t-wise Coverage by Uniform Sampling

Jeho Oh
jeho@cs.utexas.edu
University of Texas at Austin
Austin, Texas, USA

Paul Gazzillo
paul.gazzillo@ucf.edu
University of Central Florida
Orlando, Florida, USA

Don Batory
batory@cs.utexas.edu
University of Texas at Austin
Austin, Texas, USA

ABSTRACT

Efficiently testing large configuration spaces of *Software Product Lines* (SPLs) needs a sampling algorithm that is both scalable and provides good t-wise coverage. The 2019 SPLC Sampling Challenge provides large real-world feature models and asks for a t-wise sampling algorithm that can work for those models.

We evaluate t-wise coverage with one of the provided feature models using the Smarch algorithm that uniformly samples SPL configurations. While uniform sampling alone is not enough to produce 100% 1-wise and 2-wise coverage, we use standard probabilistic analysis to explain our experimental results and to conjecture how uniform sampling may enhance the scalability of existing t-wise sampling algorithms.

CCS CONCEPTS

• **Theory of computation → Logic and verification**; **Automated reasoning**; • **Software and its engineering** → *Abstraction, modeling and modularity*; *Model-driven software engineering*; Software performance.

KEYWORDS

software product lines, t-wise coverage, uniform sampling.

ACM Reference Format:
Jeho Oh, Paul Gazzillo, and Don Batory. 2019. t-wise Coverage by Uniform Sampling. In *Proceedings of 23nd International Conference on Software Product Lines (SPLC'19)*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Software Product Lines (SPLs) are highly configurable. Building blocks of SPL products are *features* that are increments of product functionality. Each product of an SPL is defined by a unique set of features called a *configuration*. A *feature model* declares each feature and constraints among features, so that a user can identify legal configurations with desired feature combinations [4]. As the number of features increase, the size of the *configuration space*, which is the set of all possible configurations, grows exponentially.

A large configuration space could have over a trillion ($>10^{12}$) configurations and is a challenge for testing, as testing every configuration is infeasible. Instead, prior work produced a small set of configurations in order to test selected features and their interactions. The aim is to get a 'high' t-wise coverage, ideally meaning 100% of *all* combinations of t features are covered by at least one configuration of the set. Sometimes less than 100% is accepted. Common values for t include feature-wise (t=1), pair-wise (t=2), and three-wise coverage (t=3).

Different approaches start with a feature model and derive samples for t-wise coverage [1, 2, 6, 9, 10]. However, they do not scale well for many features and complex constraints, which limited their applicability to the real-world SPLs. Thus, the proposed Challenge [16] provides large real-world feature models and asks for a sampling algorithm that can generate configuration sets with good t-wise coverage for those models.

In this paper, we explore t-wise coverage using *uniform sampling* ($\mathbb{US}$). $\mathbb{US}$ ensures that all configurations in a configuration space have equal probability of being selected, yielding a *statistically representative sample* of the space. $\mathbb{US}$ can be used as a baseline against which other sampling algorithms can compare as a benchmark.

Despite its utility, $\mathbb{US}$ for large SPLs was considered infeasible for large SPLs until recently [11, 13]. Prior work tried different methods to make sampling as random as possible, but none achieved $\mathbb{US}$ for large SPLs. We use a recently developed algorithm called Smarch [8], the first to perform $\mathbb{US}$ of configuration spaces of size 10^{245}. Smarch utilizes a #SAT solver, which counts the number of solutions to a propositional formula. [15].We believe we are the first to explore t-wise coverage of $\mathbb{US}$ with probabilistic analyses to explain its coverage results.

Our contributions to the 2019 SPLC Sampling Challengeare:

- Demonstration of t-wise coverage that can be achieved by $\mathbb{US}$; and
- Probabilistic analysis of configuration spaces that predicts the t-wise coverage by $\mathbb{US}$ and that may be useful for developing a practical t-wise sampling algorithm.

2 SMARCH: A $\mathbb{US}$ ALGORITHM

Smarch [8] is a $\mathbb{US}$ algorithm for SPLs based on a #SAT solver.

Let ϕ be the propositional formula of a feature model [3]. A #SAT solver can count the number of configurations in ϕ's configuration space, namely $|\phi|$. (Each solution to ϕ is a configuration, and each configuration is a solution to ϕ). A #SAT solver extends a satisfiability solver by associating the number of solutions with each truth assignment [5]. Smarch uses sharpSAT [15], a state-of-the-art #SAT solver.

Here is how Smarch achieves $\mathbb{US}$: A uniform random number generator can select an integer r in the range $[1..|\phi|]$. Smarch creates

SPLC'19, 9–13 September, 2019, Paris, France

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

a one-to-one mapping that converts r into a unique configuration, so that $\mathbb{US}$ of range $[1..|\phi|]$ leads to $\mathbb{US}$ of configurations.

To create a one-to-one mapping, Smarch recursively partitions ϕ by a fixed order of variables. A variable $v{\in}\phi$ partitions ϕ into disjoint spaces $(\phi{\wedge}\neg v)$ and $(\phi{\wedge}v)$. #SAT can compute the number of solutions for each space, i.e., $|\phi{\wedge}\neg v|$ and $|\phi{\wedge}v|$ respectively.

Then, for a random number $r{\in}[1..|\phi|]$, if $r{\le}|\phi{\wedge}\neg v|$ the $(\phi{\wedge}\neg v)$ space is selected for recursive partitioning, otherwise $(\phi{\wedge}v)$ is selected and $|\phi{\wedge}\neg v|$ is subtracted from r to adjust the search in $(\phi{\wedge}v)$. This process is repeated for the next variable in ϕ, until all variables are considered and a unique configuration has been determined.

3 EVALUATION

3.1 Experimental Set-Up

Among the feature models provided in the Challenge, we used 'FinancialServices01' version '2018-05-09'. This feature model was given in FeatureIDE format [14], so we used the functionality of FeatureIDE to generate its propositional formula as a dimacs file. This file has 771 variables and 7,241 clauses. Using sharpSAT [15], the size of the configuration space was determined to be 97,451,212,554,676 $\approx 9.7 \times 10^{14}$, counted in a mere 46 milliseconds.

We evaluated t-wise coverage for t=1 and t=2 and did the following to find valid combinations:

(1) We derived a list of feature selections. With 771 features, there are 771×2 = 1,542 possible selections since we consider both a feature and its negation;
(2) We derived all possible 1-wise and 2-wise combinations from this list. 1-wise yields $\binom{1542}{1} = 1542$ combinations and 2-wise yields $\binom{1542}{2} = 1{,}188{,}111$; and
(3) We filtered out invalid combinations using a SAT solver. If a combination is valid, the conjunction of the combination and the feature model should be satisfiable. For example, for a feature f, a 2-wise combination $(f,\neg f)$ is invalid as these selections conflict with each other.

For 1-wise, 1,518 valid combinations were found (some features were only mandatory). For 2-wise, 914,537 valid combinations were found.

We used Smarch to produce a $\mathbb{US}$ set $\mathbb{S}_n$ of n configurations.[1] We varied n to observe the results of increasing larger sets, using n={5, 10, 20, 30, 40, 50, 100, 200, 300, 400, 500, 1000, 1518}. Then, for each set $\mathbb{S}_n$, we measured [2]:

- **Time taken to sample *n* configurations**, measured by the Linux '`time`' tool;
- **Time taken to sample a configuration**, measured for each sample by Smarch;
- **Maximum memory used during sampling**, measured by the Linux '`/usr/bin/time -v`' command; and
- ***t*-wise coverage for t=1 and t=2**, measured as the percentage of t-wise combinations covered in $\mathbb{S}_n$.

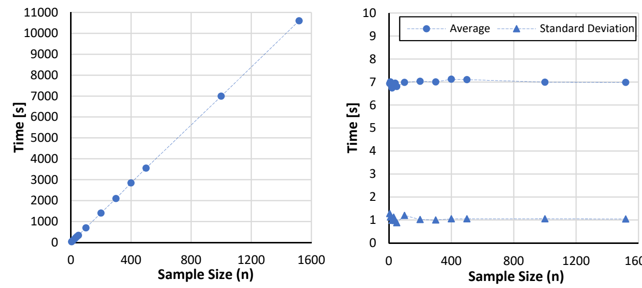
We conducted our evaluation on an Intel i7-6700@3.4Ghz Ubuntu 16.04 machine with 16GB of RAM. All the code and data for the evaluation are available at: *https://github.com/jeho-oh/Smarch_t_wise*.

[1] Smarch takes samples without replacement.
[2] The Challenge [16] explicitly requests sampling time and memory measurements.

3.2 Experimental Results

Fig. 1a shows the total sampling time and Fig. 1b the time per sample. The X-axis is the number of samples (n) and the Y-axis is the time in seconds. We observed:

- Total sampling time increases linearly with n; and
- For all n, the average sampling time for a configuration was approximately 7 seconds, with standard deviation of 1 second. For all samples, the maximum sampling time was 10.1 seconds and the minimum was 3.5 seconds.
- The number of samples taken did not affect the time to sample a configuration.



(a) Total sample time (b) Time per sample

Figure 1: Sampling time.

Fig. 2 shows the maximum memory usage of Smarch, where the X-axis is the number of samples (n) and the Y-axis is the memory size in megabytes. We observed:

- Maximum memory usage was stable, between 16.8MB and 17.1MB for all n; and
- Sampling more configurations did not increase the maximum memory usage.

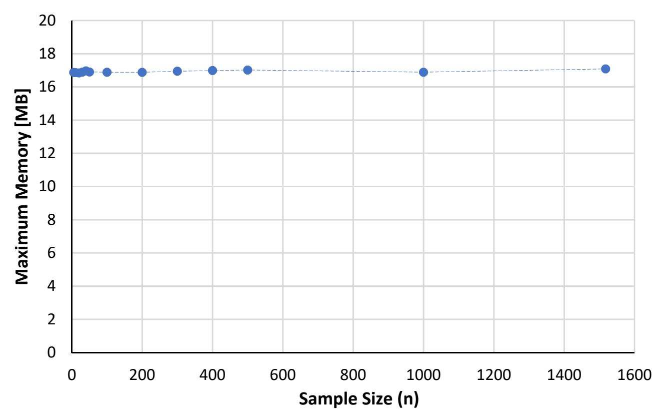


Figure 2: Maximum memory usage.

Fig. 3 shows the t-wise coverage result, where the X-axis is the number of samples (n) and Y-axis is the percentage of the coverage. Plots with different color indicates the results for different t. We observed:

- For all values of n, coverage for t=1 was higher than t=2;
- For $\mathbb{S}_5$, more than half of the feature combinations were covered for t=1 and over 35% for t=2;

- For both t=1 and t=2, increasing n yielded better coverage. With 1,518 samples, coverage for t=1 was 61.7% and t=2 was 47.6%;
- The difference in coverage between n=5 and n=1,518 was surprisingly small. For t=1, the difference was 6.4%. For t=2, the difference was 9.4%; and
- Although samples are expected to be statistically representative of the configuration space, their t-wise coverages seemed low. Both coverages improved imperceptibly after n exceeded 200.

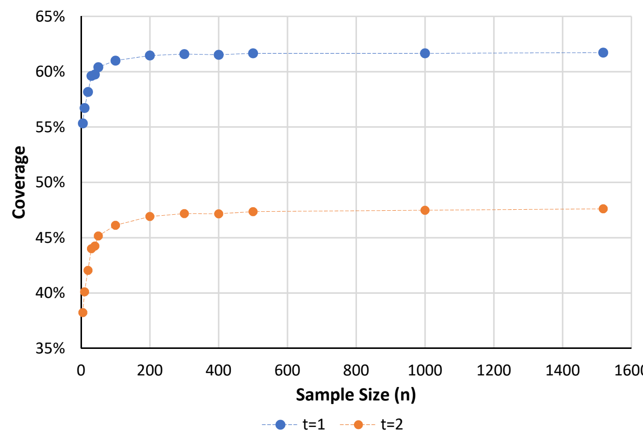


Figure 3: t-wise coverage.

We conclude that although $\mathbb{US}$ is feasible with Smarch, $\mathbb{US}$ alone is not enough to produce a 100% t-wise coverage.

4 ANALYSIS

$\mathbb{US}$ allows us to apply standard statistical analysis to explain our experimental results [7].

Let c denote a valid t-wise combination for a given t. Let ν_c denote the fraction of all valid configurations that have c in the configuration space. Since every configuration has an equal probability of being selected by $\mathbb{US}$, the probability that a sample will have c is ν_c.

ν_c can vary widely for different c because constraints among features may make certain combinations less frequent than others. A mandatory feature has ν_c=1 because it appears in all configurations. A feature with no constraints has ν_c=0.5; it can be freely enabled and disabled, making it appear in half of the valid configurations.

ν_c can be computed by a #SAT solver. Let ϕ be the propositional formula of an SPL's feature model. Let ϕ_c be the propositional formula of the conjunction of c's features. We can use a #SAT solver to compute ν_c as:

$$\nu_c = \frac{|\phi \wedge \phi_c|}{|\phi|} \tag{1}$$

The probability $p(c, n)$ that at least one of n samples includes combination c is:

$$p(c, n) = 1 - (1 - \nu_c)^n \tag{2}$$

where the more samples taken, the higher the probability we will encounter combination c. The final probability, however, largely depends on how often this combination appears in the configuration space, i.e., ν_c.

In our experiments of the previous section, we discovered:

- 61.5% of all 1-wise combinations have a ratio ν_c>0.9. Even with the minimum number of samples we used in the evaluation (n=5), these combinations have more than 0.99 probability of being encountered in n=5 samples; and
- 38.8% of the 1-wise combinations have a ratio of ν_c<0.0001. Even with the maximum number of samples we used in the evaluation (n=1815), they have less than 0.15 probability of being encountered in n=1815 samples.

We can use $p(c, n)$ to predict t-wise coverage. Let $\mathbb{C}_t$ denote the set of all valid t-wise combinations, where $|\mathbb{C}_t|$ is the number of combinations in $\mathbb{C}_t$. The estimated t-wise coverage $E(\mathbb{C}_t, n)$ for a given t, n is:

$$E(\mathbb{C}_t, n) = \frac{1}{|\mathbb{C}_t|} \sum_{c \in \mathbb{C}_t} p(c, n) = \frac{1}{|\mathbb{C}_t|} \sum_{c \in \mathbb{C}_t} (1 - (1 - \nu_c)^n) \tag{3}$$

Fig. 4 uses **(blue)** X markers to plot $E(\mathbb{C}_1, n)$ and **(brown)** X markers for $E(\mathbb{C}_2, n)$ with our experimental results (• for t=1 and • for t=2) overlaid. Eqn. (3) accurately predicts the results of our experiments and also explains why the coverage of t=1 is higher than that for t=2: there are many 2-way feature combinations (c_{ij}) that are much less likely than any 1-way combination (c_k), meaning $\nu_{c_k} \gg \nu_{c_{ij}}$.

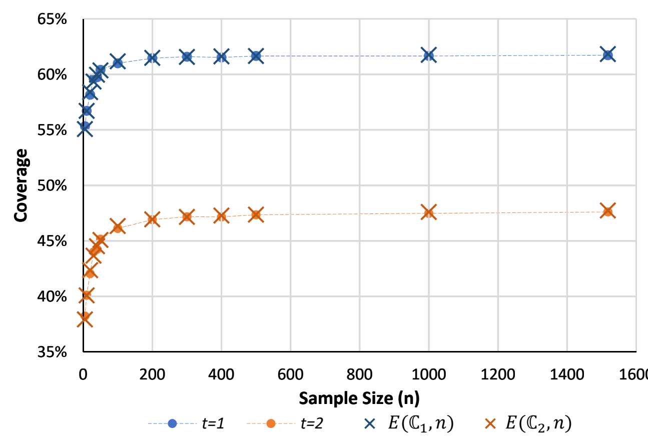


Figure 4: t-wise coverage estimation.

It is interesting to explore the relationship between coverage and larger sample set sizes *which are infeasible to explore experimentally.* Fig. 5 shows the estimated t-wise coverage for n up to 10^{14}, which is approximately 10% of the configuration space (*ie.,* 9.7×10^{14}). We observed:

- With 10^{14} samples, more than 99.99% of 1-wise and 2-wise combinations are expected to be covered. Of course, this is almost enumeration; and
- Many combinations will be covered with a small number of samples, over 30% of 2-way combinations are not likely to be covered even with 10^7 samples(!).

We could accurately predict these results because Smarch can uniform sample from a configuration space and standard probabilistic analyses rely on $\mathbb{US}$ [7].

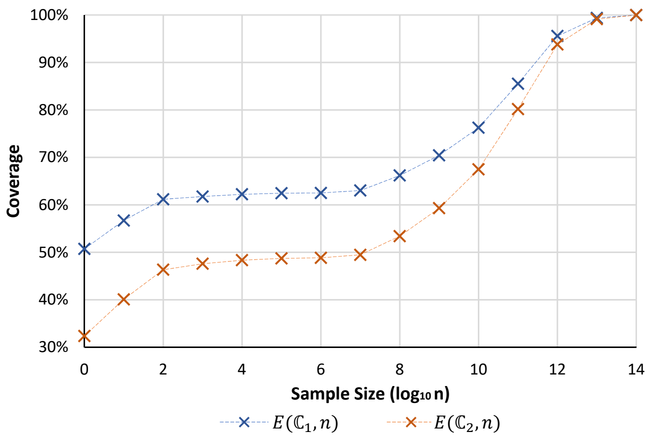


Figure 5: *t*-wise coverage estimation for large *n*.

Our analysis suggests possible enhancements to existing t-wise approaches. Once v_c values are known, we can determine which combinations can be covered by a small number of $\mathbb{US}$s. Then, for combinations that are unlikely to be found by $\mathbb{US}$, we may either: 1) constrict the configuration space with constraints to (recursively) sample configuration sub-space of interest [12] or 2) use existing approaches that do not rely on $\mathbb{US}$. As sampling with many features limits the scalability of existing approaches, $\mathbb{US}$ may improve sampling scalability by reducing the features to consider. And equally important issue is to define a reasonable t-wise coverage (percentage) for large configuration spaces (other than 100%) for practitioners to use.

5 CONCLUSIONS AND FUTURE WORK

As $\mathbb{US}$ of configurations was considered infeasible, probabilistic analyses of a configuration space based on $\mathbb{US}$ was unexplored or considered unexplorable. We used a recently developed algorithm, Smarch [8], to $\mathbb{US}$ configurations of a configuration space. We also derived probabilistic models to explain Smarch results. We showed:

- $\mathbb{US}$ alone is ***not*** enough to produce 100% t-wise coverage; and
- Distribution of v_c can be used to predict the t-wise coverage of $\mathbb{US}$.

Our work opens new possibilities on analyzing an SPL configuration space and deriving samples for testing. As $\mathbb{US}$ produces statistically representative samples of a configuration space, it may be possible to utilize the information from samples to improve the efficiency of existing approaches. As a future work, we plan to:

- Analyze other systems to validate and expand our insights on probabilistic analyses;
- Derive an algorithm that can utilize $\mathbb{US}$ for t-wise coverage; and
- Enhance the performance of the Smarch algorithm.

ACKNOWLEDGMENTS

Work by Gazzillo is supported by NSF CCF-1840934. Work by Oh and Batory is supported by NSF grant CCF-1421211.

REFERENCES

[1] Mustafa Al-Hajjaji, Sebastian Krieter, Thomas Thüm, Malte Lochau, and Gunter Saake. 2016. IncLing: efficient product-line testing using incremental pairwise sampling. In *ACM SIGPLAN Notices*. ACM, ACM, New York, NY, USA, 144–155.

[2] Mustafa Al-Hajjaji, Thomas Thüm, Malte Lochau, Jens Meinicke, and Gunter Saake. 2019. Effective product-line testing using similarity-based product prioritization. *Software & Systems Modeling* 18, 1 (2019), 499–521.

[3] Sven Apel, Don Batory, Christian Kaestner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer, Berlin, Heidelberg.

[4] Don Batory. 2005. Feature models, grammars, and propositional formulas. In *International Conference on Software Product Lines*. Springer, Springer, NY, USA, 7–20.

[5] Armin Biere, Marijn Heule, and Hans van Maaren. 2009. *Handbook of satisfiability*. Vol. 185. IOS press, IEEE.

[6] Brady J Garvin, Myra B Cohen, and Matthew B Dwyer. 2011. Evaluating improvements to a meta-heuristic search for constrained interaction testing. *Empirical Software Engineering* 16, 1 (2011), 61–102.

[7] C.M. Grinstead and J.L. Snell. 2019. Intro to Probability - Dartmouth College. https://www.dartmouth.edu/~chance/teaching_aids/books_articles/probability_book/amsbook.mac.pdf.

[8] Jeho Oh, Paul Gazzillo, Don Batory, Marijn Heule, and Maggie Myers. 2019. *Uniform Sampling from Kconfig Feature Models*. Technical Report TR-19-02. University of Texas at Austin, Department of Computer Science.

[9] Martin Fagereng Johansen, Øystein Haugen, and Franck Fleurey. 2011. Properties of realistic feature models make combinatorial testing of product lines feasible. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, Springer, Berlin, Heidelberg, 638–652.

[10] 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 *International Conference on Model Driven Engineering Languages and Systems*. Springer, Springer, Berlin, Heidelberg, 269–284.

[11] Christian Kaltenecker, Alexander Grebhahn, Norbert Siegmund, Jianmei Guo, and Sven Apel. 2019. Distance-Based Sampling of Software Configuration Spaces. In *Proceedings of the 2019 International Conference on Software Engineering*. ICSE, IEEE/ACM, Piscataway, NJ, USA, 0.

[12] Jeho Oh, Don 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*. ACM, IEEE/ACM, Piscataway, NJ, USA, 61–71.

[13] 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 *12th International Conference on Software Testing, Verification, and Validation ICST 2019*. IEEE, Piscataway, NJ, USA.

[14] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70–85.

[15] Marc Thurley. 2006. sharpSAT–counting models with advanced component caching and implicit BCP. In *International Conference on Theory and Applications of Satisfiability Testing*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 424–429.

[16] Tobias Pett, Thomas Thüm, Tobias Runge, Sebastian Krieter, Malte Lochau, and Ina Schaefer. 2019. Product Sampling for Product Lines: The Scalability Challenge. In *Software Product Line Conference, Challenge Case*. ACM, ACM, New York, NY, USA.