Fine-grained Source Code Similarity with Tree Kernels to Support Software Testing

Francesco Altiero, Luigi Libero Lucio Starace,
Anna Corazza, Sergio Di Martino, Adriano Peron
Laboratorio di Artificial Intelligence, Privacy and Applications (AIPA)
Dipartimento di Ingegneria Elettrica e delle Teconologie dell'Informazione
Università degli Studi di Napoli Federico II, Italy
{francesco.altiero, luigiliberolucio.starace, anna.corazza, sergio.dimartino, adrperon}@unina.it

Abstract

Computing a meaningful similarity among fragments of source code can be valuable in many software testing scenarios, such as regression test prioritization or model inference/test generation for web applications. In both these cases, a tree-structured representation of the source code could be exploited to provide useful structural information. Tree Kernels are a class of kernel functions, largely used in Machine Learning and Natural Language Processing, specifically suited for computing the similarity between tree-structured objects. In this work, we report on our current investigations in the definition of Tree Kernel-based solutions to improve software testing. Preliminary experimental results are encouraging and motivate further research in applying Tree Kernels to these software testing tasks.

1 Introduction

In different tasks of Software Engineering, tree-structured data arises quite naturally. For example, source code can be naturally represented by its Abstract Syntax Tree (AST) representation, and many document formats such as HTML or XML are typically defined by their tree-structured Document Object Model (DOM). Measuring the similarity between such tree-structured objects is crucial for many different software engineering tasks. Tree Kernel (TK) functions are a particular family of kernel functions specifically meant to evaluate similarity between two tree-structured objects. In this paper, we consider two of such scenarios: identifying critical changes between two subsequent versions of the same software to support regression test prioritization, and identifying near-duplicate web pages when automatically inferring state-based models for web applications.

In the following, we start by briefly presenting tree kernel functions in Section 2. Then, in Sections 3 and 4, respectively, we describe the regression test prioritization and the web application model inference scenarios, with particular emphasis on the rationale motivating the application of tree kernels. A section containing conclusions and future work ends the paper.

2 Tree Kernel functions

TKs have been extensively studied in different tasks of Natural Language Processing [Moschitti, 2006b], usually involving syntactic trees. Among their applications in the Software Engineering domain, in [Corazza *et al.*, 2010] they have been applied to Abstract Syntax Tree representations of source code for clone detection. More recently, in [Ishikawa *et al.*, 2020; Shin *et al.*, 2021] TKs are used to compute similarity between web pages aiming at fake website detection.

The computation of the similarity between two trees is based on the representation of each tree as a set of *tree fragments*, each given by a connected subset of nodes and edges of the original tree. The similarity between tree fragments is then used to build the similarity between the original trees. Different classes of TK functions can be defined, characterized by different definitions of tree fragments [Moschitti, 2006a], including:

- **Subtree Kernels**: only proper subtrees of the original trees can be tree fragments, i.e. if a node belongs to a fragment, then also all its descendants do.
- **Subset Tree Kernels**: where if a node belongs to a fragment then all or none of its children do. In this way, also incomplete subtrees, limited at any arbitrary depth, can be tree fragments.
- **Partial Tree Kernels**: this is the most general definition, where if a node belongs to a fragment, then any number of its children do, including zero.

The different definitions of tree fragments allow to capture different properties of the tree to be compared and therefore result in different performance of the system under development.

3 Tree Kernels for Regression Test Prioritization

Regression testing is a practice aimed at providing confidence that, after maintenance activities, the unchanged parts of the software still behave as expected. Anyhow, due to resources constraints and/or to the rapid evolution pace, it may not be possible to re-execute all the test cases after each change. In this scenario, we consider *prioritization*, which aims at producing an ideal ranking among test cases. With such a ranking, even when the test suite execution is interrupted (e.g.: due to time or resources constraints), the fault-revealing capability is maximized. In other words, regression test prioritization aims at ranking test cases so that the tests that reveal faults are executed as early as possible.

We envision a novel test case prioritization approach that takes into account also the structural nature of the changes introduced by programmers in the new version of the code to find the ones more prone to introduce faults. Then, we aim at prioritizing tests covering these more critical changes.

The novel proposal presented in [Altiero *et al.*, 2020] aims at assessing the structural similarity between the source code of two software versions, in order to include this finer-grained information in the evaluation of the *code churn*. In fact, it combines this information with more traditional coverage analysis to find the tests which are more likely to find errors, so that they can be given higher priority with respect to the others. To extract such structural information, we apply TK to the Abstract Syntax Tree representations of source code aiming to identify fault-prone changes. Then, the rank of tests is based on a combination of the score obtained by TKs and more traditional coverage-based metrics.

This new strategy has been assessed on real-world software artifacts with seeded faults and its performance has been compared with well-known prioritization techniques, using fault detection rate metrics. The comparison showed that the integration of TKs improves the ranking performance and therefore we can conclude that a suitable use of the TKs is effective in highlighting fault-prone changes in the code and that this information can be valuable to define a more effective prioritization of test cases.

4 Tree Kernels for Web Application Model Inference and Near-duplicate Detection

End-to-End testing of web applications often exploits state-based models of the site under test. In these models, states represent features of the web application, while transitions correspond to reachability relationships. The construction of such models can be automated by means of systematic exploration techniques (a.k.a. crawling). Typically, a crawler can infer a state-based model of a web application in a depth-first fashion by starting at a given url (the homepage of the web application), and by systematically generating user events (e.g.: click on links, fill text fields, submit forms, etc.). After executing each event, the crawler checks whether the resulting web page is already represented in the model. If not, a new state representing the feature is created. The crawling process continues by generating more events until a given time budget is exhausted or all the possible events have been generated.

Several model-based testing techniques can be applied to such state-based models, including test case generation [Biagiola *et al.*, 2017; Biagiola *et al.*, 2019] or test artifacts generation [Stocco *et al.*, 2017; Stocco *et al.*, 2016]. Of course, an accurate construction of the model improves the performance of these techniques. A known problem affecting this automated model inference task is represented by the so-called *near-duplicate* states, i.e., states representing slightly different pages that are in fact instances of the same feature [Yandrapally *et al.*, 2020].

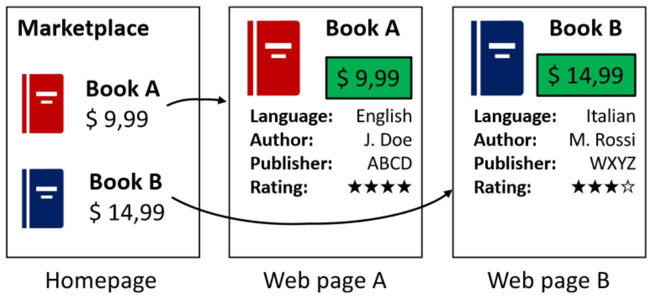

Figura 1: Example of near-duplicate states [Corazza *et al.*, 2021].

As an example, let us consider Figure 1, depicting three web pages from an imaginary bookstore web application. The homepage features a list of available books. Assuming that the crawling started at the homepage, initially model only contains the homepage state. After generating a click event on one of the books, the crawler is redirected to a detail web page with additional information on that book. The book detail state is added to the model as well. Subsequently, the crawler generates a click event on the other book, resulting in the detail page for that book. The detail pages for the two books are of course different in terms of contained text, figures, etc., 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 create a new state for the second book detail page, as that page is different from all the ones it has previously visited. Web Page A and Web Page B are near-duplicate states, and including both on them has a negative impact on the precision and completeness of the resulting model. This, of course, 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.

Hence, effective techniques to detect near-duplicate states during the crawling process could prove to be very valuable in many industrial scenarios, as also witnessed by the interest of our industrial partners NetCom Group[1], a large Italian consulting enterprise active, among other things, in the multimedia and web testing domain, and with which we are actively collaborating in these studies.

Following [Yandrapally *et al.*, 2020], we define near-duplicate detection as a multiclass classification problem, taking in input a pair of web pages from the same web applications, and producing as output one of the following distinct categories:

- **Clone**, if the two pages have the same semantic, functional and perceptible features.
- **Distinct**, if the two pages present any semantic or functional difference.
- **Near-duplicate**, if the two pages are noticeable different, but the overall functionality being exposed is the same.

[1]https://www.netcomgroup.eu/en/

TKs can be effectively applied to this classification task because a web page can be naturally represented by its tree-structured DOM representation. To experimentally assess the performance of the proposed approach in detecting near-duplicate web pages, a freely-available massive dataset of about 100k manually annotated web page pairs has been considered. The classification performance of the proposed approach has been compared with other state-of-the-art near-duplicate detection techniques [Starace, 2021; Corazza *et al.*, 2021]. Preliminary results show that our approach performs better than state-of-the-art techniques in the near-duplicate detection classification task.

5 Conclusions and Future Work

TK functions are a class of kernel functions, popular in other disciplines such as natural language processing, that can be used to measure the similarity between tree-structured objects, that arise quite naturally in many different Software Engineering scenarios. In this work, we consider two of such scenarios, namely regression test prioritization and model inference for web applications, and present the novel TK-based solutions we are currently investigating.

Our preliminary experiments show that capturing structural information by means of tree kernels leads to promising results in both the considered scenarios. However, in both cases the research is still ongoing, and currently focusing on integrating this information with other available sources and on performing more extended experimental assessments.

More in detail, for what concerns the regression test prioritization scenario, future works include significantly extending our empirical evaluation by also considering software artifacts with real faults, and not only seeded ones. Furthermore, we plan to use particular TKs which can also asses the semantical similarity between code ASTs. In this case, TKs can give insights not only on structural changes, but also detect error-prone code churns based on the nature of changes. We expect that taking into account this enchanced information can further improve the performance of our regression test prioritization techniques.

As for the model inference for web applications scenario, the promising results discussed in [Corazza *et al.*, 2021] show that TKs can be applied to near-duplicate detection, and motivate further research in this direction to assess the impact of the technique on the quality of the inferred models and on the application of test case generation and other model-based testing techniques.

In the case of near-duplicate detection and model inference, we see several research directions to further investigate the potential of Tree Kernels. The first actions to consider aim at improving the classification performance. To such aim, we plan to customize TK functions specifically geared towards detecting near-duplicate web pages. In particular, they should take into account peculiar structural properties of web pages. On the other hand, we want to explore the possibility of a more precise input representation which would give more information to include into the similarity estimation. Eventually, we want to implement the resulting modules as open-source extensions of the Crawljax web crawler [Mesbah *et al.*, 2008]. The performance of the resulting system will then be experimentally assessed on the same data and with the same experimental procedure used in [Yandrapally *et al.*, 2020], which provides a valuable state-of-the-art benchmark both for near-duplicate classification performances and for the quality of the inferred models.

Furthermore, we plan to apply the proposed techniques not only on simple open source web applications, but to verify their generalizability also in real-world industrial scenarios featuring more complex web applications, driven by our collaboration with NetCom Group. Moreover, we plan to investigate the effectiveness of TK-based near-duplicate detection also in fully-automated E2E testing [Di Martino *et al.*, 2021] of mobile applications, leveraging the tree-like layout structure of their GUI.

Riferimenti bibliografici

[Altiero *et al.*, 2020] Francesco Altiero, Anna Corazza, Sergio Di Martino, Adriano Peron, e Luigi Libero Lucio Starace. Inspecting code churns to prioritize test cases. In *Testing Software and Systems - 32nd IFIP WG 6.1 International Conference, ICTSS 2020, Naples, Italy, December 9-11, 2020, Proceedings*, volume 12543 of *Lecture Notes in Computer Science*, pages 272–285. Springer, 2020.

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

[Biagiola *et al.*, 2019] Matteo Biagiola, Andrea Stocco, Filippo Ricca, e Paolo Tonella. 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*, pages 142–153, 2019.

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

[Corazza *et al.*, 2021] Anna Corazza, Sergio Di Martino, Adriano Peron, e Luigi Libero Lucio Starace. Web application testing: Using tree kernels to detect near-duplicate states in automated model inference. In Filippo Lanubile, Marcos Kalinowski, e Maria Teresa Baldassarre, editors, *ESEM '21: ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, Bari, Italy, October 11-15, 2021*, pages 37:1–37:6. ACM, 2021.

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

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

[Mesbah *et al.*, 2008] Ali Mesbah, Engin Bozdag, e Arie Van Deursen. Crawling ajax by inferring user interface state changes. In *2008 Eighth International Conference on Web Engineering*, pages 122–134. IEEE, 2008.

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

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

[Shin *et al.*, 2021] Kilho Shin, Taichi Ishikawa, Yu-Lu Liu, e David Lawrence Shepard. Learning dom trees of web pages by subpath kernel and detecting fake e-commerce sites. *Machine Learning and Knowledge Extraction*, 3(1):95–122, 2021.

[Starace, 2021] Luigi Libero Lucio Starace. Detecting near-duplicate states in web application model inference: a tree kernel-based approach, 2021. Presented at the ISSTA'21 Doctoral Symposium.

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

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

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