A Test Generation Strategy for Pairwise Testing

Yu Lei and K. C. Tai
Department of Computer Science
North Carolina State University
Raleigh, NC 27695-7534, USA

{ylei2,kct}@eos.ncsu.edu

February 14, 2001

Abstract

Pairwise testing (or 2-way 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. Empirical results show that pairwise testing is effective for various types of software systems. In this paper, we propose a test generation strategy, called in-parameter-order (or IPO), for pairwise testing. 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 IPO strategy allows the use of local optimization techniques for test generation and the reuse of existing tests when a system is extended with new parameters or new values of existing parameters. We present practical, IPO-based test generation algorithms. We describe the implementation of an IPO-based test generation tool and show some empirical results.

1 Introduction

Pairwise testing (or 2-way 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. Empirical results show that pairwise testing is practical and effective for various types of software systems [1] [2] [3] [5] [6].

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 have been published for pairwise testing. The strategy proposed in [1] starts with an empty test set and adds one test at a time. To generate a new test, the strategy produces a number of possible 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 [8]. 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 [9]. 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. 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, after the completion of horizontal growth.

The remainder of this paper is organized as follows. Section 2 describes the IPO strategy and its advantages and also shows an optimal algorithm for vertical growth, which can be used in conjunction with any algorithm for horizontal growth. Section 3 presents two algorithms for horizontal growth. Section 4 describes an IPO-based test generation tool and shows some empirical results. Section 5 concludes this paper. This paper is an extended version of our earlier paper published in [7].

2 In-Parameter-Order (IPO) Strategy for Generating Pairwise Test Sets

In this section, we describe a test generation strategy, called in-parameter-order (or IPO), for pairwise testing. 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 a test set for the addition of a new parameter includes 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, after the completion of horizontal growth. Section 2.1 describes the IPO strategy and its advantages. Section 2.2 shows an optimal algorithm for vertical growth, which can be used in conjunction with any algorithm for horizontal growth. Section 3 presents two algorithms for horizontal growth.

2.1 Framework of the IPO Strategy

Assume that system $\mathcal{S}$ has parameters $p_1, p_2, \ldots$, and p_n, $n \geq 2$. Below is a general description of the IPO strategy for generating a pairwise test set $\mathcal{T}$ for $\mathcal{S}$.

Strategy *In-Parameter-Order*
begin
  /* for the first two parameters p_1 and p_2 */
  $\mathcal{T} := \{(v_1, v_2) \mid v_1$ and v_2 are values of p_1 and p_2 respectively$\}$
  if $n = 2$ **then** stop;
  /* for the remaining parameters */
  for parameter $p_i, i = 3, 4, \ldots, n$ **do**
  begin
    /* horizontal growth */
    for each test $(v_1, v_2, \ldots, v_{i-1})$ in $\mathcal{T}$ **do**
      replace it with $(v_1, v_2, \ldots, v_{i-1}, v_i)$, where v_i is a value of p_i;
    /* vertical growth */
    while $\mathcal{T}$ does not cover all pairs between p_i
    and each of $p_1, p_2, \ldots, p_{i-1}$ **do**
      add a new test for $p_1, p_2, \ldots, p_i$ to $\mathcal{T}$;
  end
end

The IPO test generation strategy for pairwise testing provides the following advantages:

- The IPO strategy allows the use of "local" optimization techniques for horizontal and vertical growth in order to develop practical test generation algorithms. (We will show how to do so in sections 2.2 and 3.)
- Assume that $\mathcal{T}$ is a pairwise test set for a system $\mathcal{S}$. Suppose that a new version of $\mathcal{S}$ has one or more new parameters. By applying the IPO strategy, we can easily extend $\mathcal{T}$ to produce a new pairwise test set $\mathcal{T}'$ for the new version of $\mathcal{S}$. Furthermore, since $\mathcal{T}'$ reuses tests in $\mathcal{T}$, the effort for test preparation according to $\mathcal{T}'$ is less than that according to a new pairwise test set without reusing tests in $\mathcal{T}$.
- Assume that $\mathcal{T}$ is a pairwise test set for a system $\mathcal{S}$. Suppose that $\mathcal{S}$ has a new version in which some parameters have new values. By applying the vertical growth in the IPO strategy, we can easily extend $\mathcal{T}$ by adding new tests to produce a new pairwise test set $\mathcal{T}'$ for the new version of $\mathcal{S}$. $\mathcal{T}'$ may be larger than a pairwise test set generated by applying the IPO strategy without reusing $\mathcal{T}$. However, $\mathcal{T}'$ can save effort for test preparation due to the reuse of tests in $\mathcal{T}$.

2.2 An Optimal Algorithm for Vertical Growth

We first use the example system in Section 1 to illustrate our algorithm for vertical growth. As mentioned earlier, $\{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)\}$ is a pairwise test set for parameter A and B. Assume that the horizontal growth for parameter C produces the following tests:

(A_1, B_1, C_1), (A_1, B_2, C_1), (A_2, B_1, C_2), and (A_2, B_2, C_2). These four tests do not cover pairs $(A_1, C_2), (A_1, C_3), (A_2, C_1), (A_2, C_3), (B_1, C_3)$ and (B_2, C_3). So we need to generate new tests to cover these pairs in vertical growth. (A_1, C_2) is covered by test $(A_1, -, C_2)$, where "$-$" denotes *don't care*, meaning that the value does not need to be specified for covering any pair. Similarly, (A_1, C_3), (A_2, C_1), and (A_2, C_3) are covered by $(A_1, -, C_3)$, $(A_2, -, C_1)$ and $(A_2, -, C_3)$ respectively. To cover (B_1, C_3), we change $(A_1, -, C_3)$ to (A_1, B_1, C_3) without adding a new test. Similarly, we cover (B_2, C_3) by changing $(A_2, -, C_3)$ to (A_2, B_2, C_3). Thus, we generate four new tests to cover the six missing pairs. (Note that it is impossible to use less than four tests to cover the six missing pairs.) So the generated pairwise test set has a total of eight tests.

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. (Thus, $\mathcal{T}$ is resulted from the extension of a pairwise test set for $p_1, p_2, \ldots$ and p_{i-1}.) Let π be the set of missing pairs according to $\mathcal{T}$. Each pair in π contains a value of p_i and a value of one of $p_1, p_2, \ldots, p_{i-1}$. Assume that $|\pi| > 0$. Let d_j, $1 \le j \le i$, be the domain size of parameter p_j and $m_j = max\{d_k | 1 \le k \le j\}$. The following algorithm for vertical growth shows how to construct a minimum test set $\mathcal{T}'$ such that $\mathcal{T} \cup \mathcal{T}'$ is a pairwise test set for $p_1, p_2, \ldots$ and p_i.

Algorithm *IPO_V*$(\mathcal{T}, \pi)$
/* π is the set of missing pairs according to a test set $\mathcal{T}$ for $p_1, p_2, \ldots,$ and p_i. This algorithm constructs a minimum test set $\mathcal{T}'$ such that $\mathcal{T} \cup \mathcal{T}'$ is a pairwise test set for p_1, $p_2, \ldots,$ and p_i. */
begin
    let τ be a two-dimension $(d_i \times m_i)$ array of tests, and let τ initialized with
    tests that have all the values as "$-$";
    let ι be an array of integers of size d_i initialized with 0;
    for each pair $(p_k.w, p_i.u)$ in π **do**
    begin
        let u be the jth value of p_i;
        for $l = 1$ **to** $\iota[j]$ **do**;
        begin
            if $\tau[j][l]$ has "$-$" as the value of p_k and "u" as the value of p_i
            then
            begin
                modify $\tau[j][l]$ by replacing the "$-$" with w;
                break;
            end
        end
        if $l > \iota[j]$ **then**
        begin
            $\iota[j]++$;
            modify $\tau[j][\iota[j]]$ such that w is the value of p_k and u is the value of p_i;
        end
    end;
    let $\mathcal{T}'$ be an empty set;
    for $r = 0$ **to** d_i **do**
    begin
        for $s = 0$ **to** $\iota[r]$ **do**

begin
add $\tau[r][s]$ into $\mathcal{T}'$;
end
end
return $\mathcal{T}'$;
end;

Assume that p_i contains values $u_1, u_2, \ldots$ and u_q. For $1 \leq j \leq q$, let π_j be the set of pairs in π that contain u_j as the value of p_i. If $|\pi_j| > 0$, let π_{jm}, $1 \leq m \leq i-1$, be the set of pairs of π_j that contain a value of p_m. The number of new tests according to π_j is $max\{|\pi_{jm}||m = 1, 2, \ldots, i-1\}$. The set of new tests according to π_j is minimum, since it is impossible to cover pairs in π_j with less number of tests. Furthermore, the set of new tests according to π_j cannot cover pairs that are in π, but not in π_j. Thus, the above algorithm constructs a minimum set of new tests according to π. Therefore, we have the following theorem.

Theorem 2.1 *Let S be a system with parameters $p_1, p_2, \ldots,$ and p_i. Assume that according to the IPO strategy, the vertical 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 missing pairs according to $\mathcal{T}$. Algorithm IPO_V constructs a minimum set $\mathcal{T}'$ of new tests according to π such that $\mathcal{T} \cup \mathcal{T}'$ is a pairwise test set for S.*

The time complexity of algorithm IPO_V is dominated by the first *for* loop, which has a nested loop. The size of π is at most $d_i * \sum_{j=1}^{i-1} d_j$, which is less than or equal to $m_i^2 * (i-1)$. For each pair $(p_k.u, p_i.v)$ in π, the nested loop searches the tests with v as the value of p_i to determine if there exists one with "$-$" as the value of p_k among those tests. There are at most m_i tests with v as the value of p_i, each of which can be checked in $O(1)$. So the nested loop takes $O(m_i)$. The time complexity of algorithm IPO_V is therefore $O(m_i^3 * i)$. For a system with n parameters, algorithm IPO_V is executed for $i = 3$ to n. Let $d = max\{d_i | 1 \leq i \leq n\}$. Thus, the total execution time for algorithm IPO_V is $O(d^3 * n^2)$.

Theorem 2.2 *Let S be a system with n parameters and let d be the maximum domain size of these parameters. According to the IPO strategy, the size of the pairwise test set generated for S is at most $d^2 * (n-1)$.*

Proof For $n = 2$, the size of the generated pairwise test set is at most d^2. For $n = 3$, horizontal growth does not increase the number of tests generated for $n = 2$, and vertical growth adds at most d^2 tests. So the size of the generated test set for $n = 3$ is at most $d^2 * 2$. By using the same argument, the size of the generated pairwise test set for any system with n parameters is at most $d^2 * (n-1)$. □

In practice, as shown in Section 4, the sizes of test sets generated by PairTest are $O(d^2 \times \log(n))$.

3 Two Algorithms for Horizontal Growth

Assume that $\mathcal{T}$ is a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{i-1}. As mentioned earlier, the horizontal growth for parameter p_i is to replace each test $(v_1, v_2, \ldots, v_{i-1})$ in $\mathcal{T}$ with

$(v_1, v_2, \ldots, v_{i-1}, v_i)$, where v_i is a value of p_i. Thus, the horizontal growth for p_i is to construct a list of $|\mathcal{T}|$ values of parameter p_i such that the jth value of the list, $1 \leq j \leq |\mathcal{T}|$, is the value added to the jth test in $\mathcal{T}$. Such a list is referred to as a permutation of parameter p_i for $\mathcal{T}$. There are $|\mathcal{D}(p_i)|^{|\mathcal{T}|}$ possible permutations of parameter p_i for $\mathcal{T}$. For a permutation ζ of parameter p_i for $\mathcal{T}$, let $(\mathcal{T}, \zeta)$ denote the test set resulted from adding the jth element of ζ, $1 \leq j \leq |\mathcal{T}|$, to the jth test in $\mathcal{T}$.

Ideally, we hope to find a permutation ζ of parameter p_i for $\mathcal{T}$ such that $(\mathcal{T}, \zeta)$ is a minimum pairwise test set for $p_1, p_2, \ldots$, and p_i. If such a permutation does not exist, we need to choose a permutation ζ of parameter p_i for $\mathcal{T}$ such that the set of new tests constructed by algorithm IPO_V (shown in Section 2.2) according to the set of missing pairs based on $(\mathcal{T}, \zeta)$ is minimum. Since there are $|\mathcal{D}(p_i)|^{|\mathcal{T}|}$ possible permutations of parameter p_i for $\mathcal{T}$, finding an "optimal" permutation by exhaustive search is very time-consuming. In section 4.1, we show an algorithm for horizontal growth, called IPO_H_EC ("EC" stands for "equivalence classes"). This algorithm finds an optimal permutation of parameter p_i for $\mathcal{T}$ without using exhaustive search. However, algorithm IPO_H_EC still has exponential time complexity. In section 4.2, we show algorithm IPO_H_IV ("IV" stands for "individual values"), which is practical to use.

3.1 Algorithm IPO_H_EC

Definition 1 *Let $\mathcal{T}$ be a pairwise test set for parameters $p_1, p_2, \ldots$, and p_{i-1}. For a permutation ζ of parameter p_i for $\mathcal{T}$, let the weight of ζ with respect to $\mathcal{T}$, or $weight(\zeta, \mathcal{T})$, be defined as the number of new tests constructed by algorithm IPO_V according to the set of missing pairs based on $(\mathcal{T}, \zeta)$.*

The motivation for algorithm for IPO_H_EC is that the set of permutations of parameter p_i with size $|\mathcal{T}|$ can be divided into equivalent classes such that permutations in the same class have the same weight with respect to $\mathcal{T}$. Thus, finding an optimal permutation of parameter p_i for $\mathcal{T}$ requires only the use of one permutation from each equivalent class. Below we show several definitions and theorems that are necessary for understanding algorithm IPO_H_EC. Proofs for the theorems in this section are shown in the appendix.

Definition 2 *Let ζ and ξ be two permutations of a parameter α such that $\zeta = (u_1, u_2, \ldots, u_n)$ and $\xi = (v_1, v_2, \ldots, v_n)$. An* onto mapping *from ζ to ξ is a 1-to-1 function ψ from ζ to ξ such that $v_i = \psi(u_i)$.*

Definition 3 *Two different permutations ζ and ξ of a parameter are said to be equivalent, noted as $\zeta \equiv \xi$, provided that there exists an onto mapping from ζ to ξ or vice versa.*

For the example system in Section 1, consider the following two permutations of parameter C: $C' = (C_1, C_1, C_2, C_3)$ and $C'' = (C_2, C_2, C_1, C_3)$. According to the above definitions, the two permutations are equivalent due to the existence of the following 1-to-1 function ψ from C' to C'':

$$\psi(u) = \begin{cases} C_2 & if\ u = C_1 \\ C_1 & if\ u = C_2 \\ C_3 & if\ u = C_3 \end{cases}$$

Theorem 3.1 *Let $\mathcal{T}$ be a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{k-1}. For two permutations ζ and ξ of parameter p_i with respect to $\mathcal{T}$, if ζ and ξ are equivalent, then $weight(\zeta, \mathcal{T}) = weight(\xi, \mathcal{T})$.*

To illustrate the above theorem, consider again the example system in Section 1. $\mathcal{T} = \{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2)\}$ is a pairwise test set for parameters A and B. As shown earlier, $C' = (C_1, C_1, C_2, C_3)$ and $C'' = (C_2, C_2, C_1, C_3)$ are equivalent. To calculate $weight(C', \mathcal{T})$, we construct $(\mathcal{T}, C')$, which is $\{(A_1, B_1, C_1), (A_1, B_2, C_1), (A_2, B_1, C_2), (A_2, B_2, C_3)\}$. The set of missing pairs is $\{(A_2, C_1), (A_1, C_2), (B_2, C_2), (A_1, C_3), (B_1, C_3)\}$, which is covered by the new test set $\mathcal{T}' = \{(A_2, -, C_1), (A_1, B_2, C_2), (A_1, B_1, C_3)\}$. Thus, $weight(C', \mathcal{T}) = 3$. To calculate $weight(C'', \mathcal{T})$, we construct $(\mathcal{T}, C'')$, which is $\{(A_1, B_1, C_2), (A_1, B_2, C_2), (A_2, B_1, C_1), (A_2, B_2, C_3)\}$. The set of missing pairs is $\{(A_2, C_2), (A_1, C_1), (B_2, C_1), (A_1, C_3), (B_1, C_3)\}$, which is covered by the new test set $\mathcal{T}'' = \{(A_2, -, C_2), (A_1, B_2, C_1), (A_1, B_1, C_3)\}$. Thus, $weight(C'', \mathcal{T}) = 3$. Note that although $\mathcal{T}' \neq \mathcal{T}''$, $|\mathcal{T}'| = |\mathcal{T}''| = 3$.

By Theorem 3.1, finding an optimal permutation of parameter p_i for $\mathcal{T}$ requires only the use of one permutation from each equivalent class of permutations of size $|\mathcal{T}|$ of p_i. Thus, the number of permutations considered for choosing an optimal permutation can be significantly reduced. The following theorem shows how to determine the number of equivalent classes of permutations of size $|\mathcal{T}|$ of p_i .

Theorem 3.2 *Assume that parameter α has domain $\mathcal{D}$, where $|\mathcal{D}| = d$. Let $\mathcal{M}$ be the set of all α's permutations with size m. Let $\mathcal{N}(m, d)$ be the number of equivalence classes in $\mathcal{M}$.*

$$\mathcal{N}(m,d) = \begin{cases} 1 & \text{if } d=1,\ m=1 \text{ or } 0 \\ \sum_{i=1}^{m} (C_{m-1}^{i-1} * \mathcal{N}(m-i, d-1)) & \text{otherwise} \end{cases}$$

As an example, for a parameter having 4 values, the number of different permutations of size 10 is $4^{10} = 1048576$, but the number of equivalence classes of these permutations is $\mathcal{N}(10, 4) = 43947$. Since $\mathcal{N}(m, d) \geq 2^{m-1}$, $\mathcal{N}(m, d)$ is an exponential function of m. For a given parameter and permutation size, the following theorem shows how to construct a set of permutations such that this set contains exactly one permutation from each equivalence class.

Theorem 3.3 *Assume that parameter α has domain $\mathcal{D}$, where $|\mathcal{D}| = d$. Let $\mathcal{D} = \{v_1, v_2, \ldots, v_d\}$ and $\mathcal{D}_i = \{v_{d-i+1}, v_{d-i+2}, \ldots, v_d\}$, where $1 \leq i \leq d$. Let $\mathcal{E}(m, \mathcal{D})$ be defined as follows:*

$$\mathcal{E}(m,\mathcal{D}) = \begin{cases} \{\} & \text{if } m = 0 \\ \{(v_1)\} & \text{if } m = 1 \\ \{\overbrace{(v_1, v_1, \ldots, v_1)}^{m}\} & \text{if } |\mathcal{D}| = 1 \\ \cup_{i=1}^{m} ([\mathcal{B}]_{v_1}^{(m,i)}|_{b_1=v_1} * \mathcal{E}(m-i, \mathcal{D}_{d-1})) & \text{otherwise} \end{cases}$$

$\mathcal{E}(m, \mathcal{D})$ contains exactly one permutation of α with size m from each equivalence class of permutations of α with size m. (Note that $[\mathcal{B}]_{v_1}^{(m,i)}|_{b_1=v_1}$ and "" operator are not defined here since their definitions are complicated. Please see the appendix for their definitions.)*

For the example system in Section 1, since the only pairwise test set for parameters A and B has four tests, we need to consider permutations of size 4 of parameter C. There are a total of 81 (3^4)

such permutations, but they can be divided into 14 equivalent classes according to Theorem 3.2. Below are 14 permutations of size 4 of parameter C, one from each equivalent class: $\{(C_1, C_1, C_1, C_1), (C_1, C_1, C_1, C_2), (C_1, C_1, C_2, C_1), (C_1, C_2, C_1, C_1), (C_1, C_1, C_2, C_2), (C_1, C_1, C_2, C_3), (C_1, C_2, C_1, C_2), (C_1, C_2, C_1, C_3), (C_1, C_2, C_2, C_1), (C_1, C_2, C_3, C_1), (C_1, C_2, C_2, C_2), (C_1, C_2, C_2, C_3), (C_1, C_2, C_3, C_2), (C_1, C_2, C_3, C_3)\}$. We choose one of these 14 permutations that has the minimum weight, and this permutation has the minimum weight among all possible 81 permutations. Note that for the example in Section 2.2, the horizontal growth for parameter C uses the permutation (C_1, C_1, C_2, C_2), which has 4 as its weight. It was shown earlier that the weight for (C_1, C_1, C_2, C_3) is 3. Later we will show that the weight for (C_1, C_2, C_3, C_1) is 2, which is the minimum weight.

Below we show algorithm IPO_H_EC for horizontal growth. $Partition(p, m)$ returns a set of permutations of parameter p_i with size m according to Theorem 3.3. The main loop of IPO_H_EC finds a permutation with the minimum weight, which is used to perform the horizontal growth.

Algorithm *IPO_H_EC*($\mathcal{T}$, p_i)
/* $\mathcal{T}$ is a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{i-1}. This algorithm selects an optimal permutation ζ of p_i of size $|\mathcal{T}|$ and replaces $\mathcal{T}$ with $(\mathcal{T}, \zeta)$. */
begin
    /* Initialization */
    $\zeta = null$;
    $\omega = \infty$;
    /* Main Loop */
    $\mathcal{M} := Partition(p_i, |\mathcal{T}|)$;
    for each permutation ζ' in $\mathcal{M}$ **do**
    begin
        $\omega' := weight(\zeta', \mathcal{T})$;
        if $\omega' \leq \omega$ **then**
            $\zeta = \zeta'$;
            $\omega = \omega'$;
    end
    replace $\mathcal{T}$ with $(\mathcal{T}, \zeta)$;
end

function *Partition*(p, m)
begin
  let $\mathcal{D} = \{v_1, v_2, \ldots, v_{|\mathcal{D}|}\}$ be the domain of p;
  for $i = 1$ **to** m
    $\mathcal{E}(i, \mathcal{D}_1) = \{(v_1)\}$
  for $i = 2$ **to** $|\mathcal{D}|$
    for $j = 1$ **to** m
      $\mathcal{E}(i, \mathcal{D}_j) = \{\}$;
      for $k = 1$ **to** j
        $\mathcal{E}(i, \mathcal{D}_j) = \mathcal{E}(i, \mathcal{D}_j) \cup ([\mathcal{B}]_{v_i}^{(j,k)}|_{b_1 = v_i} * \mathcal{E}(j-k, \mathcal{D}_{j-1}))$;
end

3.2 Algorithm IPO_H_IV

Since algorithm IPO_H_EC has exponential time complexity, it is not practical for systems with a large number of parameters. In this section, we show a horizontal growth algorithm that is practical to use, but may produce more tests than algorithm IPO_H_EC. Assume that $\mathcal{T}$ is a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{i-1}. Initially, the set π of missing pairs contains all possible pairs between values of p_i and $p_1, p_2, \ldots,$ and p_{i-1}. For each test in $\mathcal{T}$, we consider all values of p_i, choose one that covers the most number of pairs in π, extend the test by adding the chosen value of p_i, and remove from π pairs covered by the extended test. Thus, we incrementally construct a permutation of p_i with size $|\mathcal{T}|$ by choosing one value of p_i for each test in T.

Assume that the domain of p_i contains values $v_1, v_2, \ldots,$ and v_q. If $|\mathcal{T}| \leq q$, we can simply choose v_j, $1 \leq j \leq |\mathcal{T}|$, as the new value for the jth test in $\mathcal{T}$, since the resulting test covers exactly $i-1$ missing pairs. (In fact, we can choose any permutation of p_i of size $|\mathcal{T}|$ such that all elements of this permutation are distinct.) If $|\mathcal{T}| > q$, for $1 \leq j \leq q$, we choose v_j as the new value for the jth test in T, and for $j > q$, we consider all values of p_i and choose one that covers the most number of missing pairs.

Now we apply the above discussion 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 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. As a result, we have four missing pairs: (A_2, C_2), (B_1, C_2), (A_1, C_3), and (B_2, C_3). By applying algorithm IPO_V, we generate tests (A_2, B_1, C_2) and (A_1, B_2, C_3) to cover these missing pairs.

Below we show the complete algorithm, which is called IPO_H_IV, where "IV" stands for "individual values". For $1 \leq j \leq i$, let d_j be the domain size of parameter p_j and $m_j = max\{d_k | 1 \leq k \leq j\}$.

Algorithm *IPO_H_IV*($\mathcal{T}$, p_i)
/* $\mathcal{T}$ is a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{i-1}. This algorithm modifies $\mathcal{T}$ by adding a value of p_i to each test in $\mathcal{T}$ */.
begin
  /* initialization */
  let τ be a three-dimension $(d_i \times (i-1) \times m_i)$ array of booleans initialized with **false**
  let ι be a two-dimension $(d_i \times (i-1))$ array of integers initialized with 0
  $s = min(|\mathcal{D}(p_i)|, |\mathcal{T}|)$;
  /* no need to select values of p_i */
  for $j = 1$ **to** s **do**
  begin
    extend the jth test in $\mathcal{T}$ by adding the jth value of p_i;
    mark($\mathcal{T}$, p_i, j, τ, ι);
  end;
  if $s = |\mathcal{T}|$ then **return**;

/* necessary to select values of p_i */
for $j = s + 1$ to $|\mathcal{T}|$ **do**
begin
  $w' := 0$;
  for each value v in p_i **do**
  begin
    $w = \text{eval}(\mathcal{T}, p_i, j, \tau, \iota)$;
    if $w' < w$
    begin $w' := w; v' := v$; **end**;
  end;
  extend the jth test in $\mathcal{T}$ by adding value v';
  $\text{mark}(\mathcal{T}, p_i, j, \tau, \iota)$;
end;
end;

function $mark(\mathcal{T}, p_i, j, \tau, \iota)$
begin
  let τ be the jth test in $\mathcal{T}$;
  let v be the value of p_i in τ, and let v be the kth value of p_i;
  for $l = 1$ **to** $i - 1$ **do**
  begin
    let w be the value of p_l in τ, and let w be the nth value of p_l;
    if $w \neq$ "$-$" **then** $\tau[k][l][n] = true$;
    else
    begin
      $q = \iota[k][l]$;
      if $q \neq -1$
      begin
        replace w as the qth value of p_l;
        for $t = q + 1$ **to** d_l **do**
        begin
          if $!\tau[k][l][t]$ **break**;
        end
        if $t \leq d_l$ **then** update $\iota[k][l]$ with the tth value of p_l;
        else $\iota[k][l] := -1$;
      end
    end
  end
end

function $eval(\mathcal{T}, p_i, j, \tau, \iota)$
begin
  $r = 0$;
  let τ be the jth test in $\mathcal{T}$;
  let v be the value of p_i in τ, and let v be the kth value of p_i;
  for $l = 1$ **to** $i - 1$ **do**

```
  begin
    let w be the value of p_l in τ, and let w be the nth value of p_l;
    if w ≠ "−"
    begin
      if τ[k][l][n] = false then r++;
    end
    else
    begin
      if ι[k][l] ≠ −1 then r++;
    end
  end
  return r;
end
```

In the above algorithm, a coverage matrix τ and a pointer matrix ι are maintained for the purpose of efficiency. $\mathcal{C}$ is used to determine whether a pair is covered or not in $O(1)$. Given a pair $(p_k.u, p_i.v)$, where assume that u is the ith value of parameter p_k and v the jth value of parameter p_i, $(p_k.u, p_i.v)$ is covered if and only if $\tau[j][k][i] = true$. $\mathcal{P}$ is used to deal with "−" values. $\iota[j][k] = l$ means that the pair $(p_k.u, p_i.v)$, where u is the lth value of parameter p_k and v the jth value of parameter p_i, has not been covered yet. Therefore, a "−" value of parameter p_k can be replaced with the lth value of parameter p_k to cover a new pair.

Now we consider the time complexity of algorithm IPO_H_IV. The *mark* function is used to maintain τ and ι whenever a test is extended by adding one value of p_i. For each pair, *mark* takes $O(1)$ to update τ, and $O(m_i)$ to update ι. Considering that an extended test has $i-1$ pairs, *mark* has the time complexity of $O(m_i \times i)$. The *eval* function determines the number of new pairs covered by a test if the test were extended by a given value of p_i. It takes $O(1)$ for *eval* to check one pair, and $O(i)$ for all the $i-1$ pairs covered by a test. The time complexity of algorithm IPO_H_IV is dominated by the second *for* loop, which has a nested *for* loop. The nested loop takes $O(i \times m_i)$, since d_i values of p_i are evaluated, each of which takes $O(1) toevaluate$. The invocation of *mark* takes $O(i \times m_i)$ as mentioned earlier. Assume that the sizes of test sets generated by IPO are $O(m_i^2 \times \log(i))$, the time complexity of algorithm IPO_H_IV is $O(m_i^2 \times \log(i) \times m_i \times i) = O(m_i^3 \times \log(i) \times i)$. For a system with n parameters, algorithm IPO_H_IV is executed for $i = 3$ to n. Let $d = max\{d_i | 1 \leq i \leq n\}$. Thus, the total execution time for algorithm IPO_H_IV is $O(d^3 * n^2 * \log(n))$.

4 PairTest: An IPO-based Test Generation Tool

We have implemented an IPO-based test generation tool, called PairTest, that includes algorithm IPO_H_IV for horizontal growth and algorithm IPO_V for vertical growth. We have also implemented algorithm IPO_H_EC, but does not include it in PairTest due to its exponential time complexity.

Major features of PairTest include the following:

- PairTest supports the generation of pairwise test sets for systems with or without existing test sets and for systems modified due to changes of input parameters and/or values.

- PairTest provides information for planning the effort of testing and the order of applying test cases.

- PairTest provides a graphical user interface (GUI) to make the tool easy to use.

- PairTest was written in Java and thus can run on different platforms.

PairTest supports the following types of test generation:

- For a system without an existing test set, PairTest generates a pairwise test set.

- For a system with an existing test set, PairTest allows two options. One is to generate a new pairwise test set, without reusing the existing test set. The other is to keep the existing test set and generate additional tests, if necessary, such that the combined test set is a pairwise test set. The combined test set produced by the second option may be larger than the new test set produced by the first option. However, the second option allows the reuse of tests and thus can save the effort for test preparation.

- For a system that has an existing test set and is being modified due to changes of parameters, values, relations and constraints[1], PairTest allows two options. One is to generate a new pairwise test set, without reusing the existing test set. The other is to modify the existing test set and then generate additional tests, if necessary, such that the combined test set is a pairwise test set.

Another test generation tool for pairwise testing is AETG (Automatic Efficient Test Generator)[3] [1] [2]. We used AETG[4] 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. Since the execution time information on test sets generated by AETG is not available, it is impossible to compare AETG and PairTest in terms of actual time taken for test generation. Later we will show that Pairwise 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 statistic analysis, the values of s (number of tests) 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

[1]Relations and constraints are used to allow variations of pairwise testing according to the user's needs [2].
[3]AETG is a trademark of Telcordia Technologies Inc. and is covered by United States Patent 5,542,043.
[4]Telcordia allows free use of AETG for two weeks over the Internet.

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

Pentium II processor, Windows 98 and JDK 1.2.2. According to statistic analysis, the values of t (time for test generation) in Tables 2 and 3 grow in $O(n^2 \times \log(n))$ and $O(d^3)$ respectively. Based on the observation that the size of a test set generated by PairTest is $O(d^2 \times \log(n))$, we have shown that the time complexity of the IPO strategy is $O(d^3 \times n^2 \times \log(n))$. Thus, our empirical results match our analytic results. Based on the same observation for AETG, we have also shown that the time complexity of the AETG heuristic algorithm in [2] is $O(d^4 \times n^2 \times \log(n))$ (as shown in Theorem 4.1).

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
t (time in seconds)	0.11	0.16	0.22	0.44	0.77	0.99	1.37	1.81	2.23	2.96

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
t (time in seconds)	0.05	0.28	0.72	1.54	2.96	5.16

5 Conclusion

In this paper, we have proposed the IPO test generation strategy for pairwise testing. The IPO strategy allows the use of local optimization techniques for test generation and the reuse of existing tests when a system is extended with new parameters or new values of existing parameters. We have presented three IPO-based test generation algorithms. Two of them, algorithms IPO_H_IV and IPO_V, have polynomial time complexities and thus are practical to use. We have constructed a test generation tool, called PairTest, which implements algorithms IPO_H_IV and IPO_V, supports the reuse of existing test sets, and provides graphical user interface. Our empirical results indicate that (1) the size of a pairwise test set generated by PairTest grows logarithmically in the number of parameters and quadratically in the number of values in each parameter, and (2) PairTest is very efficient for generating pairwise test sets. As mentioned earlier, pairwise testing (or 2-way testing) is a special case of n-way testing. The IPO strategy and the algorithms presented in the paper can be easily extended for n-way testing.

Acknowledgment

This research was supported in part in part by NSF grant CCR-9320992 and a grant from IBM in Research Triangle Park, North Carolina. The authors would like to thank Mr. Ho-Yen Chang for his effort on implementing the PairTest tool.

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*, pages 83–87, September 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 Transactions on Software Engineering*, 23(7):437–443, July 1997.

[3] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton. The automatic efficient test generator. *Proc. IEEE Int. Symp. Software Reliability Engineering*, pages 303–309, 1994.

[4] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, JULY 1990.

[5] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, and C. M. Lott. Model-based testing of a highly programmable system. *Proc. IEEE Int. Symp. Software Reliability Engineering*, pages 174–178, 1998.

[6] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing. *Proceedings of the Nineteenth International Conference on Software Engineering*, pages 205–215, May 1997.

[7] Y. Lei and K. C. Tai. In-parameter-order: A test generation strategy for pairwise testing. *Proc. 3rd IEEE High-Assurance Systems Engineering Symposium*, pages 254–261, Nov 1998.

[8] R. Mandl. Orthogonal latin squares: An application of experimental design to compiler testing. *Comm. ACM*, pages 1054–1058, October 1985.

[9] A. W. Williams and R. L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. *Proc. IEEE Int. Symp. Software Reliability Engineering*, pages 246–254, 1996.

6 Appendix

Theorem 3.1 *Let $\mathcal{T}$ be a pairwise test set for parameters $p_1, p_2, \ldots,$ and p_{k-1}. For two permutations ζ and ξ of parameter p_i for $\mathcal{T}$, if ζ and ξ are equivalent, then $weight(\zeta, \mathcal{T}) = weight(\xi, \mathcal{T})$.*

Proof Assume p_i has d valid values $v_1, v_2, \ldots, v_d$. Let π be the set of missing pairs if we add ζ into $\mathcal{T}$. Let π' be the set of missing pairs if we add ξ into $\mathcal{T}$. We make a distribution of π and get φ_{ij}, where $1 \leq i \leq d$ and $1 \leq j \leq k$, to represent the set of missing pairs between p_i and p_j with the value v_i for p_i. We also make similar distribution of π' and get φ'_{ij}. Since $\zeta \equiv \xi$, there exists an onto mapping ψ from ζ to ξ. Let $v_l = \psi(v_i)$, then $|\varphi_{ij}| = |\varphi'_{lj}|$. Further, $\sum_{i=1}^{d} max_{j=1}^{k}\{|\varphi_{ij}|)\} = \sum_{l=1}^{d} max_{j=1}^{k}\{|\varphi'_{lj}|)\}$. We thus conclude that $weight(\zeta, \mathcal{T}) = weight(\xi, \mathcal{T})$. □

Theorem 3.2 *Let ζ and ξ be different permutations of α such that $\zeta = (u_1, u_2, \ldots, u_n)$, $\xi = (v_1, v_2, \ldots, v_n)$, and ζ and ξ are equivalent. For each w in the domain $\mathcal{D}$ of α, there are no i and j $(1 \leq i, j \leq n)$, such that $w = u_i = v_i \wedge w = u_j \neq v_j$.*

Proof Since $\zeta \equiv \xi$, there exists an 1-to-1 function ψ from ζ to ξ. Since $u_i = u_j = w$, then $\psi(u_i) = \psi(u_j)$, that is, $v_i = v_j$. We thus complete the proof. □

Definition 4 *Assume that parameter α has domain $\mathcal{D}$. Let $\mathcal{M}$ be the set of all α's permutations with size m. Assume that a dummy parameter has a domain of boolean values: 0 and 1. Let $\mathcal{B}$ be the set of all the dummy parameter's permutations with size m. A project function $\varphi : \mathcal{M} \times \mathcal{D} \rightarrow \mathcal{B}$ is defined as follows. Let $\zeta = (u_1, u_2, \ldots, u_m) \in \mathcal{M}$ and $v \in \mathcal{D}$. Let $\varrho = \varphi(\zeta, v)$. Then, $\varrho = (b_1, b_2, \ldots, b_m) \in \mathcal{B}$, where*

$$b_i = \begin{cases} 1 & u_i = v \\ 0 & u_i \neq v \end{cases}$$

Definition 5 *Given a project function $\varphi : \mathcal{M} \times \mathcal{D} \rightarrow \mathcal{B}$. Let $v \in \mathcal{D}$. The (appearance) pattern set of v for all α's permutations with size m is defined as $[\mathcal{B}]_v^m = \{\varrho \in \mathcal{B} | \varrho = \varphi(\zeta, v) \wedge \zeta \in \mathcal{M}\}$. ϱ is also referred as an (appearance) pattern of v.*

Definition 6 *Suppose a parameter α has the (appearance) pattern set $[\mathcal{B}]_v^m$ for a valid value v for all permutations with size m. Let $\varrho = (b_1, b_2, \ldots, b_m) \in [\mathcal{B}]_v^m$. The weight of ϱ is $|\varrho| = \sum_{i=0}^{m} b_i$. We also define $[\mathcal{B}]_v^{(m,i)} = \{\varrho | \varrho \in \mathcal{B} \wedge |\varrho| = i\}$, $[\mathcal{B}]_v^{(m,i)}|_{b_j = v_k} = \{\varrho | \varrho \in \mathcal{B} \wedge |\varrho| = i \wedge b_j = v_k\}$.*

Theorem 3.3 *Assume that parameter α has domain $\mathcal{D}$, where $|\mathcal{D}| = d$. Let $\mathcal{M}$ be the set of all α's permutations with size m. Let $\mathcal{N}(m, d)$ be the number of equivalence classes in $\mathcal{M}$.*

$$\mathcal{N}(m, d) = \begin{cases} 1 & if\ d{=}1,\ m{=}1\ or\ 0 \\ \sum_{i=1}^{m}(C_{m-1}^{i-1} * \mathcal{N}(m-i, d-1)) & otherwise \end{cases}$$

Proof Let us consider the boundary cases first. When $m = 1$, each permutation consists of a single value of α. Obviously, all single-value permutations are equivalent. Then, $\mathcal{N}(1, d) = 1$. When $d = 1$, the parameter has the only value, say v, to take. Regardless of m, the only permutation is formed by filling in all of the positions by v. Thus, $\mathcal{N}(m, 1) = 1$.

Now we consider the cases where $m > 1$ and $d > 1$. We prove by induction. Assume that we have derived $\mathcal{N}(x, y)$'s for $x < m$ and $y < d$. We are going to derive $\mathcal{N}(m, d)$. Before we proceed, we define two functions: (1) $\sigma : \mathcal{M} \to \mathcal{D}$, where $\sigma(\zeta) = v_1$ for $\zeta = (v_1, v_2, \ldots, v_m) \in \mathcal{M}$; (2) $\psi_{u\to v} : \mathcal{M} \to \mathcal{M}$, where $\psi_{u\to v}(u_1, u_2, \ldots, u_m) = (v_1, v_2, \ldots, v_m)$ with $v_i = v$ if $u_i = u$, $v_i = u$ if $u_i = v$ and $v_i = u_i$ otherwise.

Suppose α has d values $v_1, v_2, \ldots, v_d$. We divide $\mathcal{M}$ into d groups $\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_d$, where $\mathcal{M}_i = \{\zeta \in \mathcal{M} | \sigma(\zeta) = v_i\}$. We show that $\forall i, j \forall \zeta \exists \xi (\zeta \in \mathcal{M}_j \wedge \xi \in \mathcal{M}_i \wedge \zeta \equiv \xi)$. Given $\zeta \in \mathcal{M}_j$, let $\xi = \psi_{v_j \to v_i}(\zeta)$. Obviously, $\xi \equiv \zeta$. Since $\sigma(\xi) = \sigma(\psi_{v_j\to v_i}(\zeta)) = v_i$, $\xi \in \mathcal{M}_i$. Intuitively, if we pick up one group $\mathcal{M}_i$, then $\forall \zeta \in \mathcal{M}_j (j \neq i)$, we can always locate a permutation $\xi \in \mathcal{M}_i$, which makes $\xi \equiv \zeta$. By Theorem 3.1, we conclude that it is sufficient for us to examine only one group of permutations.

Without loss of generality, we pick up $\mathcal{M}_1$. Let $|\zeta_{v_1}|$ be the number of v_1's in ζ, where $\zeta \in \mathcal{M}_1$. Then, $1 \leq |\zeta_{v_1}| \leq m$. We make a distribution of $\mathcal{M}_1$ and get $\mathcal{M}_{1i}$ where $\mathcal{M}_{1i} = \{\zeta \in \mathcal{M}_1 || \zeta_{v_1}| = i\}$. Let us consider $\mathcal{M}_{1i}$. Then, $\forall \zeta (\zeta \in \mathcal{M}_{1i} \wedge \sigma(\zeta) = v_1 \wedge |\zeta_{v_1}| = i)$. By the theory of combination, $|[\mathcal{B}]_{v_1}^{(m,i)}|_{b_1=v_1}| = C_{m-1}^{i-1}$ considered that the first position is always occupied by v_1. Obviously, for any pattern $\varrho \in [\mathcal{B}]_{v_1}^{(m,i)}|_{b_1=v_1}$, ϱ fixes i positions by v_1, but leaves $m - i$ positions unfixed. Note that v_1 does not appear in the unfixed positions. Thus, ϱ has $\mathcal{N}(m - i, d - 1)$ permutations by our notation.

Note that all the permutations in $\mathcal{M}_1$ has different (appearance) patterns. Consider any two permutations $(u_1, u_2, \ldots, u_3)$ and $(v_1, v_2, \ldots, v_3)$. Since v_1 always shows up in the first position, there must be some position i such that $u_i = v_1$ (or $v_i = v_1$) but $u_i \neq v_i$. By Theorem 3.2, no permutations in $\mathcal{M}_1$ are equivalent. Therefore, the number of equivalence classes in $\mathcal{M}_1$ is $|M|_{equiv} = \mathcal{N}(m, d) = \sum_{i=1}^{m} \mathcal{M}_{1i} = \sum_{i=1}^{m} (C_{m-1}^{i-1} * \mathcal{N}(m - i, d - 1))$.

□

Definition 7 *Assume a pattern $\varrho = (b_1, b_2, \ldots, b_m)$ of a value of w. Let $|\varrho|_i = \sum_{j=0}^{i} b_j$. (Obviously, $|\varrho|_m = |\varrho|$.) Let $\mathcal{B}$ be a set of patterns of size m. Let $\mathcal{M}$ be a set of permutations of size $m - |\varrho|$ and $\mathcal{M}'$ a set of permutations of size m. We define an operator $* : \mathcal{B} \times \mathcal{M} \to \mathcal{M}'$ as follows. Let $\zeta \in \mathcal{M}$, $\xi \in \mathcal{M}'$, and $\xi = \varrho * \zeta$, Then, $\xi = (v_1, v_2, \ldots, v_m)$, where*

$$v_i = \begin{cases} w & b_i = 1 \\ u_{i-|\varrho|_i} & b_i = 0 \end{cases}$$

Theorem 3.4 *Assume that parameter α has domain $\mathcal{D}$, where $|\mathcal{D}| = d$. Let $\mathcal{D} = \{v_1, v_2, \ldots, v_d\}$ and $\mathcal{D}_i = \{v_{d-i+1}, v_{d-i+2}, \ldots, v_d\}$, where $1 \leq i \leq d$. Let $\mathcal{E}(m, \mathcal{D})$ be defined as follows:*

$$\mathcal{E}(m, \mathcal{D}) = \begin{cases} \{\} & \textit{if } m = 0 \\ \{(v_1)\} & \textit{if } m{=}1 \\ \{\overbrace{(v_1, v_1, \ldots, v_1)}^{m}\} & \textit{if } |\mathcal{D}| = 1 \\ \cup_{i=1}^{m}([\mathcal{B}]_{v_1}^{(m,i)}|_{b_1=v_1} * \mathcal{E}(m - i, \mathcal{D}_{d-1})) & \textit{otherwise} \end{cases}$$

$\mathcal{E}(m, \mathcal{D})$ contains exactly one permutation of α with size m from each equivalence class of permutations of α with size m.

Proof The conclusion can be easily drawn with being viewed as a constructive restatement of Theorem 3.3. □

Theorem 4.1 *Assume that a system S has n parameters, and each parameter has at most d valid values. The AETG algorithm presented in [2] generates a pairwise test set for S in $O(d^4 \times n^2 \times \log(n))$.*

Proof

According to [2], the AETG algorithm generates M (e.g. $M = 50$) candidate tests and selects one that covers the most number of uncovered pairs for each test. Assume that a system S has n parameters, and the ith parameter has d_i parameters. Also assume that we have selected r test cases in the resultant test set T. Below we restate the algorithm presented in [2], which is used by AETG to generate a candidate test.

Algorithm *AETG_Select* (S, T)
begin
  choose a parameter f and a value l for f such that
  l appears in the greatest number of uncovered pairs; ... (*)
  let $f_1, \ldots, f_n$ be a random order of n parameters with $f_1 = f$; ... (**)
  let l be the value selected for f_1;
  for $j = 2$ **to** n **do**
  begin
    $w' = 0$;
    for each value v of f_j **do**
    begin
      $\tau = \{(u, v) | u$ is the value selected for f_i, $1 \leq i \leq j$ $\}$;
      let w be the number of uncovered pairs in τ;
      if $w > w'$ **then**
      begin $w' := w; v' := v$; **end**
    end
    let v' be the value selected for f_j;
  end
end

Now we consider the time complexity of the algorithm *AETG_Select*. Let $d = max\{d_i | 1 \leq i \leq n\}$. The number of uncovered pairs is $O(d^2 \times n^2)$. In order to find f and l, statement (*) needs to go through the set of uncovered pairs, which takes $O(d^2 \times n^2)$. In statement (**), it takes $O(n)$ to generate a random order of n parameters. Assume that the determination of if a pair is uncovered or not takes $O(1)$ (this can be done by maintaining a coverage matrix as in algorithm IPO_H_IV). The size of τ is in $O(n)$. The inner *for* loop takes $O(d \times n)$ to calculate the number of uncovered pairs for each value of f_j and derive the one which covers the greatest number of uncovered pairs. Considering that the outer *for* loop iterates through $n - 1$ parameters, the nested *for* loop together takes $O(n^2 \times d)$. Therefore, the time complexity of *AETG_Select* is $O(n^2 \times d^2)$. Assume that the sizes of test sets generated by AETG are $O(d^2 \times \log(n)$, the time complexity of the AETG algorithm is $O(d^4 \times n^2 \log(n))$.

□