Baital: An Adaptive Weighted Sampling Platform for Configurable Systems

Eduard Baranov
Université Catholique de Louvain
Belgium

Axel Legay
Université Catholique de Louvain
Belgium

ABSTRACT

The diversity of software application scenarios has led the evolution towards highly configurable systems. Testing of such systems is challenging due to an immense number of configurations and is usually performed on a small sample set. Sampling is a promising approach for the sample set generation. t-wise coverage is often used to measure the quality of sample sets. Uniform sampling being most known method can fail to achieve high coverage in presence of complex constraints on configurations. Another challenge is a scalability hurdle for the t-wise coverage computation leaving sampling for higher values of t unexplored.

In this work, we present Baital, a platform that combines two novel techniques for sampling of configurable systems. It is based on the adaptive weighted sampling approach to generate sample sets with high t-wise coverage. The approximation techniques for the t-wise coverage computation allow the consideration of higher values of t; they improve scalability for both t-wise coverage computation and sampling process.

CCS CONCEPTS

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

KEYWORDS

configurable systems, adaptive weighted sampling, t-wise coverage

ACM Reference Format:
Eduard Baranov and Axel Legay. 2022. Baital: An Adaptive Weighted Sampling Platform for Configurable Systems. In *26th ACM International Systems and Software Product Line Conference - Volume B (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3503229.3547030

1 INTRODUCTION

In 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. Users can adapt the system to their needs by selecting features. Yet configurability does not come without a price: feature dependencies are common [15] 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 extreme number of configurations [5]. Therefore, testing is usually performed on a small sample set of configurations.

Combinatorial interaction testing (CIT) is one of the major approaches for testing of configurable systems which attempts to include all t-sized combinations of features in a sample set. Bugs caused by interactions between at most t features could be found with this approach. However, with higher values of t its application might be slow and might result in large sample sets. An alternative approach is based on sampling - a sample set is randomly selected from valid configurations. t-wise coverage is a metric measuring the quality of a sample set 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.

Uniform sampling has its limitations due to unequal distribution of features among configurations. Depending on constraints of a configurable system, multiple features can be left uncovered by any sample set due to their appearance in few configurations resulting in low t-wise coverage. For example, in the work [13] it has been shown that uniform sampling can reach only 48% 2-wise coverage with 1000 samples on a feature model from SPLC 2019 sampling challenge [14]. The reason for such low coverage is the distribution of features: in the system having almost 10^{14} valid configurations, 18% of features appear in less than 10^3 configurations and are unlikely to be sampled. In [4], we addressed the problem by proposing an adaptive weighted 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. Its evaluation on a set of public benchmarks showed a 20% increase in the number of 2-sized feature combinations appeared in sample sets on almost all benchmarks.

Another challenge in sampling is the computation of t-wise coverage. Few works perform evaluation for 3-wise coverage while the computation of coverage for $t \geq 4$ is infeasible on benchmarks with thousands of features. Yet 3-sized combinations might not be enough: authors of [9] suggest to consider up to 6-sized combinations and the work [1] describes a bug involving interaction of 5 features. This challenge has been addressed in [3] by proposing algorithms for the approximation of t-wise coverage. The evaluation showed that 6-wise coverage can be computed for systems with thousands of features within 1 hour.

In this paper, we present Baital, a platform that combines the results of the previous works and goes beyond. The tool performs sampling for configurable systems providing multiple options for the adaptation of the sampling distribution that have different tradeoffs between achievable t-wise coverage, scalability, and execution

SPLC '22, September 12–16, 2022, Graz, Austria

time. The approximation algorithms are not only used in the computation of t-wise coverage, but can also be involved in the distribution adaptation. Baital is publicly available at Github[1]. Readme file contains installation instructions, examples of use and links to benchmarks.

2 BAITAL ALGORITHMS AND HEURISTICS

In this section we describe the Baital tool. We start with concepts and algorithms behind the tool and continue with the description of its workflow. We assume that a configurable system is defined with a propositional Boolean formula describing the constraints on features. A literal is a Boolean variable or its negation. A sample is a satisfying assignment of values to variables such that the constraints formula evaluates to true. A weight function assigns non-negative weights to all literals such that for any l: $W(l) + W(\neg l) = 1$. We use $\mathrm{Cov_t}(\mathcal{S})$ to denote the number of distinct feature combinations of size t in the sample set $\mathcal{S}$ and $\mathrm{Cov_t}(F)$ to denote the number of distinct feature combinations in all satisfying assignments of the formula F. Formally, $\mathrm{Cov_t}(\mathcal{S}) = |\bigcup_{\sigma\in\mathcal{S}}\{T \subseteq \sigma \mid |T| = t\}|$, where T is a subset of literals of size t and $|\cdot|$ is a set cardinality. $\mathrm{Cov_t}(F) = \mathrm{Cov_t}(Sol(F))$, where $Sol(F)$ is a set of all satisfying assignments of F. $\mathrm{Cov_t}(\mathcal{S}, l)$ and $\mathrm{Cov_t}(F, l)$ denote the number of feature combinations involving the literal l, i.e. we consider T such that $l \in T$. t-wise coverage of a set $\mathcal{S}$ is defined as $\mathrm{Cov_t}(\mathcal{S})/\mathrm{Cov_t}(F)$.

2.1 t-wise coverage computation

t-wise coverage of a sample set $\mathcal{S}$ is defined as a ratio between $\mathrm{Cov_t}(\mathcal{S})$ and $\mathrm{Cov_t}(F)$. The first number can be computed with the iteration through samples in $\mathcal{S}$ listing the appearing combinations and filtering out duplicates. For the $\mathrm{Cov_t}(F)$, it is infeasible to enumerate all satisfying assignments of real-world feature models, thus requiring a different approach. It can be computed by checking for each feature combination the presence of a satisfying assignment involving the combination. Even with code optimisations such as data structures for fast duplicate search and heuristics to reduce the number of SAT calls, the computation of the t-wise coverage is infeasible for large configurable systems for $t \geq 4$. Note that works evaluating sampling algorithms, e.g. [2, 7, 8, 10–12], use at most 3-wise coverage.

To allow the evaluation of t-wise coverage beyond $t = 3$, we provide approximate algorithms that can estimate 6-wise coverage in less than 1 hour on systems with 1000s features [3]. In particular, Baital implements two algorithms ApproxCov and ApproxMaxCov estimating $\mathrm{Cov_t}(\mathcal{S})$ and $\mathrm{Cov_t}(F)$ respectively. ApproxCov is Monte-Carlo based; it randomly selects a set of feature combinations and checks how many of them appear in the set $\mathcal{S}$. ApproxMaxCov encodes the task into a projected model-counting problem and uses an approximate model counter for the estimation. Estimations of both algorithms are probably approximately correct (PAC): given a tolerance parameter ε and a confidence parameter δ the approximation is guaranteed to be within $(1 \pm \varepsilon)$-factor of the ground truth with probability at least $1 - \delta$. A detailed description of the algorithms and their evaluation can be found in [3].

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

2.2 Adaptive Weighted Sampling

Baital is a scalable sampler that is capable to achieve high t-wise coverage [4]. Contrary to the uniform sampling where a fixed distribution is used, Baital adapts the distribution taking into account previously generated samples. The sampling process is performed in rounds: at the first round a uniform weight function is used, and then it is updated after each round.

Sampling at each round is performed with a weighted sampler WAPS that uses a knowledge compilation approach [6]. WAPS is executed in three steps: compilation, annotation, and actual sampling. Compilation transforms the given constraints into the equivalent deterministic decomposable negation normal form (d-DNNF) with a tool d4[2]. Note that compilation is executed only during the first round and the same d-DNNF tree is used in the remaining rounds. Annotation assigns weights to every node in the d-DNNF tree starting from the leaves. Finally, sampling is a top-down sub-routine where paths are selected randomly using the distribution imposed by the node weights.

After each round Baital explores generated samples and selects weights for the next round. We provide several *strategies* for the selection based on different metrics. All strategies compute their metrics for each literal and compare values for a literal and its negation assigning higher weight to the one with lower value.

- **Strategy 1** takes into account the ratio between $\mathrm{Cov_t}(\mathcal{S}, l)$ and $\mathrm{Cov_t}(F, l)$. The latter value can be approximated with ApproxMaxCov[3].
- **Strategy 2** uses the value of $\mathrm{Cov_t}(\mathcal{S}, l)$. This strategy is much faster than the previous one since the computation of $\mathrm{Cov_t}(F, l)$ requires a lot of resources. Note that during the computation of the t-wise coverage with a non-approximate algorithm, $\mathrm{Cov_t}(F, l)$ for every l is computed as a side product. In case of expected computation of t-wise coverage, the total execution time for strategies 1 and 2 is comparable.
- **Strategies 3 and 4** are similar to Strategies 1 and 2 except that $\mathrm{Cov_t}(\mathcal{S}, l)$ is computed approximately with ApproxCov.
- **Strategy 5** uses a simple heuristic that a literal appearing in many samples has high coverage. This strategy simply counts the number of samples involving a literal.

Detailed description of strategies 1, 2, and 5 can be found in [4].

2.3 Baital Workflow

Baital combines sampling and t-wise coverage computation. A high-level view of Baital workflow is shown in Figure 1. Its first step is preprocessing, where $\mathrm{Cov_t}(F, l)$ is computed for each literal for strategies 1 and 3. Both exact and approximate computations can be used. Execution time depends on the number of features and t value; exact computation scales poorly with t, therefore approximate version is recommended for $t \geq 3$. For other strategies this step is skipped.

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

[2]https://github.com/crillab/d4

[3]Strategies 1 and 2 correspond to strategies 1 and 2 in [4], strategy 5 to strategy 4 in [4], while strategies 3 and 4 are novel.

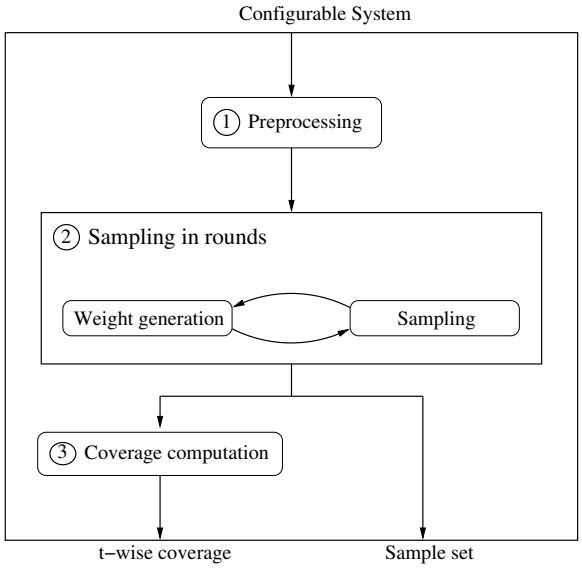

Figure 1: General workflow of Baital**.**

imposed by the weights. Its performance mainly depends on two factors: the size of the created d-DNNF tree and the selected strategy. The number of sampling rounds directly affects both factors. 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. Preprocessing results with exact computation can be reused at this step.

3 TOOL USAGE

Baital is implemented in Python 3. Its input is a file in DIMACS format containing constraints of a configurable system. A set of arguments helps to control the execution. t value can be set with *--twise* option, and the number of generated samples is controlled with *--samples* option. As an alternative, sampling process might be stopped by reaching a specified coverage given with *--desired-coverage* option. The number of rounds that is set with *--rounds* affects the sampling time: each round requires weights generation and re-annotation of the d-DNNF tree. Yet shorter rounds can improve the resulted coverage (e.g. Figures 7-9 in [4]).

The next major decision a user can take is the sampling strategy. In terms of execution time, strategy 5 which is set as default is the fastest. It does not require any preprocessing and weight selection takes milliseconds. Strategies 2 and 4 perform similar computations to strategies 1 and 3, respectively, except they skip the preprocessing step. On small systems with hundreds features and $t = 2$ strategies 1 and 2 are more efficient than strategies 3 and 4, while on larger systems or with higher values of t they are outperformed by the latter ones. Considering the resulted coverage, in our experience strategy 1 yields, in general, the best coverage followed by strategies 3, 5, 2, and 4. According to results in [4], even strategy 2 achieves significantly higher coverage than uniform sampling on 116 feature models tested out of 124 and none of the results had lower coverage than uniform sampling.

Preprocessing and coverage computation time can be controlled by switching between exact and approximate algorithms. The exact computation of $\mathrm{Cov}_t(F)$ and $\mathrm{Cov}_t(F, l)$ can require several hours even for $t = 2$, while for $t \geq 3$ they can be computed only on small configurable systems with tens of features. There are two options to avoid this computation. The first option is to use strategies that do not require $\mathrm{Cov}_t(F, l)$, i.e. strategies 2, 4, and 5, and to skip the computation of $\mathrm{Cov}_t(F)$ in the last step with *--no-maxcov*, however only $\mathrm{Cov}_t(\mathcal{S})$ would be outputted rather than t-wise coverage. The second option is to replace the exact computation with the approximate one. Arguments *--{preprocess | cov}-{approximate | epsilon | delta}* sets the approximate preprocessing and coverage computation and parameters of PAC guarantees.

In addition, Baital provides options to compute $\mathrm{Cov}_t(F, l)$ for a given system or t-wise coverage of a sample set without performing sampling. The former runs the preprocessing step and stores the result in a file that can be used later. The latter provides an opportunity to benefit from our approximate algorithms for sampling sets generated with other tools[4].

3.1 Example

We illustrate the tool application with a feature model "FinancialServices01" version "2018-05-09" ("financial" in the following) from [14] that after its transformation into CNF has 771 variables and 7241 clauses. Authors of [13] 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.

Since we are going to generate several sampling sets from the same feature model, it is convenient to precompute $\mathrm{Cov}_t(F, l)$ in order to reuse it in the consequent executions. The following two commands are used to get exact and approximate results:

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

The commands took 3697 seconds and 864 seconds, respectively. The tool created two files *results/financial_2.comb*[5] and *results/financial_2.acomb* with exact and approximate results, respectively.

It is possible to use Baital for uniform sampling by setting the number of rounds to one: the distribution for the first round is set to uniform. The following code snippet shows the command. Note that the option *−maxcov-file* uses the file computed previously for the $\mathrm{Cov}_t(F)$ computation at the final step.

```
python3 baital.py ./financial.cnf −−twise 2 −−samples 1000
    −−rounds 1 −−maxcov−file results/financial_2.comb
```

The output contains information on execution time for each of the steps in seconds, the number of combinations in the generated samples and the resulted coverage which is 47.4% corresponding to the result in [13]. Without *−maxcov-file* option coverage and total times would be approximately 3700 seconds slower.

[4] readme file describes the expected format of a sample set

[5] In our implementation the exact computation of $\mathrm{Cov}_t(F, l)$ also computes $\mathrm{Cov}_t(F, l)$ for smaller values of t that are stored in corresponding files, e.g. *results/financial_1.comb* for 1-sized combinations.

strategy	$\text{Cov}_2(F, l)$ type	coverage (%)	time (seconds)
uniform	-	47.4	14
1	exact	88.0	28
1	approximate	86.4	27
2	-	64.3	27
3	exact	88.3	48
3	approximate	87.9	46
4	-	66.6	49
5	-	70.5	17

Table 1: 2-wise coverage and execution time for sampling 1000 samples from the financial feature model. Time includes coverage computation but assumes that $\text{Cov}_2(F)$ and $\text{Cov}_2(F, l)$ are precomputed.

t	exact time (seconds)	approximate time (seconds)
2	3703	111
3	-	245
4	-	431
5	-	700
6	-	1645

Table 2: t-wise coverage computation time (in seconds) on a sample set for the financial feature model.

```
Results
Sampling time 2.4583940505981445
Coverage time 11.62496829032898
Total time 14.083527326583862
(Approximate) number of combinations 435183
(Approximate) 2−wise coverage 0.47444490268767375
```

Table 1 shows results for sampling 1000 samples on the financial feature model with different strategies in 10 rounds. Option *--seed* allows users to set random seed for repeatability. For strategies 1 and 3 we used both versions of the precomputed $\text{Cov}_t(F, l)$. Note that the execution times in Table 1 assumes that $\text{Cov}_t(F, l)$ is precomputed and passed with *--preprocess-file* option for strategies 1 and 3 and with *--maxcov-file* for the coverage computation. Coverage computation takes about 12 seconds in all rows of the table. The latter can be replaced with approximate computation, for example the last row with strategy 5 would yield coverage 70.4 with the same random seed and would require 13 seconds in total. Note that all strategies achieved a much higher coverage than uniform sampling.

In order to show the performance of the approximate t-wise coverage computation, we fixed a sample set generated with strategy 5 and run the computation of t-wise coverage for various values of t. Execution time is shown in Table 2. Exact computation took more than 1 hour for $t = 2$ and timed out of 4 hours for $t \geq 3$. Approximate method is able to get the estimation for $t = 6$ within 30 minutes. The command to compute coverage of a given sample set is shown below.

```
python3 baital.py ./financial.cnf −−twise 2 −−no−sampling
    −−samples−file results/financial.samples
```

4 CONCLUSION

In this paper, we present Baital, an adaptive weighted sampler for configurable systems that can achieve high t-wise coverage. 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.

In the future work we plan to incorporate other adaptive sampling algorithms, such as LS-Sampling [11]. In addition we plan to explore the possibility to extend our tool for a direct support of features that has a finite number of possible values.

ACKNOWLEDGMENTS

This work was supported by EU H2020 project Serums (826278-SERUMS-H2020-SC1-FA-DTS-2018-2020) and by Région Wallonne project Deep Construct (C-8560).

REFERENCES

[1] Iago Abal, Claus Brabrand, and Andrzej Wasowski. 2014. 42 variability bugs in the linux kernel: a qualitative analysis. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*. 421–432.

[2] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: efficient product-line testing using incremental pairwise sampling. *ACM SIGPLAN Notices* 52, 3 (2016), 144–155.

[3] Eduard Baranov, Sourav Chakraborty, Axel Legay, Kuldeep S Meel, and Vinodchandran N. Variyam. 2022. A Scalable t-wise Coverage Estimator. In *Proc. 44th International Conference on Software Engineering (ICSE 2022)*.

[4] Eduard Baranov, Axel Legay, and Kuldeep S. Meel. 2020. 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, Virtual Event, USA, November 8-13, 2020*. 1114–1126. https://doi.org/10.1145/3368089.3409744

[5] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wasowski. 2013. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*. 1–8.

[6] Rahul Gupta, Shubham Sharma, Subhajit Roy, and Kuldeep S. Meel. 2019. WAPS: Weighted and Projected Sampling. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*.

[7] Yue Jia, Myra B. Cohen, Mark Harman, and Justyna Petke. 2015. Learning Combinatorial Interaction Test Generation Strategies Using Hyperheuristic Search. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*. IEEE Computer Society, 540–550.

[8] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2012. An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference*. 46–55.

[9] D Richard Kuhn, Raghu N Kacker, and Yu Lei. 2010. Practical combinatorial testing. *NIST special Publication* 800, 142 (2010), 142.

[10] Jinkun Lin, Chuan Luo, Shaowei Cai, Kaile Su, Dan Hao, and Lu Zhang. 2015. TCA: An efficient two-mode meta-heuristic algorithm for combinatorial test generation (t). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 494–505.

[11] Chuan Luo, Binqi Sun, Bo Qiao, Junjie Chen, Hongyu Zhang, Jinkun Lin, Qingwei Lin, and Dongmei Zhang. 2021. LS-sampling: an effective local search based sampling approach for achieving high t-wise coverage. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1081–1092.

[12] Dusica Marijan, Arnaud Gotlieb, Sagar Sen, and Aymeric Hervieu. 2013. Practical pairwise testing for software product lines. In *Proceedings of the 17th international software product line conference*. 227–235.

[13] 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, 2019*. 15:1–15:4.

[14] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product sampling for product lines: the scalability challenge. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. 78–83.

[15] Iran Rodrigues, Márcio Ribeiro, Flávio Medeiros, Paulo Borba, Baldoino Fonseca, and Rohit Gheyi. 2016. Assessing fine-grained feature dependencies. *Inf. Softw. Technol.* 78 (2016), 27–52. https://doi.org/10.1016/j.infsof.2016.05.006