Automatic Test Generation for N-way Combinatorial Testing

Changhai Nie

changhainie@seu.edu.cn

Dept. of Computer Sci. & Eng. Southeast University

1 Introduction

- Automatic test generation: select a small number of tests to cover all test aspects to the greatest degree.
- The combinatorial design approach for testing has been studied and applied widely for its efficiency and effectiveness with a quite small test suite.

Combinatorial Testing

- R. Mandl defined the concept of pair-wise testing for the first time in 1985.
  - require that every combination of valid values of arbitrary two system parameters be covered by at least one test.
  - use the *Orthogonal Latin Squares* to generate tests in the Ada compiler testing.

Combinatorial Testing

- D. Cohen, et al. provided the AETG system for pair-wise, triple, or *n*-way combinations of a system's parameters.
- Y. Lei, et al. offered PAIRWISE TEST.
- We proposed an algorithm for pair-wise testing based on the NETWORK model.
- Toshiaki Shiba et al. also suggested two different algorithms GA and ACA for *n*-way testing.

Combinatorial Testing

- Because the problem of generating a minimum test suit for combinatorial testing is *NP-complete*[4], none of these methods can assure that the generated test suites are the best.
- In most cases, the sizes of the test suites generated by them are approximately equal.

Combinatorial Testing

- D.R. Kuhn et al. investigated the applicability of combinatorial testing. In their experiments:
  - about 20~40% software failures were caused by a certain value of one parameter,
  - about 20~40% software failures were caused by a certain combination of two parameters,
  - and about 20% software failures were caused by a certain combination of three parameters.

N-way Combinatorial Testing

- Pair-wise testing is fit for most general applications.
- In some safety-critical situations, testers may require N-way Combinatorial testing:
  - every combination of valid values of arbitrary *n* ($n \geq 2$) parameters be covered by at least one test.

2 Concepts of N-way Testing

- Suppose a system has *k* parameters, each of which has a suite of discrete values.
- In most conditions, it is too expensive to test all the combinations of all parameters.
- It is a more practical method to generate a small test suit to cover all the possibility to the greatest degree.

Pair-wise Testing

- **Definition 1** Suppose A is a $m \times k$ matrix, and column j represents parameter P_j, whose element is from finite symbol set T_j (j = 1, 2,…, k). If column i and column j are arbitrary two columns in A that satisfy: all the combinations of symbols between set T_i and set T_j appear in the ordered pairs formed by column i and column j, A is called a ***pair-wise test table***, which satisfies ***the pair-wise coverage criterion***. Every row of A is a piece of test, and m is the number of test cases.

N-way Testing

- **Definition 2** Suppose A is a $m \times k$ matrix, and column j represents parameter P_j, whose element is from finite symbol set $T_j(j = 1, 2, \ldots, k)$. If column $i_1, i_2, \ldots, i_n$ are arbitrary $n(2 \le n \le k)$ columns in A that satisfy: all the combinations of symbols between set T_1, set T_2,…, set T_n appear in the combinations formed by column i_1, column i_2,…, column i_n, A is called a ***n-way test table***, which satisfies ***the n-way coverage criterion***. Every row of A is a piece of test, and m is the number of test cases.

Theorem

- **Theorem 1** Suppose a system has k parameters $P_1, P_2, \ldots, P_k$, and every parameter P_i has $t_i (i = 1,2,\ldots,k)$ values, then there exists an n-way test table A whose rows satisfy: $r \geq \max(t_{i1} \times t_{i2} \times \ldots \times t_{in})$, $(1 \leq i_1 \neq i_2 \ldots \neq i_n \leq k)$.

Theorem

- **Theorem 2** Suppose a system has k parameters, and every parameter P_i has t_i ($i = 1,2,\ldots,k$) values, where $t_1 \ge t_2 \ge \ldots \ge t_k$, then the n-way test table A of the system has coverage p for the all combinations of arbitrary m $n \le m \le k$ parameters:

$$p \ge (t_1 \times t_2 \times \ldots \times t_n) / (t_1 \times t_2 \times \ldots \times t_m).$$

Dept. of Computer Sci. & Eng.
Southeast University

3 N-way Testing Algorithms

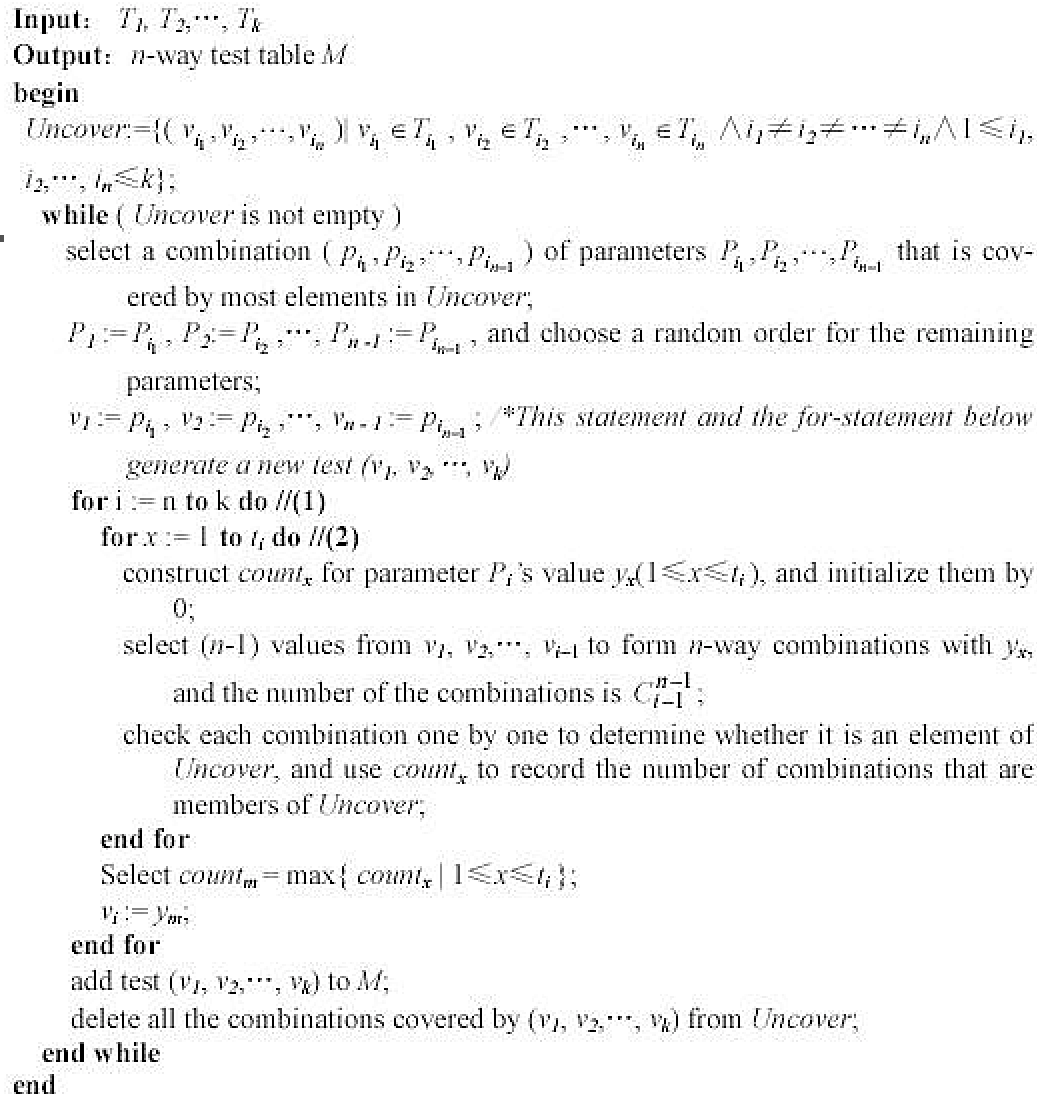
- We propose two different test generation algorithms based on the combinatorial design for the *n*-way testing.
  - Generate Tests using Greedy Algorithm (GA-N)
  - Generate Tests in Parameter Order (IPO-N)

3.1 Generate Tests using Greedy Algorithm (GA-N)

- This test generation method for *n*-way testing is extended from AETG for pair-wise testing.

3.1 Generate Tests using Greedy Algorithm (GA-N)

- Process (Greedy algorithm)
  - Firstly, construct a set *Uncover* that contains all the combinations of valid values of arbitrary *n* parameters.
  - Secondly, generate one test such that it can cover most combinations in *Uncover*. Add the test into the *n*-way test table *M*, and delete the combinations covered by it from *Uncover*.
  - Finally, generate tests continuously in the same way until *Uncover* is empty.



Input: $T_1, T_2, \cdots, T_k$
Output: n-way test table M
begin
$Uncover := \{(v_{i_1}, v_{i_2}, \cdots, v_{i_n}) \mid v_{i_1} \in T_{i_1}, v_{i_2} \in T_{i_2}, \cdots, v_{i_n} \in T_{i_n} \wedge i_1 \neq i_2 \neq \cdots \neq i_n \wedge 1 \leqslant i_1, i_2, \cdots, i_n \leqslant k\}$;
  while (*Uncover* is not empty)
    select a combination $(p_{i_1}, p_{i_2}, \cdots, p_{i_{n-1}})$ of parameters $P_{i_1}, P_{i_2}, \cdots, P_{i_{n-1}}$ that is covered by most elements in *Uncover*;
    $P_1 := P_{i_1}, P_2 := P_{i_2}, \cdots, P_{n-1} := P_{i_{n-1}}$, and choose a random order for the remaining parameters;
    $v_1 := p_{i_1}, v_2 := p_{i_2}, \cdots, v_{n-1} := p_{i_{n-1}}$; /**This statement and the for-statement below generate a new test* $(v_1, v_2, \cdots, v_k)$
    for i := n **to** k **do** //(1)
      for $x := 1$ **to** t_i **do** //(2)
        construct $count_x$ for parameter P_i's value $y_x (1 \leqslant x \leqslant t_i)$, and initialize them by 0;
        select $(n-1)$ values from $v_1, v_2, \cdots, v_{i-1}$ to form n-way combinations with y_x, and the number of the combinations is C_{i-1}^{n-1};
        check each combination one by one to determine whether it is an element of *Uncover*, and use $count_x$ to record the number of combinations that are members of *Uncover*;
      end for
      Select $count_m = \max\{ count_x \mid 1 \leqslant x \leqslant t_i \}$;
      $v_i := y_m$;
    end for
    add test $(v_1, v_2, \cdots, v_k)$ to M;
    delete all the combinations covered by $(v_1, v_2, \cdots, v_k)$ from *Uncover*;
  end while
end

Fig. 1. Greedy algorithm for n-way test table generation (GA-N)

3.2 Generate Tests in Parameter Order (IPO-N)

- This test generation method for *n*-way testing is extended from PAIRTEST for pair-wise testing

3.2 Generate Tests in Parameter Order (IPO-N)

- Process
  - Firstly, construct n-way test table M for the first n parameters.
  - Secondly, extend M by another parameter to assure that every combination of valid values of n parameters in M is covered by at least one test.
    - (1) horizontal growth: extend each existing test by one value of the new parameter,
    - (2) vertical growth: add new tests to satisfy the n-way coverage criterion.
  - Extend M continuously, until it contains all the parameters.

Input: $T_1, T_2, \cdots, T_k$

Output: n-way test table M

begin

  //Construct M for the first n parameters

  $M := \{(v_1, v_2, \cdots, v_n) \mid v_1 \in T_1, v_2 \in T_2, \cdots, v_n \in T_n\}$;

  for $i := n+1$ **to** k **do** //(1) *Extend M by other parameters*

    $\pi := \{(p_{m_1}, p_{m_2}, \cdots, p_{m_{n-1}}, p_{m_n}) \mid p_{m_1} \in T_{m_1}, p_{m_2} \in T_{m_2}, \cdots, p_{m_{n-1}} \in T_{m_{n-1}} \wedge m_1 \neq m_2 \neq \cdots \neq m_{n-1} \wedge 1 \leqslant m_1, m_2, \cdots, m_{n-1} \leqslant i-1 \wedge p_{m_n} \in T_i\}$;

    for $j := 1$ **to** $|M|$ **do** //(2) *Horizontal growth*

      if (π is not empty) **then**

        $\pi' := \{\}$;

        for each v **in** T_i **do** //(3)

          $\pi'' := \{z \mid z \in \pi \wedge z$ is covered by the extended test $(v_1, v_2, \cdots, v_{i-1}, v_i)\}$;

            //(4)

          if ($|\pi''| \geqslant |\pi'|$) **then**

            $\pi' := \pi''$; $v' := v$;

          end if

        end for

        extend the j-th test in M by v': $(v_1, v_2, \cdots, v_{i-1})$ is extended to $(v_1, v_2, \cdots, v_{i-1}, v')$;

        $\pi := \pi - \pi'$; //(5)

      else

        select $v \in T_i$ arbitrarily, and extend the j-th test in M by v;

      end if

    end for

    while (π is not empty) **do** //(6) *Vertical growth*

      generate one test t such that it can cover most combinations in π;

      add t into M, and delete all the combinations covered by t from π;

    end while

  end for

end

Fig. 2. N-way test table generation In Parameter Order (IPO-N)

4 Effectiveness Analysis

- To evaluate the efficiency of the algorithms, we implemented two automatic test generators based on them.
- We experimented with a computer consisting of a PIII processor of 733Mhz, and obtained some empirical results.

Empirical Data: IPO-N & GA-N

Table 1. Comparison between GA-N and IPO-N(n=2 or 3)

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
GA-N	29	58	12	50	264	31	56	66
(n=2)	1″	1″	1″	10″	6″	1″	4″	7″
IPO-N	22	55	10	21	275	33	47	36
(n=2)	1″	1″	1″	3″	1″	1″	2″	2″
GA-N	103	375	22	305	5424	172	390	418
(n=3)	3″	19″	1″	2539″	17767″	2″	1534″	1638″
IPO-N	86	288	19	128	2774	153	244	216
(n=3)	1″	1″	1″	859″	904″	1″	296″	592″

S_1: $3^{(13)}$; S_2: $5^{(10)}$; S_3: $2^{(10)}$; S_4: $2^{(100)}$; S_5:$10^{(20)}$;

S_6: $5^{(3)} \times 4^{(3)} \times 3^{(1)} \times 2^{(2)}$; S_7: $4^{(15)} \times 3^{(17)} \times 2^{(29)}$; S_8: $4^{(1)} \times 3^{(39)} \times 2^{(35)}$;

IPO-N Exceeds GA-N

- When n equals 2, the tests generated by GA-N are usually a few more than IPO-N, and the consumed time is also a little more, and the difference is not distinct.
- When n equals 3, the tests generated by GA-N are much more than IPO-N, and the consumed time is also more in evidence.
- Some other experiment results also show that IPO-N exceeds GA-N at the test number and the consumed time, when n is above 3.

Empirical Data: Related Methods

Table 2. Comparison with existing algorithms (n=3)

	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
AETG	38	77	194	330	1473	218	114	377
GA	33	64	125	331	1501	218	108	360
ACA	33	64	125	330	1496	218	106	361
GA-N	52	85	223	389	1769	366	120	373
IPO-N	47	64	173	271	1102	199	113	368

X_1: $3^{(6)}$; X_2: $4^{(6)}$; X_3: $5^{(6)}$; X_4: $6^{(6)}$; X_5:$10^{(6)}$; X6: $5^{(7)}$;
X_7: $5^{(2)}\times 4^{(2)}\times 3^{(2)}$; X_8: $10^{(1)}\times 6^{(2)}\times 4^{(3)}\times 3^{(1)}$.

IPO-N Performs Well

- The experimental results also suggest that GA-N need to be optimized carefully.
- IPO-N can generate the smallest test suites for some systems such as X_2, X_4, X_5 and X_6, and in other cases the sizes of test suites generated by IPO-N are only a bit more than the smallest ones.
- These results seem to indicate that IPO-N is an effective algorithm, which performs well with respect to the size of generated test suites.

5 Conclusions

- We propose two different test generation algorithms based on combinatorial design approaches for the *n*-way combination testing, and suggest the algorithm IPO-N may be more feasible based on the empirical results.

Dept. of Computer Sci. & Eng.
Southeast University

- As shown in Table 1, with the increase of n, the number of tests increases rapidly, and the time consumed for the test generation grows sharply.
- We are searching novel algorithms for the n-way combination testing to improve the performance of test generators.

Changhai Nie

changhainie@seu.edu.cn

Dept. of Computer Sci. & Eng. Southeast University