Baital: Sampling Configurable Systems with high t-wise coverage[1]

Eduard Baranov, Axel Legay

UCLouvain, Belgium

Abstract

Testing of highly configurable systems is challenging due to an immense number of configurations and is usually performed on a small sample set. Its quality is often measured with t-wise coverage. We propose a tool Baital for sampling highly configurable systems with thousands of features capable of achieving high t-wise coverage. The effectiveness and scalability of the tool are based on two novel techniques: the adaptive weighted sampling and the approximation techniques for the t-wise coverage computation. Baital can easily handle sampling and computation for 6-wise coverage. The latest version supports multi-valued features.

Keywords: configurable systems, adaptive weighted sampling, t-wise coverage

Metadata

1. Motivation and significance

In the modern world, software is involved in every aspect of our lives including critical domains such as automotive, electric power, and healthcare. The diversity of application scenarios and economic factors has led to the design of highly-configurable systems which can be adapted to user's needs by selecting features. Yet configurability does not come without a price: feature dependencies are common [1] and could lead to variability bugs appearing only in some configurations. The complexity of testing and finding such bugs is amplified by the fact that it is common in modern systems to have thousands of features resulting in an extreme number of configurations [2]. It is infeasible to test all system configurations, therefore, testing is usually performed on a small sample set of configurations.

[1]Revised version available at https://doi.org/10.1016/j.scico.2024.103209

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v2.0.0
C2	Permanent link to code/repository used for this code version	https://github.com/meelgroup/baital
C3	Permanent link to Reproducible Capsule	
C4	Legal Code License	MIT
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	Python 3
C7	Compilation requirements, operating environments and dependencies	Linux OS (Tested on Ubuntu, Debian, and CentOS); CMSGen, z3, ApproxMC4, WAPS, d4
C8	If available, link to developer documentation/manual	
C9	Support email for questions	eduard.baranov@uclouvain.be

Table 1: Code metadata (mandatory)

One of the common approaches for the sample set selection is based on sampling: configurations are randomly selected from valid ones. Sample set quality is often measured with t-wise coverage that computes a proportion of t-sized combinations of features covered by the sample set. Uniform sampling uses uniform distribution for the selection process and until recently it has been the dominant sampling approach, however, its effectiveness is limited due to the unequal distribution of features among configurations. For example, in the work [3] it has been shown that uniform sampling can reach only 48% 2-wise coverage with 1000 samples on a feature model from [4].

The computation of t-wise coverage is also challenging. Only few works consider 3-wise coverage while the computation for $t \geq 4$ is infeasible on benchmarks with thousands of features. Yet 3-sized combinations might not be enough: authors of [5] suggest considering up to 6-sized combinations and the work [6] describes a bug involving the interaction of 5 features. Existing tools, e.g. [7, 3, 8, 9], either utilize uniform sampling or do not scale above $t > 3$.

In this paper, we present Baital, originally presented in [10], a tool for sampling configurable systems and computing t-wise coverage. The tool provides efficient and scalable sampling and t-wise coverage computation for systems with thousands of features and high values of t. The latest updates include multiple optimizations and support for multi-valued features, i.e.

features taking values from a discrete domain.

For sampling the tool utilizes the novel approach called adaptive weighted sampling [11] that shows a significant increase in coverage in comparison with uniform sampling. The main idea of the approach is to periodically modify the distribution during the sampling process by reducing weights for features with already high coverage and by increasing weights for features that have low coverage. For a selected distribution, Baital can utilize either WAPS[12] or CMSGen[13] sampling tools. In the case of multi-valued features, sampling is preceded by the conversion of system constraints into conjunctive normal form (CNF) with z3[14]. To address the challenge of t-wise coverage computation, Baital uses approximation algorithms ApproxCov and ApproxMaxCov [15]. The evaluation showed that 6-wise coverage can be computed for systems with thousands of features within 1 hour.

Baital is publicly available at Github[2]. The Readme file contains installation instructions, and examples of use. Benchmarks and evaluation results are available at the separate repository[3].[4]

2. Software description

Baital is written in Python and combines sampling and t-wise coverage computation. In this section, we provide a high-level description of the tool.

2.1. Software functionalities

- Given a configurable system, Baital uses adaptive weighted sampling to generate a sample set of configurations. Multiple parameters can guide the sampling process with various heuristics.

- Given a configurable system and a sample set of configurations, Baital computes or approximates t-wise coverage of the sample set. While exact computation does not scale and, to the best of our knowledge, none of the tools is capable of computing it for $t > 3$, the approximation algorithms can easily scale above this value [15]. Approximation algorithms have theoretical guarantees of the approximation error bound that can be selected by the user.

2.2. Software architecture

The workflow of Baital is shown in Figure **??**. The tool performs three steps: preprocessing, sampling, and coverage computation.

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

[3]https://github.com/edbaranov/feature-model-benchmarks

[4]For the reviewing purposes we provide a virtual machine with the tool installed here.

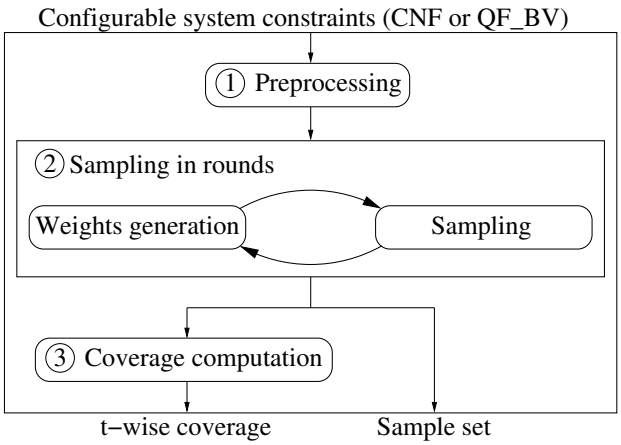

Figure 1: General workflow of Baital.

An input to Baital is a file with a set of constraints on feature dependencies. For systems with binary features, the constraints are given in a CNF form. For systems with multi-valued features, the constraints are given in a Quantifier-Free BitVector logic and are transformed into CNF with bit-blasting. This approach is more efficient than using SMT or CP solvers [16].

The first step is preprocessing which required for some of the sampling heuristics. It computes an exact value or an approximation of the number of distinct feature combinations involving a feature f in all configurations satisfying the constraints for all features of the system. This step can be skipped if the results are not required by the sampling step.

The second step generates a sample set with Adaptive Weighted Sampling. Its execution is performed in rounds. At the beginning of each round weights are assigned to all literals and the sampling process generates a part of the sample set following the distribution imposed by the weights. Originally, Baital used WAPS tool for sample generation, however in the latest versions we have switched to a more efficient CMSGen tool maintaining the support of WAPS. There are several strategies to select weights for literals with different trade-offs between performance and the resulting t-wise coverage. This step outputs a generated sample set that is a part of the final output and is an input to the third step.

The final step computes the t-wise coverage of the sample set using either exact or approximate computations. Approximation algorithms are scalable and can be applied to high values of t.

3. Illustrative examples

We illustrate the tool application with a feature model "FinancialServices01" version "2018-05-09" ("financial" in the following) from [4] that after its transformation into CNF has 771 variables and 7241 clauses. Authors of [3] showed that uniform sampling could not achieve high 2-wise coverage: only 48% coverage can be reached with 1000 samples. We used a laptop with Intel(R) Core i7-8650U CPU to report the runtime in the following examples.

The main entry point is *baital.py* script (*baital_nf.py* for systems with multi-valued features). The tool is running by calling it from Python3 providing a path to a file with configuration system constraints as an argument. Optional arguments are used to control the execution and are identical for *baital.py* and *baital_nf.py*.

Assuming that we want to generate several sampling sets for the same configurable system, it is possible to run the preprocessing step separately and reuse its results in the following executions. The following two commands are used to get exact and approximate results where *--twise* argument defines the size of combinations:

```
$ python3 baital.py ./financial.cnf --twise 2 --preprocess-only
$ python3 baital.py ./financial.cnf --twise 2 --preprocess-only
    --preprocess-approximate
```

On our laptop, the commands take about 60 and 15 minutes, respectively. The tool creates two files *results/financial_2.comb* and *results/financial_2.acomb* with exact and approximate results, respectively. The output directory can be changed with the *--outputdir* parameter.

There are several options controlling the sampling process, the most important ones are *--samples*, *--rounds*, and *--strategy* defining the number of configurations to be generated, the number of weights updates, and the strategy for weight update, respectively. The argument *--preprocess-file* allows the user to reuse the file with preprocessing results. The example of the command is shown below:

```
$ python3 baital.py ./financial.cnf --twise 2 --samples 1000 --rounds 10
                    --strategy 1 --preprocess-file results/financial_2.comb
```

The tool generates a file *results/financial.samples* where each line represents a configuration with a list of selected features. For systems with multi-valued features, two files are generated with *.nsamples* and *.rsamples* extensions. The former has a numerical format used by coverage computation, while the latter has a readable format *feature_name=value*.

The output contains information on the execution time for each of the steps in seconds, the number of combinations in the generated samples and the resulting coverage. Note that the preprocessing results are reused for coverage computation (otherwise it would perform 60 minutes of computations from preprocessing).

```
Sampling time 24.77701187133789
Coverage time 23.858088731765747
Total time 48.635111570358276
(Approximate) number of combinations 900988
(Approximate) 2-wise coverage 0.9823780188627814
```

It is also possible to compute the coverage for a given set of configurations by setting *--no-sampling* and *--samples-file* arguments. It order to perform the approximate computation *--cov-approximate*, *--cov-delta*, and *--cov-epsilon* arguments. The latter two control the approximation error.

```
$ python3 baital.py ../../benchmarks/smarch/financial.cnf --twise 2
  --no-sampling --samples-file results/financial.samples --cov-approximate
...
Coverage time 119.61899900436401
Total time 119.61900901794434
(Approximate) number of combinations 905106
(Approximate) 2-wise coverage 0.9875894727653631
```

In this example, Baital achieves above 98% of coverage in comparison with 48% achieved by uniform sampling.

Coverage computation for $t \geq 3$ requires a lot of resources, however, the approximation can be computed relatively quickly on a laptop. The computation for the financial benchmark takes less than 4, 6, 10, and 18 minutes for t values from 3 to 6, respectively.

4. Impact

Testing of configurable systems is limited to a small set of configurations and, therefore, the quality of the test suite is of utmost importance. It has to cover as many feature combinations as possible to detect variability bugs. Baital improves test suite generation in two directions.

- Adaptive weighted sampling significantly improves the quality of test suites in comparison with previously used uniform sampling. On a large set of publicly available benchmarks, almost all test suites generated with uniform sampling do not reach 80% coverage, while Baital achieves coverage above 95%.

- While bugs can appear due to an interaction of multiple features, only few works measured 3-wise coverage and none went beyond. Baital provides an efficient approximation with theoretically proved error bound that scales for higher values of t.

5. Conclusions

In this paper, we present Baital for sampling configurable systems that can achieve high t-wise coverage, can be applied to the high values of t, and supports multi-valued features. Sample sets generated with Baital have significantly higher t-wise coverage than ones obtained with uniform sampling. The novel approximation algorithms allow users to estimate the t-wise coverage for the values of t beyond the capabilities of other tools.

Acknowledgements

This work was supported by Walloon Region (Belgium) projects DeepConstruct (Convention n°8560). Computational resources have been provided by the supercomputing facilities of the Université catholique de Louvain (CISM/UCL) and the Consortium des Équipements de Calcul Intensif en Fédération Wallonie Bruxelles (CÉCI) funded by the Fond de la Recherche Scientifique de Belgique (F.R.S.-FNRS) under convention 2.5020.11 and by the Walloon Region.

References

[1] I. Rodrigues, M. Ribeiro, F. Medeiros, P. Borba, B. Fonseca, R. Gheyi, Assessing fine-grained feature dependencies, Inf. Softw. Technol. 78 (2016) 27–52.

[2] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, A. Wasowski, A survey of variability modeling in industrial practice, in: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, 2013, pp. 1–8.

[3] J. Oh, P. Gazzillo, D. S. Batory, t-wise coverage by uniform sampling, in: Proceedings of the 23rd International Systems and Software Product Line Conference, SPLC 2019, Volume A, 2019, 2019, pp. 15:1–15:4.

[4] T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, I. Schaefer, Product sampling for product lines: the scalability challenge, in: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, 2019, pp. 78–83.

[5] D. R. Kuhn, R. N. Kacker, Y. Lei, Practical combinatorial testing, NIST special Publication 800 (142) (2010) 142.

[6] I. Abal, C. Brabrand, A. Wasowski, 42 variability bugs in the linux kernel: a qualitative analysis, in: Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014, pp. 421–432.

[7] D. Achlioptas, Z. S. Hammoudeh, P. Theodoropoulos, Fast sampling of perfectly uniform satisfying assignments, in: International Conference on Theory and Applications of Satisfiability Testing, Springer, 2018, pp. 135–147.

[8] R. Dutra, K. Laeufer, J. Bachrach, K. Sen, Efficient sampling of sat solutions for testing, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), IEEE, 2018, pp. 549–559.

[9] C. Luo, B. Sun, B. Qiao, J. Chen, H. Zhang, J. Lin, Q. Lin, D. Zhang, Ls-sampling: An effective local search based sampling approach for achieving high t-wise coverage, ESEC/FSE 2021, 2021, p. 1081–1092.

[10] E. Baranov, A. Legay, Baital: an adaptive weighted sampling platform for configurable systems, in: Proceedings of the 26th ACM International Systems and Software Product Line Conference-Volume B, 2022, pp. 46–49.

[11] E. Baranov, A. Legay, K. S. Meel, Baital: an adaptive weighted sampling approach for improved t-wise coverage, in: ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020, pp. 1114–1126.

[12] R. Gupta, S. Sharma, S. Roy, K. S. Meel, Waps: Weighted and projected sampling, in: Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS), 2019.

[13] P. Golia, M. Soos, S. Chakraborty, K. S. Meel, Designing samplers is easy: The boon of testers, in: Proceedings of Formal Methods in Computer-Aided Design (FMCAD), 2021.

[14] L. De Moura, N. Bjørner, Z3: An efficient SMT solver, in: Proc. of TACAS, Springer, 2008, pp. 337–340.

[15] E. Baranov, S. Chakraborty, A. Legay, K. S. Meel, V. N. Variyam, A scalable t-wise coverage estimator, in: Proc. 44th International Conference on Software Engineering (ICSE 2022), 2022.

[16] D.-J. Munoz, J. Oh, M. Pinto, L. Fuentes, D. Batory, Uniform random sampling product configurations of feature models that have numerical features, in: Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A, 2019, pp. 289–301.