Test data combination strategy for effective test suite generation

Jae Hoon Yoon and Jeong Seok Kang
Electronic and Telecommunication Engineering
Kangwon National University
Chuncheon, Republic of Korea
{jaehoon, sleeper82}@control.kangwon.ac.kr

Hong Seong Park*
Electronic and Telecommunication Engineering
Kangwon National University
Chuncheon, Republic of Korea
hspark@kangwon.ac.kr

Abstract—It is important to reduce the size of a test suite in order to reduce the cost and time in software testing. Test data combination strategy is important because the size of a test suite is different by test data combination strategies even if their fault detection rates are equal. But in many cases, combination strategies include redundancy because they do not consider relationship between inputs and outputs of software under test. Also, some combination strategies considering relationship between inputs and outputs do not reduce the size of a test suite enough. In this paper, we propose test data combination strategy that improves existing test data combination strategy considering relationship between inputs and outputs. Also, we compared our strategy with existing test data combination strategies in robot software, and the experimental result shows the advantage of our algorithm.

Keywords— Software testing; Test suite; Test data combination; IO relationship

I. INTRODUCTION

Test data is data which has been specifically identified for use in tests, typically of a computer program. As a large and complex system, software testing may need many costs and many times. It is important to reduce the number of test data in order to reduce the costs and times in software testing. Test suite is combination set of test data, test data combination strategy is important because the size of test suite is different by test data combination strategies even if their fault detection rates are equal.

Software have inputs, outputs and relationship between inputs and outputs(I/O relationship) variously. These software testing may be efficient by test suite considering I/O relationship. Many studies have been proposed test data combination strategies such as AETG[1], CA[2], MCA[2], VCA[3], and IPO[4]. But the studies are lack of consideration about I/O Relationship. Recently, Test data combination strategy considering I/O Relationship has been studies[5][6].

Union[6] is union of each test data combination from individual outputs. Union may not create a small test suite because the size of generated test suite relies on how the values are assigned to “don’t care” parameters, which are the parameters that do not belong to current coverage requirement. ReqOrder[5] improve Union. ReqOrder may avoid the redundancy of combinations in “don’t care” parameters and reduce the size of test suite evidently.


In this paper, we propose a test data combination strategy for test suite generation, called Advanced ParaOrder(APO). APO improve ParaOrder[5] and reduce the size of test suite more than ParaOrder. The remainder of this paper is organized as follows: Section 2 presents proposed algorithm. Section 3 compared our algorithm with other algorithms by robot software. Section 4 concludes this paper.

II. TEST SUITE GENERATION ALGORITHM

We propose advanced ParaOrder[5] for test suite generation(APO). APO generates minimal test suite more than ParaOrder. Let \( F = \{f_1, f_2, ..., f_n\} \) denote the set of parameters, \( V_i = \{1, 2, ..., v[1](1 \leq i \leq n)\} \) denotes the set of values in \( f_i \) and size

Algorithm 1. Advanced ParaOrder(APO) for test suite generation
Input: \( F = \{f_1, f_2, ..., f_n\}, R = \{r_1, r_2, ..., r_m\} \)
Output: Test suite \( T \) that satisfy \( R \)
Begin
\( f_i = \text{SelectFactor}(F) \); Create a test suite \( T \) with \( v[f] \) empty test datas;
Assign \( v[f] \) values of \( f_i \) into \( T \);
\( F = F - \{f_i\}; \ F_{\text{deal}} = \{f_i\}; \)
while \( F \neq \phi \)
\( F_i = \text{SelectFactor}(F) \);
\( \text{Inters} = \{r_k \cap (F_{\text{deal}} - \{f_i\})|f_i \in r_k, k=1,2,..,m\}; \)
\( \text{CombSet} = \{(v_{i1},...,v_{ik})|v_{ij} \in V_{i1},..., v_{ij} \in V_{ij}\}; \)
End
\{f_i, \ldots, f_m\} \in \text{Inters};$

// Extending $T$ Horizontally

for $j = 1$ to $|T|$  
Check all parameters (except $f_i$) in elements of $\text{Inters}$ in $j$-th test data combination;

If all above positions are not empty then

\[ W[j] = \text{sets of largest size} \{v_{i,j}, \ldots, v_{i,m}\} \]

for $v = 1$ to $v[i]$ // select value of $f_i$, greedily

$\text{count}_v$ = the number of combinations in $\text{CombSet}$ that will be covered by $T$ if $v$ is selected as the value of $f_i$ in $j$-th test data combination. + the number of combinations in $\text{CombSet}$ that will be covered by $T$ if $v$ is selected as the value of $f_i$ in $j$-th test data combination.

end

Select a value $v$ with the largest $\text{count}_v$;

$\text{CombSet} = \text{CombSet} - \{\text{combinations that covered by the $j$-th test data combination}\}$;

if $\text{CombSet} = \emptyset$ then break;
end

// add new test data combination to extending $T$ vertically

for each combination $\text{com}$ in $\text{CombSet}$

If there exist test data combinations in $T$ that $\text{com}$ can be added into then

Select a test data combination that match the $\text{com}$ mostly from above ones, and fill $\text{com}$ into selected test data combination;

else

Add a new empty test data combination into $T$;

Fill $\text{com}$ into this test data combination;

end

$F = F - \{f_i\}; F_{\text{deal}} = F_{\text{deal}} \cup \{f_i\}$;
end

of $V_i$ different from each parameters, $R = \{r_1, r_2, \ldots, r_m\}$ denote the set of relationship among parameters, and $T$ is finished test suite.

APO and ParaOrder[5] are similar, with Extending $T$ Horizontally step. Extending $T$ Horizontally step of ParaOrder selects value that mostly covers combinations in $\text{CombSet}$ by test data combination included value. Whereas APO assign weight that is priority of to cover to largest combinations in $\text{CombSet}$. The smaller size of largest set in $\text{CombSet}$, size of adding test data combinations in Extending $T$ Vertically part show a tendency to reduce.

![Fig. 1. System 1 with multiple inputs](image)

**TABLE I. INPUT PARAMETER SPECIFICATIONS OF ROBOT GUIDE SOFTWARE**

<table>
<thead>
<tr>
<th>Component</th>
<th>Input ID</th>
<th>Input name</th>
<th>Equivalence partitions</th>
<th>The number of test data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>fbs_in1</td>
<td>Forward left sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 4$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>fbs_in2</td>
<td>Forward middle sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 4$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>fbs_in3</td>
<td>Forward right sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 4$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>fbs_in4</td>
<td>Back left sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 3$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>fbs_in5</td>
<td>Back middle sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 3$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>fbs_in6</td>
<td>Back right sensor</td>
<td>$0 &lt; x \leq 2, 2 &lt; x \leq 3$</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>fbs_in7</td>
<td>Forward range boundary</td>
<td>$0 &lt; x \leq 3$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>fbs_in8</td>
<td>Back rage boundary</td>
<td>$0 &lt; x \leq 3$</td>
<td>2</td>
</tr>
<tr>
<td>Mobile</td>
<td>Ss_in1</td>
<td>Statement language</td>
<td>KOREAN, ENGLISH, JAPANESE, CHINESE</td>
<td>4</td>
</tr>
</tbody>
</table>
remaining combinations are \((A_2, D_1)(B_1, D_2)(C_1, D_3)(C_2, D_3)\). ParaOrder selects \(D_1\) for to cover \((A_2, D_1)(C_3, D_3)\). Whereas APO select \(D_3\) for to cover \((B_1, D_3)(C_3, D_3)\). ParaOrder generates test suite comprised of 13 test data combinations. APO generates 12 test data combinations as follows. \{(A_1, B_1, C_1, D_1), (A_2, B_1, C_2, D_2), (A_1, B_2, C_2, D_3), (A_2, B_2, C_2, D_3), (A_1, B_2, C_3, D_3), (A_2, B_1, C_1, D_1), (A_1, B_3, C_1, D_2), (A_2, B_1, C_2, D_3)\}.

### III. Experiment and Evaluation

![Robot guide software](image)

**Fig. 2.** Robot guide software.

We experimented with component-based software. As shown Fig. 2 and Fig. 3, the case study has nine inputs and two outputs. Table 4 shows the input specification and Table 3 shows the output specification of the case study. For pairwise,

#### TABLE II. OUTPUT PARAMETER SPECIFICATIONS OF ROBOT GUIDE SOFTWARE

<table>
<thead>
<tr>
<th>Output ID</th>
<th>Output name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mc_out1</td>
<td>Mobile control command</td>
<td>This is movement command of mobile robot. This output is string type.</td>
</tr>
<tr>
<td>ss_out1</td>
<td>Speech statement</td>
<td>This is guiding statement, string type.</td>
</tr>
</tbody>
</table>

![Robot Guide Composite Component](image)

**Fig. 3.** I/O relationship of robot guide software.

#### TABLE III. TEST SUITE OF CASE STUDY GENERATED BY APO

<table>
<thead>
<tr>
<th>No</th>
<th>fbs_in1</th>
<th>fbs_in2</th>
<th>fbs_in3</th>
<th>fbs_in4</th>
<th>fbs_in5</th>
<th>fbs_in6</th>
<th>fbs_in7</th>
<th>fbs_in8</th>
<th>ss_in1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>KOREAN</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>4</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>2</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>15</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>


we define $R=\{(fbs_{\text{in1}},fbs_{\text{in2}}),(fbs_{\text{in1}},fbs_{\text{in3}}),(fbs_{\text{in1}},fbs_{\text{in7}}), (fbs_{\text{in1}},ss_{\text{in1}}),(fbs_{\text{in2}},fbs_{\text{in3}}),(fbs_{\text{in2}},fbs_{\text{in7}}),(fbs_{\text{in2}},ss_{\text{in1}}), (fbs_{\text{in3}},fbs_{\text{in7}}),(fbs_{\text{in3}},cc_{\text{in1}}),(fbs_{\text{in7}},cc_{\text{in1}}), (fbs_{\text{in4}},fbs_{\text{in5}}),(fbs_{\text{in4}},fbs_{\text{in6}}),(fbs_{\text{in4}},fbs_{\text{in8}}),(fbs_{\text{in5}},fbs_{\text{in6}}),(fbs_{\text{in5}},fbs_{\text{in8}}),(fbs_{\text{in6}},fbs_{\text{in8}})\}$. Table 6 shows the generated test suite. ParaOrder[5] generate test suite 20 test data combinations.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Size of test suite</th>
</tr>
</thead>
<tbody>
<tr>
<td>APO</td>
<td>17</td>
</tr>
<tr>
<td>ParaOrder</td>
<td>20</td>
</tr>
<tr>
<td>Jenny</td>
<td>25</td>
</tr>
<tr>
<td>PICT</td>
<td>24</td>
</tr>
</tbody>
</table>

Jenny[7] and PICT[8] generate test suite not considering I/O relationship. Table 4 shows the comparison result among APO, ParaOrder[5], Jenny and PICT. APO, ParaOrder, Jenny, and PICT generated test suite that covered all pairs that satisfy R. Jenny and PICT are bigger than APO and ParaOrder considering I/O relationship. Also, size of test suite generated by ParaOrder is bigger than APO. Because APO do not includes redundancy and assign weight to largest combinations in CombSet. APO is more efficient than test suite algorithm not considering I/O relationship and ParaOrder.

IV. CONCLUSION

This paper proposed the APO test data combination strategy for test suite generation by improving ParaOrder[5]. Test data combination strategies considering I/O Relationship may reduce the size of test suite without decrease of the fault detect ability. The experimental data shown APO produces competitive test suites for fixed-level inputs more than ParaOrder. As part of future work, we will generate test suite when systems are modified due to changes of parameter, value, and I/O Relationship.

REFERENCES


