Seminar Paper on Uniform Random Sampling

1st Haufe Max
Department of Mathmatics and Computer Science
Leipzig University
Leipzig, Germany

***Abstract*—Modern software systems' extensive configurability enables users to tailor systems to specific needs, but it also complicates identifying optimal configurations for performance or energy efficiency. The vastness of configuration spaces and complex interactions among options pose significant challenges. This paper examines uniform random sampling strategies, focusing on distance-based and diversity promotion methods, to create representative sample sets that facilitate accurate performance modeling without exhaustive measurement. Implementation and evaluation results highlight the effectiveness of these techniques in achieving representative samples.**

***Index Terms*—Uniform Random Sampling, Configurable Software Systems, Performance Modeling, Distance-based Sampling, Diversity Promotion.**

I. Introduction

Modern software systems offer extensive configurability, enabling end users, developers, and administrators to tailor the system to meet specific functional and performance requirements. However, this flexibility also introduces complexity in identifying the optimal configuration for performance or energy efficiency. The vastness of configuration spaces and intricate interactions among configuration options make this a challenging task [1].

Traditionally, one could resort to a brute-force approach, measuring the performance of every possible configuration to find the best one. However, this method is often impractical due to the sheer number of configurations, making it infeasible for real-world applications. To address this, machine learning techniques have been employed to develop performance models based on a subset of configurations. Techniques such as multiple linear regression and classification and regression trees can predict the performance of configurations, aiding in the identification of the performance-optimal setup without exhaustive measurements [2].

Creating an accurate performance model hinges on the selection of a representative sample set from the configuration space. This selection process is particularly challenging without domain knowledge, as it requires a sample set that reflects the diversity and interactions of the entire configuration space. Effective sampling is crucial since performance measurements are typically costly and time-consuming .

Sampling strategies vary, each with its strengths and weaknesses. A common objective is to achieve a uniform coverage of the configuration space to ensure the sample set is unbiased and representative. However, achieving such unbiased uniformity is difficult, especially when complex constraints exist among configuration options.

To overcome these challenges, distance-based sampling [3] and diversity promotion [4] techniques have been explored and benchmarked for their effectiveness in producing representative sample sets. Distance-based sampling aims to uniformly cover the configuration space or follow a specified probability distribution. This approach utilizes a distance metric and a discrete probability distribution to select configurations based on their distance values, ensuring a spread across the configuration space that reflects the overall performance distribution. Notably, this method does not require an analysis of the entire population and employs a constraint solver to maintain efficiency and avoid clustered samples.

Complementing this, diversity promotion seeks to enhance the variety of solutions in a cost-effective manner. By randomly permuting parameters that control the search for constraint-satisfying solutions processed by a SAT solver, this technique maximizes the diversity of configurations. The parameters include the order of constraints, the order of literals within each constraint, and the phase selection for variable assignments. These permutations aim to ensure a broad exploration of the configuration space, increasing the likelihood of finding diverse and representative samples.

This paper delves into the implementation and evaluation of these sampling strategies, assessing their ability to produce representative sample sets that can inform accurate performance models.

II. Preliminaries

A configurable software system has a set of $\mathcal{O}$ configuration options. In this work, we focus solely on binary configuration options, where each option can be either 0 (disabled) or 1 (enabled). The collection of all possible (valid) configurations is denoted as $\mathcal{C}$, which we refer to as the configuration space.

However, not all combinations of configuration options are valid due to constraints imposed by the system. These constraints can be effectively expressed using CNF (Conjunctive Normal Form) propositional formulas. A CNF formula is a conjunction of clauses, where each clause is a disjunction of literals, and a literal is either a variable or its negation.

Given the constraints, acquiring the entire set of valid configurations $\mathcal{C}$ is computationally expensive. This makes it impractical to analyze the complete configuration space directly. Instead, it is often more practical to select a smaller subset of valid configurations for analysis. A configuration option $o \in \mathcal{O}$ is termed mandatory if $\forall c \in \mathcal{C} : c(o) = 1$, meaning the option is selected in every configuration. [3]

A. Sampling Strategies

Sampling involves selecting a subset $\mathcal{S} \subseteq \mathcal{C}$ of all configurations $c \in \mathcal{C}$ from a configurable software system. Several strategies are used to achieve this:

Random Sampling [5]: This approach randomly assigns values to each configuration option. However, it often results in many invalid configurations due to unsatisfied constraints. Variants include using hash functions to divide the configuration space or binary decision diagrams for compact representation, but both methods have limitations in scalability and efficiency.

Solver-Based Sampling [6]: This strategy employs constraint solvers like SAT4J to generate configurations. While it avoids invalid configurations, it can lead to locally clustered samples and lacks true randomness. Enhancements like randomized solver-based sampling increase diversity by altering the order of configuration options, constraints, and values in each solver run, though at a high computational cost. Both sampling approaches dealt with in this work are solver-based.

Coverage-Oriented Sampling [7]: These strategies optimize the sample set based on specific coverage criteria. A notable example is t-wise sampling, which ensures all combinations of tt configuration options are covered. These strategies, however, require prior knowledge and may emphasize specific regions of the configuration space, potentially missing out on broader performance variations.

Each of these sampling strategies has its advantages and challenges, influencing the efficiency and representativeness of the resulting sample set. This paper further explores and benchmarks these methods to evaluate their effectiveness in producing accurate performance models for configurable software systems.

B. Distance-based Sampling

The following section is based on the methodology described in [3].

Distance-based sampling involves selecting configurations from the configuration space $\mathcal{C}$ based on a distance metric. This method utilizes a constraint solver to ensure the sampled configurations meet the constraints of the variability model (VM).

1) Distance Metric: We employ the Manhattan distance to quantify the distance d of a configuration c from the origin (where all configuration options are disabled). Formally, the distance is defined as:

$$\text{dist}(c) = \sum_{o \in \mathcal{O}} c(o)$$

where $c \in \mathcal{C}$ and $c(o)$ indicates whether option o is enabled (1) or disabled (0). The set of all possible distances is given by:

$$D = \{\text{dist}(c) \mid c \in \mathcal{C}\}$$

Due to the unknown nature of the configuration space, D is unknown, however its maximum and minimum can be estimated.

To estimate the range of possible distances, we calculate the maximum and minimum distances:

$$\max(D) = |\mathcal{O}|$$

$$\min(D) = \text{card}(\{o \in \mathcal{O} \mid o \text{ is mandatory}\})$$

where $\text{card}(\cdot)$ denotes the cardinality function. The maximum distance corresponds to all configuration options being enabled, while the minimum distance reflects the number of mandatory options that must be enabled according to the constraints.

2) Probability Distribution: For selecting distances, we utilize a uniform probability distribution. This approach ensures each distance $d \in D$ has an equal probability of being chosen, expressed as:

$$P(X = d) = \frac{1}{|D|}$$

This uniform distribution helps in achieving an unbiased sampling of distances across the configuration space.

3) Configuration Selection: To select a configuration with a specific distance d, an additional constraint is added to the constraint solver, enforcing that exactly d options are enabled. If no configuration with the specified distance is found, a new distance is selected, and the process is repeated until the desired number of configurations is obtained. This method ensures a diverse set of configurations is sampled.

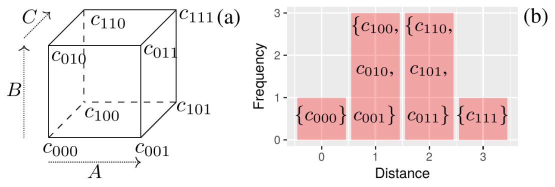

Fig. 1. Illustration of the distance function for a Software System with 3 configuration options and no constraints [3].

Algorithm 1 Distance-based Sampling [3]

```
1:  Input: VM, numSamples, probabilityDistr
2:  Output: sampleSet
3:  sampleSet ← $\emptyset$
4:  while otherSolutionsExist(VM, sampleSet) and size(sampleSet) $<$ numSamples do
5:      $d$ ← selectDistance(probabilityDistr)
6:      $c$ ← searchConfigWithDistance(VM, $d$)    ▷ Search for configuration $c$ with exactly $d$ configuration options selected
7:      if $c \neq \emptyset$ then
8:          sampleSet ← sampleSet $\cup \{c\}$
9:      end if
10: end while
11: return sampleSet
```

C. Sampling with Diversity Promotion

The following section is based on the methodology described in [4].

To promote diversity among SAT solutions efficiently, we employ random permutation of key parameters controlling the SAT solver's search process. Specifically, we permute the constraint order, the order of literals within each constraint, and the phase selection, which is the order ({true, false}) in which variable assignments are made. By randomly permuting these parameters at each iteration, we enhance the diversity of the solutions generated.

Algorithm 2 Diversity-Promoted Sampling [4]

```
1:  Input: VM, numSamples
2:  Output: sampleSet
3:  sampleSet ← $\emptyset$
4:  while otherSolutionsExist(VM, sampleSet) and size(sampleSet) < numSamples do
5:      permuteConstraints(VM)
6:      permuteLiterals(VM)
7:      permutePhases(VM)
8:      $c$ ← searchConfig(VM)
9:      if $c \neq \emptyset$ then
10:         sampleSet ← sampleSet $\cup \{c\}$
11:     end if
12: end while
13: return sampleSet
```

D. Representativeness

Representativeness is defined as the extent to which a sample set reflects the population [8], in this case, the entire configuration space. The core question is how well different sampling algorithms create representative sample sets. The goal is to achieve uniformity; however, validating this is challenging since statistical testing is not feasible without knowing and enumerating all configurations.

Solver-based sampling often encounters problems such as clustering and deterministic behavior, which can hinder the generation of representative samples [4]. This issue is particularly significant in the context of software performance modeling. Here, the goal is to cover the range of performance values, but the true distribution of these values is often unknown, and measuring them is expensive. While synthetic data could be generated using distributions like Poisson or normal for enumerated configurations, this approach does not accurately represent real data.

Given these challenges, we focus on coverage of the configuration space. Uniformity serves as a good proxy for representativeness, ensuring a more comprehensive exploration of the configuration space.

III. Results

The implementation is carried out using Python and Z3Py as the constraint solver.

A. Deliverables - Code

All code is available on GitHub at the following link: https://github.com/MaxHaufe/UniformRandomSampling. The source directory contains all the code, with the main area of interest being the 'samplers' directory. This directory includes the implementation of both the distance-based and diversity-promoted sampling strategies.

Additionally, it contains helper methods and classes such as `NaiveSampler`, which is not intended for real-world applications but rather for sampling all configurations of a given system to enable later comparisons. This approach is only feasible if the configuration space is small enough to enumerate all configurations.

The `distance_based.py` file contains the implementation of the distance-based sampling strategy, where the `sample(n)` method returns a sample set of size n. Similarly, the `diversity_promoted.py` file contains the implementation of the diversity-promoted sampling strategy, also utilizing the `sample(n)` method to return a sample set of size n.

Both classes require a variability model as input, which must be instantiated with a file path to a CNF file.

```
vm_db = VariabilityModel("path/to/cnf/file")
dbs = DistanceBasedSampler(vm_db)
sample_set = dbs.sample(1000)

vm_dp = VariabilityModel("path/to/cnf/file")
dps = DiversityPromotedSampler(vm_dp)
sample_set = dps.sample(1000)
```

It is important that different samplers do not share an instance of the `VariabilityModel` class since it contains the state of the current solver constraints. New constraints are added after each successful sampling iteration to avoid sampling the same configuration twice.

To implement the distance-based sampling strategy, we leverage Z3Py's pseudo-Boolean equality constraint, which is crucial for enforcing that exactly k configuration options are enabled, corresponding to a distance of k from the origin. This constraint is represented as a pseudo-Boolean equality constraint of the form:

$$\sum_{o \in O} c(o) = k$$

where $c(o)$ denotes whether the configuration option o is enabled (1) or disabled (0), and k is the desired distance from the origin in the configuration space. By adding this constraint to the constraint solver, we ensure that only those configurations where exactly k options are enabled are considered.

B. Deliverables - Evaluation regarding Representativeness

To evaluate the representativeness of sampling strategies, three benchmark systems were selected, as shown in Table I. These benchmarks are sourced from the GitHub repository of [10]. It is unknown which software or configuration problem these benchmarks represent, however this does not matter for the evaluation. Although more systems were initially considered, some had configuration spaces that were too small to be sufficiently complex, while others were too large to evaluate feasibly due to prolonged computation times for obtaining the ground truth.

For benchmarks with manageable configuration spaces, a large number of samples is generated, and the frequency of each possible configuration is counted. These counts are then compared to those obtained from a uniform random number generator. Achieving statistical significance with this method requires generating at least four times as many samples as there are configurations in the benchmark. However, this approach is often impractical for real-world systems because of the enormous number of possible configurations.

System	$\lvert\mathcal{O}\rvert$	**Constraints**	$\lvert\mathcal{C}\rvert$	**Source**
blasted_case_4	103	316	32768	[10]
blasted_case11	105	371	16384	[10]
blasted_case21	126	411	8192	[10]

TABLE I
BENCHMARK SYSTEMS USED FOR EVALUATION.

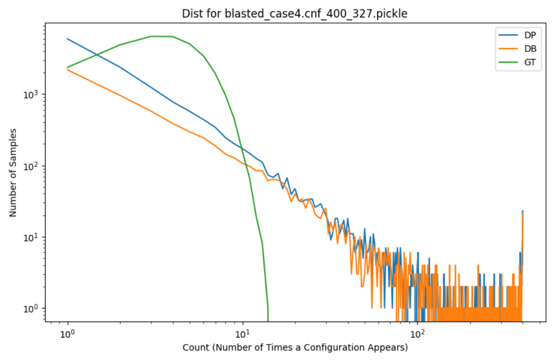

Fig. 2. Sampling results for blasted_case_4

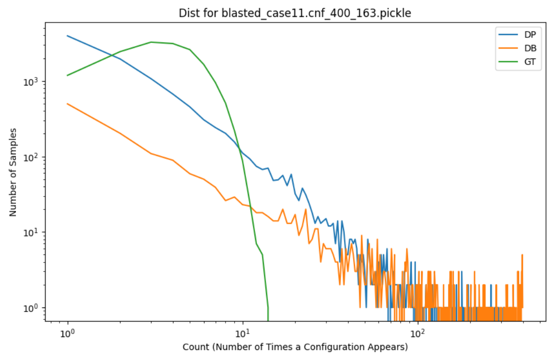

Fig. 3. Sampling results for blasted_case11

For each benchmark system listed in Table I, 1% of the configuration space was sampled 400 times. This ensured robustness, with four times as many samples generated as there are configurations. We maintain this degree of freedom fixed for consistency.

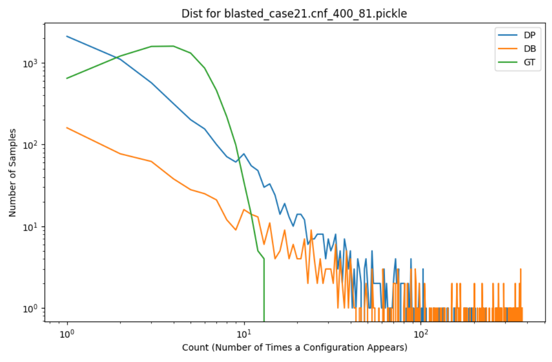

Fig. 4. Sampling results for blasted_case21

The results were evaluated in the same manner as in [9]. The sampling results were plotted in histograms to count the frequency of each configuration sampled. The x-axis represents the number of times the same configuration was sampled, and the y-axis represents the number of configurations this count applies to, using a log scale for both axes. We compare the Diversity Promoted (DP) and Distance Based (DB) samplers to the ground truth (GT) histogram, which represents a uniform distribution of all configurations.

The histograms are compared visually, and we also compute the L2 norm [11] between each sampler's results and the ground truth to assess the goodness of fit. Results are presented in Figures 2, 3, and 4. While the distributions for both the distance based sampler and diversity promoted sampler are largely deviant from the ground truth, one could argue that due to a large number of samples for large counts the distance based sampler is less representative than the diversity based sampler. The L2 norm, as shown in II is lower for the diversity based sampler for all cases, indicating a better fit to the ground truth.

An interesting observation which underlines the better representativeness of the diversity promoted sampler is the coverage of the configuration space. The diversity promoted sampler covers the configuration space more uniformly than the distance based sampler, as it generates more unique samples for all cases, as shown in Figure 5. It is also interesting to note that the coverage for the diversity promoted sampler decreases with the complexity of the benchmark system, whereas the coverage for the DB sampler increases. This, however could be coincidence, given the small number of benchmark systems considered.

IV. Discussion

In this work, we did not evaluate the performance of the samplers concerning performance measurements, as no dataset with complete configurations and performance measurements was available. Additionally, it is not feasible to measure the performance of all configurations due to the vast number of possibilities as stated earlier.

Potential future work could involve using synthetic data with a known distribution of performance values to evaluate

Name	GT Samples	Unique GT	Unique DP	Unique DB	$L2_{DP}$	$L2_{DB}$
blasted_case4	32768	32193	14049	6770	10487	11082
blasted_case11	16384	16047	10232	1785	5067	5887
blasted_case21	8192	8043	5281	747	2589	2975

TABLE II

COMPARISON OF DIFFERENT BLASTED CASES. GT: GROUND TRUTH, DP: DIVERSITY PROMOTION, DB: DISTANCE BASED, $L2_{DP}$: L2 NORM BETWEEN DP AND GT, $L2_{DB}$: L2 NORM BETWEEN DB AND GT.

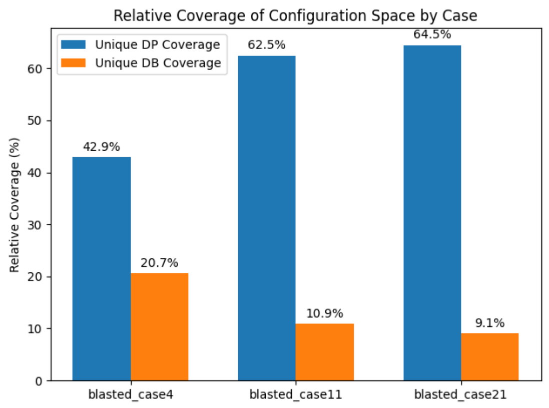

Fig. 5. Coverage Comparison Between diversity promoted and distance based Samplers

the samplers' coverage of performance values. However, this approach relies heavily on the distribution of the synthetic data, which may not accurately represent real-world data. Therefore, while informative, such evaluations might not provide a complete picture of the samplers' effectiveness in practical scenarios.

A notable observation is that distance-based sampling is heavily dependent on the uniformity of the number of configurations at certain distances from the origin. As illustrated in Figure 1, configurations with distances 0 and 3 are heavily favored because they have no competitors in their distance group. In contrast, distances 1 and 2 have three configurations each, resulting in a selection probability of 1/3 given the distance. This dependency highlights a limitation of the distance-based approach, whereas the Diversity-Promoted (DP) approach is more robust in this regard, as it does not rely on such uniformity.

To further improve the evaluation of the samplers, it would be beneficial to use more benchmark systems. This would provide a broader range of scenarios and help in understanding the strengths and limitations of each sampling strategy in various contexts. Expanding the set of benchmarks would also enhance the robustness of the conclusions drawn from the evaluations.

REFERENCES

[1] Xu, T., Jin, L., Fan, X., Zhou, Y., Pasupathy, S., & Talwadker, R. (2015, August). Hey, you have given me too many knobs!: Understanding and dealing with over-designed configuration in system software. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (pp. 307-319).

[2] Kolesnikov, S., Siegmund, N., Kästner, C., Grebhahn, A., & Apel, S. (2019). Tradeoffs in modeling performance of highly configurable software systems. Software & Systems Modeling, 18, 2265-2283.

[3] Kaltenecker, C., Grebhahn, A., Siegmund, N., Guo, J., & Apel, S. (2019, May). Distance-based sampling of software configuration spaces. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (pp. 1084-1094). IEEE.

[4] Henard, C., Papadakis, M., Harman, M., & Le Traon, Y. (2015, May). Combining multi-objective search and constraint solving for configuring large software product lines. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 517-528). IEEE.

[5] Gogate, V., & Dechter, R. (2006). A new algorithm for sampling csp solutions uniformly at random. In Principles and Practice of Constraint Programming-CP 2006: 12th International Conference, CP 2006, Nantes, France, September 25-29, 2006. Proceedings 12 (pp. 711-715). Springer Berlin Heidelberg.

[6] Henard, C., Papadakis, M., Harman, M., & Le Traon, Y. (2015, May). Combining multi-objective search and constraint solving for configuring large software product lines. In 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (Vol. 1, pp. 517-528). IEEE.

[7] Marijan, D., Gotlieb, A., Sen, S., & Hervieu, A. (2013, August). Practical pairwise testing for software product lines. In Proceedings of the 17th international software product line conference (pp. 227-235).

[8] Von der Lippe, P., & Kladroba, A. (2002). Repräsentativität von stichproben. Marketing, 24(1), 227-238.

[9] Dutra, R., Laeufer, K., Bachrach, J., & Sen, K. (2018, May). Efficient sampling of SAT solutions for testing. In Proceedings of the 40th international conference on software engineering (pp. 549-559).

[10] Plazar, Q., Acher, M., Perrouin, G., Devroey, X., & Cordy, M. (2019, April). Uniform sampling of sat solutions for configurable systems: Are we there yet?. In 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST) (pp. 240-251). IEEE.

[11] Deitmar, A. [2021]. Analysis(3., überarbeitete und erweiterte Auflage.). (pp. 183,). Berlin: Springer Spektrum.