AutoSMP: An Evaluation Platform for Sampling Algorithms

Tobias Pett
TU Braunschweig
Germany

Sebastian Krieter
Harz University of Applied Science
Germany

Thomas Thüm
University of Ulm
Germany

Malte Lochau
University of Siegen
Germany

Ina Schaefer
TU Braunschweig
Germany

ABSTRACT

Testing configurable systems is a challenging task due to the combinatorial explosion problem. Sampling is a promising approach to reduce the testing effort for product-based systems by finding a small but still representative subset (i.e., a sample) of all configurations for testing. The quality of a generated sample wrt. evaluation criteria such as run time of sample generation, feature coverage, sample size, and sampling stability depends on the subject systems and the sampling algorithm. Choosing the right sampling algorithm for practical applications is challenging because each sampling algorithm fulfills the evaluation criteria to a different degree. Researchers keep developing new sampling algorithms with improved performance or unique properties to satisfy application-specific requirements. Comparing sampling algorithms is therefore a necessary task for researchers. However, this task needs a lot of effort because of missing accessibility of existing algorithm implementations and benchmarks. Our platform *AutoSMP* eases practitioners and researchers lifes by automatically executing sampling algorithms on predefined benchmarks and evaluating the sampling results wrt. specific user requirements. In this paper, we introduce the open-source application of *AutoSMP* and a set of predefined benchmarks as well as a set of T-wise sampling algorithms as examples.

CCS CONCEPTS

• **Software and its engineering** → Software testing and debugging; **Software product lines**; **Sampling evaluation**.

KEYWORDS

product lines, sampling, sampling evalutaion

ACM Reference Format:
Tobias Pett, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Ina Schaefer. 2021. AutoSMP: An Evaluation Platform for Sampling Algorithms. In *25th ACM International Systems and Software Product Line Conference - Volume B (SPLC '21), September 6–11, 2021, Leicester, United Kingdom.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3461002.3473073

SPLC '21, September 6–11, 2021, Leicester, United Kingdom

ACM ISBN 978-1-4503-8470-4/21/09…$15.00
https://doi.org/10.1145/3461002.3473073

1 INTRODUCTION

Configurable systems enable the creation of customized variants by means of selecting features. The configuration space of a configurable system is often represented by a variability model, which describes the relations and constraints between any combination of features [4, 13]. Testing a highly configurable system is challenging because interactions between features can have unforeseen influences on the resulting configuration. Complete testing comprises the execution of every possible configuration, which is not feasible due to the combinatorial explosion problem [3, 5, 16]. Sampling can be used to restrict the effort of testing highly configurable systems by generating a preferably small yet representative subset of configurations (i.e., a sample) for testing [15, 16, 24]. Calculating a small but representative sample in a reasonable time is an NP-hard problem [11]. Therefore, different approaches to generate samples were developed, for instance search-based algorithms [7, 9, 10, 18]. Search-based sampling algorithms typically define multiple optimization goals wrt. properties of the sample, which is approximated using evolutionary algorithms or machine learning techniques. T-Wise sampling algorithms are based on combinatorial-interaction testing to generate samples that cover interactions between features [17]. Over the past decades, developers proposed many sampling algorithms which fulfill user requirements to different degrees [24]. These sampling algorithms are assessed using various evaluation criteria, for instance, run time, sample size, feature coverage, which often are mutually exclusive. For instance, a sample with low T-wise coverage (e.g., T=1) can be calculated fast, and the resulting sample size is small. Calculating a sample with high T-wise coverage (e.g., T=5) takes a long time, and the resulting samples are large. The results of evaluating a sampling algorithm depend on the benchmark (i.e., variability model) for which the sample is calculated. Hence, comparing the performance of sampling algorithms wrt. defined evaluation criteria is only reasonable when using the same variability models as input. To this end, we identify two significant challenges for users and developers of sampling algorithms: 1) Evaluating a new sampling algorithm consists of redundant and tedious tasks such as collecting existing sampling algorithms and benchmark data and executing the sampling algorithms, which can slow down the development of new sampling algorithms. 2) Practitioners often miss the time and insights to evaluate which sampling algorithms are best suited for their application scenario.

Previous and current work presents tool support for practitioners and researchers to evaluate sampling algorithms [1, 8]. For instance, Fereira et al. [8] propose NautilusVTSPL as tool support for practitioners to help them selecting search-based sampling algorithms more efficiently. Acher et al. [1], propose BURST to help researchers evaluate the scalability and uniformity of their uniform random sampling approaches.

We propose *AutoSMP* to address the challenges of comparing sampling algorithms by providing a platform to automatically calculate evaluation reports for researchers and practitioners. Users of *AutoSMP* register their sampling algorithms and benchmarks to receive results for their specific use-cases.

In contrast to NautilusVTSPL [8] and BUSRT [1], *AutoSMP* aims to be an extensible platform for evaluating any kind sampling algorithms. In addition to simply evaluating sampling algorithms, *AutoSMP* can calculate customized evaluation reports, based on user specifications, which consist of a ranked list of sampling algorithms.

AutoSMP is designed as a coherent and extensible platform for evaluating sampling algorithms. As such, it addresses the current challenges of evaluating sampling algorithms for researchers and practitioners by providing the following functionalities:

(1) Automated evaluation of sampling algorithms.
(2) Predefined benchmarks and sampling algorithms can be used as comparison/baseline.
(3) Recommendation of most suitable sampling algorithms for specific requirements.
(4) Provision of an extensible knowledge base for the community.

We provide *AutoSMP* on GitHub[1]. The repository contains the executable file, the sources for the platform, and a set of state-of-the-art T-wise sampling algorithms such as Chvatal [6], ICPL [12], IncLing [2], and YASA [14]. Further, it contains several example benchmarks consisting of multiple large-scale variability models.

2 AUTOSMP - LOOKING UNDER THE HOOD

In this section, we introduce the Auto-SMP platform. We describe the main concepts of our platform, the technical design behind the concepts, and the customized evaluator.

2.1 Concept of the AutoSMP Platform

Our platform consists of three main parts, namely Algorithms, Benchmarks, and Evaluation Results. In the following those three parts are described. *(1) Algorithms:* Our platform contains a set of registered T-wise sampling algorithms including of Chvatal [6], ICPL [12], IncLing [2], and YASA [14]. Users of the platform can extend this set by registering new sampling algorithms using a provided interface definition. *(2) Benchmarks:* We provide multiple benchmarks as input for sampling algorithms. The benchmarks are provided in the well-known DIMACS format and are categorized into data packages based on their size and evolution history. For instance, we provide small, medium and large benchmarks as well as benchmarks representing an evolution histories of BusyBox, Fiasco, ToyBox, UcLibc, and Soletta on GitHUb[2], grouped in respective data packages. Our data packages consist of a broad variety of feature models that are already utilized by researchers in the area of variability management [2, 14, 20, 22]. Evaluation results achieved by using all of the benchmarks provided in our platform can be generalized based on their diversity. In addition, users of our platform can add a new feature model if their particular use case needs to be addressed. *(3) Evaluation Results:* Sampling algorithms are frequently evaluated based on metrics such as sampling efficiency, sampling effectiveness, and T-wise coverage [24]. Our platform supports those three metrics to compare sampling algorithms. In addition, *AutoSMP* supports the evaluation criterion *sampling stability*, introduced by Pett et al. [19], which considers the evolution of samples over time.

2.2 Process of using AutoSMP

Assume, we have an example researcher Bob, who has developed a new sampling algorithm called *SAOB*. Bob wants to evaluate his algorithm against at least three other well-known sampling algorithms. The evaluation process involves implementing the software of the algorithms, learning how to use them, and evaluating them. Bob can use our evaluation platform, *AutoSMP*, to automatically evaluate his algorithm against a set of sampling algorithms registered in our platform. Bob needs to request a customized evaluation report for his algorithm. He registers SAOB as a new sampling algorithm using a provided interface definition. Then, he specifies the sets of benchmarks and the evaluation criteria, which will be used to evaluate SAOB, using the *AutoSMP*'s configuration file. Our platform automatically executes all registered sampling algorithms on the specified benchmarks and stores the evaluation results.

2.3 Registering a Sampling Algorithm

Users can register an implementation of a sampling algorithm, which can be run from the command line. The executable file for the sampling algorithm must exist, and an adapter class must be implemented, using the provided adapter-interface of *AutoSMP*. For instance, Bob provides his sampling algorithm (SAOB) as an executable jar file, which takes the input- and output- directory and the T-wise coverage as parameters. To register his algorithm in *AutoSMP*, Bob needs to implement an adapter class using the *ASamplingAlgorithm* interface. Bob specifies the commands required to execute his sampling algorithm from the command line in the adapter class. The adapter interface provides Bob with utility methods to access variables such as the input path, the output path, T-wise coverage, etc., specified in the configuration file of *AutoSMP*. After implementing the adapter class, Bob needs to copy the class file with its whole class path (e.g., de/tubs/isf/algorithmAdapter/saob.class) into the algorithms folder of *AutoSMP*.

2.4 Configuring an Evaluation Run

The configuration of *AutoSMP* is structured as a JAVA properties file, where each line represents a key-value pair. Users can change the execution behavior of *AutoSMP* by adjusting the values of a configuration entry. In Listing 1, we provide an excerpt of the configuration file for *AutoSMP* containing the most important configuration options. The *inputDir* option defines the path to the benchmarks for evaluation. *outputDir* defines the path to stored output files of the executed algorithms (e.g., samples). The path given for *csvDir* points to the output directory for results calculated by *AutoSMP*. The user specifies the location of registered algorithms by providing a path

[1] https://github.com/TUBS-ISF/AutoSMP
[2] https://github.com/TUBS-ISF/busybox-case_study, https://github.com/TUBS-ISF/fiasco-case-study, https://github.com/TUBS-ISF/toybox-case-study, https://github.com/TUBS-ISF/uclibc-case-study, https://github.com/TUBS-ISF/soletta-case-study

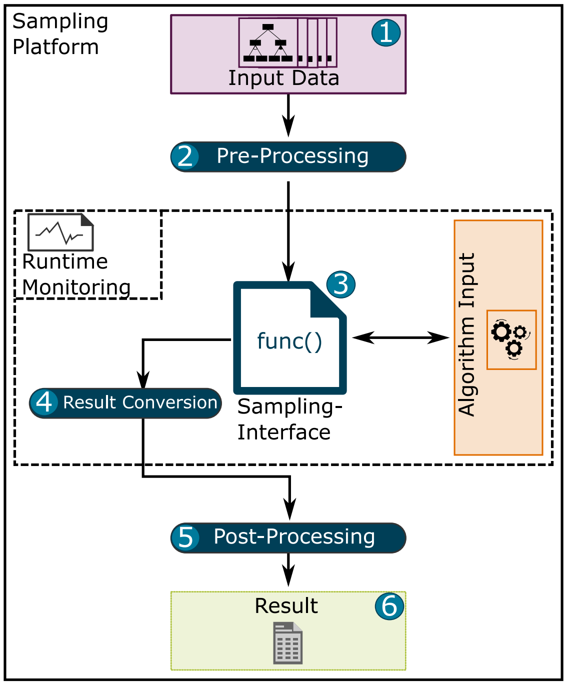


Figure 1: General architecture and workflow of AutoSMP

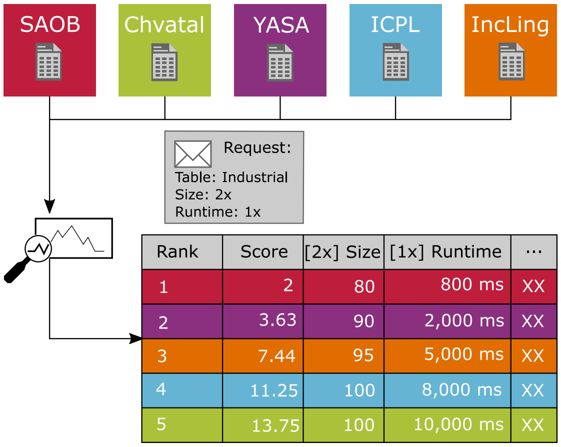
as value for *algoDir*. *algorithms* defines which adapters for sampling algorithms are executed. It is essential to specify the whole classpath of the algorithm adapter.

```
inputDir = ./benchmarks/Test_Car/
outputDir = ./output/
algoDir = ./algorithms/
csvDir  = ./sampling-evaluation-results/
algorithms = de.tubs.isf.algorithmAdapters.AdapterSaob
```

Listing 1: Excerpt of the configuration file of *AutoSMP*

2.5 Framework Realization

In Figure 1, we present the detailed workflow of evaluating a sampling algorithm with *AutoSMP*. The evaluation process takes a benchmark and a sampling algorithm as input for an evaluation run. The user provides algorithm parameters (e.g., T-coverage and sample size) and execution parameters (e.g., input- and output paths) via a configuration file, which is loaded during the start of *AutoSMP*. (1) *AutoSMP* starts by reading variability models that it evaluates in this run. (2) It checks the input model's validity and logs an error report for invalid models. If no valid variability model is provided, the platform exits the evaluation run. (3) For each valid variability model, *AutoSMP* calls the sampling interface with the input benchmark and algorithm configuration. (4) The platform transforms the output of the sampling algorithms into a required format for further processing. After the output is transformed, it is forwarded to the evaluation component of our platform. (5) In post-processing *AutoSMP* verifies each result by checking every sample for validity and calculating the achieved feature interaction coverage (Verification). Further, it extracts sample size, sample run time, and sample similarity (if calculated) from the sampling results (Assessment). (7) The results are stored as a data table in CSV format with the following headline information:



Request:
Table: Industrial
Size: 2x
Runtime: 1x

Rank	Score	[2x] Size	[1x] Runtime	···
1	2	80	800 ms	XX
2	3.63	90	2,000 ms	XX
3	7.44	95	5,000 ms	XX
4	11.25	100	8,000 ms	XX
5	13.75	100	10,000 ms	XX

Figure 2: Process of generating a custom evaluation report

- Algorithm: Name
- Model: ID, Name, #Features, #Constraints
- Sampling: Size, Runtime, Feature Interaction Coverage
- Other: Timeout, Validity, T-Value, etc.

3 CUSTOMIZED EVALUATION REPORT

Determining an appropriate algorithm to calculate a sample is challenging [24]. We provide customized recommendations for sampling algorithms with our evaluator by considering the user's priorities on evaluation criteria.

Our example user Bob requires a sampling algorithm that produces small samples for large feature models. Bob uses AutoSMP to obtain a prioritized list of sampling algorithms by specifying a request to evaluate all registered sampling algorithms on the large data package. His request contains the following settings:

Benchmarks: Bob defines for which benchmarks a custom evaluation will be calculated (i.e., *Large* Benchmark). If this setting is not existent or left empty, the evaluator calculates a custom recommendation for all registered benchmarks.
Custom Weights: Bob specifies which sampling evaluation criteria are prioritized by providing a list of tuples. A tuple contains the name of an evaluation criterion and a positive number as its weight (i.e., Size,2; run Time,1). Sampling criteria not specified in the prioritization list are considered with a weight of one.

In Figure 2, we show the result table of our evaluator, based on the example request of Bob. Each colored entry at the top represents evaluation results for a sampling algorithm. The evaluator of *AutoSMP* constructs a recommendation table based on the sampling evaluation results provided as input and specified request by computing the score of each sampling algorithm. We calculate the score of a sampling algorithm using the *Weighted Rank-Based Score (WRBS)* proposed by Sprey [23]. As defined by Sprey [23], WRBS for a sampling algorithm is calculated by adding the sub-scores of each evaluation criteria of the sampling algorithm. The subscore is calculated by normalizing the weighted arithmetic mean value of the evaluation criterion with the best-weighted arithmetic mean value of all sampling algorithms under consideration. The recommendation table contains the following entries:

Rank: a numerical index showing the ranking of the scores. The lowest rank is the most recommended algorithm.
Score: a numerical value representing the overall performance of a sampling algorithms wrt. the request (less is better).
Weighted Arithmetic Mean for each Evaluation Criteria: a numerical value that represents the performance of a sampling algorithm for a specific evaluation criteria. The value is computed by calculating the arithmetic mean value of all runs in the evaluation result table and multiplying it with the user-specified weight.

4 CONCLUSION - WHAT COMES NEXT?

In this paper, we present *AutoSMP*, a platform to evaluate sampling algorithms automatically. Our platform contributes to solving two significant challenges of sampling highly configurable systems. 1) Evaluating a new sampling algorithm includes redundant and time-consuming work. We provide researchers with a tool to automatically evaluate a new sampling algorithm on representative data and against other state sampling algorithms. 2) Choosing a sampling algorithm that fulfills user requirements best is difficult due to the vast amount of sampling algorithms available, and each algorithm fulfills evaluation criteria to a different degree. Our tool provides users with a customized recommendation table to select the best algorithm for their requirements. Currently, we provide the platform as a stand-alone Java application on GitHub[3]. Users can contribute to the platform's development by pull requests to propose new benchmarks and sampling algorithms. In the future, we aim to make our platform even more open for the community by setting it up as a web service or as a coordinated-shared task [21] using a participant-in-charge software submission system such as TIRA[4]. Further we plan to contact the authors of BURST [1], and NautilusVTSPL [8] to discuss a collaboration to build one coherent evaluation platform for sampling algorithms of any kind.

ACKNOWLEDGMENTS

This work is partially founded by the *MWK*[5] of lower saxony as part of the *ZDIN*[6] initiative under the award number ZN3495. We thank Joshua Sprey who developed the concepts and implementations for our tool during his master's thesis. We also thank Martin Potthast discussing the applicability of TIRA to transition our AutoSMP platform to being a organized shared task.

REFERENCES

[1] M. Acher, G. Perrouin, and M. Cordy. 2021. BURST: A BenchmarkingPlatform for Uniform Random Sampling Techniques. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM.

[2] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: Efficient Product-line Testing Using Incremental Pairwise Sampling. In *Proc. Int'l Conf. on Generative Programming: Concepts & Experiences (GPCE)*. ACM, 144–155.

[3] Sven Apel, Alexander von Rhein, Philipp Wendler, Armin Größlinger, and Dirk Beyer. 2013. Strategies for Product-Line Verification: Case Studies and Experiments. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. IEEE, 482–491.

[4] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *Proc. Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 7:1–7:8.

[5] Lianping Chen, Muhammad Ali Babar, and Nour Ali. 2009. Variability management in software product lines: a systematic review. In *13th International Software Product Line Conference*. ACM.

[6] Vasek Chvatal. 1979. A Greedy Heuristic for the Set-Covering Problem. *Mathematics of operations research* 4, 3 (1979), 233–235.

[7] Thiago N. Ferreira, Jackson A. Prado Lima, Andrei Strickler, Josiel N. Kuk, Silvia R. Vergilio, and Aurora Pozo. 2017. Hyper-Heuristic Based Product Selection for Software Product Line Testing. *Comp. Intell. Mag. (CIM)* 12, 2 (2017), 34–45.

[8] Thiago Nascimento Ferreira, Silvia Regina Vergilio, and Mauroane Kessentini. 2020. Many-objective Search-based Selection of Software Product Line Test Products with Nautilus. In *SPLC*. 1–4.

[9] Christopher Henard, Mike Papadakis, and Yves Le Traon. 2014. Mutation-Based Generation of Software Product Line Test Configurations. In *Search-Based Software Engineering*, Claire Le Goues and Shin Yoo (Eds.). Lecture Notes in Computer Science, Vol. 8636. Springer International Publishing, 92–106.

[10] Christopher Henard, Mike Papadakis, Gilles Perrouin, Jacques Klein, and Yves Le Traon. 2013. Multi-Objective Test Generation for Software Product Lines. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 62–71.

[11] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of Realistic Feature Models Make Combinatorial Testing of Product Lines Feasible. In *Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS)*. Springer, 638–652.

[12] Martin Fagereng Johansen, Øystein Haugen, Franck Fleurey, Anne Grete Eldegard, and Torbjørn Syversen. 2012. Generating Better Partial Covering Arrays by Modeling Weights on Sub-Product Lines. In *Proc. Int'l Conf. on Model Driven Engineering Languages and Systems (MODELS)*. Springer, 269–284.

[13] 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 CMU/SEI-90-TR-21.

[14] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Gunter Saake, and Thomas Leich. 2020. YASA: Yet Another Sampling Algorithm. In *Proc. Int'l Working Conf. on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, Article 4, 10 pages.

[15] Jihyun Lee, Sungwon Kang, and Danhyung Lee. 2012. A Survey on Software Product Line Testing. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 31–40.

[16] Flávio Medeiros, Christian Kästner, Márcio Ribeiro, Sarah Nadi, and Rohit Gheyi. 2015. The Love/Hate Relationship with the C Preprocessor: An Interview Study. In *29th European Conference on Object-Oriented Programming (ECOOP 2015)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 495–518.

[17] Changhai Nie and Hareton Leung. 2011. A Survey of Combinatorial Testing. *Comput. Surveys* 43, 2, Article 11 (2011), 11:1–11:29 pages.

[18] José A Parejo, Ana B Sánchez, Sergio Segura, Antonio Ruiz-Cortés, Roberto E Lopez-Herrejon, and Alexander Egyed. 2016. Multi-objective test case prioritization in highly configurable systems: A case study. *Journal of Systems and Software* 122 (2016), 287–310.

[19] Tobias Pett, Sebastian Krieter, Tobias Runge, Thomas Thüm, Malte Lochau, and Ina Schaefer. 2021. Stability of Product-Line Samplingin Continuous Integration. In *15th International Working Conference on Variability Modelling of Software-Intensive Systems*. 1–9.

[20] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product Sampling for Product Lines: The Scalability Challenge. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 78–83.

[21] Martin Potthast, Tim Gollub, Matti Wiegmann, and Benno Stein. 2019. TIRA integrated research architecture. In *Information Retrieval Evaluation in a Changing World*. Springer, 123–160.

[22] Reimar Schröter, Sebastian Krieter, Thomas Thüm, Fabian Benduhn, and Gunter Saake. 2016. Feature-Model Interfaces: The Highway to Compositional Analyses of Highly-Configurable Systems. In *Proc. Int'l Conf. on Software Engineering (ICSE)*. ACM, 667–678.

[23] Joshua Sprey. 2020. Automated Comparison of Product Sampling Algorithms: Master's Thesis. (2020).

[24] Mahsa Varshosaz, Mustafa Al-Hajjaji, Thomas Thüm, Tobias Runge, Mohammad Reza Mousavi, and Ina Schaefer. 2018. A Classification of Product Sampling for Software Product Lines. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, 1–13.

[3] https://github.com/TUBS-ISF/AutoSMP

[4] https://www.tira.io/

[5] Ministerium für Wissenschaft und Kultur

[6] Zentrums für digitale Innovationen Niedersachsen