A Test Generation Strategy for Pairwise Testing

Kuo-Chung Tai and Yu Lei

Abstract—Pairwise testing is a specification-based testing criterion, which requires that for each pair of input parameters of a system, every combination of valid values of these two parameters be covered by at least one test case. In this paper, we propose a new test generation strategy for pairwise testing.

Index Terms—Software testing, pairwise testing, test generation.

1 INTRODUCTION

PAIRWISE testing requires that, for each pair of input parameters of a system, every combination of valid values of these two parameters be covered by at least one test case. Empirical results indicate that pairwise testing is practical and effective for various types of software systems [1], [2]. To illustrate the concept of pairwise testing, consider a system with parameters and values as shown below:

- parameter A has values A_1 and A_2,
- parameter B has values B_1 and B_2, and
- parameter C has values C_1, C_2, and C_3.

For parameters A and B, $\{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)\}$ is the only pairwise test set. For parameters A, B, and C, a large number of pairwise test sets exist. Below are three of them with the numbers of tests being 6, 7, and 8, respectively:

- $\{(A_1, B_1, C_1), (A_1, B_2, C_2), (A_2, B_1, C_3), (A_2, B_2, C_1), (A_2, B_1, C_2), (A_1, B_2, C_3)\}$,
- $\{(A_1, B_1, C_1), (A_1, B_2, C_1), (A_2, B_1, C_2), (A_2, B_2, C_3), (A_2, B_1, C_1), (A_1, B_2, C_2), (A_1, B_1, C_3)\}$,
- $\{(A_1, B_1, C_1), (A_1, B_2, C_1), (A_2, B_1, C_2), (A_2, B_2, C_2), (A_2, B_1, C_1), (A_1, B_1, C_2), (A_1, B_1, C_3), (A_2, B_2, C_3)\}$.

Different test generation strategies for pairwise testing have been published. The strategy proposed in [2] starts with an empty test set and adds one test at a time. To generate a new test, the strategy produces a number of candidate tests according to a greedy algorithm and then selects one that covers the most uncovered pairs.

Another approach to generating a pairwise test set is to use orthogonal arrays. The original method of orthogonal arrays requires that all parameters have the same number of values and that each pair of values be covered the same number of times [4]. The first requirement can be relaxed by adding *don't care* values for missing values. But, the use of *don't care* values creates extra tests [5]. The second requirement is considered unnecessary for software testing and also creates extra tests for pairwise testing [1].

In this paper, we propose a new test generation strategy, called in-parameter-order (or IPO), for pairwise testing. The remainder of this paper is organized as follows: Section 2 presents the IPO strategy. Section 3 describes an IPO-based test generation tool and shows some empirical results. Section 4 concludes this paper. An extended version of this paper is available [3].

- *The authors are with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7534.*
E-mail: {kct, ylei2}@eos.ncsu.edu.

Manuscript received 5 Jan. 1999; revised 12 Feb. 2001; accepted 26 Mar. 2001.
Recommended for acceptance by R. Hamlet.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 108587.

2 THE IPO STRATEGY

For a system with two or more input parameters, the IPO strategy generates a pairwise test set for the first two parameters, extends the test set to generate a pairwise test set for the first three parameters, and continues to do so for each additional parameter. The extension of an existing pairwise test set for an additional parameter contains the following two steps: a) horizontal growth, which extends each existing test by adding one value of the new parameter and b) vertical growth, which adds new tests, if necessary, to the test set produced by horizontal growth. Sections 2.1 and 2.2 show algorithms for horizontal and vertical growth, respectively.

2.1 An Algorithm for Horizontal Growth

Assume that $\mathcal{T}$ is a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{i-1}. The horizontal growth of $\mathcal{T}$ for parameter p_i is to extend each test in $\mathcal{T}$ by adding a value of p_i. Fig. 1 shows a high-level algorithm called IPO_H for horizontal growth of $\mathcal{T}$ for parameter p_i.

Now, we apply algorithm IPO_H to the example system in Section 1. $\{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)\}$ is the only pairwise test set for A and B. Since C has three values C_1, C_2, and C_3, we extend (A_1, B_1), (A_1, B_2), and (A_2, B_1) by adding C_1, C_2, and C_3, respectively. The extended tests are (A_1, B_1, C_1), (A_1, B_2, C_2), and (A_2, B_1, C_3), and the resulting set of missing (or uncovered) pairs is $\{(A_2, C_1), (B_2, C_1), (A_2, C_2), (B_1, C_2), (A_1, C_3), (B_2, C_3)\}$. Now, we need to choose one of C_1, C_2, and C_3 for (A_2, B_2). If we add C_1 to (A_2, B_2), the extended test (A_2, B_2, C_1) covers two missing pairs (A_2, C_1) and (B_2, C_1). If we add C_2 to (A_2, B_2), the extended test (A_2, B_2, C_2) covers only one missing pair (A_2, C_2). If we add C_3 to (A_2, B_2), the extended test (A_2, B_2, C_3) covers only one missing pair (B_2, C_3). Thus, we choose (A_2, B_2, C_1) as the fourth test. The following four pairs are not covered yet: (A_2, C_2), (A_1, C_3), (B_1, C_2), and (B_2, C_3). How to generate new tests to cover these four pairs is discussed next.

2.2 An Algorithm for Vertical Growth

Assume that the horizontal growth for parameter p_i has produced a test set $\mathcal{T}$ for $p_1, p_2, \ldots,$ and p_i. Let π be the set of pairs not covered by $\mathcal{T}$. Each pair in π contains a value of p_i and a value of $p_1, p_2, \ldots,$ or p_{i-1}. Assume that $|\pi| > 0$. The vertical growth of $\mathcal{T}$ according to π is to construct new tests for covering pairs in π and add these new tests to $\mathcal{T}$. Thus, the resulting $\mathcal{T}$ is a pairwise test set for $p_1, p_2, \ldots,$ and p_i. Fig. 2 shows a high-level algorithm called IPO_V for the vertical growth of $\mathcal{T}$ according to π. In this algorithm, "$-$" denotes an unspecified value of a parameter.

After the completion of algorithm IPO_V, $\mathcal{T}$ may contain "$-$" values. If p_i is the last parameter, each "$-$" value for p_k, $1 \le k \le i$, is replaced by any value of p_k. Otherwise, these "$-$" values are replaced by parameter values in the horizontal growth for p_{i+1} as follows: Assume that value v of p_{i+1} is chosen for the horizontal growth of a test that contains "$-$" as the value for p_k, $1 \le k \le i$. If there are uncovered pairs involving v and some values of p_k, the "$-$" for p_k is replaced by one of these values of p_k. Otherwise, the "$-$" for p_k is replaced by any value of p_k.

We continue our discussion of the example system defined in Section 1. In Section 2.1, we show that the horizontal growth for parameter C generates the following four tests: (A_1, B_1, C_1), (A_1, B_2, C_2), (A_2, B_1, C_3), and (A_2, B_2, C_1), and that these four tests do not cover the following four pairs: (A_2, C_2), (A_1, C_3), (B_1, C_2), and (B_2, C_3). Now, we apply algorithm IPO_V to construct new tests to cover these four pairs. To cover (A_2, C_2), we generate test $(A_2, -, C_2)$. To cover (A_1, C_3), we generate test $(A_1, -, C_3)$. To cover (B_1, C_2), we change $(A_2, -, C_2)$ to (A_2, B_1, C_2) without adding a new test. To cover (B_2, C_3), we change $(A_1, -, C_3)$ to (A_1, B_2, C_3) without adding a new test. Thus, we generate two new tests to cover the four pairs not covered by horizontal growth. So, the generated pairwise test set has a total of six tests.

0098-5589/02/$17.00 © 2002 IEEE

```
Algorithm IPO_H(T, p_i)
// T is a test set. But T is also treated as a list with elements in arbitrary order.
{ assume that the domain of p_i contains values v_1, v_2, ..., and v_q;
  π = { pairs between values of p_i and values of p_1, p_2, ..., and p_{i-1} };
  if (|T| ≤ q)
  { for 1 ≤ j ≤ |T|, extend the jth test in T by adding value v_j and
    remove from π pairs covered by the extended test;
  }
  else
  { for 1 ≤ j ≤ q, extend the jth test in T by adding value v_j and
    remove from π pairs covered by the extended test;
    for q < j ≤ |T|, extend the jth test in T by adding one value of p_i
    such that the resulting test covers the most number of pairs in π, and
    remove from π pairs covered by the extended test;
  }
}
```

Fig. 1. Algorithm IPO_H.

```
Algorithm IPO_V(T, π)
{ let T' be an empty set;
  for each pair in π
  { assume that the pair contains value w of p_k, 1 ≤ k < i, and value u of p_i;
    if (T' contains a test with "−" as the value of p_k and u as the value of p_i)
      modify this test by replacing the "−" with w;
    else
      add a new test to T' that has w as the value of p_k, u as the value of p_i,
      and "−" as the value of every other parameter;
  };
  T = T ∪ T';
};
```

Fig. 2. Algorithm IPO_V.

TABLE 1
Sizes of Pairwise Test Sets Generated by AETG and PairTest

System	S1	S2	S3	S4	S5	S6
AETG	11	17	35	25	12	193
PairTest	9	17	34	26	15	212

S1: 4 3-value parameters
S2: 13 3-value parameters
S3: 61 parameters (15 4-value parameters, 17 3-value parameters, 29 2-value parameters)
S4: 75 parameters (1 4-value parameter, 39 3-value parameters, 35 2-value parameters)
S5: 100 2-value parameters
S6: 20 10-value parameters

3 PairTest: An IPO-Based Test Generation Tool

We have implemented an IPO-based test generation tool, called PairTest, that includes algorithm IPO_H for horizontal growth and algorithm IPO_V for vertical growth. PairTest was written in Java and it provides a graphical user interface to make the tool easy to use. PairTest also supports the reuse of tests sets when systems are modified due to changes of input parameters and/or values.

Another test generation tool for pairwise testing is AETG (Automatic Efficient Test Generator)[1] [1], [2]. We used AETG[2] to produce pairwise test sets for the six systems mentioned in [2]. We also used PairTest to generate pairwise test sets for the same six systems. Table 1 shows the size information produced by AETG and PairTest for these six systems. As shown in Table 1, each of AETG and PairTest produces smaller test tests than the other for some systems. Later, we will show that PairTest has lower time complexity than AETG.

It was shown that, for a system with n parameters, each having d values, the size of a minimum pairwise test set grows at most logarithmically in n and quadratically in d [2]. Empirical results based on AETG indicates that when the number of candidate test cases for a new test case is 50, the number of test cases grows logarithmically in n [2]. We have carried out empirical studies to determine the growth function for the size of a pairwise test set generated by PairTest in terms of n and d. Table 2 shows the sizes of test sets generated by PairTest for systems with $d = 4$ and different values of n. Table 3 shows the sizes of test sets generated by PairTest for systems with $n = 10$ and different values of d. According to statistical analysis,[3] the values of s (number of tests)

1. AETG is a trademark of Telcordia Technologies Inc. and is covered by United States Patent 5,542,043.
2. Telcordia allows free use of AETG for two weeks over the Web. AETG does not provide the length of time used for test generation.

3. Curve fitting using the SAS package was performed on data in Tables 2 and 3.

TABLE 2
Results of PairTest for Systems with n 4-Value Parameters

n (# of parameters)	10	20	30	40	50	60	70	80	90	100
s (# of tests)	31	34	41	42	48	48	51	51	51	53

in Tables 2 and 3 grow in $O(\log(n)$ and $O(d^2)$, respectively. These empirical results match the theoretical results mentioned earlier.

Tables 2 and 3 also show time information for test sets generated by PairTest. The execution time information was collected when PairTest was compiled and run on a PC with Intel 450MHZ Pentium II processor, Windows 98, and JDK 1.2.2. According to statistical analysis, the values of t (time for test generation) in Tables 2 and 3 grow in $O(n^2 \log(n))$ and $O(d^3)$, respectively. Based on the observation that the size of a test set generated by PairTest is $O(d^2 \log(n))$, we have shown that the time complexity of the IPO strategy is $O(d^3 n^2 \log(n))$ [3]. Based on the same observation for AETG, we have also shown that the time complexity of the AETG heuristic algorithm in [2] is $O(d^4 n^2 \log(n))$ [3].

4 Conclusion

In this paper, we have presented the IPO test generation strategy for pairwise testing. Our empirical results indicate that the IPO strategy performs well according to the sizes of generated test sets and the amount of time taken for test generation. In addition, the IPO strategy can be easily adapted to reuse existing test sets when systems are modified due to changes of input parameters and/or values. More information on IPO-based test generation and our pairwise testing tool can be found in [3]. Pairwise testing (or 2-way testing) is a special case of n-way testing, which requires that for each set of n input parameters of a system, every combination of valid values of these n parameters be covered by at least one test case. The IPO strategy can be easily extended for n-way testing.

One problem with pairwise testing is that, if the domains of input parameters are large, the number of generated tests is huge. For a system with each parameter having d values, the number of tests required for pairwise testing is at least d^2. Thus, if each parameter has 1,000 values, at least 1 million tests are required for pairwise testing. To alleviate this test explosion problem, one solution is to divide each input domain into partitions, select one representative value from each partition, and generate tests according to representative values for input parameters. By controlling the number of partitions for each input parameter, we can determine the number of tests needed for pairwise testing.

Acknowledgments

The authors would like to thank Mr. Ho-Yen Chang for his effort on implementing the PairTest tool. They also wish to thank reviewers and Professor Richard Hamlet for their helpful comments. This research was supported in part by US National Science Foundation grants CCR-9320992 and CCR-9901004 and a grant from IBM in Research Triangle Park, North Carolina.

TABLE 3
Results of PairTest for Systems with 10 Parameters, Each Having d Values

d (# of values)	5	10	15	20	25	30
s (# of Tests)	47	169	361	618	956	1355

References

[1] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The Combinatorial Desgin Approach to Automatic Test Generation," *IEEE Software*, vol. 13, no. 5, pp. 83–89, Sept. 1996.

[2] D.M. Cohen, S.R. Dalal, M.L. Fredman, and G.C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," *IEEE Trans. Software Eng.*, vol. 23, no. 7, pp. 437–443, July 1997.

[3] Y. Lei and K.C. Tai, "In-Parameter-Order: A Test Generation Strategy for Pairwise Testing," Technical Report TR-2001-03, Dept. of Computer Science, North Carolina State Univ., Raleigh, North Carolina, Mar. 2001.

[4] R. Mandl, "Orthogonal Latin Squares: An Application of Experimental Design to Compiler Testing," *Comm. ACM*, vol. 28, no. 10, pp. 1054–1058, Oct. 1985.

[5] A.W. Williams and R.L. Probert, "A Practical Strategy for Testing Pair-Wise Coverage of Network interfaces," *Proc. IEEE Int'l Symp. Software Reliability Eng.*, pp. 246–254, 1996.

▹ **For more information on this or any computing topic, please visit our Digital Library at** http://computer.org/publications/dlib.