Web Application Testing: Using Tree Kernels to Detect Near-duplicate States in Automated Model Inference

Anna Corazza*
anna.corazza@unina.it
Università degli Studi di Napoli Federico II
Naples, Italy

Sergio Di Martino*
sergio.dimartino@unina.it
Università degli Studi di Napoli Federico II
Naples, Italy

Adriano Peron*
adrperon@unina.it
Università degli Studi di Napoli Federico II
Naples, Italy

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

ABSTRACT

Background: 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 site under test. These models, in which states represent features of the web application and transitions represent reachability relationships, can be used for several model-based testing tasks, such as test case generation. However, current exploration techniques often lead to models containing many near-duplicate states, i.e., states representing slightly different pages that are in fact instances of the same feature. 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. **Aims:** As a web page can be naturally represented by its tree-structured DOM representation, we propose a novel near-duplicate detection technique to improve the model inference of web applications, based on Tree Kernel (TK) functions. TKs are a class of functions that compute similarity between tree-structured objects, largely investigated and successfully applied in the Natural Language Processing domain. **Method:** To evaluate the capability of the proposed approach in detecting near-duplicate web pages, we conducted preliminary classification experiments on a freely-available massive dataset of about 100k manually annotated web page pairs. We compared the classification performance of the proposed approach with other state-of-the-art near-duplicate detection techniques. **Results:** Preliminary results show that our approach performs better than state-of-the-art techniques in the near-duplicate detection classification task. **Conclusions:** These promising results show that TKs can be applied to near-duplicate detection in the context of web application model inference, and motivate further research in this direction to assess the impact of the technique on the quality of the inferred models and on the subsequent application of model-based testing techniques.

CCS CONCEPTS

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

KEYWORDS

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

ACM Reference Format:
Anna Corazza, Sergio Di Martino, Adriano Peron, and Luigi Libero Lucio Starace. 2021. Web Application Testing: Using Tree Kernels to Detect Near-duplicate States in Automated Model Inference . In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21), October 11–15, 2021, Bari, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3475716.3484187

1 INTRODUCTION

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, 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 for 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 [28].

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, 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.). To do so, testers typically develop test scripts that, leveraging test automation libraries such as Selenium [7], automate the set of manual operations that the end-user would perform on the GUI of the web application.

*All authors contributed equally to this research.

ESEM '21, October 11–15, 2021, Bari, Italy

ACM ISBN 978-1-4503-8665-4/21/10…$15.00
https://doi.org/10.1145/3475716.3484187

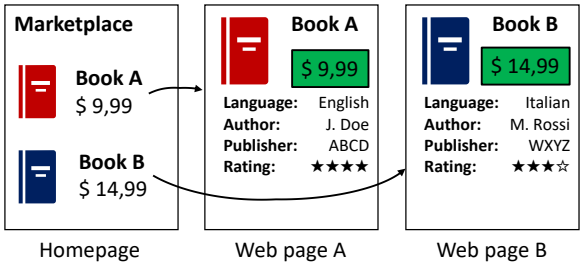

Figure 1: Example of near-duplicate web pages

Developing such test scripts manually, however, is a time consuming and expensive task, often neglected in web projects because of resource constraints [8]. To support these E2E web testing activities, several model-based approaches have been proposed in the software engineering community, including test case generation [3–5, 23, 29] and test artifact generation [31, 32]. To infer the AUT models underlying these approaches, automated exploration techniques [37], also referred to as web application crawling [20], are widely used. Broadly speaking, these crawling-based techniques dynamically and systematically analyze the AUT starting at an initial page, and then explore the application by generating GUI events and checking the responses. When, as a consequence of a fired event, changes in the web page are detected, a new state is added to the model. In these models, a state represents a 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).

From a testing view-point, these inferred models should contain a minimal set of significantly different states, yet adequate to cover all the functionalities of the AUT. In practice, however, models inferred automatically through crawling are often affected by *near-duplicates* [12, 15, 18, 22], i.e., replicas of the same functional web page differing only by minor changes [37]. 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 in terms of contained text, but from a functional testing view-point they are conceptually the same, as both are an instance of the "Show book details" functionality. Nevertheless, a "naive" crawler would assign them to different states, with negative consequences on subsequent model-based testing activities. For instance, test suites generated from these models with many near-duplicate states can be noticeably worse in terms of size and, running time [37].

Despite being crucial for effective model inference activities, few research efforts have been directed towards detecting and discarding such near-duplicate web pages during the crawling process for E2E testing purposes. A first study in this direction was recently presented at ICSE 2020 [37]. In that study, 10 widely used similarity measures for web pages (e.g. Simhash [9], which is used by Google during its indexing process) taken from different domains are applied and compared in the context of near-duplicate detection for web application model inference. The study shows that all of the techniques exhibited limitations and highlighted that there is a need for further research in devising novel approaches geared specifically towards model inference.

To address this issue, in our ongoing research we are investigating novel similarity measures for web pages, specifically designed for supporting model inference for web applications. In particular, as web pages can be naturally represented using their tree-structured Document Object Model (DOM), we 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 [26]. We envision that TKs, thanks to their flexibility and customizability in the definition of what features to weight more for computing the similarity, might be effective tools to capture different types of near-duplicate web pages, improving the overall detection performance.

To understand the potentialities of our proposal, we conducted some preliminary experiments using a freely-available massive dataset of about 100k web page pairs, in which each pair was manually labelled as distinct, near-duplicate, or clone [33]. We compared the classification performance of the proposed approach with the one achieved by the similarity measures investigated in [37]. Preliminary results show that our TK-based approach outperforms all other techniques in the near-duplicate detection classification task.

The remainder of this paper is organized as follows. In Section 2 we survey related works on near-duplicate detection of web pages. In Section 3 we introduce the TK-based near duplicate detection approach we propose, while in Section 4 we describe the preliminary empirical evaluation we conducted. In Section 4.3 we present the emerging results we obtained, and in Section 5 we provide some concluding remarks and a road-map detailing future research efforts.

2 RELATED WORKS

Many techniques from different domains have been defined, in different contexts, to the near-duplicate detection of web pages. 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 have been found to be quite effective [18]. Since performance is a crucial issue in this domain, due to the amount of involved data, content hashing techniques have been widely adopted thanks to their design simplicity and speed of comparison. Notable examples include the shingling algorithm presented by Broder et al. in [6], and the Simhash algorithm [9], which is also used by Google in its web page indexing process [22]. In [18], Henzinger carried out a large scale evaluation of these algorithms on a set of 1.6B distinct web pages, showing that both achieve high precision in detecting near-duplicate web page pairs across different websites, while performing significantly worse in detecting near-duplicated within the same website.

Detecting near-duplicate pages is also a challenge for automatic phishing detection. In this context, 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, techniques from the Computer Vision domain have often been applied to screen captures of the web pages with good results [1, 36].

Among those visual-based techniques, the most fine grained approaches focus on individual pixels composing the image. Examples of such techniques are *color-histogram* [34] and *Perceptual Diff* (PDiff) [39], which have also been successfully applied in a previous web testing work for detecting cross-browser incompatibilities [21]. Some visual approaches operate at a coarser-grained scale, aiming at quantifying structural similarity or at extracting features from images. Structural similarity-based techniques leverage the intuition that images (and in particular screen captures of web applications) are typically highly structured, and their pixels, especially when they are spatially close, exhibit strong dependencies that convey important information about the structure of the represented objects. Similarity measures such as *Structural Similarity Index* (SSIM) [35], which has been successfully applied in the detection of phishing websites [10], take into account these spatial correlations.

Other visual techniques are based on image hashing, aiming at computing identical or nearly-identical digests for similar images, e.g. the screen captures corresponding to near-duplicate web pages [17]. Examples of image hashing algorithms include block-mean hash [38] and perceptual hash (pHash) [40].

Less work, however, has been directed towards the detection of near duplicate web pages *within* the same web application and with the specific goal of supporting automated model inference. Notable examples are the works presented in [14, 31], in which the authors apply the *Robust Tree Edit Distance* (RTED) [27] metric to respectively detect duplicate and near-duplicate web pages during dynamic crawling of web applications, and to reduce the size of the inferred models by clustering duplicate and near-duplicate states. More recently, in [37], Yandrapally et al. presented a comparative study in which 10 different near-duplicate detection techniques from different domains (including the aforementioned simhash, PDiff, color-histogram, SSIM, pHash, RTED) are applied and evaluated in the context of model inference. That study showed that none of the considered algorithms borrowed from the domains of information retrieval and computer vision "*is able to accurately detect all functional near-duplicates within apps*", and "*underlined the need for further research in devising techniques geared specifically toward web test models*".

3 NEAR-DUPLICATE DETECTION WITH TREE KERNELS

As done in the work of Yandrapally et al. [37], we frame the near-duplicate detection problem as a multiclass classification problem. In particular, given a pair of web pages from the same web applications, 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.

In this section, we start by giving some preliminary notions on Tree Kernel functions in 3.1, and then we introduce the Tree Kernel-based near-duplicate detection approach we are investigating in 3.2.

3.1 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 [26], 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 [11], and their usage is also being investigated for test case prioritization tasks [2]. More recently, [19, 30] 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 [25]:

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

3.2 The proposed approach

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 3, can be naturally modelled using their tree-structured Document Object Model (DOM) representation. Since some preliminary experiments highlighted that none of the most common TK functions was able to detect all kinds of near-duplicates effectively, we devised a solution combining the similarity score computed by three different TK functions, namely 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

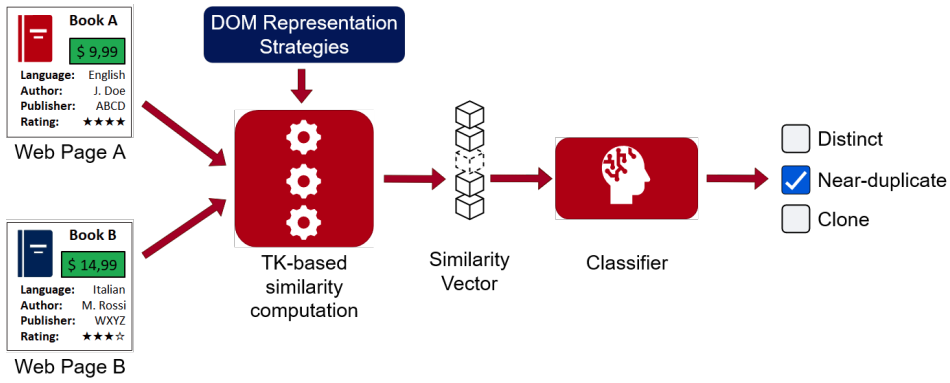

Figure 2: Overview of the proposed TK-based approach

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

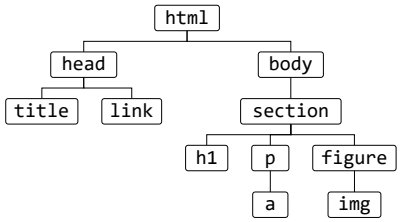

Figure 3: An HTML document and its DOM representation

Table 1: Considered DOM representation strategies

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.

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 the three considered TK functions and the three DOM representation strategies, the similarity vector feeding the classifier has nine components for each pair of web pages. Leveraging these similarity vectors and existing open datasets with annotated web page pairs, we use supervised learning approaches to train an *ad-hoc* classifier. The proposed approach is summarized in Figure 2.

4 PRELIMINARY EMPIRICAL EVALUATION

The goal of our preliminary experiment is to evaluate the effectiveness of the proposed TK-based approach in correctly detecting near-duplicate web pages, w.r.t. other state-of-the-art techniques. To this end, we formulated and investigated the following research question:

RQ. As compared to state-of-the-art approaches (i.e., the 10 techniques investigated in [37]), does the TK-based approach we propose allow achieving better near-duplicate detection performance?

4.1 Employed Data

We preliminarily evaluate the proposed approach using the same data and experimental procedure presented by Yandrapally et al. in [37]. This way, our results can be directly compared with the state-of-the-art. In particular, [37] compared 10 different near-duplicate detection techniques (which we use as a baseline) from the different domains of Computer Vision and Information Retrieval, and provided a large dataset of about 100k manually annotated same-website web page pairs, obtained by crawling both real-world websites and open source applications in a controlled environment.

The dataset they made available consists 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. The considered web applications were used in many previous works on web testing, and are briefly described in Table 2.
- $\mathcal{DS}$ 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 $\mathcal{DS}$.

For more details on the dataset and on the classification procedure the authors employed, we refer the interested reader to [37].

Table 2: The considered open source web applications

Web App	Description
Addressbook	Simple address and phone book.
PetClinic	Management of a veterinary clinic.
Claroline	Collaborative e-learning platform.
Dimeshift	Expense tracker.
PageKit	Modular Content Management System.
Phoenix	Project management.
PPMA	Password Manager.
MRBS	Meeting Room Booking System.
MantisBT	Bug Tracker.

Table 3: Macro-averaged F_1 scores on $\mathcal{SS}$ and $\mathcal{TS}$

Technique	$\mathcal{SS}$	$\mathcal{TS}$	Average
PDiff	0.53	0.67	0.60
BlockHash	0.54	0.62	0.58
SSIM	0.53	0.62	0.57
Levenshtein	0.48	0.59	0.54
RTED	0.50	0.57	0.54
SIFT	0.47	0.61	0.54
pHASH	0.40	0.63	0.52
TLSH	0.44	0.56	0.50
Color-histogram	0.37	0.52	0.44
Simhash	0.17	0.48	0.33
TK-based SVM	0.58	0.68	0.63

4.2 Experimental Procedure

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 vector we defined in Section 3. To do so, we leveraged the well-known open-source KeLP library, which features a state-of-the-art implementation of the tree kernel functions we employed [16]. Then, similarly to [37], we used $\mathcal{DS}$ to train a SVM classifier, representing each web page pair with its computed similarity vector. Finally, we evaluated classification performance on both $\mathcal{SS}$ and $\mathcal{TS}$, measuring for each of these datasets the macro-averaged F_1 classification score, as done in [37].

The software component we implemented to extract the similarity vectors, as well as the R scripts we used to train and evaluate the SVM classifier, is open-source and available in the replication package at the public doi: https://doi.org/10.6084/m9.figshare.14975178.

4.3 Emerging results and discussion

The results of this preliminary evaluation are reported in Table 3, in which the first 10 rows show the best results obtained by the 10 state-of-the-art techniques in [37], and the last one the results obtained by our TK-based approach. These figures show that the proposed TK-based classification solution performs better than all the considered baseline approaches, achieving a 5% improvement on $\mathcal{SS}$ and a 1% improvement on $\mathcal{TS}$ w.r.t. *Perceptual Diff* (PDiff), the best technique among those investigated in [37].

It is worth noting that PDiff is a computationally expensive visual-based technique, and this can limit its application to model inference of large applications. Indeed, as reported in [37], when using PDiff the model inference process could only explore 4 states per minute on average, as compared to faster, DOM-based approaches such as RTED, which could explore 25 states per minute. The approach we propose is based on the DOM of the web pages, and thus is more efficient than PDiff. When considering the best DOM-based techniques, namely RTED and the Levenshtein distance, our TK-based approach improves classification performance on both datasets by approximately 10%.

4.4 Threats to Validity

External validity threats concern the generalizability of the results. In this study, the employed dataset consists mainly of web page pairs from nine open-source web applications, which may not be representative of complex, real-world commercial applications. To mitigate this threat, the authors of the original dataset selected nine open source web applications from different domains, having different sizes, and implemented with different technologies. To further improve the generalizability of the results, in future works we plan to extend the original dataset by considering more web applications, including possibly commercial solutions.

A possible threat to *internal validity*, concerning uncontrolled factors that may have affected the results, is represented by the manually created web page pair annotations. This threat is unavoidable, since there exists no automated method to compute the ideal classification of web pages. To minimize this threat, the authors of the original dataset created, in isolation, a ground truth, and then established a discussion to reach an agreement [37].

5 CONCLUSIONS AND FUTURE RESEARCH

Automated exploration techniques (a.k.a. crawling) are widely used to infer state-based models of web applications. From a functional testing viewpoint, such inferred models should be as compact as possible, i.e., contain a minimal set of significantly different states, while maintaining completeness, adequately covering all the functionalities of the web application. In practice, however, models inferred automatically through state exploration are often affected by *near-duplicate* states, i.e., states corresponding to replicas of the same functional web page differing only by minimal, insignificant changes. This has a negative impact on the quality of the models and on the subsequent model-based testing tasks, adversely affecting, for example, size, running time, and achieved coverage of generated test suites.

In this paper, we introduced a novel approach to near-duplicate detection, based on Tree Kernel functions, a class of kernel functions largely used in the Natural Language Processing domain to measure the similarity of tree-structured objects. To preliminarily assess the effectiveness of the proposed approach, we conducted an empirical evaluation based on an open dataset of approximately 100k annotated web page pairs, in which we compared the near-duplicate detection performance of our approach against 10 baseline techniques. Preliminary results were promising and showed that the TK-based approach performs better than all the baselines.

In future works, 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 extending the current approach, including more components in our similarity vectors. To this end, for example, we intend to experiment with more refined DOM representation strategies and with different kinds of TK functions, such as *Subpath Kernels*, which were also recently applied, although in a different context, to web pages [30]. We aim at implementing the solutions emerging from these studies as open-source extensions of the well-known Crawljax web crawler [24], 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 [37], which provides a valuable state-of-the-art benchmark both for near-duplicate classification performances and for the quality of the inferred models. Moreover, we plan to investigate the effectiveness of TK-based near-duplicate detection also in fully-automated E2E testing [13] of mobile applications, leveraging the tree-like layout structure of their GUI.

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] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. 1997. Syntactic clustering of the web. *Computer networks and ISDN systems* 29, 8-13 (1997), 1157–1166.

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

[8] 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.

[9] 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.

[10] Teh-Chung Chen, Scott Dick, and James Miller. 2010. Detecting visually similar web pages: Application to phishing detection. *ACM Transactions on Internet Technology (TOIT)* 10, 2 (2010), 1–38.

[11] 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.

[12] 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.

[13] Sergio Di Martino, Anna Rita Fasolino, Luigi Libero Lucio Starace, and Porfirio Tramontana. 2021. Comparing the effectiveness of capture and replay against automatic input generation for Android graphical user interface testing. *Software Testing, Verification and Reliability* 31, 3 (2021), e1754.

[14] Amin Milani Fard and Ali Mesbah. 2013. Feedback-directed exploration of web applications to derive test models.. In *ISSRE*, Vol. 13. 278–287.

[15] 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.

[16] 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.

[17] Abhishek Gangwar, Eduardo Fidalgo, Enrique Alegre, and Vıctor González-Castro. 2018. PhishFingerprint: A Practical Approach for Phishing Web Page Identity Retrieval Based on Visual Cues. In *International Conference of Applications of Intelligent Systems*.

[18] 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.

[19] 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.

[20] 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.

[21] Sonal Mahajan and William GJ Halfond. 2014. Finding HTML presentation failures using image comparison techniques. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 91–96.

[22] 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.

[23] 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.

[24] 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.

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

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

[27] Mateusz Pawlik and Nikolaus Augsten. 2015. Efficient computation of the tree edit distance. *ACM Transactions on Database Systems (TODS)* 40, 1 (2015), 1–40.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] Near Duplicate Study. 2019. Near-Duplicate Study DataSet. https://doi.org/10.5281/zenodo.3376730

[34] Michael J Swain and Dana H Ballard. 1992. Indexing via color histograms. In *Active perception and robot vision*. Springer, 261–273.

[35] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing* 13, 4 (2004), 600–612.

[36] Liu Wenyin, Guanglin Huang, Liu Xiaoyue, Zhang Min, and Xiaotie Deng. 2005. Detection of phishing webpages based on visual similarity. In *Special interest tracks and posters of the 14th international conference on World Wide Web*. 1060–1061.

[37] 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.

[38] Bian Yang, Fan Gu, and Xiamu Niu. 2006. Block mean value based image perceptual hashing. In *2006 International Conference on Intelligent Information Hiding and Multimedia*. IEEE, 167–172.

[39] 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.

[40] Christoph Zauner. 2010. Implementation and benchmarking of perceptual image hash functions. (2010).