Generating Variable Strength Covering Array for Combinatorial Software Testing with Greedy Strategy

Ziyuan Wang[1, 2+] and Haixiao He[1]
[1]School of Computer, Nanjing University of Posts and Telecommunications, Nanjing, 210006, China
[2]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, 210093, China
Email: wangziyuan@njupt.edu.cn

***Abstract*—Combinatorial testing is a practical and efficient software testing techniques, which could detect the faults that triggered by interactions among factors in software. Compared to the classic fixed strength combinatorial testing, the variable strength combinatorial testing usually uses less test cases to detect more interaction faults, because it considers the actual interaction relationship in software sufficiently. For a model of variable strength combinatorial testing that has been propose previously, two heuristic algorithms, which are based on one-test-at-a-time greedy strategy, are proposed in this paper to generate variable strength covering arrays as test suites in software testing. Experimental results show that, compared to some existed algorithms and tools, the two proposed algorithms have advantages on both the execution effectiveness and the optimality of the size of generated test suite.**

***Index Terms*—software testing, combinatorial testing, test generation, interaction relationship, algorithm**

I. INTRODUCTION

Software can be considered as a complex logic system, which may be affected by many factors or parameters, such as system configurations, internal events, external inputs etc. Rather than the single factors, the interaction of multiple factors may also affect the work of software systems. Combinatorial testing (or interaction testing) uses a small test suite that cover all needed parametric values and their combinations, to detect the faults that may triggered by these single factors or parameters in software and even the interactions of them. Many applications of combinatorial test approach have shown that, a carefully designed test suite which contains small number of test cases can yield high fault detection ability with reduced test cost. Therefore, combinatorial testing is an important and effective method for software testing, especially for those high-configurable systems.

Existed combinatorial testing includes fixed strength combinatorial testing and variable combinatorial testing. The former requires a test suite to cover all N-way ($N \geq 2$) combinations of factors value, by using a fixed uniform strength N. And the latter allows that the strength of different interaction to be variable. It was proposed by Cohen et al in 2003 [1], since Bach and Schroeder pointed out most successful applications of combinatorial testing require the detailed analysis of characteristic for software [12]. In the variable strength combinatorial testing, people select a series of disjoint sub-sets of factor and assigning a higher strength for interaction among those factors in sub-sets. Note that the variable strength combinatorial testing requires all sub-sets must be disjoint, so it may be helpless for the cases that interaction relationship does not satisfy such constraint.

Above all, to increase the effectiveness of existed combinatorial testing, it is necessary to mine the actual interaction relationship among factors and make more sufficient consideration on such interaction relationship. A new model of variable strength combinatorial testing (or named "interaction relationship based combinatorial testing" in Ref. [13]) was proposed by us previously [13][20]. To generate combinatorial test suite for the new model of variable strength combinatorial testing, two heuristic algorithms, which are based on one-test-at-a-time greedy strategy, are proposed in this paper. The theoretical analysis and experimental results show there are many advantages of the proposed algorithms.

The remainder of this paper is organized as follows. Section 2 describes definitions. Section 3 reviews related works. Section 4 describes the framework of one-test-at-a-time greedy strategy. And two concrete test generation algorithms which are based on one-test-at-a-time strategy are proposed in section 5. Section 6 gives experimental results. Finally, a conclusion remarks is given.

II. DEFINITIONS

Before introducing the new model of variable strength combinatorial testing, we firstly review some definitions about existed fixed strength and "narrow sense" variable strength combinatorial testing [23][24].

A. Existing Models of Combinatorial Testing

Suppose that the system under test (SUT) has n factors (or parameters), and each factor f_i has a_i ($1 \leq i \leq n$) discrete values. Let $F = \{f_1, f_2, \ldots, f_n\}$ denote the set of factors, and $V_i = \{1, 2, \ldots, a_i\}$ ($1 \leq i \leq n$) denote the value set of factor f_i. If the cardinalities of all n value sets are equal, it is a system with fixed-level factors. Otherwise, it is a system with mixed-level factors. In this paper, we also assume that all factors are independent, which means there are no constraints between factors and their values.

Manuscript received September 4, 2013; revised October 9, 2013; accepted October 9, 2013.
+Corresponding author.

doi:10.4304/jsw.8.12.3173-3181

Definition 1. The n-tuple $test= (v_1, v_2,\ldots, v_n)$ ($v_1\in V_1$, $v_2\in V_2,\ldots, v_n\in V_n$) is a test case for the system.

Definition 2. Given $A=(a_{i,j})_{m\times n}$ is a $m\times n$ array, where the j-th column denotes the factor f_j of the SUT and all elements of this column come from the finite set V_j (j=1, 2,…, n), that is $a_{i,j}\in V_j$. If every $m\times N$ ($2\le N\le n$) sub-arrays contain all value combinations of such N columns (or factors), then A is an N-way fixed strength covering array or a fixed strength covering array with strength N. It could be denoted as $CA(m; N, F)$.

Definition 3. For an fixed strength covering array A, if it contains C, a multi-set of disjoint covering arrays that each with a larger strength than N, then A is a "narrow sense" variable strength covering array that could be denoted as $VCA(m; N, F, \{C\})$.

The definition of "narrow sense" variable strength covering array is concluded from the description of Cohen [1]. We call their approach that proposed by D. M. Cohen et al as the "narrow sense" variable strength combinatorial testing, since there is a limitation that it need a "disjoint" property, which will be demonstrated in next sub-section.

B. New Model of Variable Strength Combinatorial Testing

In this sub-section, we introduce a new model of variable strength combinatorial testing, which considers the actual interaction relationship more sufficiently than "narrow sense" variable strength combinatorial testing.

Firstly, we discuss the interaction among factors. One group of factors that have interaction with each other could form a subset r of F. It means that there is a $|r|$-way interaction (or an interaction with strength $|r|$) among all $|r|$ factors in such subset. For such subset, all value combinations of factors in r should be covered by test suite. Furthermore, for the whole SUT, there should be a collection $R=\{r_1, r_2,\ldots, r_t\}$ that contains t subsets of F, and these subsets represent all interactions in such SUT. In combinatorial testing, all such interactions in R must be covered by test suite. For example, if we test a system with n factors by 2-way (or pair-wise) combinatorial testing approach, there will be $|R|=n\times(n-1)/2$ different 2-way interactions and $R=\{\{f_i, f_j\}| f_i, f_j\in F, i\neq j\}$.

Definition 4. A subset $r_k\in R$ (k=1, 2,…, t) could be named as an interaction coverage requirement, or coverage requirement for short. And the collection R could be named as the interaction relationship of SUT.

For simplicity, we define following rules: (i) Each coverage requirement $r_k=\{f_{k,1}, f_{k,2},\ldots, f_{k,nk}\}\in R$ (k=1, 2,…, t) has n_k factors (n_k>1); (ii) For any two different coverage requirements $r_{k1}, r_{k2}\in R$ ($k_1\neq k_2$), there are $r_{k1}\not\subset r_{k2}$ and $r_{k2}\not\subset r_{k1}$; (iii) Two different factors f_i, $f_j\in F$ ($i\neq j$) interact with each other if and only if there is a coverage requirement $r\in R$ and $f_i, f_j\in r$.

Definition 5. Given $A=(a_{i,j})_{m\times n}$ is a $m\times n$ array, where the j-th column denotes the factor f_j of the SUT and all elements of this column come from the finite set V_j (j=1, 2,…, n), that is $a_{i,j}\in V_j$. For a coverage requirement $r_k\in R$, if the sub-array that consists of all factos in r_k contains all value combinations of those factors, then A satisfies r_k. If A satisfies all coverage requirements in an interaction relationship R, then A is a variable strength covering array for R and it could be denoted as $VCA(m; F, R)$.

Therefore, the variable strength covering array for R should cover all combinations in the set:

$$CombSet=\mathrm{U}^{t}_{k=1}CombSet_k$$

Where the $CombSet_k$ (k=1, 2,…, t) covers the coverage requirement r_k:

$$CombSet_k=\{(v_{k,1}, v_{k,2},\ldots, v_{k,nk})| v_{k,1}\in V_{k,1}, v_{k,2}\in V_{k,2},\ldots, v_{k,nk}\in V_{k,nk}\}.$$

Definition 6. The software testing approach that designs and runs variable strength covering array as test suite is variable strength combinatorial testing approach.

Given a SUT, the combinatorial test suite T which covers interaction relationship R could be obtained easily from the covering array A for R, by mapping each row of covering array to a test case of test suite. So we say that variable strength combinatorial test suite and variable strength covering array are equivalent in this paper.

Definition 7. For a variable strength combinatorial test suite T, if it contains minimum possible number of test cases to cover R, then T is an optimal variable strength combinatorial test suite.

An optimal test suite can help us to test software with minimal cost. But as we demonstrated before, pair-wise testing could be considered as a special case ($R=\{\{f_i, f_j\}| f_i, f_j\in F, i\neq j\}$) of variable strength combinatorial testing. Therefore, the problem of generating optimal test suite for variable strength combinatorial testing is as hard as NP-C, since the problem of generating optimal pair-wise test suite has been proven to be a NP-C problem [9].

Different from fixed strength combinatorial testing, the new variable strength combinatorial testing approach allows the strengths of interactions to be variable. And such approach is also more general than existed "narrow sense" variable strength combinatorial testing, since it dose not require the property of "disjoint". For example, consider a given interaction relationship $R=\{r_1, r_2\}$, where two coverage requirements $r_1=\{f_1, f_2\}$ and $r_2=\{f_1, f_3\}$ intersect with each other, in a system with $F=\{f_1, f_2, f_3\}$. Such interaction relationship is difficult to be described by the "narrow sense" variable strength combinatorial testing, for it requires two sub-arrays of covering array are disjoint.

III. RELATED WORKS

Except the existed "narrow sense" variable strength combinatorial testing, there are also many works related to variable strength combinatorial testing approach. E.g., as a special application of interaction relationship, Input-Output relationships has been discussed by Schroeder et al [14][15][16]. The open-sourced tool TVG that managed by Software Eva[1], and the tool PICT that developed by J. Czerwonka [6] are also both available to generate variable strength combinatorial test suite.

In the model of IO relationship testing, each output variable is influenced by a group of input variables,

[1] http://sourceforge.net/projects/tvg/

which could be considered as a coverage requirement in the model of variable strength combinatorial testing. People need an optimal test suite, which covers all value combinations of input variables that influence each output variable. However, generating an optimal test suite to satisfy a IO relationship has been proven to be NP-C [15]. Therefore, we can conclude again that generating an optimal variable strength combinatorial test suite is also a NP-C problem, by mapping each coverage requirement to a output variable, which is influenced by a set of input variables that corresponding to factors in such a coverage requirement.

There are totally 3 different test generation algorithms proposed to generate test suite for IO relationship testing [15]. Except a brute force algorithm that generates optimal test suite, other two heuristic ones UNION and GREEDY. The UNION can generate a result very quickly with time complexity $O(\sum_{k=1}^{t}(m\times n\times|CombSet_k|))$, but the generated test suite is usually very big. The GREEDY can generate much smaller test suites than UNION, but the worst time complexity is bad as $O(m\times(\sum_{k=1}^{t}|CombSet_k|)\times(\prod_{i=1}^{n} a_i))$. To make the algorithm become more efficient, a color graph based problem reduction method was proposed [16]. However, the reduction technique is suitable only when IO relationships are "simple", which means that the number of edges in color graph is much smaller than that of complete graph. Another limitation of such reduction technique is that it may lead to the redundancy of generated test cases.

Test Vector Generator (TVG) is a project-based MDI application managed by Software Eva. TVG provides a test generation tool with GUI to generate combinatorial test suite based on the input-output relationship or fixed strength coverage. The main disadvantage of TVG, which will be displayed in our experiment, is that the size of generated test suite is usually not as small as expected. PICT is a combinatorial test generation tool. By editing the option of command line and the "Sub-Models" field of "model file", it can also generate variable strength (or "mixed strength" described in [6]) combinatorial test suite. The limitation of PICT, which will be shown in experimental results, is that its performance in variable strength combinatorial test generation is much worse than that in fixed strength combinatorial test generation for some unidentified reasons.

In recent years, we have made study on combinatorial testing. Especially, the characteristic of factor interaction relationship has been considered in our works [13][17][20]. E.g., we proposed the new model of variable strength combinatorial testing (or called "interaction relationship based combinatorial testing" in Ref. [13]) [20]. And correspondingly, several variable strength combinatorial test suite generation algorithms, including the ReqOrder with a worst time complexity $O(\sum_{k=1}^{t}(m\times |CombSet_k|\times|r_k|))$ [13], the ParaOrder with a worst time complexity $O(\sum_{k=1}^{t}(m\times a_i\times|CombSet_k|\times\max_{1\le k\le t}\{|r_k|\}))$ [13][20], and the Density [20], which can be considered as an initial version of algorithms that will be described in this paper, were proposed previously. This paper will mainly improve the Density to generate smaller test suite.

IV. ONE-TEST-AT-A-TIME GREEDY STRATEGY

Generating optimal fixed strength combinatorial test suite has been proved to be NP-C [9], and many heuristic strategies were proposed. The one-test-at-a-time strategy is one that has been most widely used for its simplicity, accuracy, efficiency, and consistency. Rather than test generation, it could provide some additional functions such as seed test cases, constraint handling, and test prioritization, etc. Therefore, we apply one-test-at-a-time strategy on problem of variable strength combinatorial test generation.

In one-test-at-a-time strategy, a set of combinations that should be covered by combinatorial test suite is required, which is just the set *CombSet* that has been mentioned before. The process starts with an empty initial test suite. Then at each time, one single test case will be selected and added into test suite, and the covered combinations will be removed form set *CombSet.* Such step repeats until set *CombSet* becoming empty. The framework of one-test-at-a-time strategy is described as Algorithm 1.

Algorithm 1. One-test-at-a-time Strategy
Start with an empty test suite T;
Initialize the set CombSet according to SUT;
While (CombSet ≠ ∅)
    Select a single test case, and add it into *T*;
    Modify *CombSet* by deleting combinations that covered by selected test case;
End While

To generate test suite as small as possible, some people adopt a greedy strategy in one-test-at-a-time strategy, which selects a "best" single test case each time to cover the greatest number of uncovered combinations in *CombSet*. This "best" greedy method may generate a small test suite in fixed strength and variable strength combinatorial testing. For example, in theory, D. M. Cohen et al proved that the size of fixed strength combinatorial test suite that generated by the "best" greedy method grows logarithmically in the number of factors [2]. And in experimental, the GREEDY, which adopts the "best" greedy method, could generate much smaller variable strength combinatorial test suite than other heuristic algorithms [15].

But unfortunately, selecting such a "best" test case is difficult too. C. J. Colbourn proved that, in pair-wise testing, for a given set of uncovered pairs and a given positive integer *p*, the problem that determining whether there exists a test case which covers *p* pair-wise combinations is a NP-C problem. It means that there exists no efficient polynomial time algorithm to select such a "best" test case to cover the greatest number of pair-wise combinations [2][5]. Note that fixed strength combinatorial testing could be considered as special cases of variable strength combinatorial testing, so selecting such a "best" test case in variable strength combinatorial testing is also a NP-C problem. E.g., the GREEDY, which is an exponential-time algorithm, can be hardly used in practice.

For above limitations, instead of selecting the "best" test case by searching the exponential number ($\prod^{n}_{i=1} a_i$) of usable candidates, a feasible approach is to generate approximate "best" test case with some more efficient heuristic approaches, such as determining an order of factors and fixing values in the determined turn [10]. This kind of approximate algorithms include AETG[2], TCG[3], DDA[5], and PICT[6]. The algorithms that will be proposed in this paper are approximate algorithms too.

V. Algorithms to Generate Single Test Case in One-test-at-a-time Strategy

We proposed two concrete deterministic algorithms, which are based on "density", to generate single test cases in one-test-at-a-time strategy. The concept of "density" was firstly proposed by C. J. Colbourn et al firstly [5], but it is only available for fixed strength combinatorial test generation. Therefore, we define a new concept of "density" for variable strength combinatorial test generation.

When generating a single test case with one-test-at-a-time strategy, the priorities of different coverage requirements are different. For a coverage requirement (assuming it is r_k ($1 \le k \le t$) without loss of generality), if there are a greatest number of uncovered combinations in $CombSet_k$, then it should be handled as early as possible in our intuition. We can illustrate it by an extreme example. Support two coverage requirements $r_1=\{f_1, f_2\}$ and $r_2=\{f_1, f_3\}$ in R. Considering a step that there are two uncovered combinations ($f_1=1$, $f_2=1$) and ($f_1=1$, $f_2=2$) in $CombSet_1$ but none in $CombSet_2$. In such situation, the values of factors in r_2 could be selected randomly. If we fix values for r_2 firstly and the fixed value of f_1 is 2, the generated test case will not cover any uncovered combination, and the final test suite may be redundant (see Figure 1). Therefore, a priority number, which should grow as the growth of the number of uncovered combinations, is required to measure such priority. And for a given coverage requirement r_k, the priority number could be selected as a density of such coverage requirement: a ratio of current number of uncovered combinations in $CombSet_k$ to the $\max_{1\le k\le t}\{\prod_{fi\in rk} a_i\}$. It is evident that such ratio ranges from 0 to 1.

Note that there may be intersection between two different coverage requirements. It means that, when handle a given coverage requirement, the value of some factors in such coverage requirement may have been fixed already. And it is reasonable that only the available combinations, in which the values of such factors are equal to the fixed values in current test case, should be counted when calculating density. And in the extreme case that the values of all factors have been fixed, there will be at most one available combination. If such available combination exists, which means that it covers a new uncovered combination, the density should be the upper bound 1; else it should be the low bound 0. Therefore, after the values of all factors have been fixed in current test case, the density of each coverage requirement should be as big as possible to make test case cover more combinations.

And in another aspect, only the factors, whose values have not been fixed, should be counted when calculating density. So we could construct a sub coverage requirement by collecting these factors, and then calculate density for the sub coverage requirement to instead original one. The number of factors in sub coverage requirement is less than that in original one, so the denominator of density should also be modified to a smaller value, in order to increase the density of coverage requirement to a balanced level.

Therefore, we could define the density (local density) of a given coverage requirement r_k as:

$$LD_k = \frac{num_k}{(\max_{1\le k\le t}\{\prod_{f_i\in r_k} a_i\})^{(n_k-p_k)/n_k}}$$

In which, the symbol num_k is the number of available uncovered combinations in set $CombSet_k$, and the symbol p_k is the number of factors whose values have been fixed. The special case of $p_k=n_k$ means that the values of all factors in r_k have been fixed already.

For simplicity, we call the density of coverage requirement as the local density. And based on the definition of local density, we define the global density for the whole system as:

$$GD = \sum_{k=1}^{t} LD_k$$

After introducing the concept of "density", we will present two different algorithms to generate single test case. And when generating a single test case, we endeavor to take the global density as great as possible, to make the generated test case cover uncovered combinations as most as possible.

A. Fix Value in The Order of Coverage Requirements

To generate a single test case with one-test-at-a-time strategy efficiently, the value of factors should be fixed in

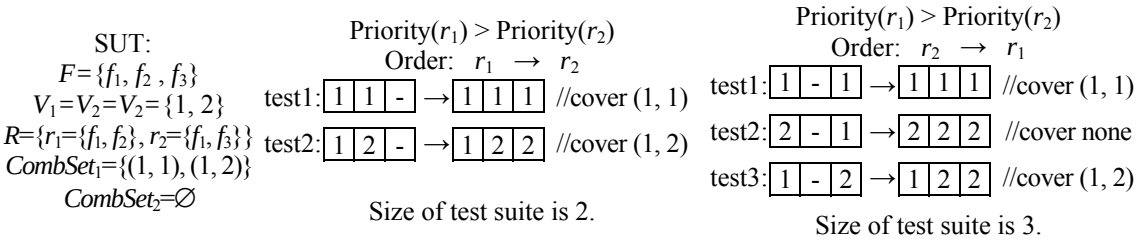

Figure 1. Different priorities of coverage requirements
(Values of factors in r_2 are selected randomly)

turn. In the first proposed algorithm DA-RO, the values will be fixed in the order of coverage requirements, and such order will be determined by the local densities of each coverage requirement.

At each stage, one coverage requirement, in which there is at least one factor whose value has not been fixed, will be selected for its greatest local density. Once a coverage requirement (assuming it is r_k ($1 \le k \le t$) without loss of generality) is selected, then the values of factors in selected coverage requirement should be fixed according to the global density. For each one of totally $\prod_{fi \in rk} a_i$ combinations in set $CombSet_k$, if it is available in current test case, then calculate the global density that assume we fix values for factors in r_k as such combination. The available combination that takes the greatest global density will be selected and assigned to the corresponding factors. Note that the local and global density may change after fixing values for factors in a coverage requirement, so the densities should be modified over again.

Above operations will repeat until all coverage requirements have been handled in current test case. The process of generating a single test case is also described as the Algorithm 2. After run such algorithm, there may still be some independent factors, which are not involved in any coverage requirement, have not been assigned. Note that these factors can not reduce the coverage ability of generated test suite, so we can fix values for them after all test cases have been generated, to guarantee all valid values of each independent factor appear at least once.

Algorithm 2. Generate One Test Case by Fixing Value in the Order of Coverage Requirements

```
Start with an empty test case test, in which the values
of all factors have not been fixed;
While (there are coverage requirements not been dealt)
    For k=1 to t
        If (in r_k, there are factors whose value has not
        been fixed) then
            Calculate local density for r_k∈R;
        End If
    End For
    Select a new coverage requirement r_k with the
    greatest local density;
    For Each combination comb∈CombSet_k
        If (comb is available in test) then
            Calculate the global density by assuming the
            values of factors in r_k are fixed as comb;
        End If
    End For
    Select a combination comb that takes the greatest
    global density;
    Fix factors in r_k as the selected combination;
End While
```

The step, which selecting the coverage requirement with the greatest local density, may suffer from the problem of ties that there exist more than one coverage requirements with the equal greatest local densities. There are several methods to break ties, such as the *First* strategy that selecting the first one that with the greatest local density, the *MostFactor* strategy that selecting the one that with most fixed factors, and the *Random* strategy that selecting one from all that with greatest local density randomly. Another step that may suffer from ties is the step that selecting combination for the selected coverage requirement to increases global density as great as possible. The possible available tie-break methods in such step are similar to above three strategies. We did some experiments to test tie-break methods in above two steps, and the results showed that there are not obvious differences between different methods in aspect of size of generated test suite. To make algorithm to be simple and deterministic, we usually adopt the *First* strategy in both two steps.

Then we analyze the time performance of algorithm DA-RO. It is very difficult to find the time complexity of calculating a local density, but we can find a upper bound $O(|r_k| \times |CombSet_k|)$. When selecting coverage requirement, the local density of at most t coverage requirements should be compared, that is $O(\sum_{k=1}^{t}(|r_k| \times |CombSet_k|))$. And the time complexity of selecting combination for a selected r_i is $O(|CombSet_i| \times \sum_{k=1}^{t}(|r_k| \times |CombSet_k|))$, since the time complexity of calculating global density is also $O(\sum_{k=1}^{t}(|r_k| \times |CombSet_k|))$. So the worst time complexity of DA-RO is $O(\sum_{i=1}^{t}\sum_{k=1}^{t}(|r_k| \times |CombSet_k| \times |CombSet_i|))$.

A. Fix Value in The Order of Factor

By analyzing the time complexity of algorithm DA-RO, we can conclude that the time performance of DA-RO is not as good as expected. Therefore, a concrete test generation algorithm with a better time performance is required. This sub-section proposes the DA-FO, another variable strength combinatorial test generation algorithm. The new algorithm is similar to DDA, which is a pair-wise test generation algorithm, for they both generate single test case by fixing value one by one as a given order of factors.

To fix value in the order of factors, the order of factors must be determined firstly, so we should define a priority number to measure the priority of different factors and determine such an order. We define a factor density, which could be described as the summation of local densities of coverage requirements that contain such a factor, to measure the priority of factors:

$$FD_i = \sum_{k=1}^{t} LD'_k$$

Where, the mutation of local density LD'_k (k=1, 2,…, t) is defined as:

$$LD'_k = \begin{cases} LD_k, & f_i \in r_k \\ 0, & f_i \notin r_k \end{cases}$$

It is evidently that the factor, which involved in large number of coverage requirements that with high priorities and great local densities, will have a great factor density. And such a factor should have a high priority to be handled when generating a single test case.

Therefore, after a factor, whose value has not been fixed, with the greatest factor density is selected, the value of selected factor should be fixed. Without loss of generality, assume that the selected factor is f_i and the value could be selected from V_i ($1 \le i \le n$). Then for each

possible value in V_i, calculate the global density that assume the value of factor f_i is fixed as such a value. The value in V_i that takes the greatest global density will be selected and assigned to factor f_i. The same as the algorithm DA-RO, local densities, factor densities, and the global density should be modified after fixing value for each factor.

Above operation will repeat until all factors have been selected and the values of all these factors have been fixed. The process of generating a single test case is also described as the Algorithm 3. The problem of tie-breaking in steps that selecting factor and fixing value can be treated similar as the DA-RO.

Algorithm 3. Generate One Test Case by Fixing Value in the Order of Factors

Start with an empty test case *test*, in which the values of all factors have not been fixed;
While (there exists at least one factor whose value has not been fixed)
  For i=1 to n
    If (value of f_i has not been fixed) **then**
      Calculate factor density for $f_i \in F$;
    End If
  End For
  Select a new factor f_i with the greatest factor density;
  For Each value $v \in V_i$
    Calculate global density by assuming the value of f_i is fixed as v;
  End For
  Select a value v that takes the greatest global density;
  Fix factor f_i as the selected value v;
End While

Next, we analyze the time performance of algorithm DA-FO to check whether it is better than that of DA-RO. According to the definition of factor density, in the worst case, there are t local densities of all t coverage requirements should be calculated to obtain a factor density. So the worst time complexity of selecting a unfixed factor is $O(n \times t \times tm\_ld)$, for all unfixed factors are required to be compared. And there are a_i global densities required to be calculated when selecting value for the selected factor f_i ($1 \le i \le n$), for there are a_i possible values in V_i. So the worst time complexity of DA-FO to generate a single test case is $O(t \times tm\_ld \times (n^2+\sum_{i=1}^{n} a_i))$, which is better than DA-RO when $(n^2+\sum_{i=1}^{n} a_i)<(t+\sum_{k=1}^{t}\prod_{fi \in rk} a_i)$.

VI. EXPERIMENTS

To assess the efficiency of proposed algorithms, we compare them to some existed algorithms and tools. We experiment with a computer consisting of 2.66GHz Pentium IV processor and 1G memory.

In first experiment, in order to assess their practicality in variable strength combinatorial test generation, we compare two proposed algorithms to UNION, GREEDY, PICT, and TVG. Besides, the ReqOrder and ParaOrder, two algorithms that were proposed in our earlier paper [13], are also included in this experiment. The tools PICT and TVG are both downloaded from internet. The algorithms UNION and GREEDY are implemented according to their description [15]. In the implementation of UNION, when constructing test suite for a single coverage requirement, the values of factors that excluded in such coverage requirement will be selected randomly. And in the implementation of GREEDY, although the problem reduction technique is not included, but the time and space performance of this version has been improved to be much better than that of earlier version[13].

Though there are some published experimental results about UNION and GREEDY in relevant literature [15], but the inputs of this experiment are not published. Therefore, the inputs of experiment have to be designed ourselves. In the first step, two factor set $F_1=\{3^{10}\}$ and $F_2=\{2^3\times3^3\times4^3\times5\}$ are chosen to represent the systems with fixed-level factors and mixed-level factors respectively. And in the second step, we create the interaction relationship by selecting a given number of coverage requirements from a pool of coverage requirements (see Appendix). There are two reasons why the sizes of all coverage requirements in the pool range from 2 to 4. First, D. R. Kuhn et al claimed that the FTFI number (strength of failure-triggering fault interaction) in most systems are usually less than 4~6, and most faults were triggered by the interactions with low strength [18][19]. Second, rather than capabilities of different algorithms, coverage requirements with high strength may have a greater impact on size of generated test suite.

Table 1 and Table 2 show the sizes of generated test suites and the time consumed of each algorithm. The size of test suite generated by TVG is obtained by selecting a best one from totally 10 runs. The consumed time of TVG is not available, since we can not measure it from GUI exactly. And for PICT, though it is claimed that "people can define as many sub-models as they want" in model file, but in fact, PICT is too inefficient to use in practice when the number of coverage requirements is much (especially when it is more than 2) and there is intersection between different coverage requirements. So, the most data about PICT is not available since it require excessive amount of time (more than 1 hour).

As displayed in those tables, though GREEDY can generate the smallest test suite almost all the time, its consumed time is much longer than that of all the other algorithms. DA-RO generates the smallest test suites for more than half cases (9 of all 16 inputs), and DA-FO generates smallest test suites 7 times. In the aspect of time performance, the consumed time of DA-FO is always less than that of DA-RO, which supports the theoretical results about the time complexity of each algorithm. We can also find out that for some inputs, the ParaOrder and TVG can also generate small test suite with an excellent time performance. And the sizes of test suite generated by the ReqOrder and UNION are always much bigger than others.

Rather than the first experiment, some further ones are also designed to assess the practicality of proposed

TABLE 1.
COMPARISON OF DIFFERENT ALGORITHMS FOR $F=\{3^{10}\}$ AND DIFFERENT SIZE OF R

$\|R\|$	DA-RO	DA-FO	ReqOrder	ParaOrder	Union	Greedy	TVG	PICT
2	81 (0.08s)	81 (0.02s)	81 (0.01s)	81 (0.01s)	162 (0.01s)	81 (1.04s)	81 (-)	81 (0.69s)
3	81 (0.11s)	81 (0.02s)	81 (0.01s)	81 (0.01s)	242 (0.01s)	81 (1.91s)	84 (-)	-
10	86 (0.44s)	84 (0.07s)	99 (0.01s)	96 (0.01s)	503 (0.01s)	93 (6.77s)	86 (-)	-
20	95 (0.98s)	99 (0.14s)	128 (0.01s)	105 (0.02s)	858 (0.02s)	91 (18.0s)	105 (-)	-
30	116 (1.98s)	120 (0.30s)	157 (0.01s)	111 (0.04s)	1599 (0.04s)	109 (34.0s)	125 (-)	-
40	126 (3.02s)	123 (0.44s)	163 (0.01s)	120 (0.05s)	2057 (0.06s)	111 (50.7s)	135 (-)	-
50	135 (3.74s)	135 (0.60s)	172 (0.04s)	132 (0.06s)	2635 (0.10s)	125 (70.2s)	139 (-)	-
60	141 (4.96s)	142 (0.77s)	190 (0.05s)	144 (0.08s)	3257 (0.15s)	141 (110s)	150 (-)	-

TABLE 2.
COMPARISON OF DIFFERENT ALGORITHMS FOR $F=\{2^3\times3^3\times4^3\times5\}$ AND DIFFERENT SIZE OF R

$\|R\|$	DA-RO	DA-FO	ReqOrder	ParaOrder	Union	Greedy	TVG	PICT
2	64 (0.04s)	64 (0.01s)	64 (0.01s)	64 (0.01s)	104 (0.01s)	64 (1.35s)	64 (-)	64 (0.71s)
3	144 (0.24s)	144 (0.04s)	144 (0.01s)	144 (0.01s)	248 (0.01s)	144 (2.06s)	144 (-)	-
10	144 (0.71s)	144 (0.11s)	144 (0.01s)	144 (0.02s)	505 (0.01s)	144 (8.60s)	144 (-)	-
20	160 (1.78s)	160 (0.23s)	166 (0.01s)	161 (0.03s)	929 (0.01s)	160 (26.3s)	161 (-)	-
30	165 (4.35s)	175 (0.50s)	204 (0.01s)	179 (0.06s)	1861(0.06s)	162 (66.8s)	179 (-)	-
40	165 (5.67s)	172 (0.65s)	209 (0.02s)	183 (0.11s)	2244(0.08s)	167 (111s)	181 (-)	-
50	182 (7.96s)	186 (0.87s)	229 (0.02s)	200 (0.14s)	2820(0.13s)	183 (161s)	194 (-)	-
60	197 (11.15s)	200 (1.22s)	237 (0.05s)	204 (0.16s)	3587(0.21s)	197 (259s)	209 (-)	-

algorithms in fixed strength and "narrow sense" variable strength combinatorial test generation, which could be considered as the special cases of variable strength combinatorial test generation. In the following experiments, the UNION and ReqOrder will not be included, since the sizes of test suite that generated by them are always much bigger.

Table 3 depicts the sizes of 3-way combinatorial test suites that generated by two proposed algorithms and 11 existed algorithms and tools. In such table, the data of AETG, GA (generic algorithm) and ACA (ant colony algorithm) is collected from [11], and that of GA-N and IPO-N is collected from [7]. The data of IPO[8] is obtained by running a tool TConfig[1], which integrates an algebraic recursive algorithm and an in-parameter-order strategy based algorithm. And the data of Jenny[2] is obtained by running an open-sourced program.

As displayed in Table 3, DA-FO can generate smallest test suite for S_2, and generate smaller test suites, which are only a bit larger than the smallest ones, for some inputs such as S_6, S_7 and S_8. And for most inputs except S_2 and S_8, the test suites generated by DA-RO are a bit larger than that generated by DA-FO. Therefore, it could be concluded that, the performance of two proposed algorithms approximate to many classic algorithms for fixed strength combinatorial testing. And compared to ParaOrder, PICT and TVG, two proposed algorithms can generated smaller test suite almost all the time. GREEDY can generate the smallest test suites sometimes, but its bad time performance is also a main shortcoming. For example, it consumed nearly 1 hour for input S_5.

Table 4 displays the size of "narrow sense" variable strength combinatorial test suites that generated by 7 algorithms and tools. In which, the data of SA (simulated annealing) is collected from [1]. And some data about GREEDY is not available since it require excessive amount of time (more than 1 hour).

As demonstrated in Table 4, we find out SA generates the smallest test suite for all inputs. When ignoring the data about SA, DA-RO generates the smallest test suites for 17 of all 22 inputs, and DA-FO generates the smallest ones for 12 inputs. Besides, the PICT, TVG and ParaOrder can generate the smallest test suites for 1, 4 and 7 inputs respectively, and GREEDY can generate the smallest ones for only 1 of 8 valid inputs. Therefore, it could be concluded from experimental result that, the test suites generated by two proposed algorithms are much smaller than that generated by PICT, TVG, ParaOrder, and even GREEDY, though their performances are worse than that of SA in the field of "narrow sense" variable strength combinatorial test generation.

In a conclusion, it is clear that two proposed algorithms have some advantages in variable strength combinatorial test generation. Besides, they are also competitive in fixed strength and "narrow sense" variable strength combinatorial test generation. The experiment results also suggest that DA-RO usually generate smaller test suite than DA-FO in variable strength (include "narrow sense" variables strength) combinatorial test generation, while DA-FO is usually better in fixed strength combinatorial test generation.

VII. CONCLUSION

Variable strength combinatorial testing, which has been proposed in our earlier paper, may avoid some limitation of existing classic combinatorial testing models including fixed strength combinatorial testing and "narrow sense" variable strength combinatorial testing.

[1] http://www.site.uottawa.ca/~awilliam/
[2] http://burtleburtle.net/bob/math/jenny.html

TABLE 3.

SIZES OF GENERATED 3-WAY FIXED STRENGTH COMBINATORIAL TEST SUITES

	DA-RO	DA-FO	ParaOrder	Greedy	TVG	PICT	AETG	GA	ACA	GA-N	IPO-N	IPO	Jenny
S_1	50	47	53	43	48	48	38	33	33	52	47	48	51
S_2	64	64	106	64	120	111	77	64	64	85	64	64	112
S_3	213	211	225	184	239	215	194	125	125	223	173	200	215
S_4	362	359	363	325	409	369	330	331	330	389	371	366	373
S_5	1592	1587	1624	1474	1949	1622	1473	1501	1496	1769	1502	1678	1572
S_6	242	237	225	220	269	241	218	218	218	336	199	239	236
S_7	119	116	108	106	133	119	114	108	106	120	113	120	130
S_8	365	369	377	388	429	368	377	360	361	373	368	464	397

(S_1: 3^6; S2: 4^6; S3: 5^6; S_4: 6^6; S_5: 10^6; S_6: 5^7; S_7: $5^2 4^2 3^2$; S_8: $10^1 6^2 4^3 3^1$)

TABLE 4.

SIZES OF GENERATED "NARROW SENSE" VARIABLE STRENGTH COMBINATORIAL TEST SUITES

	C	DA-RO	DA-FO	ParaOrder	Greedy	TVG	PICT	SA
VSCA(*m*;2, 3^{15},*C*)	∅	21	20	33	-	22	35	16
	CA(3, 3^3)	28	29	27	-	27	81	27
	CA(3, 3^3)^2	28	29	33	-	30	729	27
	CA(3, 3^3)^3	28	30	33	-	30	785	27
	CA(3, 3^4)	32	34	27	-	35	105	27
	CA(3, 3^5)	40	42	48	-	41	131	33
	CA(3, 3^4), CA(3, 3^5), CA(3, 3^6)	46	46	49	-	53	1376	34
	CA(3, 3^6)	46	46	53	-	48	146	34
	CA(3, 3^7)	53	53	54	-	54	154	41
	CA(3, 3^9)	60	60	62	-	62	177	50
	CA(3, 3^{15})	70	78	82	-	81	83	67
VSCA (*m*;2, $4^3 5^3 6^2$,*C*)	∅	41	40	40	44	44	43	36
	CA(3, 4^3)	64	64	64	67	67	384	64
	CA(3, $4^3 5^2$)	131	132	140	119	132	781	100
	CA(3, 5^3)	125	125	125	126	125	750	125
	CA(3, 4^3), CA(3, 5^3)	125	125	129	126	125	8000	125
	CA(3, $4^3 5^3 6^1$)	207	211	220	209	237	1266	171
	CA(3, $5^1 6^2$)	180	180	180	181	180	900	180
	CA(3, $4^3 5^3 6^2$)	256	261	264	258	302	261	214
VSCA (*m*;2, $3^{20} 10^2$,*C*)	∅	100	100	100	-	101	100	100
	CA(3, 3^{20})	100	105	119	-	103	940	100
	CA(3, $3^{20} 10^2$)	401	409	445	-	423	423	304

The reason is that such a new model makes more sufficient consideration on actual interaction relationship in software. To address the problem of variable strength combinatorial test generation, this paper proposed two test generation algorithms based on one-test-at-a-time strategy. The experience results show the advantages of two proposed algorithms in both the aspect of size of generated test suite and the aspect of time performance. And rather than the variable strength combinatorial test generation, two proposed algorithms are also available in fixed strength and "narrow sense" variable strength combinatorial test generation.

Above all, many works on combinatorial testing have been done in recent years, but there are also many problems to study in the future. The first one is test generation for different model of combinatorial testing, and there is a limitation that most works in this field focus on pair-wise testing. Secondly, the combinatorial testing techniques for test prioritization, constraint handling, and fault location are also important. Furthermore, automatic integration tool for combinatorial testing, which need to support the automation of test generation, execution, measurement, and fault location etc, is also required to be developed.

APPENDIX

There are totally10 factors in both F_1 and F_2, that is $F=\{f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, f_{10}\}$. We map these 10 factors to its sequence number for simplicity, so we could described them as $F=\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

The pool of coverage requirements that used in our first experiment is:

POOL={{1, 2, 7, 8}, {0, 1, 2, 9}, {4, 5, 7, 8}, {0, 1, 3, 9}, {0, 3, 8}, {6, 7, 8}, {4, 9}, {1, 3, 4}, {0, 2, 6, 7}, {4, 6}, {2, 3, 4, 8}, {2, 3, 5}, {5, 6}, {0, 6, 8}, {8, 9}, {0, 5}, {1, 3, 5, 9}, {1, 6, 7, 9}, {0, 4}, {0, 2, 3}, {1, 3, 6, 9}, {2, 4, 7, 8}, {0, 2, 6, 9}, {0, 1, 7, 8}, {0, 3, 7, 9}, {3, 4, 7, 8},

{1, 5, 7, 9}, {1, 3, 6, 8}, {1, 2, 5}, {3, 4, 5, 7}, {0, 2, 7, 9}, {1, 2, 3}, {1, 2, 6}, {2, 5, 9}, {3, 6, 7}, {1, 2, 4, 7}, {2, 5, 8}, {0, 1, 6, 7}, {3, 5, 8}, {0, 1, 2, 8}, {2, 3, 9}, {1, 5, 8}, {1, 3, 5, 7}, {0, 1, 2, 7}, {2, 4, 5, 7}, {1, 4, 5}, {0, 1, 7, 9}, {0, 1, 3, 6}, {1, 4, 8}, {3, 5, 7, 9}, {0, 6, 7, 9}, {2, 6, 7, 9}, {2, 6, 8}, {2, 3, 6}, {1, 3, 7, 9}, {2, 3, 7}, {0, 2, 7, 8}, {0, 1, 6, 9}, {1, 3, 7, 8}, {0, 1, 3, 7}}.

ACKNOWLEDGMENT

The work described in this paper is supported by the National Natural Science Foundation of China (61003020, 61300054); Natural Science Foundation of Jiangsu Province (BK2011190, BK20130879); Natural Science Foundation for Colleges and Universities in Jiangsu Province (13KJB520018); Foundation of Nanjing University of Posts and Telecom. (NY212023); Open Foundation of Guangxi Key Lab of Trustworthy Software (KX201328).

REFERENCES

[1] M. B. Cohen, P. B. Gibbons, W. B. Mugridge, C. J. Colbouns, J. S. Collofello. Variable Strength Interaction Testing of Components. In Proceedings of 27th Annual International Computer Software and Applications Conference (COMPSAC2003): 413-418.

[2] D. M. Cohen, S. R. Dalal, M. L. Fredman, G. C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. IEEE Transaction on Software Engineering, 1997, 23(7): 437-444.

[3] Yu-Wen Tung, Wafa S. Aldiwan. Automating Test Case Generation for the New Generation Mission Software System. In Proceedings of IEEE Arospace Conference, 2000: 431-437.

[4] Chuanqi Tao, Bixin Li, Jerry Gao. A Systematic State-Based Approach to Regression Testing of Component Software. Journal of Software, 2013, 8(3): 560-571.

[5] R C. Bryce, C. J. Colbourn. The Density Algorithm for Pairwise Interaction Testing. Software Testing, Verification and Reliability, 2007, 17(3): 159-182.

[6] J. Czerwonka. Pairwise Testing in Real World: Practical Extensions to Test Case Generator. In Proceedings of 24th Pacific Northwest Software Quality Conference, Portland, Oregon, USA, October 9-11, 2006: 419-430.

[7] Changhai Nie, Baowen Xu, Liang Shi, Guowei Dong, Automatic Test Generation for N-way Combinatorial Testing. Second International Workshop on Software Quality (SOQUA2005): 203-211.

[8] Kuo-Chung Tai, Yu Lei. A Test Generation Strategy for Pairwise Testing. IEEE Transaction on Software Engineering, 2002, 28(1): 109-111.

[9] Yu Lei, K. C. Tai. In-Parameter-Order: A Test Generation Strategy for Pairwise Testing. In: Proceedings 3rd IEEE International Symposium on High-Assurance Systems Engineering, November 13-14, 1998: 254-261.

[10] R. C. Bryce, C. J. Colbourn. One-test-at-a-time Heuristic Search for Interaction Test Suites. In Proceedings of 9th Annual Conference on Genetic and Evolutionary Computation (GECCO2007): 1082-1089.

[11] Toshiaki Shiba, Tatsuhiro Tsuchiya, Tohru Kikuno. Using Artificial Life Techniques to Generate Test Cases for Combinatorial Testing. In Proceedings of 28th Annual International Computer Software and Applications Conference (COMPSAC2004), Volume 1: 72-78.

[12] J. Bach, P. J. Schreoder. Pairwise testing: A Best Practice That Isn't. In Proceedings of 22nd Pacific Northwest Software Quality Conference: 180-196.

[13] Wang Ziyuan, Nie Changhai, Xu Baowen. Generating Combinatorial Test Suite for Interaction Relationship. In: Proceeding of 4th International Workshop on Software Quality Assurance (SOQUA 2007): 55-61.

[14] P. J. Schroeder, B. Korel. Black-Box Test Reduction Using Input-Output Analysis. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA2000):21-22.

[15] P. J. Schroeder. Black-box Test Reduction Using Input-output Analysis. Dissertation for PhD, Department of Computer Science, Illinois Institute of Technology, Chicago, Illinois, USA, December, 2001.

[16] C. Cheng, A. Dumitrescu, P. J. Schroeder. Generating Small Combinatorial Test Suites to Cover Input-Output Relationships. In Proceedings of 3rd International Conference on Quality Software (QSIC2003): 76-82.

[17] Wang Zi-Yuan, Nie Chang-Hai, Xu Bao-Wen, Shi Liang. Optimal test suite generation methods for neighbor factors combinatorial testing. Chinese Journal of Computers, 2007, 30(2): 200-211.

[18] Kuhn D R, Reilly M J. An investigation of the applicability of design of experiments to software testing. In: Proceedings of 27th NASA/IEEE Software Engineering Workshop: 91-95.

[19] D R Kuhn, D R Wallace. Software fault interaction and implication for software testing. IEEE Transaction on Software Engineering, 2004, 30(6): 1-4.

[20] Wang Ziyuan, Xu Baowen, Nie Changhai. Greedy Heuristic Algorithms to Generate Variable Strength Combinatorial Test Suite. In Proceedings of the 8th International Conference on Quality Software (QSIC2008), Oxford, UK, August 12-13, 2008: 155-160.

[21] Chang-ai Sun. A Constraint-based Test Suite Reduction Method for Conservative Regression Testing. Journal of Software, 2011, 6(2): 314-321.

[22] Abdul Azim Abdul Ghani, Reza Meimandi Parizi. Aspect-Oriented Program Testing: An Annotated Bibliography. 2013, 8(6): 1281-1300.

[23] Changhai Nie, Hareton Leung. A Survey of Combinatorial Testing. ACM Computing Survey, 2011, 43(2).

[24] Ziyuan Wang, Lin Chen, Baowen Xu, Yan Huang. Cost-Cognizant Combinatorial Test Case Prioritization. International Journal of Software Engineering and Knowledge Engineering, 2011, 21(6): 829-854.

Ziyuan Wang (male, born in 1982) received his Ph. D. degree in Computer Science & Technology from Southeast University in 2009. He was a post-doctoral fellow in Nanjing University form 2009 to 2012. Now he works for School of Computers, Nanjing University of Posts and Telecommunications. His research interests include software testing.

Haixiao He (male, born in 1992) is an undergraduate student in the School of Computers, Nanjing University of Posts and Telecommunications.