Adaptive Behavioral Model Learning for Software Product Lines (Artifact Submission)

ABSTRACT

In this submission, the artifacts of the paper "Adaptive Behavioral Model Learning for Software Product Lines" (Submission number 16) are described. The provided artifacts include the structural and behavioral models of subject systems, the source code artifacts, experimental results, and source code for performing statistical tests and drawing diagrams. In this submission, each of these artifacts is explained. The steps required to replicate the experiments are also described.

1 INTRODUCTION

In this submission, the artifacts of the paper "Adaptive Behavioral Model Learning for Software Product Lines" (Submission number 16) are described. The artifacts include models of the subject systems, source code of the experiments, and statistical tests used. In this paper, the structure of the benchmarks is described; the functionality of the different parts of the source code is explained; and the steps of the performed experiments are elaborated. We also specify how the statistical tests are performed.

The artifact is available through the following URL:
https://github.com/sh-t-20/artifacts

2 SUBJECT SYSTEMS

The two subject systems used in the experiments are the Minepump SPL and the BCS SPL. For each of these SPLs, there is a folder of the same name in the `experiments` folder. The contents of these folders are described below:

The Minepump SPL: The artifacts of the Minepump SPL are available in the `Minepump_SPL` directory. This folder contains a file named `model.xml` which is the feature model [7, 10] of this SPL. Feature model files are available in ".xml" format [11] and are created using the FeatureIDE [11] library. Model learning experiments are performed using finite state machines (FSM) [4]. The FSM files and the configuration files for the products in the sample are in folder `products_3wise`. The FSM files are saved in ".dot" format [3?] and the configuration files are in ".config" format [11].

The BCS SPL: The artifacts of the BCS SPL are stored in the `BCS_SPL` folder. The `model.xml` file is the feature model of this SPL. The FSM files for the BCS components are available in the `Complete_FSM_files` folder. The component FSMs are created using the I/O transition systems which are available in [8]. The FSM of each valid product, can be created by merging the FSMs of its components.

3 THE SOURCE CODE ARTIFACTS

In the paper, the model learning experiments are performed using the `ExtensibleLStarMealyBuilder` class of the LearnLib [9] library version 0.16.0. The source code artifacts of this paper are in the `ir.ac.ut.fml` package from the `src` directory. The function of classes in this package is described below:

The `MergeMultipleFSMs` class is used to produce the FSMs of the BCS SPL products. The input parameters of this class are listed below:

- -dir: Directory of the configuration files
- -dir2: Folder containing FSMs of SPL components
- -out: Output directory for storing FSMs of SPL products.
- -fm: Feature model

Using the `MergeMultipleFSMs` class, for each of the product configuration files, the FSM files of its features are merged and the FSM of that product is constructed. The configuration files of the BCS SPL are available in the `products_3wise` folder of the `BCS_SPL` directory. The FSM files for the components of the BCS SPL are available in the `Complete_FSM_files`.

The `LearningOrderSampling` class, as input, takes a folder containing the products sampled from an SPL (i.e., the subject system SPLs). The products in this sample are learned based on different random orders using the PL* method and the non-adaptive method. For each learning order, the values of the learning cost metrics for each learning method are measured and stored in a log file. By running the `FixedLearningOrder` class, the products in a sample are learned based on a fixed learning order using the PL* method. This process is repeated several times and the values of the learning cost metrics are stored in a log file. The `Calculate order metric` class is used to calculate the values of parameter D for a number of random learning orders. After running this class, the calculated values of parameter D are stored in a log file.

The three classes `ConvertToExcelFile`, `ConvertToExcelFile2` and `ConvertToExcelFile3` are used in Experiments 4.1, 4.2.1 and 4.2.2, respectively, to convert the created log file to a ".csv" file.

4 REPLICATING THE EXPERIMENTS

To replicate the experiments, the repository must be downloaded. All the files needed for these experiments are in the `experiments` folder. The steps of replicating the experiments is described below:

4.1 Comparing the learning methods (RQ1-RQ3)

To replicate this experiment, the `LearningOrderSampling` class must be run using the following parameters:

- -dir: Directory of the SPL products (sampled products)
- -out: The output directory (the log file will be saved in this directory)
- -sot2: A folder for storing observation tables and learned FSM files

It is not necessary to specify the values of other arguments because they have default values. If the "-help" argument is used, the help menu is displayed. The number of learning orders tested is determined using the `samples_count` variable, which in this experiment is set to 200.

SPLC'22, 12-16 September, 2022, Graz, Austria
2022. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The learning orders used and their corresponding metric values, which are stored in the log file, must be saved as a ".csv" file. Each row of this ".csv" file contains a learning order and the corresponding learning cost metric values (for the PL* method and the non-adaptive method) To do this, the `ConvertToExcelFile` class must be run with the following parameters:

- -file: The input log file
- -out: The output directory (the ".csv" file will be saved in this directory)

The ".csv" files of this experiment are in the `results_1` folder.

4.2 The effect of learning order (RQ4)

To evaluate the effect of product learning order on the efficiency of the PL* method, two experiments are performed. Experiment 4.2.1 shows that the order of learning products can affect the total cost of learning in the PL* method. In Experiment 4.2.2, the value of parameter D is calculated for the 200 learning orders used in Experiment 4.1. At the end of this experiment, the Pearson correlation coefficient between D and the learning cost metrics can be calculated (using the Python codes explained in 5).

4.2.1 The effect of learning order on the efficiency of the PL method.* To replicate this experiment, a fixed learning order must first be considered. This learning order is stored in a variable of type int array (called `learning_order_array`) in the `FixedLearningOrder` class. The parameters required to run this class are similar to the parameters described for the `LearningOrderSampling` class. After running the `FixedLearningOrder` class, the measured values for the learning cost metrics are stored in a log file. Using the `ConverToExcelFile2` class, these values can be stored as a ".csv" file. The above steps must be performed for two learning orders: an order with a high learning efficiency (order 1) and an order with a relatively low learning efficiency (order 2). To determine these two learning orders, the results of Experiment 4.1, which are sorted by efficiency, can be used. The sorted "csv" files of Experiment 4.1 are in the `results_1` directory. The results of Experiment 4.2.1 for both subject SPLs are available in the `results_2_1` directory.

4.2.2 Calculating the parameter D. In this experiment, using the `CalculateOrderMetric` class, the values of parameter D is calculated for the 200 learning orders used in Experiment 4.1. The parameters required to run the `CalculateOrderMetric` class are similar to the parameters explained for the `LearningOrderSampling` class. In order for random learning orders produced in this experiment and Experiment 4.1 to be the same, the following conditions must be considered for the `CalculateOrderMetric` and `LearningOrderSampling` classes:

- The initial value of the seed must be the same in both classes. The seed value is stored in a `long` variable of the same name.
- The number of random learning orders generated in both classes must be the same. The number of product learning orders is stored in a variable called `samples_count`.

After running the `CalculateOrderMetric` class, the values calculated for parameter D are stored in a log file. Then, using the `ConvertToExcelFile3` class, theses values can be saved as a ".csv" file. The results of Experiment 4.2.2 are available as ".csv" files in the `results_2_2` folder.

5 STATISTICAL TESTS

The source codes for performing statistical tests and plotting diagrams are in the `SPL_Learning.ipynb` file (which is located in the `statistical_tests` folder). These codes are written in Python using the Jupyter Notebook. To perform statistical tests, the ".csv" files generated in the experiments described in Section 4 are first loaded using into DataFrames using the `read_csv` method of the Pandas [12] library. Statistil tests are performed using the Scipy [6] library. The Matplotlib [5] library is used to draw the diagrams.

6 LICENSING

The artifacts are all available under GNU Public License 3.0. It makes use of the following two repositories, which are also available under the same license and are properly attributed in the artifact:

https://github.com/damascenodiego/DynamicLstarM [1]
https://github.com/damascenodiego/learningFFSM [2]

REFERENCES

[1] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. 2019. Learning to Reuse: Adaptive Model Learning for Evolving Systems. In *Integrated Formal Methods - 15th International Conference, IFM 2019, Bergen, Norway, December 2-6, 2019, Proceedings (Lecture Notes in Computer Science, Vol. 11918)*, Wolfgang Ahrendt and Silvia Lizeth Tapia Tarifa (Eds.). Springer, 138–156. https://doi.org/10.1007/978-3-030-34968-4_8

[2] Carlos Diego Nascimento Damasceno, Mohammad Reza Mousavi, and Adenilso da Silva Simão. 2021. Learning by sampling: learning behavioral family models from software product lines. *Empir. Softw. Eng.* 26, 1 (2021), 4. https://doi.org/10.1007/s10664-020-09912-w

[3] Emden R. Gansner and Stephen C. North. 2000. An open graph visualization system and its applications to software engineering. *SOFTWARE - PRACTICE AND EXPERIENCE* 30, 11 (2000), 1203–1233.

[4] Arthur Gill et al. 1962. Introduction to the theory of finite-state machines. (1962).

[5] J. D. Hunter. 2007. Matplotlib: A 2D graphics environment. *Computing in Science & Engineering* 9, 3 (2007), 90–95. https://doi.org/10.1109/MCSE.2007.55

[6] Eric Jones, Travis Oliphant, Pearu Peterson, et al. 2001–. SciPy: Open source scientific tools for Python. http://www.scipy.org/

[7] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study.* Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.

[8] Sascha Lity, Remo Lachmann, Malte Lochau, and Ina Schaefer. 2012. *Delta-oriented software product line test models-the body comfort system case study.* Technical Report 2012-07. TU Braunschweig.

[9] Harald Raffelt, Bernhard Steffen, and Therese Berg. 2005. LearnLib: a library for automata learning and experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems, FMICS '05, Lisbon, Portugal, September 5-6, 2005*, Tiziana Margaria and Mieke Massink (Eds.). ACM, 62–71. https://doi.org/10.1145/1081180.1081189

[10] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA*. IEEE Computer Society, 136–145. https://doi.org/10.1109/RE.2006.23

[11] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Program.* 79 (2014), 70–85. https://doi.org/10.1016/j.scico.2012.06.002

[12] Wes McKinney. 2010. Data Structures for Statistical Computing in Python. In *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman (Eds.). 56 – 61. https://doi.org/10.25080/Majora-92bf1922-00a