Design of Prioritized N-Wise Testing

Eun-Hye Choi, Takashi Kitamura, Cyrille Artho, Yutaka Oiwa

Design of Prioritized N-Wise Testing

Eun-Hye Choi, Takashi Kitamura, Cyrille Artho, and Yutaka Oiwa

Nat. Inst. of Advanced Industrial Science and Technology, Amagasaki, Japan
{e.choi, t.kitamura, c.artho, y.oiwa}@aist.go.jp

Abstract. N-wise testing is a widely used technique for combinatorial interaction testing. Prioritizing testing reorders test cases by relevance, testing important aspects more thoroughly. We propose a novel technique for N-wise test case generation to satisfy the three distinct prioritization criteria of interaction coverage, weight coverage, and KL divergence. The proposed technique generates small N-wise test cases, where high-priority test cases appear early and frequently. Our early evaluation confirms that the proposed technique improves on existing techniques based on the three prioritization criteria.

Keywords: Combinatorial testing, N-wise testing, prioritized testing

1 Introduction

N-wise testing ($N = 1, 2, ...$), e. g., *pairwise testing* when $N = 2$, is a widely used technique for combinatorial interaction testing. It is based on a coverage criterion called *interaction coverage*, which stipulates to test all N-tuples of parameter values at least once, given a system under test (SUT) model as sets of parameters, parameter values, and constraints to express parameter interactions.

Recently, techniques for *prioritized N-wise testing*, which covers prioritized test aspects earlier and more thoroughly, have been proposed. This is motivated by practical demand, as resources for testing are often limited in real-world software development. Previous work on prioritized N-wise testing mainly falls into two categories: (1) reordering of test cases based on coverage criteria such as code coverage [2, 8], and (2) generating prioritized test cases given SUT models with user-defined priorities assigned to parameters or values [1, 3, 5, 6]. (We call such models *weighted SUT models*.)

In this paper, we propose a novel approach to prioritizing N-wise testing, along the line of category (2). Our technique for prioritized N-wise test case generation, given weighted SUT models, satisfies the following criteria:

- CO: Higher-priority test cases should appear earlier.
- CF: Higher-priority parameter values should appear more frequently.
- CS: The number of test cases should be as small as possible.

Our proposed technique provides a strategy to achieve a good balance between the three criteria. By considering CO, CF, and CS together, our strategy obtains important test cases early and frequently in a small test suite.

Table 1. An example SUT model with value pairs and weights.

(a) An example SUT model. N=2.

Parameter: *Value(Weight)*	
CPU:	AMD(2), Intel(6)
OS:	Mac(1), Ubuntu(1), Win(2)
Browser:	IE(3), Firefox(3), Chrome(3), Safari(3), Opera(3)

(b) All value pairs and weights.

CO	W
AM	3
AU	3
AW	4
IM	7
IU	7
IW	8

CB	W
AI	5
AF	5
AC	5
AS	5
AO	5

CB	W
II	9
IF	9
IC	9
IS	9
IO	9

OB	W
MI	4
MF	4
MC	4
MS	4
MO	4
UI	4
UF	4
UC	4

OB	W
US	4
UO	4
WI	5
WF	5
WC	5
WS	5
WO	5

The next section describes the state-of-the-art techniques for prioritized N-wise test case generation. It shows that existing techniques only consider some of the criteria. In Section 3, we explain the proposed technique in detail, together with a comparison of our technique with the state-of-the-art techniques using sample data. The last section concludes this paper and proposes future work.

2 Related Work

Our running example, the weighted SUT model shown in Tab. 1-(a), has three parameters: *(OS, CPU, Browser).* Each parameter has two, three, and five values, respectively. Weights are assigned to values, as specified in parentheses, e. g., value *Win* has weight 2. (For brevity, we do not consider constraints; N-wise constraints can be managed in the initial enumeration phase of our algorithm.) The weight of a value pair is the sum of the weights of its values. Tab. 1-(b) shows all the value pairs and their weights, W, in the model. A test case is a set of value assignments for all parameters. A pairwise (N-wise) test suite is a set of test cases that covers all pairs (N-tuples) of values in the model.

For prioritized N-wise test generation given a weighted SUT model, several techniques have been proposed [1, 3, 5, 6]. Some techniques use priority for *ordering* test cases [1, 6], while others use it for balancing the occurrences *(frequency)* of test cases [3, 5, 7]. Tab. 2(c)−(f) show the test suites generated by these techniques. They all generate test cases one at a time, until all value pairs are covered. We briefly introduce the difference among these methods here. Prioritization criteria are considered when selecting a new test case as follows.

Order-focused prioritization: CTE-XL [6] aims to obtain test cases ordered by weight. It focuses on the weight of values assigned to a new test case. On the other hand, Bryce's technique [1] aims to obtain test cases with better *weight coverage*, as defined in Fig. 1. To this aim, it provides an approximate algorithm to maximize the weight of newly-covered pairs in a new test case.

Both techniques share the advantage that important test cases appear at an earlier stage of a test suite, satisfying criterion CO. The disadvantage is that they do not consider the frequency aspect (CF) at all (see Tab. 2). Furthermore, CTE-XL generates a test suite that is larger than necessary for interaction coverage; this is because it focuses on ordering test cases, rather than improving interaction coverage. Actually, the size of CTE-XL's test suite (18 test cases) is larger than Bryce's (15 test cases), which suffices for satisfying pairwise coverage.

Table 2. Generated test suites by the proposed and previous methods.

(a) A1. (CS>CO>CF)

	C O B	n	w	*p-cov*	*w-cov*	D
1	I W I	3	22	0.097	0.132	2.590
2	I M F	3	20	0.194	0.251	1.551
3	I U C	3	20	0.290	0.371	0.855
4	A W F	3	14	0.387	0.455	0.570
5	A U S	3	12	0.484	0.527	0.385
6	A M I	3	12	0.581	0.599	0.480
7	I W S	2	14	0.645	0.683	0.194
8	I W O	2	14	0.710	0.766	0.088
9	A W C	2	10	0.774	0.826	0.124
10	A M O	2	9	0.839	0.880	0.154
11	I U I	1	4	0.871	0.904	0.117
12	I M C	1	4	0.903	0.928	0.107
13	I U F	1	4	0.935	0.952	0.089
14	I M S	1	4	0.968	0.976	0.085
15	I U O	1	4	1	1	0.074

(b) A2. (CO>CS>CF)

	C O B	n	w	*p-cov*	*w-cov*	D
1	I W I	3	22	0.097	0.132	2.590
2	I M F	3	20	0.194	0.251	1.551
3	I U C	3	20	0.290	0.371	0.855
4	A W F	3	14	0.387	0.455	0.570
5	I W S	2	14	0.452	0.539	0.304
6	I W O	2	14	0.516	0.623	0.126
7	A M I	3	12	0.613	0.695	0.097
8	A U S	3	12	0.710	0.766	0.088
9	A W C	2	10	0.774	0.826	0.124
10	A M O	2	9	0.839	0.880	0.154
11	I U I	1	4	0.871	0.904	0.117
12	I M C	1	4	0.903	0.928	0.107
13	I U F	1	4	0.935	0.952	0.089
14	I M S	1	4	0.968	0.976	0.085
15	I U O	1	4	1	1	0.074

(c) Bryce's.

	C O B	n	w	*p-cov*	*w-cov*	D
1	I W I	3	22	0.097	0.132	2.590
2	I M F	3	20	0.194	0.251	1.551
3	A W C	3	14	0.290	0.335	0.816
4	I W S	2	14	0.355	0.419	0.527
5	I W O	2	14	0.419	0.503	0.338
6	I U C	3	20	0.516	0.623	0.126
7	A W F	2	10	0.581	0.683	0.158
8	A M I	3	12	0.677	0.754	0.141
9	A U S	3	12	0.774	0.826	0.124
10	A M O	2	9	0.839	0.880	0.154
11	A U I	1	4	0.871	0.904	0.217
12	A U F	1	4	0.903	0.928	0.290
13	A M C	1	4	0.935	0.952	0.343
14	A M S	1	4	0.968	0.976	0.399
15	A U O	1	4	1	1	0.440

(d) CTE XL.

	C O B	n	w	*p-cov*	*w-cov*	D
1	I W I	3	22	0.097	0.132	2.590
2	I M F	3	20	0.194	0.251	1.551
3	A W I	2	9	0.258	0.305	1.278
4	A M F	2	8	0.333	0.353	1.407
5	A U C	3	12	0.419	0.425	0.882
6	A M S	2	9	0.484	0.479	0.807
7	A M O	2	9	0.548	0.533	0.766
8	I U C	2	16	0.613	0.629	0.536
9	I U S	2	13	0.677	0.707	0.411
10	I U O	2	13	0.742	0.784	0.337
11	A M I	1	4	0.774	0.808	0.437
12	A M C	1	4	0.806	0.832	0.529
13	A U I	1	4	0.839	0.856	0.609
14	A U F	1	4	0.871	0.880	0.661
15	A W F	1	5	0.903	0.910	0.622
16	A W C	1	5	0.935	0.940	0.603
17	A W S	1	5	0.968	0.970	0.575
18	A W O	1	5	1	1	0.556

n: Number of new value pairs
w: Weight of new value pairs
p-cov: Pairwise coverage
w-cov: Weight coverage
D: KL Divergence

(e) PictMaster.

	C O B	n	w	*p-cov*	*w-cov*	D
1	I W F	3	22	0.097	0.132	2.590
2	I W C	2	14	0.161	0.216	1.897
3	I U C	2	11	0.226	0.281	1.548
4	I W S	2	14	0.290	0.365	1.161
5	I M C	2	11	0.355	0.431	0.967
6	I M O	2	13	0.419	0.509	0.683
7	I U S	1	4	0.452	0.533	0.630
8	I U F	1	4	0.484	0.557	0.620
9	A W C	2	9	0.548	0.611	0.415
10	A W S	1	5	0.581	0.641	0.347
11	I M I	2	13	0.645	0.719	0.159
12	I M S	1	4	0.677	0.743	0.204
13	A U F	2	8	0.742	0.790	0.179
14	I W O	1	5	0.774	0.820	0.111
15	I U O	1	4	0.806	0.844	0.111
16	I W I	1	5	0.839	0.874	0.051
17	A U I	2	9	0.903	0.928	0.038
18	I M F	1	4	0.935	0.952	0.039
19	A M F	1	3	0.968	0.970	0.055
20	A W I	0	0	0.968	0.970	0.039
21	A M C	0	0	0.968	0.970	0.064
22	I U I	0	0	0.968	0.970	0.067
23	A M S	0	0	0.968	0.970	0.089
24	A W O	1	5	1	1	0.075
25	A U C	0	0	1	1	0.102
26	A W F	0	0	1	1	0.109
27	A M I	0	0	1	1	0.136
28	A M O	0	0	1	1	0.156

(f) PICT+Fujimoto's.

	C O B	n	w	*p-cov*	*w-cov*	D
1	A W C	3	14	0.097	0.084	3.689
2	A M I	3	12	0.194	0.156	2.649
3	I M F	3	20	0.290	0.275	1.413
4	I W I	3	22	0.387	0.407	1.060
5	I U I	2	11	0.452	0.473	0.767
6	I M C	2	13	0.516	0.551	0.759
7	A U C	2	7	0.581	0.593	0.790
8	A U O	2	9	0.645	0.647	0.629
9	I W O	2	14	0.710	0.731	0.444
10	I M O	1	4	0.742	0.754	0.439
11	A U F	2	9	0.806	0.808	0.442
12	A W F	1	5	0.839	0.838	0.424
13	A U S	2	9	0.903	0.892	0.328
14	I W S	2	14	0.968	0.976	0.200
15	A M S	1	4	1	1	0.239
16	I W I	0	0	1	1	0.183
17	I W F	0	0	1	1	0.139
18	I W C	0	0	1	1	0.105
19	I W S	0	0	1	1	0.077
20	I W O	0	0	1	1	0.054
21	I M I	0	0	1	1	0.049
22	I U F	0	0	1	1	0.042
23	I W C	0	0	1	1	0.030
24	I W S	0	0	1	1	0.021
25	I M O	0	0	1	1	0.015
26	I U I	0	0	1	1	0.014
27	I W F	0	0	1	1	0.010
28	I W C	0	0	1	1	0.007

Frequency-focused prioritization: PICT [3], PictMaster [7], and Fujimoto's method [5] obtain higher-priority values more frequently; in our example, for parameter *OS*, value *Win* with weight 2 should appear twice as frequently as values *Mac* and *Ubuntu* with weight 1. In PICT, given weights are considered only if two value choices are identical, and thus the frequency is reflected only approximately. To improve this, PictMaster constructs a test suite to redundantly cover pairs according to their weights, and thus the size is unnecessary large as shown in Tab. 2-(e). On the other hand, Fujimoto developed a method to add test cases to an existing test suite that more accurately reflect given weights for value frequency. The test suite generated by Fujimoto's method in Tab. 2 shows that *Win* appears 14 times, twice as frequently as *Mac* (7 times), but the test suites by PICT and PictMaster contains the same numbers of *Win* and *Mac*.

In the example, PictMaster generates 28 test cases, while Fujimoto's method uses 15 test cases by PICT plus 13 optional test cases (to more accurately reflect given weights for value frequency); see Tab. 2-(e),(f). The strength of these approaches is that more important values appear more frequently; however, the order of important test cases is completely disregarded.

Algorithm 1 Prioritized N-wise test generation.

Input: A weighted SUT model, Combinatorial strength N.
Output: A N-wise test suite.
1) For each parameter, order all parameter values by weight.
2) Enumerate all N-tuples of parameter values and their weights.
3) Set a set of uncovered combinations, UC, as the set of all the value N-tuples.
while $UC \neq \emptyset$ **do**
  4) List test case candidates to cover the max. number of combinations in UC. [CS]
  4-1) If there is one candidate, choose it as t.
  4-2) If there are several candidates, list candidates with the max. weight of new combinations. [CO]
  4-2-1) If there is one candidate, choose it as t.
  4-2-2) If there are several candidates, choose any candidate with min. KL divergence as t. [CF]
  5) Set t as the next test case and add it to the test suite.
  6) Remove the value N-tuples covered by t from UC.
end while

3 Proposed approach to prioritized test generation

We propose a novel approach to prioritized N-wise test generation, which integrates the three prioritization criteria of *order,* CO, *frequency,* CF, and *size,* CS. For CS, we select a new test case to cover as many uncovered value N-tuples as possible. For CO, we select a new test case to cover value N-tuples with the highest weight. For CF, a new test case is selected according to the occurrence ratio of values in a test suite. To rigorously consider CF, we employ the notion of *KL Divergence* used in [5]. KL divergence, $D_{KL}(P||Q)$ defined in Fig. 1, expresses the distance between the current value occurrence distribution, P, and an ideal distribution, Q, according to given weights. In the ideal distribution, for each value, the number of occurrences is proportional to its weight. By integrating the three criteria CO, CF, and CS, our approach obtains small test suites where high-priority test cases appear early and frequently.

Algorithm 1 shows the pseudo code of the algorithm with priority order CS>CO>CF. Given a weighted SUT model, the algorithm generates test cases one by one until all N-tuples of parameter values are covered. For a new test case, we choose the best one w. r. t. CS, i. e., the test case that covers the most new N-tuples of parameter values (step 4). If there are several candidates, choose the best one for CO, i. e., the test case that covers the new N-tuples of parameter values with the highest weight (step 4-2). If there are still several candidates, choose the best one for CF; i. e., the test case with min. KL divergence (step 4-2-2) to ensure that the value frequency distribution is closest to the ideal state.

Tab. 2-(a) shows the pairwise test suite generated from the example model in Tab. 1-(a), by Algorithm 1 (A1). For the first test case, all the test cases in which the number of newly-covered value pairs (n) is 3 become candidates, since this is the max. number at this stage. Among them, (*Intel, Win,* $-$[1]) is selected as it has the max. value of w, 22. For the 11th test case, test cases 11–15 have the same n and w. In the first ten test cases, *AMD* and *Intel* are assigned equally often to parameter *CPU*, even though the weight of *Intel* is twice of that of *AMD*. Furthermore, *Ubuntu* occurs less often than *Mac* despite their weights

[1] The symbol $-$ can take any value for *Browser* as all values have the same weight.

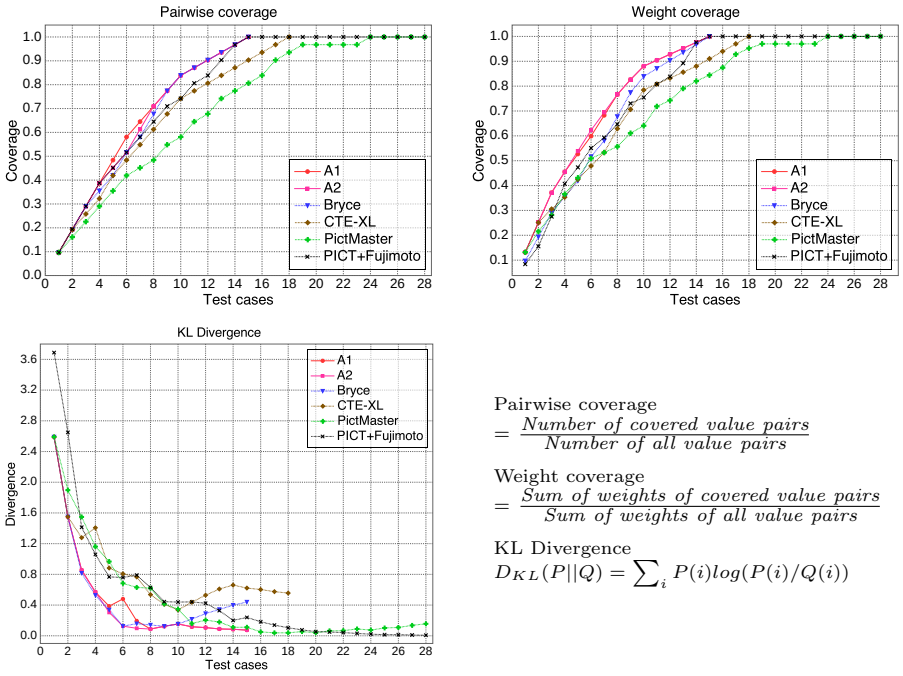

Fig. 1. Comparison of the evaluation metrics.

being the same. Thus, *Intel* and *Ubuntu* are assigned to the 11th test case to achieve a better KL divergence.

We can construct variants of the algorithm with different orders of prioritization criteria CO, CF, and CS, by swapping the conditions in steps 4, 4-2, and 4-2-2 in Algorithm 1. For example, we can construct another algorithm A2 with CO>CS>CF. Tab. 2-(b) shows the pairwise test suite generated by A2. Note that test cases by A1 (resp., A2) are ordered by n (w), since CO (CW) is the first priority criterion. Test cases with the same value of n (resp., w) are ordered by w (n) for A1 (A2), since CW (CO) is the second priority criterion. Furthermore, the test suites generated by A1 and A2 require only 15 test cases, which is fewer than those computed by most previous methods.

We compared our method with the previous methods from the three metrics of interaction coverage, weight coverage, and KL divergence, using several examples of weighted SUT models. As a result, we confirmed that our proposed method can improve on all the metrics.[2] Fig. 1 shows the evaluation result on our example in Tab. 1-(a).[3] The proposed algorithms are superior to the previous methods according to the three metrics. A1 and A2, respectively, provide the

[2] To search locally optimal test cases, our algorithm can incur a high computing cost with a high quality for large models. The cost will be evaluated in our next paper.

[3] See http://staff.aist.go.jp/e.choi/evaluationGraphs.html for larger sized graphs.

best results for pairwise coverage and weight coverage, and thus we can obtain a high coverage of higher-priority test cases even when only a small number of test cases is selected. When considering KL divergence, A2 (resp., A1) provides the best result when the number of test cases is higher than 4 (7). Our method provides a good balance of value occurrences, even in a small test suite.

4 Conclusion and Future Work

We have proposed a novel technique for prioritized N-wise test generation, which integrates the three prioritization criteria of order, CO, frequency, CF, and size, CS. Our technique is designed to generate small N-wise test suites, where high-priority test cases appear early and frequently. Our early evaluation has shown that the technique can outperform the state-of-the-art techniques on all the three metrics. The technique is currently under implementation, and will be evaluated including computing cost on practical-sized SUT models. We are also considering to develop approximate algorithms with a lower computing cost, in case our technique is not scalable for practical sized SUT models. Future work also includes handling weights attached to both of parameters and values, and to structured SUT models [4].

Acknowledgements. The authors would like to thank Goro Hatayama and Tatsuhiro Tsuchiya for their helpful comments. This work is supported by JST ASTEP grant (No. AS2524001H).

References

1. R. Bryce and C. Colbourn. Prioritized interaction testing for pair-wise coverage with seeding and constraints. *Information & Software Technology*, 48(10):960–970, 2006.
2. R. Bryce, S. Sampath, J. Pedersen, and S. Manchester. Test suite prioritization by cost-based combinatorial interaction coverage. *Int. J. Systems Assurance Engineering and Management*, 2(2):126–134, 2011.
3. J. Czerwonka. Pairwise testing in the real world: Practical extensions to test case generators. *Microsoft Corporation, Software Testing Technical Articles*, 2008.
4. N. Do, T. Kitamura, N. Tang, G. Hatayama, S. Sakuragi, and H. Ohsaki. Constructing test cases for N-wise testing from tree-based test models. In *Proc. of SoICT 2013*. ACM ICPS, 2013.
5. S. Fujimoto, H. Kojima, and T. Tsuchiya. A value weighting method for pair-wise testing. In *Proc. of APSEC 2013*, pages 99–105, 2013.
6. P. Kruse and M. Luniak. Automated test case generation using classification trees. *Software Quality Professional*, pages 4–12, 2010.
7. PictMaster. http://sourceforge.jp/projects/pictmaster/.
8. Q. Xiao, M. Cohen, and K. Woolf. Combinatorial interaction regression testing: A study of test case generation and prioritization. In *Proc. ICSM'07*, pages 255–264. IEEE, 2007.