A Comparative Study on Combinatorial and Random Testing for Highly Configurable Systems

Hao Jin, Takashi Kitamura, Eun-Hye Choi, Tatsuhiro Tsuchiya

A Comparative Study on Combinatorial and Random Testing for Highly Configurable Systems

Hao Jin[1][0000−0003−1392−0327], Takashi Kitamura[2][0000−0002−8903−3161], Eun-Hye Choi[2][0000−0001−9339−8107], and Tatsuhiro Tsuchiya[1][0000−0002−3329−9235]

[1] Osaka University, Suita, Japan {k-kou,t-tutiya}@ist.osaka-u.ac.jp
[2] AIST, Ikeda, Japan {t.kitamura,e.choi}@aist.go.jp

Abstract. *Highly configurable systems (HCSs)*, such as software product lines, have complex configuration spaces. *Combinatorial Testing* and *Random Testing* are the main approaches to testing of HCSs. In this paper, we empirically compare their strengths with respect to scalability and diversity of sampled configurations (i.e., tests). We choose ICPLand QUICKSAMPLERto respectively represent *Combinatorial Testing* and *Random Testing*. Experiments are processed to evaluate the *t-way coverage criterion* of generated test suites for HCS benchmarks.

Keywords: combinatorial testing · random testing · software product line

1 Introduction

Highly configurable systems (HCSs), such as software product lines (SPLs), have complex configuration spaces. The configuration spaces are often determined by *features* and *constraints*, which are modeled in some logical formalism over features, such as *feature models*. Figure 1 shows a simple feature model which describes the configuration space of Eclipse IDE [9]. A *configuration* (or a *test*) specifies which features are de-/selected in a given feature model. Table 1 shows a set of configurations of the feature model of Figure 1.

Configuration testing validates if systems run correctly at various configurations. However, testing all configurations in a given configuration space exhaustively is infeasible for any non-trivial HCS, since the number of configurations increases exponentially with the number of features (or parameters). Techniques to effectively sample configurations are needed to effectively test HCSs.

Combinatorial Testing (CT) [11] and *Random Testing (RT)* are main approaches to effective configuration sampling for testing of HCSs. Given a HCS and *strength* t, which is a small positive integer such as 2 or 3, CT requires all possible feature interactions of size t are tested by at least one test in a test suite. This condition is called the *t-way coverage criterion*. Note that the integer t is user-defined. The effectiveness of CT has been demonstrated empirically (e. g., [10,11]). Sampling algorithms for CT test suites (CT algorithms for short) are a central subject in the CT research. The 12 configurations in Table 1 meets the 2-way coverage criterion. We call such a set of test cases a t-way test suite.

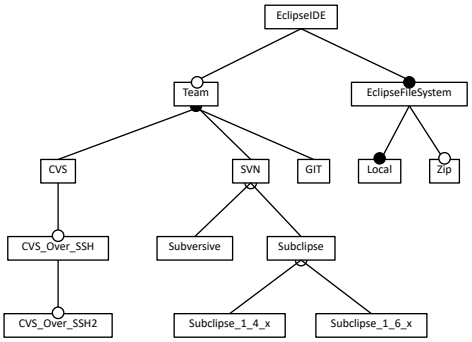

Fig. 1. A feature model of Eclipse IDE (using the notation of [9])

Table 1. Twelve configurations, sampled from Eclipse IDE feature model in Figure 1. Each column represents a configuration, where 'X' and '-' respectively mean selected and de-selected features of corresponding features in the rows.

Feature \ Configurations	1	2	3	4	5	6	7	8	9	10	11	12
EclipseIDE	X	X	X	X	X	X	X	X	X	X	X	X
Team	X	X	-	X	X	X	X	X	X	X	-	X
CVS	X	X	-	X	X	-	X	-	X	-	-	-
CVS_Over_SSH	X	-	-	X	X	-	X	-	X	-	-	-
CVS_Over_SSH2	-	-	-	X	X	-	X	-	X	-	-	-
SVN	X	X	-	X	X	X	-	X	X	X	-	-
Subversive	X	-	-	X	-	-	-	X	-	-	-	-
Subclipse	-	X	-	-	X	X	-	-	X	X	-	-
Subclipse_1_4_x	-	X	-	-	X	-	-	-	-	X	-	-
Subclipse_1_6_x	-	-	-	-	-	X	-	-	X	-	-	-
GIT	X	-	-	-	X	X	-	X	-	-	-	X
EclipseFileSystem	X	X	X	X	X	X	X	X	X	X	X	X
Local	X	X	X	X	X	X	X	X	X	X	X	X
Zip	-	X	X	X	-	-	-	X	X	-	-	-

RT for HCSs is addressed by, for example, Hirasaki et al. [8]. They developed an algorithm to randomly sample configurations from a test space specified by a logical formula. Solution sampling techniques have been actively investigated in the field of constraint solving (instead of testing), yielding several algorithms using such as SAT solvers [4,2] or Binary Decision Diagrams (BDDs) [15]. Although such sampling techniques are not originally developed for software testing, we can use them as configuration sampling algorithms because configuration spaces can be represented as Boolean formulas.

In this paper, we compare CT and RT in testing of large and complex HCSs, focusing on the following RQs:

RQ1 How scalable are CT and RT sampling techniques?
RQ2 How diverse are sampled configurations by RT in terms of t-way coverage criteria?

It is generally perceived that CT sampling is less scalable than RT sampling, since it spends computation resources to maximize t-way coverage criteria. With RQ1, we investigate scalability limits of the both approaches in the setting of HCSs in comparison. RQ2 is posed to evaluate the two approaches with respect to the quality of sampled configurations. Based on the definition of *t-way coverage criterion*, a test suite with smaller size that meets the t-way coverage criteria has better diversity as different tests in it are tending to test different t-way interactions. Hence, we choose t-way coverage criteria as the evaluation metric of the test suite diversity.

2 Related work

Several studies have compared RT and CT from different perspectives. Arcuri and Briand [1] theoretically analyzed the fault detecting ability of RT in terms of t-way coverage, showing that any t-way interaction can be detected with a

probability of 63% by an RT test suite whose size is the same as that of a theoretically-optimal CT test suite. Our work can be viewed as an empirical analogue to the theoretical analysis in the context of large HCS testing.

Wu et al. [16] empirically compared CT, RT, and Adaptive Random Testing (ART) [3] for their fault detecting abilities, in different settings concerning the proportion of parameters, constraints recognized by a tester, and fault rate (degree of injected faults in a system). They concluded, e. g., that the detecting ability of CT is high regardless of different settings and that the three techniques perform equally when a tester knows little about constraints. Contrarily, we are interested in more foundational questions about CT and RT in the context of HCS testing: e. g., scalability and diversity of sampled configurations.

3 Sampling techniques of CT and RT

A number of CT and RT algorithms have been proposed in the literature. However, their implementations are rarely publicly available. We thus select one representative algorithm from each of the two approaches and compare implementations of the two selected algorithms in our experiments. This section briefly reviews algorithms of the two approaches and present the selected algorithms.

3.1 Sampling techniques of CT

In this paper, we select the ICPL algorithm by Johansen et al. [9] as the representative to be compared with an RT algorithm in the next section. We explain the decision, by reviewing existing CT algorithms from three aspects.

Most of the existing CT algorithms can be characterized according to algorithm paradigms, into the two categories: *greedy* and *global optimization* algorithms. The main advantage of greedy algorithms is that they run fast while maintaining the generated test suites to be reasonably small. Algorithms/tools such as [19], including ICPL [9], belong to this algorithm paradigm. Global optimization algorithms put emphasis on minimizing test suites and use computationally costly search techniques such as constraint solving [14] or meta-heuristic search [7,12]. They generally can find smaller test suites than greedy algorithms; however, they suffer more from scalability. In this work, we do not consider global optimization algorithms hereafter, because it is already clear that these algorithms cannot compete with RT algorithms in terms of scalability.

ICPL is especially tailored to handle large and complex HCSs, i. e., HCSs with thousands of features and hundred thousands of clauses to specify complex configuration spaces. Some other greedy algorithms employ constraint handling mechanisms (e. g., [18]). The work [19] elaborates a constraint handling technique based on *Minimum Invalid Tuples (MITs)*. However, computing MITs is so costly that it cannot scale to large HCSs. Algorithm by [17] has a refined constraint handling mechanism using the notion of *un-satisfiability cores* [13]; but its implementation is not publicly available.

3.2 Sampling techniques of RT

Random sampling techniques from configuration spaces specified by logical formulas have been studied in the field of constraint solving. Such random sampling techniques can be classified into two types: clausal-SAT-based and BDD-based. Techniques of the former approach, such as QuickSampler [4], Unigen [2], and SearchTreeSampler (STSampler, for short) [5], take constraints in CNF as input and sample configurations. These techniques all use repetitive calls of SAT solvers to sample configurations but differ at detailed levels. The BDD-based technique by [15] performs sampling by building a BDD from the input logical formula (not necessarily in CNF) and traversing it. The technique by Hirasaki et al. [8] uses BDDs for constraint solving.

We adopt QuickSampler, a technique with the clausal-SAT-based approach, as the representative of RT sampling techniques. The clausal-SAT-based approach is superior to the BDD-based approach in terms of scalability. Among clausal-SAT-based techniques, QuickSampler is one of the most recently developed techniques, shown to be competitive with other random sampling techniques such as Unigen and STSampler[4] in efficiency and randomness.

4 Experiments

In this section, we conduct experiments to investigate the RQs raised in Section 1.

4.1 Experiment Settings

Our experiments compare Icpl and QuickSampler as the representative algorithms of CT and RT generation techniques, using publicly available implementations. As benchmark data we collected models of 11 HCSs from different sources as summarized in Table 2. For the HCS benchmarks, all features have exactly two values (binary domains) and their configuration spaces are specified in CNF, which is the format both of Icpl and QuickSampler can process.

We measure the computation time (for RQ1), and sizes of generated t-way test suites by Icpl for $t = 2$ and $t = 3$ (for RQ2) for the HCS benchmarks. We also let QuickSampler sample as many configurations as those generated by Icpl to meet the 2-way and 3-way coverage criteria. We measure the computation time of QuickSampler and the t-way coverage of the sampled configurations. We set timeout for sampling to 1 hour and timeout for measuring coverage of sampled configurations to 4 hours. For benchmarks for which Icpl failed to generate test suites within the time limit, we let QuickSampler sample 1000 and 2000 configurations for 2-way and 3-way coverage, respectively, as a enough large number to confirm its scalability.

All the experiments are conducted on a machine with 3 GHz 8 Core Intel Xeon E5 1680v2 CPU and 64 GB memory, running MacOS Sierra. QuickSampler can only run on a single thread, while Icpl internally runs on eight threads in the experiments. Both are allowed to use 64 GB memory.

[3] https://zenodo.org/record/265808#.X08JB9P7RGA

Table 2. HCS Benchmark information. The columns for '#F', '#C', '#Intr. (t=2)', and '#Intr. (t=3)' respectively show the number of features, of clauses, of 2-way interactions, and of 3-way interactions of features.

No.	HCS	#F	#C	#Intr. (t=2)	#Intr. (t=3)
1	Arcade Game Maker Pedagogical Product Line [6]	61	122	7,320	287,920
2	Berkeley DB [6]	78	151	12,012	608,608
3	Violet [6]	101	203	20,200	1,333,200
4	toybox [3]	544	1,020	590,784	213,469,952
5	axTLS [3]	684	2,155	934,344	424,815,072
6	eCos 3.0 i386pc [6]	1,244	3,146	3,092,584	2,560,659,552
7	FreeBSD Kernel [6]	1,396	62,183	3,894,840	3,619,604,640
8	Fiasco [3]	1,638	5,228	5,362,812	5,849,040,288
9	uClinux [3]	1,850	2,468	6,841,300	8,428,481,600
10	BusyBox [4]	6,796	17,836	92,357,640	418,318,537,440
11	X86 Linux Kernel 2.6.28.6 [5]	6,888	343,944	94,875,312	435,540,932,288

Table 3. Experimental results. The columns for 'time' and 'size' in 'ICPL (2-way)' respectively show the generation times (in seconds in wall time) and sizes of generated 2-way test suites, where 'T.O.' means ICPL failed in generation within the time limit. The columns for in 'ICPL (3-way)' show the same for 3-way test suites. The columns for 'QUICKSAMPLER' show (1) the computation time ('time') to sample configurations of the specified sizes ('size'), where 'T.O.' means measuring coverages does not finish within the time limit, and (2) the coverage scores of the sampled configurations for the 2-way coverage criterion ('cov. (%)'). The columns for 'QUICKSAMPLER' show the same items but include columns 't=2 cov. (%)' and 't=3 cov. (%)' for the scores of 2-way and 3-way coverage criteria.

No.	ICPL (2-way)		ICPL (3-way)		QUICKSAMPLER			QUICKSAMPLER			
	size	time(s)	size	time(s)	size	time(s)	cov. (%)	size	time(s)	t=2 cov.(%)	t=3 cov.(%)
1	22	0.33	75	7.10	22	0.01	42.33	75	0.04	44.85	30.80
2	26	0.35	122	7.03	26	0.02	35.55	122	0.05	51.42	37.78
3	31	0.49	153	593.03	31	0.02	30.10	153	0.05	32.93	19.67
4	23	2.89	T.O.		23	0.09	61.43	(2000)	0.66	70.39	T.O.
5	25	7.99	T.O.		25	0.26	57.14	(2000)	1.04	59.32	T.O.
6	72	107.04	T.O.		72	0.73	30.71	(2000)	3.15	31.85	T.O.
7	82	134.73	T.O.		82	3.20	28.71	(2000)	9.24	31.79	T.O.
8	155	124.31	T.O.		155	1.96	77.76	(2000)	3.37	80.22	T.O.
9	32	91.23	T.O.		32	0.18	57.49	(2000)	1.55	58.24	T.O.
10	T.O.		T.O.		(1000)	8.92*	T.O.	(2000)	13.25*	T.O.	T.O.
11	T.O.		T.O.		(1000)	47.83*	T.O.	(2000)	78.24*	T.O.	T.O.
avg.							46.80			51.22	29.41

4.2 Experimental results, and answers for RQs

Table 3 shows the experimental results, based on which we answer the RQs.

Answer to RQ1: "How scalable are CT and RT sampling techniques?" Observing Table 3, we can conclude the answer as follows: ICPL can scale for 2-way test suites for HCSs to up to 1850 features, and for 3-way test suites to up to 101 features. Considering also Table 2, several billions of interactions may be scalability limit of CT in our computing environment. On the other hand, QUICKSAMPLER can sample (even 2000 configurations) for all the HCSs including those with 6888 features and hundred thousands clauses.

[4] http://www.busybox.net/
[5] https://www.kernel.org

Answer to RQ2: "How diverse are sampled configurations by RT in terms of t-way coverage criteria?" Table 3 shows that when the number of sampled configurations is equal to the 2-way and 3-way test suites generated by ICPL, QUICKSAMPLER achieves only 46.8% and 29.4% for 2-way and 3-way coverage respectively. We thus conclude that *the diversity of configurations sampled by CT is 2 to 3 times higher than those sampled by RT*. It is also interesting to see the following: (1) Even sampled configurations are as many as the 3-way test suite generated by ICPL, QUICKSAMPLER achieves only 51.22% with respect to 2-way coverage. (2) The measured t-way coverages of RT test suites are much less than the theoretical bound (63%) provided by [1]; this can be explained by that the theoretical analysis assumes unconstrained configuration spaces, though.

5 Threats to Validity

A possible threat to the validity of this comparative study is the representativeness of the HCS benchmarks used in the experiments. In the HCS models, all features are modeled as binary values. Coverage results might be qualitatively different, when features are modeled with more than two values. Such models can be handled by many CT algorithms such as [17,18]. It is, however, worth noting that recent CT algorithms, including ICPL, are optimized to deal with binary features for acceleration [9,19]. To reflect such recent advances of CT algorithms, our experiments focused on HCSs with binary features.

The validity may be also threatened by the selection for representative algorithms of CT and RT, i. e., ICPL and QUICKSAMPLER. Although we have explained rationale of the selection, experiments using different algorithms may show different results. This paper reports the first step toward comparison of CT and RT in the setting of HCSs, and thus we plan to extend the study to include other algorithms from both CT and RT in experiments in our future work.

6 Conclusion and Future Work

Sampling techniques from logical formula have been an active research subject in the fields of system testing and constraint solving, and various algorithms have been proposed recently from the fields. On the other hand, relatively little attention has been paid to comparing their strengths. This situation causes a problem in testing practice such as testers have confidence in choosing right testing techniques. In this paper, we take the first step toward this problem, investigating strengths of CT and RT in testing of large HCSs.

We consider several directions for future work. First, we plan to extend this comparative study to include other representative algorithms from CT, RT, and constraint solving techniques. In addition to QUICKSAMPLER and ICPL, we plan to include other clausal-SAT-based sampling techniques (e. g., UNIGEN [2] and STSAMPLER [5]), BDD-based techniques (e. g., [8,15]), and other CT greedy algorithms (e. g., [17]) in the comparative study. We also plan to extend the study to investigate fault detection capabilities of CT and RT, as done by [16].

References

1. Arcuri, A., Briand, L.C.: Formal analysis of the probability of interaction fault detection using random testing. IEEE Trans. Software Eng. **38**(5), 1088–1099 (2012)
2. Chakraborty, S., Fremont, D.J., Meel, K.S., Seshia, S.A., Vardi, M.Y.: On parallel scalable uniform SAT witness generation. In: Proc. of TACAS 2015. pp. 304–319. Springer (2015)
3. Chen, T.Y., Leung, H., Mak, I.K.: Adaptive random testing. In: Proc. of ASIAN 2004. pp. 320–329 (2004)
4. Dutra, R., Laeufer, K., Bachrach, J., Sen, K.: Efficient sampling of SAT solutions for testing. In: Proc. ICSE 2018. pp. 549–559 (2018)
5. Ermon, S., Gomes, C., Selman, B.: Uniform solution sampling using a constraint solver as an oracle. In: Proc. of UAI 2012. p. 255–264 (2012)
6. Gargantini, A., Radavelli, M.: Migrating combinatorial interaction test modeling and generation to the web. In: 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW). pp. 308–317 (April 2018). https://doi.org/10.1109/ICSTW.2018.00066
7. Garvin, B.J., Cohen, M.B., Dwyer, M.B.: Evaluating improvements to a meta-heuristic search for constrained interaction testing. Empirical Software Engineering **16**(1), 61–102 (2011)
8. Hirasaki, Y., Kojima, H., Tsuchiya, T.: Applying random testing to constrained interaction testing. In: Proc. of SEKE 2013. pp. 193–198 (2014)
9. Johansen, M.F., Haugen, O., Fleurey, F.: An algorithm for generating t-wise covering arrays from large feature models. In: Proc. of SPLC 2012. pp. 46–55 (2012)
10. Kuhn, D.R., Bryce, R., Duan, F., Ghandehari, L.S., Lei, Y., Kacker, R.N.: Chapter one - combinatorial testing: Theory and practice. Advances in Computers, vol. 99, pp. 1 – 66. Elsevier (2015)
11. Kuhn, D.R., Kacker, R.N., Lei, Y.: Introduction to combinatorial testing. CRC Press (2013)
12. Lin, J., Cai, S., Luo, C., Lin, Q., Zhang, H.: Towards more efficient meta-heuristic algorithms for combinatorial test generation. In: Proc. of ESEC/FSE 2019. pp. 212–222 (2019)
13. Lynce, I., Silva, J.P.M.: On computing minimum unsatisfiable cores. In: Proc. of SAT 2004 (2004)
14. Nanba, T., Tsuchiya, T., Kikuno, T.: Using satisfiability solving for pairwise testing in the presence of constraints. IEICE Transactions **95-A**(9), 1501–1505 (2012)
15. Oh, J., Batory, D.S., Myers, M., Siegmund, N.: Finding near-optimal configurations in product lines by random sampling. In: Proc. of ESEC/FSE 2017. pp. 61–71 (2017)
16. Wu, H., Nie, C., Petke, J., Jia, Y., Harman, M.: An empirical comparison of combinatorial testing, random testing and adaptive random testing. IEEE Transactions on Software Engineering (2018)
17. Yamada, A., Biere, A., Artho, C., Kitamura, T., Choi, E.: Greedy combinatorial test case generation using unsatisfiable cores. In: Proc. of ASE 2016. pp. 614–624 (2016)
18. Yu, L., Lei, Y., Kacker, R.N., Kuhn, D.R.: ACTS: A combinatorial test generation tool. In: Proc. of ICST 2013. pp. 370–375 (2013)
19. Yu, L., Duan, F., Lei, Y., Kacker, R., Kuhn, D.R.: Combinatorial test generation for software product lines using minimum invalid tuples. In: Proc. of HASE 2014. pp. 65–72 (2014)