

“Session-Based Recommender Systems for Action Selection in GUI Test Generation” by Varun Nayak and Daniel Kraus, submitted to the 3rd IEEE Workshop on NEXt level of Test Automation (NEXTA) 2020. This is a preprint of the accepted version of this paper. The paper starts on the next page, after this information.

© 2020 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works.

Session-Based Recommender Systems for Action Selection in GUI Test Generation

Varun Nayak*, Daniel Kraus†
ReTest GmbH

Haid-und-Neu-Straße 7
76131 Karlsruhe, Germany

Email: *varun.nayak@retest.de, †daniel.kraus@retest.de

Abstract—Test generation at the graphical user interface (GUI) level has proven to be an effective method to reveal faults. When doing so, a test generator has to repeatedly decide what action to execute given the current state of the system under test (SUT). This problem of action selection usually involves random choice, which is often referred to as monkey testing. Some approaches leverage other techniques to improve the overall effectiveness, but only a few try to create human-like actions—or even entire action sequences. We have built a novel session-based recommender system that can guide test generation. This allows us to mimic past user behavior, reaching states that require complex interactions. We present preliminary results from an empirical study, where we use GitHub as the SUT. These results show that recommender systems appear to be well-suited for action selection, and that the approach can significantly contribute to the improvement of GUI-based test generation.

Index Terms—Test generation, testing and debugging, information filtering.

I. INTRODUCTION

System tests through the *graphical user interface (GUI)* are important since they stimulate software from end to end, i.e., somewhat from a user’s perspective down to persistence layers such as databases. When used wisely, they can be a powerful part of a testing strategy. However, such tests usually have a bad reputation because they tend to be “[...] brittle, expensive to write, and time consuming to run.” [1] Both academia and the industry try to overcome these issues by automatically generating GUI tests; not just to free developers and testers from the burden of test creation and maintenance, but to reduce the overall costs—without compromises and at the pace required [2].

While generated tests cannot fully compensate hand-crafted test cases, the future of testing is said to drastically increase the use of automation [3]. Nowadays, test generators already yield impressive results in a wide range of application areas. Sapienz [4], for example, found 558 previously unknown crashes in an empirical study with more than 1,000 Android apps from the Google Play store. Meanwhile, the former research project has been deployed at Facebook, where it is now used to automatically test the mobile apps of, e.g., Instagram, WhatsApp and Facebook itself [5].

When it comes to GUI-based test generation, a crucial part is to decide what action to execute next given the current state of the system under test (SUT). Many of today’s approaches rely on random choice, a.k.a. *monkey testing*. This is

sometimes combined with techniques like (meta-)heuristics or machine learning (ML) to improve the generated tests. For instance, ant colony optimization [6], genetic programming [7], ML-enhanced evolutionary computing [8], data mining [9], deep learning [10], Q-learning [11] or other reinforcement learning algorithms [12]. Yet, only a few actually focus on creating human-like sequences of actions, e.g., to allow a test generator to get behind “gate GUIs” [13] such as login screens or non-trivial forms. And although random testing is effective in finding relevant faults, it tends to miss bugs humans do reveal [14]. Therefore, generating sequences that mimic past user behavior might help to reduce this gap.

We propose a novel approach to action selection in GUI-based test generation by leveraging *recommender systems*. Recommender systems are a well-studied field and they form the core of many successful businesses like Netflix [15] or YouTube [16], for which targeted recommendations are indispensable. We investigate a possible intersection between recommender systems and the problem of action selection by mapping GUI actions to items and sessions within a SUT to users. Provided an adequate amount of data, our approach is able to predict actions a user likely would perform in the current state. By using a *session-based* recommender system as our model, we not just suggest single actions, but sequences of actions. This allows a test generator to be guided through states that require complex user interactions.

First, we give a brief introduction to recommender systems and some advances relevant for this paper in Section II. Section III outlines our technical approach, where we describe the overall design and various implementation details. In Section IV, we conduct a first empirical study on top of GitHub by mixing real-world and synthetic data. Afterwards, we summarize our findings and report on our planned future work in Section V.

II. RECOMMENDER SYSTEMS

Receiving recommendations of different forms has become a part of our daily online experience in a variety of application domains such as e-commerce, social media and content streaming. Internally, such systems analyze the past behavior of individual users to detect patterns in data. On typical online sites, various types of user actions can be recorded, e.g., that a user views an item or makes a purchase. These recorded

actions and the detected patterns are then used to provide recommendations to the user. In this context, the entity being recommended is called *item*, and the entity that receives the recommendation is referred to as the *user*.

The basic models for recommender systems work primarily with user-item interactions such as ratings or like/dislike, or attributes like user interests or item properties. Based on this, there are two main approaches to recommender systems: *collaborative filtering* and *content-based*. Traditional techniques such as matrix factorization have treated user-item interactions as flat, matrix-structured data, often ignoring the temporal structure and order within the data [17]. Being able to predict a user’s short-term interests in an online session is a highly relevant problem in practice, e.g., to adapt to item viewing and purchase activities in e-commerce. Within such application domains, the items have to be recommended in a certain order, or the recommendation of one item only makes sense after some other event has happened.

Session-based recommender systems consider the information embedded in between sessions and treat sessions as the basic recommendation unit. A session could be a set of items or a collection of actions consumed in one event or in a particular period of time. When dealing with such sequential data, recurrent neural networks (RNNs) are being heavily used [18]. But practical applications involve temporal dependencies spanning many time steps where the network is often unable to propagate useful information from the output end of the model back to the layers near the input end, known as the vanishing gradient problem [19].

In 2016, Hidasi et al. [20] presented a *gated recurrent unit (GRU)*-powered RNN for session-based recommendations and called this method *GRU4Rec*. A GRU is a more elaborate model of an RNN that can deal with the aforementioned problem(s) [21]. The gatings within these units essentially learn when and how much to update the hidden state of the unit. This enables more accurate recommendations for session-based data.

III. OUR TECHNICAL APPROACH

We design our approach on top of the work by Hidasi et al. as illustrated in Figure 1. We adopt their general network architecture, but specialize it for our purposes.

The input to our model is a batch of sessions where each session is encoded as a sequence of action IDs. Action IDs are derived from the targeted GUI element and the action performed on it. That is, two actions only produce the same ID if they target the same element with the same action (ignoring possible input data like text). An RNN layer is added next, which consists of GRU or long short-term memory (LSTM). (In our evaluation below, we explore multiple network types.) This layer is expected to learn the temporary patterns in the action-selection behavior. The following dense layer converts this information into a probability distribution over the given action IDs—the output of our model. Thus, we get a stream of ranked actions that is based on past behavior of actual users a test generator can choose from.

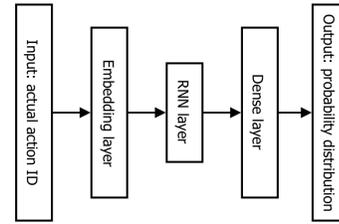


Fig. 1. General architecture of our network based on [20].

To implement our approach, we are using the GRU4Rec library based on [20], [22]. It adds several extensions to a sequence modeling architecture like ours. For example, session-parallel mini-batches, mini-batch-based output sampling and the use of a pairwise ranking loss function. Sequence-to-sequence models produce the result by one item at a time, in other words, by solving a classification problem at each time step. According to Hidasi et al., pairwise ranking losses are expected to give a better performance with the given network setup. The loss function compares the rank of pairs of a positive and a negative item and enforces that the rank of the positive item should be lower than that of the negative one.

For more details on the basic network setup, please refer to the original paper(s). Next, we conduct a first empirical study, show how the network can be trained and how it performs.

IV. A FIRST EMPIRICAL EVALUATION

According to the World Quality Report 2018-19 [23], data is a main obstacle when it comes to the adoption of artificial intelligence (AI) in testing. The problem of data scarcity is an important factor since data is at the core of any ML project. This issue is especially challenging for young or small organizations, because only rarely there are cooperations with large enterprises where sufficient data is available.

As we were struggling with small data too, we have designed an extract, transform, load (ETL) pipeline to increase the amount of real-world data by adding synthetic data, so that we get a first impression of how our approach performs. We decided to target web applications (i.e., web-based GUIs), where we picked GitHub as the SUT; a popular code hosting and development platform. This setup allows insights based on (i) a mature and widely-used target platform, (ii) a sufficiently complex and well-known SUT.

To create real-world data, we recorded 50 of our own user sessions on GitHub using the Selenium IDE, a record-and-playback tool available as a Chrome and Firefox extension. We exported these sessions as Java tests, where every test case represents a user session. Each exported test was executed with a custom Selenium WebDriver, which allows us to extract training data as CSV. Note that the Selenium IDE comes with the functionality that when the default locator—a particular GUI element property such as an ID, typically used in test scripts to locate elements—doesn’t find an element, it will fall back to other available means. This fallback mechanism ensures that most recorded tests don’t fail, e.g., due to recording inaccuracies. The exported Java code does not have this

TABLE I
REAL-WORLD DATA SAMPLES AFTER PRE-PROCESSING.

Session ID	Action IDs sequence	Timestamp
1	(151, 1, 2, 3, 4)	1568573073
2	(151, 4, 5, 3, 1, 2, 3, 4)	1568573079
3	(6, 7, 8, 9, 10, 11, 12, 2, 3, 4)	1568573088
4	(151, 4, 5, 3, 4, 1, 2, 3, 4)	1568573099
5	(6, 109, 110, 2, 3, 4)	1568573362

feature and only uses a single locator, which is why we had to manually fix many locators to avoid runtime failures. The poor code export quality when using the Selenium IDE is a major bottleneck in the proposed ETL pipeline that we aim to address as part of our future work (see Section V).

One of the key parts in the pre-processing step is the assignment of accurate action IDs. The tests exported via the Selenium IDE already contain the absolute XPath for each element, we combine this information with the web page the element appears on and the performed action type to derive the action ID. Table I illustrates some resulting data samples, where every session is of arbitrary length (the actions executed by the user) and represented by a sequence of action IDs.



Fig. 2. Action ID distribution within the recorded user sessions.

When it comes to synthetic data, Wu et al. [24] formalize the problem of generating such datasets using the maximum entropy principle for categorical data, which captures the characteristics of the underlying data. Figure 2 shows the distribution of action IDs within the recorded user sessions. As can be seen, some actions are more common across sessions. These frequent actions usually carry a deeper meaning and represent short-term user goals that correspond to common use cases like a login procedure. The assumption we made is that most sessions will have such recurring patterns in the recorded interactions and in-between a user will perform arbitrary actions. The synthetic sessions we generated still hold these properties, but have been mixed up with randomly created action IDs. In practice, click botnets may also create a considerable amount of traffic [25], so we additionally interjected random noise to represent spam and the like.

The resulting dataset is summarized in Table II. We split this data into a training set (80%) and a test set (20%) for evaluation. In the context of recommender systems, we are most likely interested in recommending an item from the top- n list of items. Therefore, we calculate relevant metrics with regards to the first n actions instead of all actions. Precision

TABLE II
SUMMARY OF THE USED DATASET.

Real-world sessions	50	Avg. no. of actions	14.12
Synthetic sessions	3,476	Min. no. of actions	1
Distinct actions	522	Max. no. of actions	49

at n is the proportion of recommended actions present in the top- n list that are relevant. Recall at n is the proportion of relevant actions found in the top- n recommendations. Mean reciprocal rank (MRR), which is important in cases where the order of recommendations matter, is the mean of reciprocals of the rank from all queries. The reciprocal rank is set to zero if the rank is above n .

We further followed the practice of Quadrana et al. [26], where items from each session in the test set are grouped together to form a sequence, and each sequence is further split into the user profile and ground truth. The user profile is composed of the first event in the sequence that is fed into the system and used to compute recommendations. The ground truth is composed of the remainder of the sequence that is used for performance evaluation. Items are revealed incrementally, then the evaluation is performed after each new item. This helps to evaluate the recommendation quality in a setting where user profiles are revealed sequentially. Metrics are averaged over each sequence and then averaged over all.

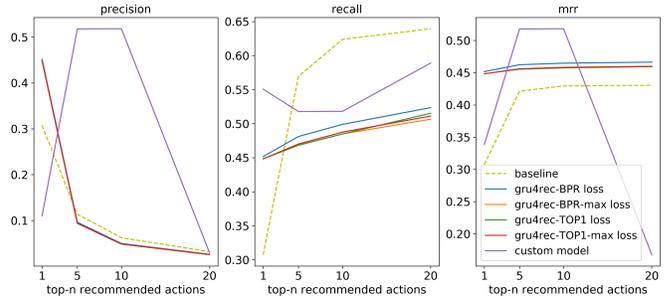


Fig. 3. Precision, recall, MRR at 1, 5, 10, 20 for the different network types.

As a baseline, we used a simple k -nearest neighbor recommender based on an item-to-item similarity. In this setting, the similarity matrix is pre-computed from the available session data, i.e., actions that are often executed together in sessions are deemed to be similar. This similarity matrix is then used during the session to recommend the most similar actions to the one the user has currently performed. We compared this model to different GRU-based network types with different losses, as well as a custom model using LSTM and cross-entropy loss. The models converged between 5 – 25 epochs, depending on the loss function and the amount of data. The evaluation results are shown in Figure 3.

An observation we made is that the MRR is roughly within the range $[0.2, 0.5]$ across all the evaluated recommendation list lengths. This indicates that the best relevant actions were retrieved between top-5 and top-10. The MRR, however, seems to saturate, which we assume is a consequence of the data deficit. The baseline performance being on-par with the other models is most likely owing to short average session length.

Precision and recall both indicate the accuracy of the models. Precision for all models except the custom one is very high at the top-1 recommendation, then it continues to drop as the recall climbs up. This is because in order to recall everything, it is required to keep generating results which are not accurate, hence, lowering the precision.

We also observed that the models did not significantly outperform the baseline in the top-5 recommendation and beyond. In all experiments, the best-choice model performed better only by 8 – 18%. This could possibly have to do with the used dataset, but the lack of data remains a threat to validity and adds some uncertainty. Apart from this, the models performed well. The recommended sequences reflect many of the recorded use cases, mastering also complex states.

V. CONCLUSION AND FUTURE WORK

We have built a novel prototype for action selection in GUI test generation using a session-based recommender system. We conducted a first empirical study on top of GitHub, for which we presented preliminary results. These results suggest that action selection, when seen as a sequence modeling task, can guide a test generator through states that require complex interactions by mimicking past user behavior.

Based on our current approach and findings, we identified several tasks for future work. First, there is an overall need for more real-world data in order to develop sophisticated models. Therefore, we strive for cooperations with owners of big web applications and other researchers. Second, a major bottleneck was the poor quality of the Selenium IDE code export. We plan to develop (i) a native Selenium IDE plugin to leverage fallback locators and (ii) a script for web application owners to extract anonymous usage traces. Third, adopting additional GRU4Rec extensions from [22] could improve the results. Moreover, hyperparameter optimization tools may be used to further improve the models' performance. Last and most importantly, the presented results are preliminary. To prove its effectiveness, the approach must be evaluated as part of a large-scale study using multiple, diverse SUTs, ideally in comparison to other test generators.

We believe that addressing these tasks can significantly contribute to the improvement of GUI-based test generation.

ACKNOWLEDGEMENTS

As part of the joint research project “Surili”, this work is supported by a grant (no. 01IS17092A) from the German Federal Ministry of Education and Research.

REFERENCES

- [1] M. Fowler. (May 2012). Testpyramid, [Online]. Available: <https://martinfowler.com/bliki/TestPyramid.html>.
- [2] A. Walgude and S. Natarajan, “World quality report 2019–20,” Paris, France, Tech. Rep., 2019.
- [3] K. Wiklund and M. Wiklund, “The next level of test automation: What about the users?” In *Proceedings of the 2018 International Conference on Software Testing, Verification, and Validation Workshops*, ser. ICSTW '18, Vasteras, Sweden: IEEE, 2018, pp. 159–162.
- [4] K. Mao, M. Harman, and Y. Jia, “Sapienz: Multi-objective automated testing for android applications,” in *Proceedings of the 25th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '16, Saarbrücken, Germany: IEEE, 2016, pp. 94–105.
- [5] N. Alshahwan, X. Gao, M. Harman, Y. Jia, K. Mao, A. Mols, T. Tei, and I. Zorin, “Deploying search based software engineering with sapienz at facebook,” in *Proceedings of the 10th International Conference on Search-Based Software Engineering*, ser. SSBSE '18, Montpellier, France: Springer Verlag, 2018, pp. 3–45.
- [6] S. Bauersfeld, S. Wappler, and J. Wegener, “A metaheuristic approach to test sequence generation for applications with a gui,” in *Proceedings of the 3rd International Conference on Search-Based Software Engineering*, ser. SSBSE '11, Szeged, Hungary: Springer Verlag, 2011, pp. 173–187.
- [7] A. I. Esparcia-Alcázar, F. Almenar, T. E. J. Vos, and U. Rueda, “Using genetic programming to evolve action selection rules in traversal-based automated software testing: Results obtained with the testar tool,” *Memetic Computing*, vol. 10, no. 3, pp. 257–265, Sep. 2018.
- [8] D. Kraus, *Machine learning and evolutionary computing for gui-based regression testing*, 2018. arXiv: 1802.03768.
- [9] M. Ermuth and M. Pradel, “Monkey see, monkey do: Effective generation of gui tests with inferred macro events,” in *Proceedings of the 25th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '16, Saarbrücken, Germany: ACM, 2016, pp. 82–93.
- [10] Y. Li, Z. Yang, Y. Guo, and X. Chen, *A deep learning based approach to automated android app testing*, 2019. arXiv: 1901.02633.
- [11] A. I. Esparcia-Alcázar, F. Almenar, M. Martínez, U. R. Rueda, and T. E. J. Vos, “Q-learning strategies for action selection in the testar automated testing tool,” in *Proceedings of the 6th International Conference on Metaheuristics and Nature Inspired Computing*, ser. META '16, Marrakech, Morocco, 2016, pp. 174–180.
- [12] C. Degott, N. P. Borges Jr., and A. Zeller, “Learning user interface element interactions,” in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA '19, Beijing, China: ACM, 2019, pp. 296–306.
- [13] D. Amalfitano, V. Riccio, N. Amatucci, V. De Simone, and A. R. Fasolino, “Combining automated gui exploration of android apps with capture and replay through machine learning,” *Information and Software Technology*, vol. 105, pp. 95–116, May 2019.
- [14] I. Ciupa, B. Meyer, M. Oriol, and A. Pretschner, “Finding faults: Manual testing vs. random+ testing vs. user reports,” in *Proceedings of the 19th IEEE International Symposium on Software Reliability Engineering*, ser. ISSRE '08, Seattle, WA, USA: IEEE Press, 2008, pp. 157–166.
- [15] C. A. Gomez-Urbe and N. Hunt, “The netflix recommender system: Algorithms, business value, and innovation,” *ACM Transactions on Management Information Systems*, vol. 6, no. 4, 13:1–13:19, Dec. 2015.
- [16] P. Covington, J. Adams, and E. Sargin, “Deep neural networks for youtube recommendations,” in *Proceedings of the 10th ACM Conference on Recommender Systems*, ser. RecSys '16, Boston, MA, USA: ACM, 2016, pp. 191–198.
- [17] S. Wang, L. Cao, and Y. Wang, *A survey on session-based recommender systems*, 2019. arXiv: 1902.04864.
- [18] Z. C. Lipton, J. Berkowitz, and C. Elkan, *A critical review of recurrent neural networks for sequence learning*, 2015. arXiv: 1506.00019.
- [19] Y. Bengio, P. Y. S. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
- [20] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk, *Session-based recommendations with recurrent neural networks*, 2016. arXiv: 1511.06939.
- [21] K. Cho, B. van Merriënboer, D. Bahdanau, and Y. Bengio, *On the properties of neural machine translation: Encoder-decoder approaches*, 2014. arXiv: 1409.1259.
- [22] B. Hidasi and A. Karatzoglou, *Recurrent neural networks with top-k gains for session-based recommendations*, 2017. arXiv: 1706.03847.
- [23] M. Buenen and A. Walgude, “World quality report 2018–19,” Paris, France, Tech. Rep., 2018.
- [24] H. Wu, Y. Ning, P. Chakraborty, J. Vreeken, N. Tatti, and N. Ramakrishnan, *Generating realistic synthetic population datasets*, 2016. arXiv: 1602.06844.
- [25] S. Nagaraja and R. Shah, *Clicktok: Click fraud detection using traffic analysis*, 2019. arXiv: 1903.00733.
- [26] M. Quadrana, P. Cremonesi, and D. Jannach, “Sequence-aware recommender systems,” *ACM Computing Surveys*, vol. 51, no. 4, 66:1–66:36, Jul. 2018.