Inferring Test Models from Kate's Bug Reports Using Multi-objective Search

Yuanyuan Zhang[✉], Mark Harman, Yue Jia, and Federica Sarro

CREST, Department of Computer Science, University College London, Malet Place, London WC1E 6BT, UK
yuanyuan.zhang@ucl.ac.uk

Abstract. Models inferred from system execution logs can be used to test general system behaviour. In this paper, we infer test models from user bug reports that are written in the natural language. The inferred models can be used to derive new tests which further exercise the buggy features reported by users. Our search-based model inference approach considers three objectives: (1) to reduce the number of invalid user events generated (over approximation), (2) to reduce the number of unrecognised user events (under approximation), (3) to reduce the size of the model (readability). We apply our approach to 721 of *Kate*'s bug reports which contain the information required to reproduce the bugs. We compare our results to start-of-the-art *KLFA* tool. Our results show that our inferred models require 19 tests to reveal a bug on average, which is 98 times fewer than the models inferred by *KLFA*.

Keywords: SBSE · NLP · Topic modelling · Model inference · NSGA-II

1 Introduction and Background

Many systems allow users to submit bug reports when they encounter unexpected behaviour. Developers need to validate and fix these issues, based on these bug reports. Unfortunately, not all of the bug-fixes work as expected. A recent study suggests that up to 24 % of post-release bug-fixes of large software systems are incorrect and some of the generated patches even introduce additional faults into the software [1,2]. These bad bug fixes not only affect the reliability of the software source code but also have negative impact on their users [1].

Generating additional tests that exercise the reported buggy features could improve software developers' confidence in their bug fixes. In this paper, we adapt an event-based model inference approach for such test enhancement using search-based algorithms. Event-based model inference has been widely used in software testing [3,4]. This technique takes system logs as inputs and generates a finite state machine which recognises execution sequences observed from the log file. Such a log file is often automatically generated and contains a sequence of function calls.

M. Barros and Y. Labiche (Eds.): SSBSE 2015, LNCS 9275, pp. 301–307, 2015.
DOI: 10.1007/978-3-319-22183-0_27

In this work, our approach aims to infer models from user bug reports instead of using system logs. Bug reports submitted by users are written in natural language, many of which include a set of instructions that can be used to reproduce the bugs. An inferred model from these bug reports is a generalisation of the set of user events which has triggered software bugs. The model can be used to generate new test data targeting the user-reported buggy features of the system.

Traditional single objective inferencing approaches tend to suffer from two intertwined problems. The inferred model either misses some behaviour specified in bug reports (under generalising) and includes some infeasible behaviour (over generalising). To overcome this limitation, we adapted the multi-objective approach proposed by Tonella et al. [5] to balance these two conflicting objectives.

We apply our approach to the SSBSE 2015 Challenge program *Kate* [6], a popular multi-platform text editor. We provide empirical evidence that the model generated from our approach not only provides good trade-offs between under and over approximation but also provides a good level of fault detection ability.

2 Models Inference Framework

Our approach to bug-report model inference consists of four phases. The first phase extracts raw bug issue reports from the *Kate* bug tracking system. Then the second phase parses the raw data extracted to retrieve bug information, such as textual descriptions of the execution steps for bug reproduction, the related components, the status and severity of the bug. In the third phase, bug descriptions are used to identify execution trace information. In particular, we use topic modelling to mine and extract reproducible user events. In the final phase, we use two multi-objective search algorithms to infer models from the user events.

Phase 1 - Bug Report Extraction: *Kate* is a multi-platform text editor written in C/C++. The *KDE Bugtracking System* [7] is used by the *Kate* project to maintain and keep track of reported software bugs. A web crawler was implemented to collect raw HTML webpage data from the *KDE Kate* bug repository. There have been 5,583 bug issues reported (including those already resolved, verified and closed) since January 2000. Our crawler visits the webpage of each bug issue and saves it as raw bug report data.

Phase 2 - Raw Data Parsing: we extract bug descriptions for each bug issue by parsing the raw data according to a set of search rules. We manually developed the search rules based on HTML files to capture information about bug issue ID, status, component, importance, description. In particular, in the textual description, steps to reproduce are the most important part of the bug report. They provide valuable information for the developer in order to test and fix the issue. We retrieve such information from the *Kate* bug HTML files by locating content between the 'Steps to Reproduce' and 'Actual Results' keywords.

Phase 3 - Data Mining Trace Events: there are three steps in Phase 3: (1) preparing the training corpus; (2) clustering similar trace steps; (3) mapping trace events. First, since 'Steps to Reproduce' patterns are written in natural language, we need to refine the patterns to remove noise. The Natural Language ToolKit (NLTK) [8] was used to preprocess the raw patterns. NLTK is an open source library for Natural Language Processing (NLP) implemented in Python. We first tokenise the patterns from strings to vectors, then remove English language stop words, numbers and punctuation marks. Next, we stem tokens to their root form and filter out low-frequency words that only appear once. We save all the refined patterns together as corpus in the Vector Space Model (VSM), which will be used in the next step.

In the second step, we cluster similar preprocessed trace steps using a tool called *gensim* [9], an open source NPL topic modelling tool, supporting semantic topic detection. In order to cluster steps, we transform a pre-prepared training corpus into a term frequency-inverse document frequency (tf-idf) matrix and then project it into a Latent Semantic Indexing (LSI) space. For each trace step, we compute *similarity* against the transformed corpus. The *similarity* measure used is the *cosine* similarity between two vectors. The most similar steps are clustered. We repeatedly combine clusters if their similarity measure is greater than a predefined *similarity* threshold (*cosine* > 0.7 in the experiment on which we report here). At this stage, the user events are generated by locating shared common tokens in one cluster. We found some generated user events have the same semantics, for example, 'open_kate', 'start_kate' and 'launch_kate' all represent the same user behaviour. These events should be treated as one, otherwise the algorithm will generate many similar states. In the last step, we solve this problem by manually examining half of the user events and creating a mapping to transform duplicated user events.

Phase 4 - Model Inference: We use two multi-objective algorithms, a Genetic Algorithm and the NSGA-II algorithm to infer models from the user events generated. In this work, there are three objectives taken into account to optimise the inferred models. These objectives are those proposed by Tonella et al. [5]. The first objective is to minimise the amount of model behaviour which does not follow any existing trace events generated. This type of unobserved model behaviour is over approximation, which is unlikely to occur in reality. The second objective is to minimise the amount of behaviour which is not accepted by the model, namely under approximation. It is measured by the number of unrecognised trace events. The third objective is to minimise the number of states in a model, to ensure we favour simplicity where possible.

3 Experiments and Results

To evaluate the feasibility and effectiveness of our approach, we answer the following research questions: **RQ0:** What are the prevalence and the characteristics of the trace events generated? **RQ1:** What are the performance of multi-objective optimisation compared to the benchmark model inference technique, *KLFA* [10] in terms of the hypervolume, running time and the number of solutions? **RQ2:** What is the fault revealing ability of the models inferred?

In total, our approach takes 721 bug reports that contain ‘Steps to Reproduce’ patterns as inputs and generates 452 user event trace files containing 265 unique trace events. To answer RQ0, we manually analysed these events which can be divided into six categories, as shown in Table 1. As can be seen from the table, the user events generated from our approach cover a wide range of functionalities of *Kate*, from basic operations to advanced features.

Table 1. Example of user events generated

Category	Basic operation	Text editing	Programming
Examples	Start_Kate	copy_paste_text	select_haskell_mode
	open_multiple_files	change_input_method	show_javascript_console
	score_screen	fold_section	check_regular_expression
	drag_cursor	find_replace	fold_function
	resize_window	captialize_text	check_indentation
	close_file	set_bookmark_color	enter_vi_command
Category	**Configuration**	**Plugins**	**Shortcut**
Examples	change_keyboard_setting	enable_plugin_quickswitcher	ctrl_1
	change_background_color	enable_plug_xml	ctrl_g
	change_print_margin	enable_plugin_spellcheck	ctrl_o
	change_print_page_range	enable_plugin_tabbar	ctrl_r
	enable_command_line	enable_plugin_terminal	alt_right
	enable_static_word_wrap	enable_plugin_treeview	alt_tab

To answer RQ1, we adopt one of standard, widely-used measures of multi-objective solution quality - hypervolume. Hypervolume is the volume covered by the solutions in the objective space. It is the union of hypercubes of solutions on the Pareto front [11]. By using a volume rather than a count, this measure is less susceptible to bias when the numbers of points on the two compared fronts are very different. We also measure the running time of the algorithms and the number of solutions generated. For algorithms that produce good quality solutions, quick and diverse answers are an important algorithmic property for decision makers.

Table 2. Objectives and performance metrics results for GA, NSGA-II and KLFA

Performance / Algorithm	Objectives - Mean (Min, Max)			Quality Metrics - Mean	
	Over Approximation	Under Approximation	Size of Model	Running Time	No. of Solutions
GA	2 (0, 66)	219 (208, 225)	13 (2, 53)	3239.66s	25
NSGA-II	0.1 (0.0, 7)	215 (183, 226)	19 (1, 94)	2341.14s	17
KLFA	55707	4	289	556.30s	1

Table 2 shows the mean, lowest and highest values of three objectives and average running time and the number of solutions generated by GA and NSGA-II for 30 executions. As *KLFA* generates deterministic solutions, we only report the results with one run for *KLFA*. Figure 1 shows the distribution of models generated by three techniques, along three objectives, in the form of box plots. As can be seen from the results, both GA and NSGA-II are able to infer models which have a lower over-approximation account but relatively higher under-approximation account. By contract, the models inferred by *KLFA* have a very high over-approximation account. In terms of size of a model, both GA and NSGA-II are able to keep the size of a model small.

The statistical analysis of hypervolume results is reported in Table 3. We use Cliff's method [12] for assessing statistical significance and the Vargha-Delaney $\hat{A}_{12}$ metric for effect size measure where the result is significant (at the 0.05 α level). The results of all algorithms are significantly different. The effect size of the two search algorithms are very small and both of them outperform *KLFA*.

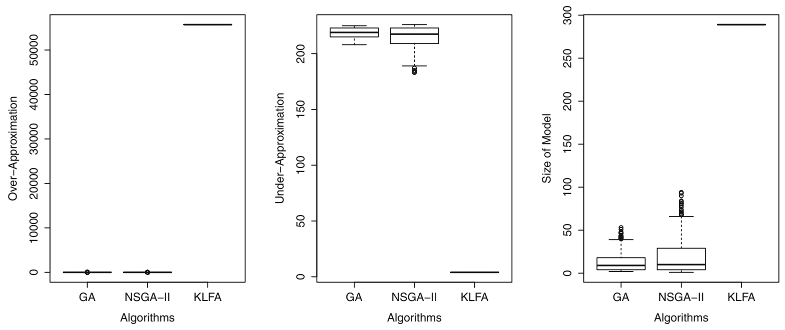

Fig. 1. Box plots of the over-approximation, under-approximation and size counts from the models inferred by multi-objective GA, NSGA-II and KLFA - 30 runs

Table 3. Hypervolume results of the statistical analysis for GA, NSGA-II and KLFA

Algorithm	Algorithm	Hypervolume	
		Cliff's method	Vargha-Delaney effect size
(x)	(y)	*p-value*	$\hat{A}_{12}$
GA	NSGA-II	**1e-04**	**0.06**
GA	KLFA	**1e-04**	**1.00**
NSGA-II	KLFA	**1e-04**	**1.00**

To answer RQ2, we investigate the fault-revealing ability of the models inferred. In software testing, the effectiveness of a test suite is assessed according to its ability to detect real bugs. We evaluate the fault-revealing ability of the models by checking the number of bug traces accepted by the models. If a bug trace is accepted by a model, it means the model can be used to generate test event traces to capture this bug. We have equally divided all valid bug reports into training and validation sets based on their submission time. We used the training set to infer models. We then check if the models inferred can capture the bug reported in the validation set. The training set contains 226 bug reports submitted between 07/2009 to 10/2012 , while the evaluation set contains 226 more recently submitted between 11/2012 and 02/2015. Table 4 shows the average number of bugs by each set of Pareto Front solution, the total number of bug detected and the average number of tests to be generated per bug revealed. Although *KLFA* generates find more bug in the validation set, it generates 500 times more tests traces. On the other hand, the models inferred using search only take less than 20 tests to reveal a bug on average where as *KLFA* takes 1,863. This makes former models preferable in place as the cost involved in checking the results of test sequence requires human effort.

Table 4. Results for fault-revealing ability of the models inferred

	Avg. # Traces (L = 4)	Avg. # Bugs Pareto Front	Total # Bugs	Avg. Test per bug revealed
GA	147	8	16	18
NSGA-II	116	6	22	19
KLFA	55,906	30	30	1863

4 Conclusion

We have studied the use of multi-objective search algorithm to infer models from software bug reports. The models inferred are well-balanced between the amount of over- and under-approximation of users behaviour. We also found that our approach generates smaller number of user event traces per bug revealed than *KLFA*, thereby these models are more preferable in practice. We believe that model inferencing techniques for documents written in natural language may prove to be widely applicable to many software documents, such as bug reports in our case.

Acknowledgements. We wish to express our gratitude to Paolo Tonella for his helpful suggestion and the search-based FSM tools provided.

References

1. Yin, Z., Yuan, D., Zhou, Y., Pasupathy, S., Bairavasundaram, L.: How do fixes become bugs? In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011), Szeged, Hungary, pp. 26–36. ACM, 5–9 September 2011
2. Buggy McAfee update whacks Wndows XP PCs: http://news.cnet.com/8301-1009_3-20003074-83.html

3. Krka, I., Brun, Y., Popescu, D., Garcia, J., Medvidovic, N.: Using dynamic execution traces and program invariants to enhance behavioral model inference. In: Proceedings of the ACM/IEEE 32nd International Conference on Software Engineering (ICSE 2010), Cape Town, South Africa, pp. 179–182. IEEE, 2–8 May 2010
4. Lorenzoli, D., Mariani, L., Pezzè, M.: Automatic generation of software behavioral models. In: Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, pp. 501–510. ACM, 10–18 May 2008
5. Tonella, P., Marchetto, A., Nguyen, D.C., Jia, Y., Lakhotia, K., Harman, M.: Finding the optimal balance between over and under approximation of models inferred from execution logs. In: Proceedings of IEEE 5th International Conference on Software Testing, Verification and Validation (ICST), Montreal, QC, Canada, pp. 21–30. IEEE, 17–21 April 2012
6. The Kate Editor: http://kate-editor.org/
7. KDE Bugtraking System: https://bugs.kde.org/
8. The Natural Language ToolKit (NLTK): http://www.nltk.org/
9. Řehůřek, R., Sojka, P.: Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, pp. 45–50. ELRA, May 2010. http://is.muni.cz/publication/884893/en
10. Mariani, L., Pastore, F.: Automated identification of failure causes in system logs. In: Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE 2008), Seattle, WA, USA, pp. 117–126. IEEE, 10–14 November 2008
11. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans. Evol. Comput. **3**(4), 257–271 (1999)
12. Cliff, N.: Ordinal Methods for Behavioral Data Analysis. Lawrence Erlbaum Associates Inc., New Jersey (1996)