a node and all of its descendants, as fragments.
- *Subset tree Kernels*, which consider as fragments a more general structure than subtree kernels, relaxing the constraint of taking all descendant of a given node and thus allowing for incomplete subtrees, limited at any arbitrary depth.

```
<html>
  <head>
    <title></title>
    <link/>
  </head>
  <body>
    <section>
      <h1></h1>
      <p><a></a></p>
      <figure>
        <img/>
      </figure>
    </section>
  </body>
</html>
```

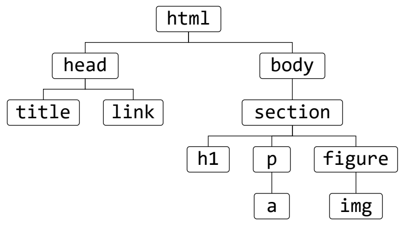

Figure 2: An HTML document and its DOM representation

- *Partial Tree Kernels*, which consider an even more general notion of fragment, in which the constraint of taking either all children of a tree node or none at all is relaxed. In this case, it is possible to include only some of the children of a node in a fragment.

3 IMPROVING NEAR-DUPLICATE DETECTION WITH TREE KERNELS

As done in the work of Yandrapally et al. [26], we frame the near-duplicate detection problem as a multinomial classification problem. In particular, given a pair of web pages, the goal is to classify it into one of the following distinct categories:

- **Clone**, if there is no semantic, functional or perceptible difference between two web pages.
- **Distinct**, if there is any semantic or functional difference between the two pages.
- **Near-duplicate**, if there are noticeable differences, but the overall functionality being exposed is the same.

To this end, our approach consists in extracting a set of features representing the similarity of the two web pages, and on using these features to classify the web page pair. We envision that Tree Kernel (TK) functions might be an effective tool to measure the similarity of two web pages which, as shown in Figure 2, can be naturally modeled using their tree-structured Document Object Model (DOM) representation. In particular, each of the similarity features we consider is a real-valued similarity score computed by a different TK function. We believe that different TKs might be able to capture different types of near-duplicate web pages, hence complementing one-another and improving the overall classification performance. In particular, we consider three standard tree kernel functions: a subtree kernel, a subset-tree kernel and a partial tree kernel. Moreover, to investigate how different portions of the DOM tree impact similarity computation and near-duplicate detection, and to make our approach more general and customizable, we also introduce the concept of DOM representation functions. Intuitively, these functions represent a pre-processing step in which the DOM of a web page can be transformed according to some meaningful strategy. Currently, we are considering three basic DOM representation strategies, as detailed in Table 1. From the pairwise combination of

Strategy	Description
As-is	This representation strategy leaves the DOM unchanged;
Only body	This representation strategy considers only the DOM subtree rooted in the body element of the web page.
Only body with no scripts	This representation strategy is the same as the only body one, but also removes `script` elements along with their subtrees.

Table 1: Considered DOM representation strategies

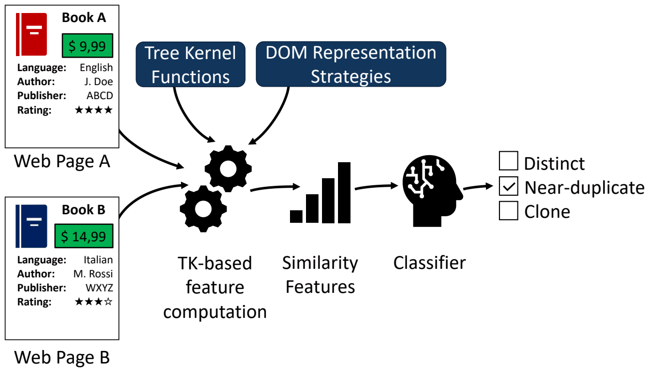

Figure 3: The proposed approach

the three considered TK functions and the three DOM representation strategies, nine different similarity features arise. Leveraging these similarity features and existing open datasets with annotated web page pairs, we use unsupervised learning approaches to train an *ad-hoc* classifier. The proposed approach is summarized in Figure 3.

4 EVALUATION AND PRELIMINARY RESULTS

We plan to evaluate the proposed approach using the same data and experimental procedure presented by Yandrapally et al. in [26], which investigates the effectiveness of a number of state-of-the-art near-duplicate detection techniques, and their impact on the model-inference process. This way, our results can be directly compared with the state-of-the-art. In particular, [26] compared 10 different near-duplicate detection techniques from the different domains of Computer Vision, Information Retrieval, and Web Testing. Moreover, that work also provided a large dataset of about 100k annotated same-website web page pairs, consisting of three main parts detailed as follows:

- $\mathcal{SS}$ is a set of ~97k annotated web page pairs extracted from 9 open source web applications in a controlled environment.
- $\mathcal{DS}$ is a set of ~1k annotated same-website web page pairs extracted from about 1k real-world websites, randomly selected from Alexa's top 1 million URLs list.
- $\mathcal{TS}$ is a set of ~500 additional annotated web page pairs extracted from the same websites as $\mathcal{DS}$.

Technique	F_1 score on $\mathcal{SS}$	F_1 score on $\mathcal{TS}$
PDiff [26]	0.53	0.67
TK-based SVM	0.58	0.68

Table 2: Macro-averaged F_1 scores on $\mathcal{SS}$ and $\mathcal{TS}$

To evaluate the effectiveness of the TK-based classification approach we devised, we firstly extracted, for each web page pair in the dataset, the TK-based similarity features we defined in Section 3. To do so, we leveraged the well-known KeLP library [12]. Then, similarly to [26], we used $\mathcal{DS}$ to train a SVM classifier, and then evaluated classification performance on both $\mathcal{SS}$ and $\mathcal{TS}$, measuring the macro-averaged F_1 classification score. The results of this preliminary evaluation, which are reported in Table 2, show that the proposed TK-based classification approach outperforms *Perceptual Diff* (PDiff) [27], the best state-of-the-art technique among those investigated in [26].

5 FUTURE RESEARCH ROAD MAP

The goal of my Ph.D. is to improve the model inference of web applications by devising novel near-duplicate detection techniques. The promising preliminary results we obtained in the classification task we described in the previous section showed that tree kernel-based similarity features can be effective for the problem at hand. Encouraged by these results, during my Ph.D. we plan to further investigate the potential of Tree Kernels in near duplicate detection and model inference along several research directions.

Firstly, we plan to improve the classification performance. Along this direction, we aim at designing custom TK functions specifically geared towards detecting near-duplicate web pages, as we believe that considering peculiar structural properties of web pages could make TKs more effective. Furthermore, we also plan on defining and assessing additional and more refined similarity features, leveraging for example different kinds of TK functions, such as *Subpath Kernels*, which were also recently applied, although in a different context, to web pages [23].

We aim at implementing the solutions emerging from these studies as open-source extensions of the well-known Crawljax web crawler [18], that will be made freely available to Software Engineering researchers and practitioners.

As for the empirical assessment, we plan to use the same data and experimental procedure used by Yandrapally et al. in their ICSE2020 paper [26], which provides a valuable state-of-the-art benchmark both for near-duplicate classification performances and for the effectiveness of the inferred models.

REFERENCES

[1] Sadia Afroz and Rachel Greenstadt. 2011. Phishzoo: Detecting phishing websites by looking at them. In *2011 IEEE fifth international conference on semantic computing*. IEEE, 368–375.

[2] Francesco Altiero, Anna Corazza, Sergio Di Martino, Adriano Peron, and Luigi Libero Lucio Starace. 2020. Inspecting Code Churns to Prioritize Test Cases. In *IFIP International Conference on Testing Software and Systems*. Springer, 272–285.

[3] Anneliese A Andrews, Jeff Offutt, and Roger T Alexander. 2005. Testing web applications by modeling with FSMs. *Software & Systems Modeling* 4, 3 (2005), 326–345.

[4] Matteo Biagiola, Filippo Ricca, and Paolo Tonella. 2017. Search based path and input data generation for web application testing. In *International Symposium on Search Based Software Engineering*. Springer, 18–32.

[5] Matteo Biagiola, Andrea Stocco, Filippo Ricca, and Paolo Tonella. 2019. Diversity-based web test generation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 142–153.

[6] Andreas Bruns, Andreas Kornstadt, and Dennis Wichmann. 2009. Web application tests with selenium. *IEEE software* 26, 5 (2009), 88–91.

[7] Hari Sankar Chaini and Sateesh Kumar Pradhan. 2015. Test script execution and effective result analysis in hybrid test automation framework. In *2015 International Conference on Advances in Computer Engineering and Applications*. IEEE, 214–217.

[8] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 380–388.

[9] Anna Corazza, Sergio Di Martino, Valerio Maggio, and Giuseppe Scanniello. 2010. A tree kernel based approach for clone detection. In *2010 IEEE International Conference on Software Maintenance*. IEEE, 1–5.

[10] Giuseppe Antonio Di Lucca, Massimiliano Di Penta, Anna Rita Fasolino, and Pasquale Granato. 2001. Clone analysis in the web era: An approach to identify cloned web pages. In *Seventh Workshop on Empirical Studies of Software Maintenance*. 107.

[11] Dennis Fetterly, Mark Manasse, and Marc Najork. 2003. On the evolution of clusters of near-duplicate web pages. In *Proceedings of the IEEE/LEOS 3rd International Conference on Numerical Simulation of Semiconductor Optoelectronic Devices (IEEE Cat. No. 03EX726)*. IEEE, 37–45.

[12] Simone Filice, Giuseppe Castellucci, Danilo Croce, and Roberto Basili. 2015. Kelp: a kernel-based learning platform for natural language processing. In *Proceedings of ACL-IJCNLP 2015 System Demonstrations*. 19–24.

[13] Monika Henzinger. 2006. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. 284–291.

[14] Taichi Ishikawa, Yu-Lu Liu, David Lawrence Shepard, and Kilho Shin. 2020. Machine learning for tree structures in fake site detection. In *Proceedings of the 15th International Conference on Availability, Reliability and Security*. 1–10.

[15] Manuel Leithner and Dimitris E Simos. 2020. XIEv: dynamic analysis for crawling and modeling of web applications. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 2201–2210.

[16] Gurmeet Singh Manku, Arvind Jain, and Anish Das Sarma. 2007. Detecting near-duplicates for web crawling. In *Proceedings of the 16th international conference on World Wide Web*. 141–150.

[17] Alessandro Marchetto, Paolo Tonella, and Filippo Ricca. 2008. State-based testing of Ajax web applications. In *2008 1st International Conference on Software Testing, Verification, and Validation*. IEEE, 121–130.

[18] Ali Mesbah, Engin Bozdag, and Arie Van Deursen. 2008. Crawling Ajax by inferring user interface state changes. In *2008 Eighth International Conference on Web Engineering*. IEEE, 122–134.

[19] Alessandro Moschitti. 2006. Efficient convolution kernels for dependency and constituent syntactic trees. In *European Conference on Machine Learning*. Springer, 318–329.

[20] Alessandro Moschitti. 2006. Making tree kernels practical for natural language learning. In *11th conference of the European Chapter of the Association for Computational Linguistics*.

[21] Filippo Ricca, Maurizio Leotta, and Andrea Stocco. 2019. Three open problems in the context of E2E web testing and a vision: NEONATE. In *Advances in Computers*. Vol. 113. Elsevier, 89–133.

[22] Filippo Ricca and Paolo Tonella. 2001. Analysis and testing of web applications. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*. IEEE, 25–34.

[23] Kilho Shin, Taichi Ishikawa, Yu-Lu Liu, and David Lawrence Shepard. 2021. Learning DOM Trees of Web Pages by Subpath Kernel and Detecting Fake e-Commerce Sites. *Machine Learning and Knowledge Extraction* 3, 1 (2021), 95–122.

[24] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 2016. Clustering-aided page object generation for web testing. In *International Conference on Web Engineering*. Springer, 132–151.

[25] Andrea Stocco, Maurizio Leotta, Filippo Ricca, and Paolo Tonella. 2017. APOGEN: automatic page object generator for web testing. *Software Quality Journal* 25, 3 (2017), 1007–1039.

[26] Rahulkrishna Yandrapally, Andrea Stocco, and Ali Mesbah. 2020. Near-duplicate detection in web app model inference. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 186–197.

[27] Hector Yee, Sumanita Pattanaik, and Donald P Greenberg. 2001. Spatiotemporal sensitivity and visual attention for efficient rendering of dynamic environments. *ACM Transactions on Graphics (TOG)* 20, 1 (2001), 39–65.