Incremental Generation and Prioritization of t-way Strategy for Web Based Application

Mrs. B. Vani[#1] Dr. R. Deepalakshmi[#2]

[#1] PG Scholar, Department of CSE, Velammal College of Engineering and Technology, Madurai, Tamil Nadu, INDIA
vani.sarguru@gmail.com

[#2] Professor, Department of CSE, Velammal College of Engineering and Technology, Madurai, Tamil Nadu, INDIA
jei.deepa@gmail.com

Abstract - The adoption of t-way strategies termed as interaction testing for combinatorial testing is the main focus of this paper. Earlier work focus only on pairwise testing and the interaction coverage achieved is also not hundred percent. This paper discusses the different t-way strategies of uniform strength, variable strength, cumulative variable strength interactions for the test suite generation and prioritization for the web based application. The paper highlights the t-way strategy implementation by two different algorithms one for the minimal test suite generation and other for test suite prioritization with the step-by-step example with experimental results and analysis.

Keywords - software testing, interaction testing, t-way strategy, combinatorial testing

I. INTRODUCTION

Combinatorial testing [1, 2] creates tests by selecting values for input parameters and by combining these values. For a system with k parameters, each of which has v values, the number of possible combinations of values of these parameters is v^k. Owing to resource constraints it is nearly impractical to exhaustively test all possible combinations [3, 4]. Thus a strategy is needed to select a subset of combinations to be tested. One such strategy, called t-way testing[5], requires every combination of values of any t parameters covered by at least one test, where t is referred to the strength of coverage and takes a small value. Each combination of values of a set of parameters is considered to represent one possible interaction among these parameters. The rationale behind t-way testing is not that every parameter contributes to every fault. The notion of t-way testing can substantially reduce the number of tests. For example, a system of 20 parameters that have 10 values each requires 10^{20} tests for exhaustive testing. It requires about 180 tests for 2-way in pairwise testing [6, 7]. Empirical studies have shown that t-way testing can effectively detect faults and generate better test suites in various types of applications. The next requirement is to prioritize the generated minimal test suite providing hundred percent of interaction coverage.

The paper is structured as follows. Section 2 gives the insight about the test suite generation and the exhaustive combination strategies. Section 3 explains the idea behind the new construction t-way strategy. Section 4 gives the details about the proposed algorithm for the test suite generation and the logic behind. Section 5 provides the details for test suite prioritization. Section 6 gives the experimental results and analysis. Section 7 draws our conclusive statements.

II. TEST SUITE GENERATION

Test suite generation for the web based banking application involves different parameters and settings. Considering the t-tuples in the suite originating from poorly combined strategies the resulting test suite will then be bigger than necessary.

A. TEST SUITE GENERATION THROUGH EXHAUSTIVE COMBINATIONS

Consider the running example of bank mortgaging application. There are 4 options or parameters with 2 settings each derived from the mathematical function f1 = (A,B,C,D) and f2 = f1(L1,L2). The Actual values of the parameters and their settings along with the symbolic representation are given in the table.

TABLE I
Parameters and Value Conversion

Actual Parameters and their values	**Symbolic Representation**
Mort App = { Customer , Property }	A = { a1, a2 }
Customer = { Income , CreditRating }	B = { b1, b2 }
Property = { Type , Location }	C = { c1, c2 }
Loan = { Term , Amount }	D = { d1, d2 }

Test suite generation by exhaustive strategy requires 2^4 = 16 combinations for the above base data values. The following table shows the exhaustive combinations of test suites.

TABLE II
Exhaustive Combinations

	Input Variables			
	A	B	C	D
Base Values	a1	b1	c1	d1
	a2	b2	c2	d2
All Combinatorial Values	a1	b1	c1	d1
	a1	b1	c1	d2
	a1	b1	c2	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c1	d2
	a1	b2	c2	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c1	d2
	a2	b1	c2	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c1	d2
	a2	b2	c2	d1
	a2	b2	c2	d2

III.TEST SUITE GENERATION THROUGH T-WAY STRATEGY

Combinatorial testing has raised from this tenet as a technique to sample, in a systematic way, some subset of the input or configuration space. In combinatorial testing, the parameters and their settings are modeled as set of factors or parameters and values; for each factor f_i, a set of values are defined $\{x_1, x_2, \ldots x_j\}$, that partition the factor space. From this model test suites[8, 9] or specific program configuration are generated by selecting a subset of the Cartesian product of the values for all factors; an application with three factors and three settings each has 3*3*3 = 27 combinations.

For the next level of interaction with three more settings have 27*27*27 = 19683 possible test suite combinations for 2-way interaction. Considering the next level of interaction with three more settings the number of combinations gets increased to (27*3)*(27*3)*(27*3) = 5,31,441 possible test suite combinations that becomes very vast for exhaustive testing and pairwise testing. In order to reduce the test suite combinations the t-way strategy finds a suitable approach for the complex web based banking application.

A. OVERVIEW OF T-WAY STRATEGIES

The adoption of t-way strategies [10] termed as interaction testing for combinatorial testing of web based application for test suite generation is the main focus. The main aim of any t-way strategy is to cover the interaction tuples of interest in an optimal manner i.e. at most once whenever possible and hence systematically minimizing the test suites for testing consideration.

B. TYPES OF T-WAY STRATEGIES FOR TEST SUITE GENERATION

- Uniform strength
- Variable strength

Uniform strength t-way interaction:

As highlighted earlier in survey, uniform strength interaction [11] forms the basis of interaction testing, where all input parameters are assumed to be uniformly interacting (i.e. with constant interaction strength throughout).

Variable Strength t-way interaction:

Unlike uniform strength interaction counterparts, variable strength interaction [12] considers more than one interaction strength in the test suite generation process. Practically, a particular subset of input parameters can have a higher interaction dependency than other parameters.

IV. CONSTRUCTION OF T-WAY TEST SUITES

The approach for the construction of a t-way covering test suite chosen in this work is based on the assumption of the large input parameter interactions [13,14]. In this context instead of building the test suite by either greedy or algorithmic approach we use incremental approach and the construction process is based on two computational steps, namely the expansion and contraction stages.

Expansion Stage: build up t-way covering test suite T by enumerating all combinatorial requirements, one per each row. As only t parameters are involved in each required combination, all others will be left unassigned.

Contraction Stage: search for the effective way to combine compatible rows together while preserving the coverage, in order to reduce the total number of rows, i.e. the total number of requited test suites.

Two rows are said to be compatible if each corresponding position in the row is either assigned to the same value or it has not been assigned don't care values in at least one of the two rows [15,16]. The peculiarity of this approach is that it builds up an intermediate test suite enumerating all t-way combinations for the parameters of the application. The final test suite is derived from the intermediate one by finding the right way to effectively merge together as many compatible rows as possible in order to reduce the total number of rows appearing in the final test suite.

At beginning the intermediate test suite does not have as many compatible rows [17] that will have to be merged. By giving the intermediate test suite the maximal redundancy that are a number of compatible rows that could be merged together and the contraction stage removes the compatible rows and performs several merge sequences in parallel to achieve the optimal test suite.

A. TEST SUITE GENERATION ALGORITHM

```
1    reduceThread(TestSuite T) {
2    T1 = local copy of T ;
3    for each row r1 in T1 {
4    for each row r2 in T1-r1 {
5    if compatible (r1,r2) {
6*   rotate = random Boolean ;
7*   if (rotate) start new reduceThread(T1) ;
8    rem r2 in r1 ;
9    }
10   }
11   }
12*  If (T1.size() <current_min) {
13*  Discard Tmin ;
14*  Tmin = t1 ;
15*  }
16*  else discard T1 ;
17   }
```

Fig 1. Multithreaded Rem Algorithm for Test Suite Generation

Algorithm Explanation:

The Original single threaded algorithm computes only one merge sequence. Parameter passed as the input value to reduceThread is previously computed, redundant test suite. The Multithreaded rem algorithm computes several possible merge sequences in parallel (marked by *). Algorithm reduceThread() searches for rows that can be merged and hence removed. In order to ensure termination it proceeds merging compatible rows with the incremental approach.

Each merge will add to test suite a completely new row. This added new row determines potential change in compatibility relations. It is not possible to determine in advance if choice(merge(r1,r2)) was the best possible until reduce Thread() is finished. In order to overcome this limitation before applying change to the local copy of test suite a new thread is started i.e. given the current version of test suite.

The new thread will skip the change and search for alternative chance of merging. The instances of the reduceThread are run in parallel. At termination, each thread will compare its resulting test suite with the one currently minimal and stored in a repository. The replacements in the repository are done in case of improvements.

The Speed up of multithreaded rem algorithm with respect to single thread execution is bound to number N i.e. N merge sequences at the same time. A running threads counter is used to monitor the termination of the whole process. Although the number of occurrences of thread will be two times the total number of thread instances and it is controlled using the stochastic variable rotate.

B. TEST SUITE GENERATION BY UNIFORM STRENGTH t-WAY INTERACTION

Consider the interaction strength t=2 and t=3 and the tables below illustrate how the test suites are generated for both and reduction is achieved.

For t=2 the interaction is broken down to AB, AC, AD, BC, BD, CD. The other two parameters in each of the set considers don't care values i.e. they both are not varied. Combining these results, note that there are some repetitions for the entries and remove those repetitive entries.

Base Values	Input Variables				+	Base Values	Input Variables			
	A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2
Combinatorial Values For AB, t=2	a1	b1	**c1**	**d1**		Combinatorial Values For AC, t=2	~~a1~~	~~**b1**~~	~~c1~~	~~**d1**~~
	a1	b2	**c2**	**d2**			~~a1~~	~~**b2**~~	~~c2~~	~~**d2**~~
	a2	b1	**c1**	**d1**			~~a2~~	~~**b1**~~	~~c1~~	~~**d1**~~
	a2	b2	**c2**	**d2**			~~a2~~	~~**b2**~~	~~c2~~	~~**d2**~~
Base Values	Input Variables				+	Base Values	Input Variables			
	A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2
Combinatorial Values For AD, t=2	~~a1~~	~~**b1**~~	~~**c1**~~	~~d1~~		Combinatorial Values For BC, t=2	~~**a1**~~	b1	~~c1~~	~~**d1**~~
	~~a1~~	~~**b2**~~	~~**c2**~~	~~d2~~			**a1**	b1	c2	**d2**
	a2	**b1**	**c1**	d1			**a2**	b2	c1	**d1**
	~~a2~~	~~**b2**~~	~~**c2**~~	~~d2~~			~~**a2**~~	~~b2~~	~~c2~~	~~**d2**~~
Base Values	Input Variables				+	Base Values	Input Variables			
	A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2
Combinatorial Values For BD, t=2	~~**a1**~~	~~b1~~	~~**c1**~~	~~d1~~		Combinatorial Values For CD, t=2	~~**a1**~~	~~**b1**~~	~~c1~~	~~d1~~
	~~**a1**~~	~~b1~~	~~**c2**~~	~~d2~~			**a1**	**b2**	c1	d2
	~~**a2**~~	~~b2~~	~~**c1**~~	~~d1~~			**a2**	**b1**	c2	d1
	~~**a2**~~	~~b2~~	~~**c2**~~	~~d2~~			~~**a2**~~	~~**b2**~~	~~c2~~	~~d2~~

On removing the repetitions the test suite generated is given in the following table.

= Removing Repetitions				
Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values With t=2	a1	b1	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b2	c2	d2
	a2	b1	c1	d1
	a1	b1	c2	d2
	a2	b2	c1	d1
	a1	b2	c1	d2
	a2	b1	c2	d1

Total Test Suites = 9

TABLE III
Uniform t-way Interaction Result (t=2)

For Interaction strength t=3 consider the test suite generation

Base Values	Input Variables				+	Base Values	Input Variables			
	A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2
Combinatorial Values for ABC, t=3	a1	b1	c1	**d1**		Combinatorial Values for ACD, t=3	~~a1~~	~~**b1**~~	~~c1~~	~~d1~~
	a1	b1	c2	**d2**			~~a1~~	~~**b2**~~	~~c2~~	~~d2~~
	a1	b2	c1	**d1**			~~a1~~	~~**b2**~~	~~c1~~	~~d1~~
	a1	b2	c2	**d2**			~~a1~~	~~**b1**~~	~~c2~~	~~d2~~
	a2	b1	c1	**d1**			~~a2~~	~~**b1**~~	~~c1~~	~~d1~~
	a2	b1	c2	**d2**			~~a2~~	~~**b1**~~	~~c2~~	~~d2~~
	a2	b2	c1	**d1**			~~a2~~	~~**b2**~~	~~c1~~	~~d1~~
	a2	b2	c2	**d2**			~~a2~~	~~**b2**~~	~~c2~~	~~d2~~
+										
Base Values	Input Variables				+	Base Values	Input Variables			
	A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2
Combinatorial Values for ABD, t=3	~~a1~~	~~b1~~	~~**c1**~~	~~d1~~		Combinatorial Values for BCD, t=3	~~**a1**~~	~~b1~~	~~c1~~	~~d1~~
	a1	b1	**c1**	d2			~~**a2**~~	~~b1~~	~~c1~~	~~d2~~
	a1	b2	**c2**	d1			~~**a1**~~	~~b1~~	~~c2~~	~~d1~~
	a1	b2	**c1**	d2			~~**a1**~~	~~b1~~	~~c2~~	~~d2~~
	~~a2~~	~~b1~~	~~**c1**~~	~~d1~~			~~**a2**~~	~~b2~~	~~c1~~	~~d1~~
	a2	b1	**c1**	d2			~~**a1**~~	~~b2~~	~~c1~~	~~d2~~
	~~a2~~	~~b2~~	~~**c2**~~	~~d1~~			~~**a2**~~	~~b2~~	~~c2~~	~~d1~~
	a2	b2	**c2**	d2			~~**a2**~~	~~b2~~	~~c2~~	~~d2~~

For t=3 the interaction is broken down between parameters ABC, ABD, ACD and BCD. Here when the parameters ABC are considered, the parameter D takes don't care values. When the parameters ABD are considered, the parameter C takes don't care values. When the parameters ACD are considered, the parameter B takes don't care values. When the parameters BCD are considered, the parameter A takes don't care values. Combining these results, note that there are some repetitions for the entries and remove these repetitive entries.

= Removing Repetitions				
Base Values	Input Variables			
	A	B	C	D
	a1	b1	c1	d1
	a2	b2	c2	d2
Combinatorial Values with t=3	a1	b1	c1	d1
	a1	b1	c2	d2
	a1	b2	c1	d1
	a1	b2	c2	d2
	a2	b1	c1	d1
	a2	b1	c2	d2
	a2	b2	c1	d1
	a2	b2	c2	d1
	a1	b1	c1	d2
	a1	b2	c2	d1
	a1	b2	c1	d2
	a2	b1	c1	d2
	a2	b2	c2	d2

Total Test Suites = 13

TABLE IV
Uniform t-way Interaction Results (t=3)

C. TEST SUITE GENERATION BY VARIABLE STRENGTH t-WAY INTERACTION

Unlike uniform strength interaction counterparts, variable strength interaction considers more than one interaction strength in the test suite generation process [18]. Practically, a particular subset of input parameters can have a higher interaction dependency than other parameters (indicating failures due to the interaction of that subset may have more significant impact to the overall system). For example, consider the bank mortgaging application where the customers and the property interaction play a vital role.

Base Values	Input Variable					Base Values	Input Variable					Base Values	Input Variable			
	A	B	C	D			A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2			a2	b2	c2	d2
Comb Values t=2	a1	b1	c1	d1		Variable Strength Comb Values BCD t=3	~~a1~~	~~b1~~	~~c1~~	~~d1~~		Variable strength Comb Values	a1	b1	c1	d1
	a1	b2	c2	d2			a1	b1	c1	d2			a1	b2	c2	d2
	a2	b1	c1	d1			~~a2~~	~~b1~~	~~c2~~	~~d1~~			a2	b1	c1	d1
	a2	b2	c2	d2			a2	b1	c2	d2			a2	b2	c2	d2
	a2	b1	c1	d1	+		a2	b2	c1	d1	=		a2	b1	c1	d1
	a1	b1	c2	d2			~~a1~~	~~b2~~	~~c1~~	~~d2~~			a1	b1	c2	d2
	a2	b2	c1	d1			a2	b2	c2	d1			a2	b2	c1	d1
	a1	b2	c1	d2			~~a1~~	~~b2~~	~~c2~~	~~d2~~			a1	b2	c1	d2
	a2	b1	c2	d1									a2	b1	c2	d1
													a1	b1	c1	d2
													a2	b1	c2	d2
													a2	b2	c1	d1
													a2	b2	c2	d1

Total Test Suite = 9 Total Test Suite = 8 Total Test Suite = 13

TABLE V
Variable Strength Interaction

To provide stronger coverage those parameters are considered with interaction strength t=3 and the remaining parameters with interaction strength t=2 [19]. In this case, we can assign variable coverage strength to each subset of parameters and to the whole application.

To illustrate variable strength t-way interaction assumes the bank mortgage application example. Assume that all interaction is uniform at t=2 for all parameters. Then consider t=3 only for parameters B, C, D. Combining both interactions the yield result is shown in the table below. Here, the test suite has been reduced from 16 (for exhaustive case) to 13, a saving of 18.75 percent.

D. TEST SUITE GENERATION BY CUMULATIVE VARIABLE STRENGTH t-WAY INTERACTION

Base Values	Input Variable				+	Base Values	Input Variable				=	Base Values	Input Variable			
	A	B	C	D			A	B	C	D			A	B	C	D
	a1	b1	c1	d1			a1	b1	c1	d1			a1	b1	c1	d1
	a2	b2	c2	d2			a2	b2	c2	d2			a2	b2	c2	d2
Comb Values	~~a1~~	~~b1~~	~~c1~~	~~d1~~		Comb Values	a1	b1	c1	d1		Comb Values	a1	b1	c1	d1
	~~a1~~	~~b2~~	~~c2~~	~~d2~~			a1	b1	c2	d2			a1	b1	c2	d2
	~~a2~~	~~b1~~	~~c1~~	~~d1~~			a1	b2	c1	d1			a1	b2	c1	d1
	~~a2~~	~~b2~~	~~c2~~	~~d2~~			a1	b2	c2	d2			a1	b2	c2	d2
	~~a2~~	~~b1~~	~~c1~~	~~d1~~			a2	b1	c1	d1			a2	b1	c1	d1
	~~a1~~	~~b1~~	~~c2~~	~~d2~~			a2	b1	c2	d2			a2	b1	c2	d2
	~~a2~~	~~b2~~	~~c1~~	~~d1~~			a2	b2	c1	d1			a2	b2	c1	d1
	~~a1~~	~~b2~~	~~c1~~	~~d2~~			a2	b2	c2	d1			a2	b2	c2	d1
	a2	**b1**	**c2**	**d1**			a1	b1	c1	d2			a1	b1	c1	d2
							a1	b2	c2	d1			a1	b2	c2	d1
							a1	b2	c1	d2			a1	b2	c1	d2
							a2	b1	c1	d2			a2	b1	c1	d2
							a2	b2	c2	d2			a2	b2	c2	d2
													a2	**b1**	**c2**	**d1**

Total Test Suite = 9 Total Test Suite = 13 Total Test Suite =14

TABLE VI
Variable Strength Interaction (Cumulative t=2 and t=3) Results

As a special case of variable strength interaction, consider the cumulative strength t=3 and t=2. Combining the test suite it yields the result as shown in the table above. Note that the test suite t=2 is not necessarily the subset of test suite t=3. In this case the test suite has been reduced from 16 (for exhaustive case) to 14 proving 100 percent coverage and a saving of 12.5 percent.

V. TEST SUITE PRIORITIZATION

Web Based applications routinely undergo changes as part of their maintenance process. New versions of the applications are created as result of bug fixes or requirement modification. In such situation, a large number of test suites may be available from testing previous versions of the application which are often reused to test the new version of the application. However running such test suites may take a significant amount of time. A tester needs to select and execute only a subset of those test suites. Here comes need for test suite prioritization. Test suite prioritization is the process of scheduling the execution of test suites according to some criterion in order to satisfy the goal. We have already generated the CIT test suite that is already smaller in size. The next step is to prioritize the generated test suite in order to improve the likelihood that faults will be detected early in the testing process [20].

Consider the function for test prioritization as formally defined. Given T, a test suite, Π, the set of all test suites obtained by permuting the tests of T, and f, a function from Π to the set of real numbers, the problem is to find $\pi \in \Pi$ such that $\forall\ \pi' \in \Pi$, $f(\pi) \geq f(\pi')$. In this definition, Π refers to the possible prioritizations of T and f is a function that is applied to evaluate the orderings. The selection of the function f leads to many criteria to prioritize software tests. For instance, the prioritization criteria may consider frequency of occurrence. The applications that cover a smaller percentage of interactions in the test suites do not benefit from prioritization by interaction [21,22]. The interaction based prioritization results in fastest fault detection rate [23, 24] by providing hundred percent of interaction coverage using the metric APCC. The function order suite selects a test suite that covers the maximum number of criteria elements not yet covered by already selected test suites. The function iterates until all test suites have been ordered.

The function Order Suite takes four parameters:

1) The suite to be ordered – note the set from the repository
2) A function f that takes a single test suite as input and return set of elements of interest to the criterion.
3) Another function F that operates on the sequence of test suites S.
4) An operation ⊕ assigns a fitness value to test suite.

Function Order Suite starts with an unordered sequence and involves BestTests until all the test suites have been ordered.

	Input Parameters Suite : Test Suite to be prioritized f: Function returns criteria elements of a single test suite F: Function returns criteria elements in sequence of test suites ⊕ : Operation combines results of f and F. **Output:**Priority ordered sequence containing all test suites. **Computation:**
1	S ← EMPTY;
2	T ← Suite ;
3	Repeat
4	t ← BestTest(S,T,f,F,⊕);
5	S ← Insert at end (S,t);
6	T ← T – t ;
7	Until (T == ∅);
8	Ordered Sequence ← S;

Fig 2. Function – Order Suite

	Input Parameters: S : Priority ordered sequence of test suites selected T : set of remaining test suites f : Function returns criteria elements in single test F : Function returns criteria element in sequence of test suites ⊕ : Operation combines results of f and F **Output:** t= bestTest added to test suite **Computation:**
1	Max ← Min Int ;
2	f(s) ← F(s) ;
3	∀ x ∈ T {
4	y ← f(x) ⊕ fs ;
5	if (max < y) || (max ==y) && (Rand() <= 0.5) {
6	max ← y ;
7	t ← x ;
8	}
9	}
10	Return(t) ;

Fig 3. Function – bestTest

A. TEST SUITE PRIORITIZATION ALGORITHM

The Proposed test suite prioritization algorithm keeps a more than one test suite that covers largest number of currently uncovered t-tuples. In original algorithm ties being broken at random and the test suites later in the list have higher chance of getting picked [25, 26]. i.e. instead of $T_0 - T_{12}$ is picked where T_0 most important. Consider the case when all n tests cover the same amount of uncovered t-tuples. The CurrentMaximum is picked first. The probability of being picked = 0.5^n, since at each tie breaking point it has to win over the next test suite. In order to reduce the probability to 1/n the array denoted by # is added. The array holds the test suites that covers same amount of interactions. Boolean mapping is kept from test suites to t-tuples to mark currently uncovered t-tuples. These mappings are updated whenever a new test suite is marked. This avoids constantly recalculating the number of uncovered t-tuples for each test suite that is generated.

```
1   CA = test suite to prioritize
2   gather all valid t-tuples based on CA
3   mapping = []
4   order = []
5   for all tests in CA do
6   mapping[test] = [ true if t-tuples(i)  in test, else false]
7   order [test] = f(mapping[test])
8   end for
9   bestTest = a test function that covers the most unique t-tuples
10  bestTests = [bestTest]
11  add bestTest to TestSuite
12  selected TestCount = 1
13  while selected TestCount < size(CA) do
14  update order,mapping
15  remove order[bestTest], mapping[bestTest]
16  tCountMax = F(max(order))
17  bestTests = []
18  for all tests in order do
19  if order[test] == tCountMax then
20  add test to bestTests
21  end if
22  end for
23  bestTest = random test from bestTests
24  add bestTest to TestSuite
25  selected TestCount++
26  end while
```

Fig 4. Test Suite Prioritization Algorithm

B. TEST SUITE PRIORITIZATION FOR THE GENERATED MINIMAL TEST SUITE

Considering the generated test suite, prioritizing the test suite using the above algorithm the flow is describing by the following figure. The parameter Customer and Income is considered along with the frequency of occurrence then the order sequence is $T_0T_1T_8$. Considering the next parameter Customer and Credit rating the order sequence is $T_2T_3T_9T_{10}$.Considering the parameter Property and Income the order sequence is $T_4T_5T_{11}T_{13}$. Considering the parameter Property and Credit rating the order sequence is $T_6T_7T_{12}$. The ordered sequence is not the complete prioritized order. The next parameter is considered for ordering that sequence. Then the ordered sequence is $T_0T_8T_1 - T_2T_{10}T_9T_3 - T_4T_{11}T_{13}T_5 - T_6T_7T_{12}$. Then each of the sequence is added to bestTest and finally added to the prioritized TestSuite. The final prioritized order of the test suite is $T_0, T_8, T_1, T_2, T_{10}, T_9, T_3, T_4, T_{11}, T_{13}, T_5, T_6, T_7, T_{12}$.

Base Values	Input Variables				Test Suite		Base Values	Input Variables				Prioritized Order
	A	B	C	D				A	B	C	D	
	a1	b1	c1	d1				a1	b1	c1	d1	
	a2	b2	c2	d2				a2	b2	c2	d2	
Comb Values	a1	b1	c1	d1	T_0		Comb Values	a1	b1	c1	d1	T_0
	a1	b1	c2	d2	T_1			a1	b1	c1	d2	T_8
	a1	b2	c1	d1	T_2			a1	b1	c2	d2	T_1
	a1	b2	c2	d2	T_3			a1	b2	c1	d1	T_2
	a2	b1	c1	d1	T_4			a1	b2	c1	d2	T_{10}
	a2	b1	c2	d2	T_5			a1	b2	c2	d1	T_9
	a2	b2	c1	d1	T_6			a1	b2	c2	d2	T_3
	a2	b2	c2	d1	T_7			a2	b1	c1	d1	T_4
	a1	b1	c1	d2	T_8			a2	b1	c1	d2	T_{11}
	a1	b2	c2	d1	T_9			a2	b1	c2	d1	T_{13}
	a1	b2	c1	d2	T_{10}			a2	b1	c2	d2	T_5
	a2	b1	c1	d2	T_{11}			a2	b2	c1	d1	T_6
	a2	b2	c2	d2	T_{12}			a2	b2	c2	d1	T_7
	a2	b1	c2	d1	T_{13}			a2	b2	c2	d2	T_{12}

TABLE VII
Prioritized Test Suite with the order and Test Suite Number

C. INTERACTION COVERAGE METRIC

To calculate the t-way interaction coverage of a the given test suite we use the following formula. Rate = coverage / number of all valid t-tuples * 100%, whereas coverage = number of t-way interaction. Rate = 14 / 14 * 100% = 100% for the generated test suite of the web based banking mortgage application. To compare how quickly each prioritized test suite achieves the interaction coverage of specific strength the metric used is Average Percentage Covering array Coverage – APCC. APCC = ($1 - \sum_{i}^{m} \left(\frac{\text{CAi}}{\text{nm}} + \frac{1}{2n}\right)$) * 100, where m is the number of covering arrays and n is the number of test suites in each CA. It takes 14 test suites to achieve 100% coverage for both 3-way and pairwise interaction coverage i.e. 3-way and 2-way cumulative.

VI. EXPERIMENTAL RESULTS AND ANALYSIS

The Experimental results have shown that for the interaction strength t=2 and t=3 using the incremental generation algorithm the total number of test suites generated are 871 for the maximum domain size 10 that has covered about 2,31,516 combinations of parameters and settings with the execution time of 0.219 seconds with 100% coverage. Similarly for the IPO algorithm and its variants the numbers of test suites generated are very large for interaction strength t=2 and t=3. For t=2 the number of combinations is 6530 for the maximum domain size of 10 and the number of test suites generated is 1,72,881 considering the don't care values. The execution time is also 249 seconds. For t=3 the number of combinations is 2,31,516 for the maximum domain size of 10 and the number of test suites generated is 1,73,637. The execution time is also 406 seconds. The coverage achieved is also not 100%. From the above experimental results the incremental generation algorithm and prioritization algorithm performs the best and achieves speed up and 100% interaction coverage. Thus the incremental generation algorithm serves to be more efficient in terms of coverage and reduction of test suites. Since the number of test suites is already minimal the prioritization also consumes only less time and 100% coverage is achieved.

VII. CONCLUSION

The proposed technique for building the test suite of size t for the application under test focuses on the ability to reduce the number of test suites, merging the redundant t-tuples. The algorithm has intrinsic benefits over the traditional and greedy approaches. This algorithm is both mathematical and practical. It is surely able to arrive to the minimal test suite with 100% coverage for the web based banking application. The test suite prioritization algorithm uses the Boolean mapping to avoid recalculation of interaction benefit of the test suites every time. The Prioritization also preserves 100% coverage of combinations and verified by the CCM tool using the rate of coverage and APCC metric.

REFERENCES

[1] Nie, Changhai, and Hareton Leung. "A survey of combinatorial testing." ACM Computing Surveys (CSUR) 43.2 (2011): 11.

[2] Dumlu, Emine, et al. "Feedback driven adaptive combinatorial testing." Proceedings of the 2011 International Symposium on Software Testing and Analysis. ACM, 2011.

[3] Groce, Alex, et al. "Establishing flight software reliability: testing, model checking, constraint-solving, monitoring and learning." Annals of Mathematics and Artificial Intelligence 70.4 (2014): 315-349.

[4] Borazjany, Mehra N., et al. "An Input Space Modeling Methodology for Combinatorial Testing." Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on. IEEE, 2013.

[5] Yu, Linbin, et al. "Acts: A combinatorial test generation tool." Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. IEEE, 2013.

[6] Lopez-Herrejon, Roberto E., et al. "Comparative analysis of classical multi-objective evolutionary algorithms and seeding strategies for pairwise testing of software product lines." Evolutionary Computation (CEC), 2014 IEEE Congress on. IEEE, 2014.

[7] Oster, Sebastian, Florian Markert, and Philipp Ritter. "Automated incremental pairwise testing of software product lines." Software Product Lines: Going Beyond. Springer Berlin Heidelberg, 2010. 196-210.

[8] Fraser, Gordon, Andrea Arcuri, and Phil McMinn. "Test suite generation with memetic algorithms." Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference. ACM, 2013.

[9] Anand, Saswat, et al. "An orchestrated survey of methodologies for automated software test case generation." Journal of Systems and Software 86.8 (2013): 1978-2001.

[10] Al-Sewari, AbdulRahman A., and Kamal Z. Zamli. "An Orchestrated Survey on T-Way Test Case Generation Strategies Based on Optimization Algorithms." The 8th International Conference on Robotic, Vision, Signal Processing & Power Applications. Springer Singapore, 2014.

[11] Yilmaz, Cemal, et al. "Moving forward with combinatorial interaction testing." (2013): 1-1.

[12] Abdullah, Syahrul AC, Zainal HC Soh, and Kamal Z. Zamli. "Variable-Strength Interaction for T-Way Test Generation Strategy." International Journal of Advances in Soft Computing & Its Applications 5.3 (2013).

[13] Zhang, Zhiqiang, et al. "Generating combinatorial test suite using combinatorial optimization." Journal of Systems and Software (2014).

[14] Petke, Justyna, et al. "Efficiency and early fault detection with lower and higher strength combinatorial interaction testing." Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013.

[15] Abad, Pablo, et al. "Improving test generation under rich contracts by tight bounds and incremental SAT solving." Software Testing, Verification and Validation (ICST), 2013 IEEE Sixth International Conference on. IEEE, 2013.

[16] Anielak, Grzegorz, Grzegorz Jakacki, and Sławomir Lasota. "Incremental test case generation using bounded model checking: an application to automatic rating." International Journal on Software Tools for Technology Transfer (2014): 1-11.

[17] Chaturvedi, Animesh, and Atul Gupta. "A tool supported approach to perform efficient regression testing of web services." Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), 2013 IEEE 7th International Symposium on the. IEEE, 2013.
[18] Wang, Ziyuan, and Haixiao He. "Generating Variable Strength Covering Array for Combinatorial Software Testing with Greedy Strategy." Journal of Software 8.12 (2013): 3173-3181.
[19] Mayo, Quentin, Ryan Michaels, and Renee Bryce. "Test Suite Reduction by Combinatorial-Based Coverage of Event Sequences." Software Testing, Verification and Validation Workshops (ICSTW), 2014 IEEE Seventh International Conference on. IEEE, 2014.
[20] Jacob, T. Prem, and T. Ravi. "A Novel Approach for Test Suite Prioritization using Clustering." Indian Journal of Advances in Computer Sciences and Technology 1.1 (2013): 59-66.
[21] Huang, Rubing, et al. "Adaptive random prioritization for interaction test suites." Proceedings of the 29th Annual ACM Symposium on Applied Computing. ACM, 2014.
[22] Huang, Rubing, et al. "Prioritizing Variable-Strength Covering Array." COMPSAC. 2013.
[23] Badhera, Usha, and Annu Maheshwari. "PERFORMANCE ANALYSIS OF PRIORITIZED TEST SUITES BASED ON FAULT DETECTION." International Journal 3.4 (2014).
[24] Qu, Xiao, and Myra B. Cohen. "A study in prioritization for higher strength combinatorial testing." Software Testing, Verification and Validation Workshops (ICSTW), 2013 IEEE Sixth International Conference on. IEEE, 2013.
[25] Huang, Rubing, et al. "Prioritization of combinatorial test cases by incremental interaction coverage." International Journal of Software Engineering and Knowledge Engineering 23.10 (2013): 1427-1457.
[26] Huang, Rubing, et al. "Aggregate-Strength Interaction Test Suite Prioritization." Journal of Systems and Software (2014).