# BURST: A Benchmarking Platform for Uniform Random Sampling Techniques

Mathieu Acher
IRISA, University of Rennes I, France
mathieu.acher@irisa.fr

Gilles Perrouin
PReCISE, NaDI,
Faculty of Computer Science,
University of Namur
gilles.perrouin@unamur.be

Maxime Cordy
SnT, University of Luxembourg
maxime.cordy@uni.lu

## ABSTRACT

We present BURST, a benchmarking platform for uniform random sampling techniques. With BURST, researchers have a flexible, controlled environment in which they can evaluate the scalability and uniformity of their sampling. BURST comes with an extensive — and extensible — benchmark dataset comprising 128 feature models, including challenging, real-world models of the Linux kernel. BURST takes as inputs a sampling tool, a set of feature models and a sampling budget. It automatically translates any feature model of the set in DIMACS and invokes the sampling tool to generate the budgeted number of samples. To evaluate the scalability of the sampling tool, BURST measures the time the tool needs to produce the requested sample. To evaluate the uniformity of the produced sample, BURST integrates the state-of-the-art and proven statistical test Barbarik. We envision BURST to become the starting point of a standardisation initiative of sampling tool evaluation. Given the huge interest of research for sampling algorithms and tools, this initiative would have the potential to reach and crosscut multiple research communities including AI, ML, SAT and SPL.

## CCS CONCEPTS

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

## KEYWORDS

configurable systems, software product lines, variability model, sampling, SAT, benchmark

**ACM Reference Format:**
Mathieu Acher, Gilles Perrouin, and Maxime Cordy. 2021. BURST: A Benchmarking Platform for Uniform Random Sampling Techniques. 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, 5 pages. https://doi.org/10.1145/3461002.3473070

## 1 INTRODUCTION

Uniform Random Sampling (URS) is a family of techniques to sample from the set of solutions of a logical formulae, such that each solution gets the same probability of being selected. URS has numerous applications in various domains including deep learning [3] and Software Product Line (SPL) engineering [27, 29, 32]. As such, multiple research communities have worked on approaches and tools for URS.

In SPL engineering, URS – and more generally, sampling approaches – have been an important support to quality assurance techniques like testing [16, 32], verification [10, 30] and performance analysis [2, 19, 40]. This support consists of computing a representative sample of SPL variants (or *configurations*). Because these variants are too numerous to be all considered for analysis, sampling offers an adequate compromise between completeness and efficiency.

Various artifacts can drive the sampling of SPL variants, such as feature models [20], source code, test suites, behavioural models, etc. Feature models, however, remain the most commonly used input for SPL sampling techniques. The main reason is that the semantics of feature models can be expressed in first-order logic [4, 35], whose set of solutions corresponds to the set of valid SPL variants. This makes feature models inherently amenable to URS.

There exist also approaches that combine URS with metaheuristics to obtain a sample optimizing several objectives (e.g. coverage and cost of configurations) [15, 17, 34]. Metaheuristics – such as evolutionary algorithms – start from initial set of random solutions, and the choice of these initial solutions can have a significant impact on effectiveness [5]. Indeed, past research has demonstrated that URS can produce an initial set of solutions that ultimately improve the effectiveness of the metahuristics by 10% on average [11, 24].

As the interest towards URS has been growing, different research communities have developed algorithmic solutions and evaluated those solutions on community-specific datasets. For example, researchers in SPL engineering typically use feature models, whereas researchers in artificial intelligence and SAT solvers typically focus on circuits or embedded system design problems, resulting in very different formulas. Such lack of standardization has inevitably led to poor generalization beyond the specifically used datasets. For instance, our past study has revealed that UniGen – one of the most effective URS tools that AI researchers had developed – failed to scale on large feature models [32]. UniGen's authors have, since then, developed new approaches that perform better on feature models [37, 38]. This experience illustrates that researchers need the ability to experiment with various datasets and, ideally, under a common protocol and technological ground. Such experimental control would circumvent the threats to generalization of previous studies and lead to a standardization of sampling tools evaluation.


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

ACM ISBN 978-1-4503-8470-4/21/09.
https://doi.org/10.1145/3461002.3473070

To address this need, we present BURST, a benchmarking platform for uniform random sampling techniques. With BURST, researchers have a flexible, controlled environment in which they can evaluate the scalability and uniformity of their sampling. As noted by Pett *et al.* [31], evaluating sampling algorithms require collecting them, benchmarks data and executing them. BURST thus comes with an extensive — and extensible — benchmark dataset comprising 128 feature models, including challenging, real-world models of the Linux kernel stored as CNF formulas in the well-known DIMACS format. The platform also includes datasets from the artificial intelligence community [12] for comparison. BURST takes as inputs a sampling tool, a set of CNF formulas and a sampling budget to generate samples and/or evaluate their conformance to uniform distributions. BURST relies on external tools (such as FeatureIDE [26]) to transform feature models to CNF formulas. To evaluate the uniformity of the produced samples, BURST integrates the state-of-the-art and proven statistical test Barbarik [7]. BURST is available as open source and as a Docker image: https://github.com/FAMILIAR-project/usampling-exp. A short video explaining BURST motivation and usage is also available: https://www.youtube.com/watch?v=sSKosyrfitA.

## 2 BURST: INPUTS, PROCESS, OUTPUTS

### 2.1 Feature models

BURST's benchmark dataset comprises 128 publicly available feature models with a varying complexity scale, including models of real-world configurable systems (Linux, eCos, toybox, JHipster, etc.). These models come from three sources:

- 117 feature models come from previous research [21, 22]. The majority of these 117 models contain between 1,221 and 1,266 features. Of these 117 models, 107 comprise between 2,968 and 4,138 cross-tree constraints, one has 14,295, and the other nine have around 50,000 [21, 22].
- 10 additional models come from [23]. These models contain more than 6,000 features.
- the last model is the JHipster feature model [16, 33]. It is real-world but smaller: 45 variables, 26,000+ configurations.

Once transformed in conjunctive normal form (CNF), these instances typically contain between 1 and 15 thousand variables and up to 340 thousand clauses. For instance, the Linux kernel models – the hardest model for sampling that we have – contains more than 6 thousand variables, 340 thousand clauses.

### 2.2 Other Models

BURST also offers a variety of models obtained when evaluating tools such as QuickSampler and Unigen. The models are not translation of feature models but represent electronic circuits, embedded design or algorithmic problems. These models have different characteristics from feature models, inducing a very different behavior from uniform samplers. Feature models are generally more challenging. Since uniform sampling have applications in many areas and not only SPLs, diversity of models is key to practical significance of sampling tools.

### 2.3 Uniform random sampling tools

BURST integrates multiple URS tools that researchers from different communities have proposed in the recent years. The underlying techniques offer various trade-offs between efficiency and uniformity guarantees. Therefore, BURST comes with multiple state-of-the-art baselines that can support the evaluation of new techniques. The techniques and tools that BURST includes are:

- **UniGen** [6, 8] is a hashing-based algorithm to generate samples in a nearly uniform manner with strong theoretical guarantees: it either produces samples satisfying a nearly uniform distribution or it produces no sample at all. These strong theoretical properties come at a cost: the hashing based approach requires adding large clauses to formulas so they can be sampled. These clauses grow quadratically in size with the number of variables in the formula, which can raise scalability issues.
- **QuickSampler** is an algorithm based on a strong set of heuristics, which are shown to produce samples quickly in practice on a large set of industrial benchmarks [13]. However, the tool offers no guarantee on the distribution of generated samples, or even on the termination of the sampling process and the validity of generated samples (they have to be checked with a SAT solver after the generation phase).
- **Distance-based Sampling** (DbS) [19] is a recent sampling technique designed to improve configuration-performance prediction models. The techniques is based on a distance metric and a probability distribution to control the spread of the random sampling over the configuration space. By doing so, DbS covers different kinds of interactions among configuration options in the sample set. DbS does not aim to generate uniform samples. However, it is an efficient metrics that is competitive to uniform sampling in terms of improving the performance predictions models.
- **SMARCH** [28] follows a divide-and-conquer approach establihing mapping between integers and configurations and exploit multiple CPUs facilities by running computations in parallel.
- **SPUR** [1] is a perfect uniform random sampler that is based on the sharpSAT model counter .
- **KUS** is an uniform sampler based on knowledge compilation techniques: it rewrites CNF formulas in d-DNNF format and iterates over this representation, allowing order of magnitude compared to UniGen [36].
- **ApproxMC/UniGen3** is a near uniform sampler based on approximate model counting. Since model counting of large formulas is computationally heavy, such a tool can drastically improve sampling performance [9, 37, 38]. Because it also integrates hahsing-based techniques of UniGen we refer to it as 'UniGen3'.
- **STS** [14] explores the space of variable assignments in a breadth-first manner and use a SAT solver to complete solutions.
- **CMS** CryptoMiniSAT is a SAT solver originally dedicated to cryptographic applications [39]. Now in version 5.x[1], it integrates with ApproxMC/Unigen3.

[1] https://github.com/msoos/cryptominisat/releases/tag/5.8.0

Overall, support the evaluation of ten distinct sampling tools, issued both from the SPL and SAT/artificial intelligence communities.

## 2.4 Quality Metrics

BURST enables the evaluation of URS techniques according to two qualities: uniformity and efficiency. Uniformity refers to the quality of the samples that a given technique generates, that is, whether it actually samples any configuration with the same likelihood. Efficiency refers to the speed at which the tool can produce such samples.

To assess uniformity, we use a recent statistical test named Barbarik [7]. Barbarik is an algorithmic framework that can test whether a sample's distribution is $\epsilon$-close or $\eta$-far from the uniform distribution where $\epsilon$ and $\eta$ are thresholds. Compared to other statistical methods, Barbarik requires a smaller number of samples, i.e. $O(\frac{1}{(\eta-\epsilon)^4})$. To realize this test, Barbarik invokes a reference, ideal uniform sampling tools to produce a uniform sample. BURST reuses the open-source implementation of Barbarik as available in the official repository.[2]

To assess efficiency, BURST invokes the sampling tool and records the number of samples the tool can generate within a predefined time budget.

## 2.5 BURST configuration parameters

BURST has two execution modes, one for sampling efficiency (**S**) and the other for uniformity assessment (**U**). BURST main configuration parameters are the following:

- S/U . The formulas or folder of CNF formulas in DIMACS format to be analysed, allowing for large scale experiments in batch mode.
- S/U The sampler to evaluated.
- U The sampler that will serves as the reference for uniformity evaluation. SPUR is the reference by default.
- S/U A timeout for experiments, which is useful for slow samplers and large formulas.

Additionally, BURST supports all options provided by Barbarik (such as $\eta$ and $\epsilon$), using default values if omitted. Parameters $\eta$ and $\epsilon$ determine the number of samples required for the evaluation of the uniformity. All supported options are described on BURST's website.

# 3 BURST IMPLEMENTATION

BURST is implemented a set of command-line tools written in Python.

## 3.1 Architecture

BURST utilities are decomposed as follows:

- **Samplers**. All samplers are in samplers directory (and all utilities/dependencies are also in this folder).
- **Sampling experiments.** The `usampling-experiments.py` script pilots the scalability study of samplers over different models.
- **Uniformity experiments.** The file `barbarikloop.py` performs uniformity experiments and store results in a CSV file. It is based on the barbarik tool from Kuldeep Meel et al: https://github.com/meelgroup/barbarik. This version supports uniformity check for all the 10 solvers above and uses SPUR as a reference uniform solver, if not specified.
- **Models.** Our models come from different sources: https://github.com/diverse-project/samplingfm/ (including non-SPL formulas and hard feature models) and https://github.com/PettTo/Feature-Model-History-of-Linux gathering difeertn version of linux models of more than 6,000 features.

To ease usage, these artifacts are embedded in a docker image.

## 3.2 Extending BURST

It is relatively easy to add a new sampler to BURST. One needs to implement a Python function – `getSolutionFromXXX` – in the `SolutionRetriver` class of the `barbarik.py` file. Existing implementations of that function can serve as examples. It is straightforward to add new satisfiability formulae (e.g., coming from feature models), using the 'flas' option allowing to process all files in a given folder.

# 4 CONCLUSION

We presented BURST, a benchmarking platform for assessing random uniform samplers. BURST provides flexible and extensible tools to run samplers and to evaluate their performance as well as the uniformity of generated samples. This platform joins the effort of research aiming at providing systematic and reproducible evaluation and comparison of sampling approaches [31]. Our initial experiments [32] demonstrated that it is challenging for a sampler to achieve both performance and uniformity goals. BURST is an opportunity to evaluate algorithmic developments and demonstrate practical advances on a variety of SPL and non-SPL models, including the most challenging ones. BURST is under development and there is room for future work. First, We would like to include visualisation tools, to understand in a more fine-grained way uniformity deviations [32]. Second, since the development of uniformity testing approaches is a research area in itself [18, 25], we would like to integrate them in BURST. Indeed, akin to model diversity, such tests have different features: Barabarik works with CNF formulas and requires a reference sampler, Heradio *et al.*'s technique works with BDDs and does not need such a reference. We believe that this platform can contribute to the synergistic development of new samplers and testing approaches [25], and more generally to more collaboration between the SAT solving and SPL engineering communities.

# ACKNOWLEDGMENTS

This research was partly funded by the ANR-17-CE25-0010-01 VaryVary project. Gilles Perrouin is a Research Associate at the FNRS. Maxime Cordy was supported by FNR Luxembourg (grant C19/IS/13566661/BEEHIVE/Cordy).

# REFERENCES

[1] D. Achlioptas, Zayd Hammoudeh, and P. Theodoropoulos. 2018. Fast Sampling of Perfectly Uniform Satisfying Assignments. In *SAT*.

[2] https://github.com/meelgroup/barbarik

[2] Juliana Alves Pereira, Mathieu Acher, Hugo Martin, and Jean-Marc Jézéquel. 2020. Sampling effect on performance prediction of configurable systems: A case study. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. 277–288.

[3] Teodora Baluta, Zheng Leong Chua, Kuldeep S. Meel, and Prateek Saxena. 2021. Scalable Quantitative Verification For Deep Neural Networks. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 312–323. https://doi.org/10.1109/ICSE43902.2021.00039

[4] D. Batory, D. Benavides, and A. Ruiz-Cortés. 2006. Automated Analysis of Feature Models: Challenges Ahead. *Commun. ACM* (December 2006).

[5] Erick Cantú-Paz. 2002. On Random Numbers and the Performance of Genetic Algorithms. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation* (New York City, New York) *(GECCO'02)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 311–318. http://dl.acm.org/citation.cfm?id=2955491.2955546

[6] Supratik Chakraborty, Daniel J. Fremont, Kuldeep S. Meel, Sanjit A. Seshia, and Moshe Y. Vardi. 2015. On Parallel Scalable Uniform SAT Witness Generation. In *Tools and Algorithms for the Construction and Analysis of Systems*, Christel Baier and Cesare Tinelli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 304–319.

[7] Sourav Chakraborty and Kuldeep S. Meel. 2019. On Testing of Uniform Samplers. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 7777–7784. https://doi.org/10.1609/aaai.v33i01.33017777

[8] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2014. Balancing scalability and uniformity in SAT witness generator. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1145/2593069.2593097

[9] Supratik Chakraborty, Kuldeep S. Meel, and Moshe Y. Vardi. 2016. Algorithmic Improvements in Approximate Counting for Probabilistic Inference: From Linear to Logarithmic SAT Calls. In *Proceedings of International Joint Conference on Artificial Intelligence (IJCAI)*.

[10] Maxime Cordy, Mike Papadakis, and Axel Legay. 2020. Statistical Model Checking for Variability-Intensive Systems. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 294–314.

[11] Axel de Perthuis de Laillevault, Benjamin Doerr, and Carola Doerr. 2015. Money for Nothing: Speeding Up Evolutionary Algorithms Through Better Initialization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* (Madrid, Spain) *(GECCO '15)*. ACM, New York, NY, USA, 815–822. https://doi.org/10.1145/2739480.2754760

[12] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient sampling of SAT solutions for testing. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 549–559. https://doi.org/10.1145/3180155.3180248

[13] Rafael Dutra, Kevin Laeufer, Jonathan Bachrach, and Koushik Sen. 2018. Efficient sampling of SAT solutions for testing. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 549–559. https://doi.org/10.1145/3180155.3180248

[14] Stefano Ermon, Carla Gomes, and Bart Selman. 2012. Uniform Solution Sampling Using a Constraint Solver as an Oracle. In *Proceedings of the Twenty-Eighth Conference on Uncertainty in Artificial Intelligence* (Catalina Island, CA) *(UAI'12)*. AUAI Press, Arlington, Virginia, USA, 255–264.

[15] Jianmei Guo, Jia Hui Liang, Kai Shi, Dingyu Yang, Jingsong Zhang, Krzysztof Czarnecki, Vijay Ganesh, and Huiqun Yu. 2017. SMTIBEA: a hybrid multi-objective optimization algorithm for configuring large constrained software product lines. *Software & Systems Modeling* (22 Jul 2017). https://doi.org/10.1007/s10270-017-0610-0

[16] Axel Halin, Alexandre Nuttinck, Mathieu Acher, Xavier Devroey, Gilles Perrouin, and Benoit Baudry. 2019. Test them all, is it worth it? Assessing configuration sampling on the JHipster Web development stack. *Empirical Software Engineering* 24, 2 (2019), 674–717.

[17] Christopher Henard, Mike Papadakis, Mark Harman, and Yves Le Traon. 2015. Combining Multi-objective Search and Constraint Solving for Configuring Large Software Product Lines. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1* (Florence, Italy) *(ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 517–528. http://dl.acm.org/citation.cfm?id=2818754.2818819

[18] Ruben Heradio, David Fernandez-Amoros, José A Galindo, and David Benavides. 2020. Uniform and scalable SAT-sampling for configurable systems. In *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. 1–11.

[19] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-based sampling of software configuration spaces. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1084–1094.

[20] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. 1990. *Feature-Oriented Domain Analysis (FODA)*. Technical Report CMU/SEI-90-TR-21. SEI.

[21] Alexander Knüppel, Thomas Thüm, Stephan Mennicke, Jens Meinicke, and Ina Schaefer. 2017. Is there a mismatch between real-world feature models and product-line research?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 291–302. https://doi.org/10.1145/3106237.3106252

[22] Sebastian Krieter, Thomas Thüm, Sandro Schulze, Reimar Schröter, and Gunter Saake. 2018. Propagating configuration decisions with modal implication graphs. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*. 898–909. https://doi.org/10.1145/3180155.3180159

[23] Jia Hui Liang, Vijay Ganesh, Krzysztof Czarnecki, and Venkatesh Raman. 2015. SAT-based Analysis of Large Real-world Feature Models is Easy. In *Proceedings of the 19th International Conference on Software Product Line* (Nashville, Tennessee) *(SPLC '15)*. ACM, New York, NY, USA, 91–100. https://doi.org/10.1145/2791060.2791070

[24] H. Maaranen, K. Miettinen, and M. M. Mäkelä. 2004. Quasi-random Initial Population for Genetic Algorithms. *Comput. Math. Appl.* 47, 12 (June 2004), 1885–1895. https://doi.org/10.1016/j.camwa.2003.07.011

[25] Kuldeep S. Meel, Yash Pote, and Sourav Chakraborty. 2020. On Testing of Samplers. In *Advances in Neural Information Processing Systems(NeurIPS)*.

[26] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. [n.d.]. *Mastering software variability with FeatureIDE*. Springer.

[27] Jeho Oh, Don S. Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding near-optimal configurations in product lines by random sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, Eric Bodden, Wilhelm Schäfer, Arie van Deursen, and Andrea Zisman (Eds.). ACM, 61–71. https://doi.org/10.1145/3106237.3106273

[28] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Margaret Myers. 2020. *Scalable Uniform Sampling for Real-World Software Product Lines*. Technical Report TR-20-01.

[29] Jeho Oh, Paul Gazzillo, and Don S. Batory. 2019. *t*-wise coverage by uniform sampling. In *Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, Paris, France, September 9-13, 2019*, Thorsten Berger, Philippe Collet, Laurence Duchien, Thomas Fogdal, Patrick Heymans, Timo Kehrer, Jabier Martinez, Raúl Mazo, Leticia Montalvillo, Camille Salinesi, Xhevahire Tërnava, Thomas Thüm, and Tewfik Ziadi (Eds.). ACM, 15:1–15:4.

[30] Johan Oudinet, Alain Denise, Marie-Claude Gaudel, Richard Lassaigne, and Sylvain Peyronnet. 2011. Uniform Monte-Carlo Model Checking. In *Fundamental Approaches to Software Engineering - 14th International Conference, FASE 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6603)*, Dimitra Giannakopoulou and Fernando Orejas (Eds.). Springer, 127–140.

[31] Tobias Pett, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Ina Schaefer. 2021. AutoSMP: An Evaluation Platform for Sampling Algorithms. In *Proc. Int'l Systems and Software Product Line Conf. (SPLC)*. ACM, New York, NY, USA. To appear.

[32] Quentin Plazar, Mathieu Acher, Gilles Perrouin, Xavier Devroey, and Maxime Cordy. 2019. Uniform Sampling of SAT Solutions for Configurable Systems: Are We There Yet?. In *ICST '19 (to appear)*.

[33] Matt Raible. 2015. *The JHipster mini-book*. C4Media.

[34] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. 2013. On the value of user preferences in search-based software engineering: a case study in software product lines. In *ICSE'13*. 492–501.

[35] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE'06)*. IEEE Computer Society, Washington, DC, USA, 136–145. https://doi.org/10.1109/RE.2006.23

[36] Shubham Sharma, Rahul Gupta, Subhajit Roy, and Kuldeep S. Meel. 2018. Knowledge Compilation meets Uniform Sampling. In *Proceedings of International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR)*.

[37] Mate Soos, Stephan Gocht, and Kuldeep S. Meel. 2020. Tinted, Detached, and Lazy CNF-XOR solving and its Applications to Counting and Sampling. In *Proceedings of International Conference on Computer-Aided Verification (CAV)*.

[38] Mate Soos and Kuldeep S. Meel. 2019. BIRD: Engineering an Efficient CNF-XOR SAT Solver and its Applications to Approximate Model Counting. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*.

[39] Mate Soos, Karsten Nohl, and Claude Castelluccia. 2009. Extending SAT solvers to cryptographic problems. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 244–257.

[40] Paul Temple, José A. Galindo, Mathieu Acher, and Jean-Marc Jézéquel. 2016. Using Machine Learning to Infer Constraints for Product Lines. In *Proceedings of the 20th International Systems and Software Product Line Conference* (Beijing, China) *(SPLC '16)*. Association for Computing Machinery, New York, NY, USA, 209–218. https://doi.org/10.1145/2934466.2934472

# APPENDIX

## A BURST DEMONSTRATION

Our demonstration has four steps. We begin with an introduction to uniform random sampling and why URS is an important problem for SPL research and other communities. We, then, present the different sampling tools that exist in the literature and are included in BURST. We provide an overview of BURST, its components and functionalities. Finally, we detail BURST usage scenarios as described below.

## B BURST USAGE SCENARIOS

### B.1 Setup

BURST is based on Docker to host data, scripts, and samplers. The Python scripts in charge of executing samplers and gathering results are available on Github.

```
git clone https://github.com/FAMILIAR-project/
    ↪ usampling-exp
docker pull macher/usampling:squashed
```

Once the Docker is pulled, one can run a container with the last up-to-date scripts of Github repository:

```
cd usampling-exp
docker run -it -v $(pwd):/home/usampling-exp macher
    ↪ /usampling:squashed /bin/bash
```

SAT formulae are located in a specific folder and come from different benchmarks or feature models:

```
ls -d /home/samplingfm/Benchmarks/*/
  /home/samplingfm/Benchmarks/Blasted_Real/
  /home/samplingfm/Benchmarks/FMEasy/
  /home/samplingfm/Benchmarks/FeatureModels/
  /home/samplingfm/Benchmarks/V15/
  /home/samplingfm/Benchmarks/V3/
  /home/samplingfm/Benchmarks/V7/
```

Please note that the setup phase is out of the scope of the tool demonstration since it requires to download large docker images (up to 10GB).

### B.2 Sampling Performance

It is possible to run a Docker container without the interactive mode, typically to measure performance of samplers:

```
docker run -v $(pwd):/home/usampling-exp:z macher/
    ↪ usampling:squashed /bin/bash -c 'cd␣/home/
    ↪ usampling-exp/;␣echo␣STARTING;␣python3␣
    ↪ barbarikloop.py␣-flas␣/home/samplingfm/
    ↪ Benchmarks/FeatureModels/␣--sampler␣9␣--ref-
    ↪ sampler␣6␣--seed␣1␣--timeout␣5600;␣echo␣END'
```

The current list of supported samplers is as follows.

```
SAMPLER_UNIGEN = 1
SAMPLER_QUICKSAMPLER = 2
SAMPLER_STS = 3
SAMPLER_CMS = 4
SAMPLER_UNIGEN3 = 5
SAMPLER_SPUR = 6
SAMPLER_SMARCH = 7
SAMPLER_UNIGEN2 = 8
SAMPLER_KUS = 9
SAMPLER_DISTAWARE = 10
```

Typical outcomes are:

```
cat usampling-data/experiments-DBS.csv
formula_file,timeout,execution_time_in,exception_dbs
/home/samplingfm/Benchmarks/FeatureModels/FM-3.6.1-
    ↪ refined.cnf,False,0.6199491024017334,False

cat usampling-data/experiments-KUS.csv
formula_file,timeout,execution_time_in,dnnf_time,
    ↪ sampling_time,model_count,counting_time,
    ↪ dnnfparsing_time
/home/samplingfm/Benchmarks/FeatureModels/FM-3.6.1-
    ↪ refined.cnf,False, 0.1399824619293213,
    ↪ 0.011404275894165039, 0.0007951259613037109,
    ↪ 26256, 0.0008006095886230469,
    ↪ 0.0012776851654052734

cat usampling-data/experiments-SPUR.csv
formula_file,execution_time_in,timeout
/home/samplingfm/Benchmarks/Blasted_Real/
    ↪ blasted_case141.cnf,1,True
/home/samplingfm/Benchmarks/Blasted_Real/
    ↪ blasted_case142.cnf,1,True

cat usampling-data/experiments-Unigen2.csv
formula_file,timeout,execution_time_in
/home/samplingfm/Benchmarks/FeatureModels/FM-3.6.1-
    ↪ refined.cnf,False,0.045912742614746094
```

### B.3 Uniformity Analysis

The following command performs uniformity analysis on the JHipster FM using CMS as the target sampler and SPUR as reference for a sampling budget of 5000 samples:

```
python3 barbarikloop.py --maxSamples 50000 --
    ↪ minSamples 0 --ref-sampler 6 --sampler 4 --
    ↪ seed 1 --delta 0.05 --epsilon 0.3 --eta 0.9 -
    ↪ flas /home/samplingfm/Benchmarks/
    ↪ FeatureModels/FM-3.6.1-refined.cnf
```

The content of the generated CSV file should look something like this:

```
cat output/c1f1b9a13035439383912ef57a98535d/Uniform-
    ↪ CustomSampler.csv
file,time,cmd_output,err_output,Uniform,Timeout
/home/gilles/FeatureModels/FM-3.6.1-refined.cnf
    ↪ ,1.782,...,b'',True,FALSE
```