BURST

Acher, Mathieu; Perrouin, Gilles; Cordy, Maxime

Published in:
Science of Computer Programming

DOI:
10.1016/j.scico.2022.102914

Publication date:
2023

Document Version
Peer reviewed version

BURST: Benchmarking Uniform Random Sampling Techniques

Mathieu Acher[a], Gilles Perrouin[b], Maxime Cordy[c]

[a] *Univ Rennes, CNRS, Inria, IRISA, Institut Universitaire de France (IUF), France*
[b] *PReCISE, NaDI, Faculty of Computer Science, University of Namur*
[c] *SnT, University of Luxembourg*

Abstract

BURST is a benchmarking platform for uniform random sampling (URS) techniques. Given: i) the description of a sampling space provided as a Boolean formula (DIMACS), and ii) a sampling budget (time and strength of uniformity), BURST evaluates ten samplers for scalability and uniformity. BURST measures scalability based on the time required to produce a sample, and uniformity based on the state-of-the-art and proven statistical test Barbarik. BURST is easily extendable to new samplers and offers: i) 128 feature models (for highly-configurable systems), ii) many other models mined from the artificial intelligence/satisfiability solving benchmarks. BURST envisions supporting URS assessment and design across multiple research communities.

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

---

Code metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.1
C2	Permanent link to code/repository used for this code version	`https://archive.softwareheritage.org/swh:1:dir:c7e0d46e4cfa63c5365073422bfba73140a6d3ab;origin=https://github.com/diverse-project/BURST;visit=swh:1:snp:ab9b0c4b268f096da2c3f0cf943bdd3c28f486ab;anchor=swh:1:rev:8ceeb56500d25432f220eb480da95ff496417eb7`
C3	Permanent link to Reproducible Capsule	Github for command line interface (CLI) use: `https://github.com/diverse-project/BURST` and via Docker at `https://hub.docker.com/repository/docker/macher/usampling`
C4	Legal Code License	MIT
C5	Code versioning system used	Git and Github
C6	Software code languages, tools, and services used	Python, Jupyter notebooks, Bash, SAT samplers written in Python, C++, and .NET, and statistical testing procedure (Barbarik)
C7	Compilation requirements, operating environments & dependencies	Python > 3, Docker image
C8	If available Link to developer documentation/manual	`https://github.com/diverse-project/BURST/#README.md`
C9	Support email for questions	mathieu.acher@irisa.fr

Table 1: Code metadata (mandatory)

Software metadata

Nr.	(Executable) software metadata description	Please fill in this column
S1	Current software version	1.0
S2	Permanent link to executables of this version	`https://github.com/diverse-project/BURST/releases/latest`
S3	Permanent link to Reproducible Capsule	`https://github.com/diverse-project/BURST/`
S4	Legal Software License	MIT
S5	Computing platforms/Operating Systems	Docker (solution tested on MacOS and Linux)
S6	Installation requirements & dependencies	Docker (that includes necessary tools, including Python); samplers are embedded into the Docker
S7	If available, link to user manual - if formally published include a reference to the publication in the reference list	`https://github.com/diverse-project/BURST/#README.md`
S8	Support email for questions	mathieu.acher@irisa.fr

Table 2: Software metadata (optional)

1. Background & Motivation

This paper presents a platform to assess uniform random sampling (URS) algorithms for highly-configurable systems (HCS). HCS allow customisation of their behaviour though the activation of options or features. These features are organised within Feature Models (FMs) [1], amenable to formal analysis via a translation to propositional logic [2, 3]. HCS customisation comes at the price of the explosion of the number of variants one can derive from combinations of options, i.e., *configurations*. This explosion leads to the impossibility to assess all configurations. As a result, different sampling strategies for HCS have been devised [4, 5]. Uniform random sampling attributes the same selection probability to each configuration, and therefore makes no hypothesis on its characteristics, allowing an unbiased exploration of the configuration space. URS also supports verification [6] or search-based techniques [7]. To this end, several authors offered URS algorithms, either in the HCS community [8, 9] or in the constraint solving one [10, 11, 12]. Each algorithm exposes trade-offs in terms of scalability and statistical guarantees of uniformity.

Based on our previous experience [13], we designed BURST to explore the following research questions:

RQ1: How scalable are the different URS algorithms?

RQ2: How uniform are the different URS algorithms?

2. Implementation

BURST answers our research questions above by providing: *i)* a set of command line tools written in Python, *ii)* a set of 10 URS tools (see Section 5) to perform comparisons, and *iii)* a set of 128 feature models [14, 15, 16] capturing the configuration spaces of various HCS such Linux kernel versions, command line tools such as busybox, and a model of Jhipster [17, 5], a configurable WebApp generator. These models are provided as Conjunctive Normal Form (CNF) Boolean formulas in the DIMACS format. The whole BURST platform is integrated within a docker image to ease replicability. The BURST platform is organised 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.* [18]: `https://github.com/meelgroup/barbarik`. This version supports uniformity check for all the 10 solvers above and uses SPUR as a reference uniform sampler named SPUR [19] by default. The reference solver can be specified in BURST calls.
- **Models.** A set of various HCS and non-HCS models as introduced above.

3. Results

Early experiments with the platform tend to confirm our initial observations [13]: *it is generally difficult to marry scalability (i.e., producing samples fast on large formulas) with uniformity (i.e., ensuring that all configurations have equal selection probabilities).*

However, recent development of the CMS sampler, named CMSgen [20], seem to offer a very encouraging trade-off of producing samples fast with good uniformity properties.

4. Related Literature

BURST has been originally presented at the SPLC 2021 tool demonstrations track [21]. This original software publication adds more information about technical details of the platform (metadata) and the possibility to be updated as the platform evolves.

We are not aware other initiatives to benchmark uniform random sampling tools with the following characteristics: *i)* a diverse and extendable set of samplers and *ii)* a varied set of CNF formulas to choose from, including feature models of various size and models issued from the SAT/AI communities.. However, we would like to point the work of Meel *et al.* who demonstrate, through the use of Barbarik tool BURST also relies on, that better evaluation of uniformity leads to better samplers, that in turn suggests improvement of benchmarking tools and evaluation metrics [20, 22]. We believe that BURST can contribute to this virtuous circle.

BURST also takes place in the wide initiative to evaluate sampling techniques for HCS [4, 5, 13, 23].

5. Illustrative Example

We illustrate the usage of the platform using docker for scalability and uniformity.

5.1. Sampling Performance

The following command performs scalability analysis on the JHipster feature model using KUS [24] as the target sampler:

```
docker run macher/usampling:squashed /bin/bash -c 'cd␣/home/
   ↪ usampling-exp/;␣echo␣STARTING;␣python3␣usampling-
   ↪ experiments.py␣-flas␣/home/samplingfm/Benchmarks/
   ↪ FeatureModels/FM-3.6.1-refined.cnf␣--kus;␣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-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
```

The meaning of columns of the CSV file is as follows: 'formula_file': the name of the processed model; 'timeout': whether a timeout has been reached or not; 'execution_time_in' overall execution time; 'dnnf_time': the time required by KUS to compile the corresponding DNNF formula; 'sampling_time': time taken to produce the samples; 'model_count': the number of solutions in the model; 'counting_time': time taken to count solutions; 'dnnfparsing_time': time taken to parse the compiled DNNF formula. All times are reported in seconds.

5.2. Uniformity Analysis

Within the Docker image, 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
```

The meaning of columns of the CSV file is as follows: 'file': the name of the processed formula; 'cmd_output': the output of the command (debugging purposes); 'err_output': captures of possible errors (debugging purposes); 'Uniform': whether the model is uniform; 'Timeout': whether the timeout (in seconds) has been reached.

6. Conclusion

In this paper, we presented BURST, a generic Uniform Random Sampling evaluation platform for a variety of domains. Provided with a SAT formula describing the configuration space and a testing budget (time and strength of uniformity desired), BURST evaluates a variety of uniform random samplers with respect to scalability and uniformity. BURST records results in CSV file to ease further analyses. BURST is conveniently provided in Docker container, integrates 10 random uniform samplers, provides a large choice (128) of feature models and is easily to extensible to new samplers. We hope to make BURST a tool of choice to support the design of new uniform random sampling techniques.

Acknowledgements

The authors would particularly like to thank Kuldeep S. Meel from National University of Singapore, Mate Soos from Zalando Germany and their colleagues for their help setting up and fixing Barbarik as well as the CMS samplers. 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] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature-Oriented Domain Analysis (FODA), Tech. Rep. CMU/SEI-90-TR-21, SEI (Nov. 1990).

[2] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, 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, 2006, pp. 136–145. `doi:http://dx.doi.org/10.1109/RE.2006.23`.

[3] D. S. Batory, Feature models, grammars, and propositional formulas, in: SPLC'05, Vol. 3714 of LNCS, 2005, pp. 7–20.

[4] F. Medeiros, C. Kästner, M. Ribeiro, R. Gheyi, S. Apel, A comparison of 10 sampling algorithms for configurable systems, in: ICSE'16.

[5] A. Halin, A. Nuttinck, M. Acher, X. Devroey, G. Perrouin, B. Baudry, Test them all, is it worth it? assessing configuration sampling on the jhipster web development stack, Empirical Software Engineering 24 (2) (2019) 674–717.

[6] M. Cordy, M. Papadakis, A. Legay, Statistical model checking for variability-intensive systems, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2020, pp. 294–314.

[7] A. de Perthuis de Laillevault, B. Doerr, C. Doerr, Money for nothing: Speeding up evolutionary algorithms through better initialization, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15, ACM, New York, NY, USA, 2015, pp. 815–822. `doi:10.1145/2739480.2754760`.
URL `http://doi.acm.org/10.1145/2739480.2754760`

[8] J. Oh, D. S. Batory, M. Myers, N. Siegmund, Finding near-optimal configurations in product lines by random sampling, in: E. Bodden, W. Schäfer, A. van Deursen, A. Zisman (Eds.), Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE

2017, Paderborn, Germany, September 4-8, 2017, ACM, 2017, pp. 61–71. doi:10.1145/3106237.3106273.
URL https://doi.org/10.1145/3106237.3106273

[9] R. Heradio, D. Fernandez-Amoros, J. A. Galindo, D. Benavides, 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, 2020, pp. 1–11.

[10] M. Soos, K. S. Meel, Bird: Engineering an efficient cnf-xor sat solver and its applications to approximate model counting, in: Proceedings of AAAI Conference on Artificial Intelligence (AAAI), 2019.

[11] M. Soos, S. Gocht, K. S. Meel, Tinted, detached, and lazy cnf-xor solving and its applications to counting and sampling, in: Proceedings of International Conference on Computer-Aided Verification (CAV), 2020.

[12] R. Dutra, K. Laeufer, J. Bachrach, K. Sen, 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, 2018, pp. 549–559. doi:10.1145/3180155.3180248.
URL http://doi.acm.org/10.1145/3180155.3180248

[13] Q. Plazar, M. Acher, G. Perrouin, X. Devroey, M. Cordy, Uniform sampling of sat solutions for configurable systems: Are we there yet?, in: ICST '19, 2019.

[14] A. Knüppel, T. Thüm, S. Mennicke, J. Meinicke, I. Schaefer, 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, 2017, pp. 291–302. doi:10.1145/3106237.3106252.
URL http://doi.acm.org/10.1145/3106237.3106252

[15] S. Krieter, T. Thüm, S. Schulze, R. Schröter, G. Saake, 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, 2018, pp. 898–909. doi:10.1145/3180155.3180159.
URL http://doi.acm.org/10.1145/3180155.3180159

[16] J. H. Liang, V. Ganesh, K. Czarnecki, V. Raman, Sat-based analysis of large real-world feature models is easy, in: Proceedings of the 19th International Conference on Software Product Line, SPLC '15, ACM, New

York, NY, USA, 2015, pp. 91–100. doi:10.1145/2791060.2791070.
URL http://doi.acm.org/10.1145/2791060.2791070

[17] M. Raible, The JHipster mini-book, C4Media, 2015.

[18] S. Chakraborty, K. S. Meel, 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, 2019, pp. 7777–7784. doi:10.1609/aaai.v33i01.33017777.
URL https://doi.org/10.1609/aaai.v33i01.33017777

[19] D. Achlioptas, Z. Hammoudeh, P. Theodoropoulos, Fast sampling of perfectly uniform satisfying assignments, in: SAT, 2018.

[20] P. Golia, M. Soos, S. Chakraborty, K. S. Meel, Designing samplers is easy: The boon of testers, in: 2021 Formal Methods in Computer Aided Design (FMCAD), IEEE, 2021, pp. 222–230.

[21] M. Acher, G. Perrouin, M. Cordy, Burst: a benchmarking platform for uniform random sampling techniques, in: Proceedings of the 25th ACM International Systems and Software Product Line Conference-Volume B, 2021, pp. 36–40.

[22] K. S. Meel, Y. Pote, S. Chakraborty, On testing of samplers, in: Advances in Neural Information Processing Systems(NeurIPS), 2020.

[23] R. Heradio, D. Fernandez-Amoros, J. A. Galindo, D. Benavides, D. Batory, Uniform and scalable sampling of highly configurable systems, Empirical Software Engineering 27 (2) (2022) 1–34.

[24] S. Sharma, R. Gupta, S. Roy, K. S. Meel, Knowledge compilation meets uniform sampling, in: Proceedings of International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR), 2018.