DOI: http://dx.doi.org/10.26483/ijarcs.v8i9.4977

ISSN No. 0976-5697

Volume 8, No. 9, November-December 2017

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

AN EFFICIENT BLACK-BOX REGRESSION MAXIMIZATION (BBM) FOR COMBINATORIAL TESTING USING GREEDY SEARCH ALGORITHM

K.Rekha
M.Phil Research Scholar, Department of Computer Science
Sri,RamakrishnaCollege of Arts and Science forWomen
Coimbatore, India

Dr.D.Gayathri Devi
Assistant Professor, Department of Computer Science
Sri,Ramakrishna College of Arts and Science for Women
Coimbatore, India

Abstract: Comparing behaviours of program models has become an important task in software maintenance and regression testing. Combinatorial testing focuses on recognizing faults that happen due to interaction of values of a small number of input parameters.In this paper presents the Black-Box Regression Maximization (BBM) Algorithm with Density-based Spatial Clustering Algorithm (DSC) using Greedy Search optimization algorithm focuses on combinatorial testing and proactively exposes behavioural deviations by checking inside block transitions. In this method presents new approach of BBM with Internal block transitions to measure the dissimilarity statements in large program data. To identify specific faults, an adaptive testing rule repeatedly constructs and tests configurations in order to determine, for each interaction of interest, whether it is faulty or not.

Keywords: Testing, Greedy, Black-box, DD Path.

1. INTRODUCTION

Software testing is an expensive and time consuming activity that leads to production of reliable software systems [1,2]. Due to its importance, testing process is allocated a large share of the software development resources [3]. However, it is often observed that when the usage of large data-intensive software increases, the modules which have passed conventional testing methods start developing undetected errors [4]. The possible reasons include addition of records with an oddball combination of values that has not occurred before in the software. It is observed that these rare combinations of values which have escaped testing process and usage of software can cause interaction failures. To avoid such failures, it is desirable to test all combinations of values in an exhaustive manner. However, exhaustive testing is not feasible either due to time or resources availability. Thus a technique is required that focuses on testing combination of values.

Testing software is a very important and challenging activity. Nearly half of the software production development cost is spent on testing. The main objective of software testing with clustering approach is to eliminate as many errors as possible to ensure that the tested software meets an acceptable level of quality.

Table 1: Pair-wise test set for a problem where each variable can have 2 possible values.

A	B	C
0	0	0
0	1	0
1	0	1
1	1	0

There are two types of interaction failures as defined in [11]. These are type 1 interaction failures and type 2 interaction failures. Type 1 interaction failures occur when a code segment in which a fault exists is executed. Due to interaction among variables, the faulty code is executed. For the pseudo code given in Fig. 1, a software system is observed to fail only for a set of customers residing at a particular location. Due to an interaction of two or more variables, a block of code is executed in which fault exists. Type 2 interaction failures occur when performing some computation on two or more variables leads to an incorrect result.

Combinatorial (t-way) testing focuses on testing combinations of values. It is based on the observation that a large number of faults are caused by interactions of a few input parameters. Hence rather than testing all combinations in an exhaustive manner, combinations of only few parameters are tested. In order to generate test set, values for input parameters are selected such that every possible combination of values of any t parameters occurs at least once [5]. It is also known as the strength of coverage or interaction strength.

As an example, let us consider 3 input parameters, *A*, *B* and *C*, each can have 2 possible values, 0 and 1. Pairwise testing (where t = 2) would require the following 4 test cases as given in Table 1. Test cases are designed such that all possible pairs of values are getting covered. As per studies, it is observed that maximum value of interaction strength is 4–6 for most of the systems [4]. As the value of interaction strength increases, the total number of detectable errors increases. But, an increase in interaction strength leads to an increase in the test set size, and hence increases the cost of testing. On the other hand, lower interaction strength leads to reduction in test set size which affects faults detection rate. Thus an optimal value of interaction strength can substantially reduce the testing costs without compromising fault detection capability. However, not much research is done in this area.

```
  Begin
  if (customer belongs to set B){
  //some code here
  if(customer.habitation==US){
  //defective code here
  }
  else{
  //block of code-executes normal
  }
  }
End
```

Fig. 1: Pseudo code illustrating type 1 interaction failure.

Type 2 interaction failures are illustrated using pseudo code in Fig. 2. Here, placing an erroneous operator causes the set of variables involved in computation to produce an incorrect result.

```
  Begin
  float x, y, z;
  float result=(x*y)/z// it should be (x*y)-z
  //block of code
End
```

Fig. 2: Pseudo code illustrating type 2 interaction failure.

In this paper, we aim to identify interactions that may cause these two types of interaction failures to occur. Data flow analysis techniques derive the information about the flow of data that exist in the program [6]. At each step in the program, the information about definition and usage of variables is obtained.

The rest of this paper is organized as follows. In Section 2 review the related work. The Proposed methodology described in Section 3. Finally conclude the paper in Section 4.

2. RELATED WORK

D.M. Cohen, S.R. Dalal, J. Parelius, G.C. Patton [5] discussedthe combinatorial design method substantially reduces testing costs. The authors describe an application in which the method reduced test plan development from one month to less than a week. In several experiments, the method demonstrated good code coverage and fault detection ability.

S. Varshney, M. Mehrotra[6] presented a search-based approach that generates test data for data-flow dependencies of a program using dominance concepts, branch distance, and elitism. Genetic algorithm is used for the proposed approach and Gray encoding is used to encode test data. A set of subject programs is taken from the research literature to evaluate efficiency and effectiveness of the proposed approach. For the proposed approach, the measures considered are the mean number of generations and mean percentage coverage achieved. The performance of the proposed approach is evaluated by comparing the results with those of random search and earlier studies on data-flow testing.

S. Sabharwal, M. Aggarwal[7] proposeda Combinatorial (t-way) testing has been proved to be an effective technique that identifies faults caused by interactions among a small number of input parameters. However, the degree of interaction to be considered for testing is still an open issue. Although higher strength t-way testing improves fault detection, it leads to a considerable increase in number of interactions to be tested and hence the test set size. The authors proposed that attempts to reduce the number of interactions to be tested. The source code is transformed into a flow graph and data flow analysis is applied to it to identify the interactions that exist in the system. The initial results suggest that the approach is able to achieve a considerable reduction in the number of interactions to be tested.

C. Nie, H. Leung[8] discussed Combinatorial Testing (CT) can detect failures triggered by interactions of parameters in the Software Under Test (SUT) with a covering array test suite generated by some sampling mechanisms. It has been an active field of research in the last twenty years. This article aims to review previous work on CT, highlights the evolution of CT, and identifies important issues, methods, and applications of CT, with the goal of supporting and directing future practice and research in this area. First, we present the basic concepts and notations of CT. Second, we classify the research on CT into the following categories: modeling for CT, test suite generation, constraints, failure diagnosis, prioritization, metric, evaluation, testing procedure and the application of CT. For each of the categories, we survey the motivation, key issues, solutions, and the current state of research. Then, we review the contribution from different research groups, and present the growing trend of CT research. Finally, we recommend directions for future CT research, including: (1) modeling for CT, (2) improving the existing test suite generation algorithm, (3) improving analysis of testing result, (4) exploring the application of CT to different levels of testing and additional types of systems, (5) conducting more empirical studies to fully understand limitations and strengths of CT, and (6) combining CT with other testing techniques.

L.G. Hernandez, N.G. Valdez, J.T. Jimenez [9] proposed adevelopment of a new software system involves extensive tests of the software functionality in order to identify possible failures. Also, a software system already built requires a fine tuning of its configurable options to give the best performance in the environment where it is going to work. Both cases require a finite set of tests that avoids testing all the possible combinations (which is time consuming); to this situation mixed covering arrays (MCAs) are a feasible alternative. MCAs are combinatorial structures having a case per row. MCAs are small, in comparison with exhaustive search, and guarantee a level of interaction among the involved parameters (a difference with random testing). We present a tabu search algorithm (TSA) for the construction of MCAs. Also, we report the fine tuning process used to identify the best parameter values for TSA. The analyzed TSA parameters were three different initialization functions, five different tabu list sizes and the mixture of four neighborhood functions. The performance of TSA was evaluated with two benchmarks previously reported.

S. Chen, Z. Chen, Z. Zhao, B. Xu, and Y. Feng[10] discussed a component based software development is prone to unexpectedinteraction faults. The goal is to test as manypotential interactions as is feasible within time and budgetconstraints. Two combinatorial objects, the orthogonalarray and the covering array, can be used to generate testsuites that provide a guarantee for coverage of all t-setsof component interactions in the case when the testing ofall interactions is not possible. Methods for constructionof these types of test suites have focused on two main areas.The first is finding new algebraic constructions that producesmaller test suites. The second is refining computationalsearch algorithms to find smaller test suites more quickly. Inthis paper authors explored one method for constructing coveringarrays of strength three that combines algebraic constructionswith computational search. This method leveragesthe computational efficiency and optimality of size obtainedthrough algebraic constructions while benefiting from thegenerality of a heuristic search. We present a few examplesof specific constructions and provide some new bounds forsome strength three covering arrays.

3. PROPOSED METHODOLOGY

This paper aims to collect and consider papers that deal with combinatorial testing using Black-Box Regression Maximization (BBM) Algorithm and Density-based Spatial Clustering Algorithm (DSC) with Greedy Search optimization algorithm. Our objective is not to undertake a logical review, but quite to provide a broad state-of-the-art view on these related fields. Many different approaches have been projected to assistcombinatorial testing, which has mentioned in a body of literature that is spread over a wide variety of fields and periodicallocations. However, the combinatorial testing attempts to reduce the number of interactions to be tested. The source code is transformed into a flow graph and data flow analysis is applied to it to identify the interactions that exist in the system. In this proposed methodology we will discuss about the Black-Box Regression Maximization (BBM) Algorithm in detail.The overall architecture in figure 3 follows combinatorial testing from begins to end state. The users initialize the input parameter instances, features and classes as initial parameters in which the testing process is to be evaluated.

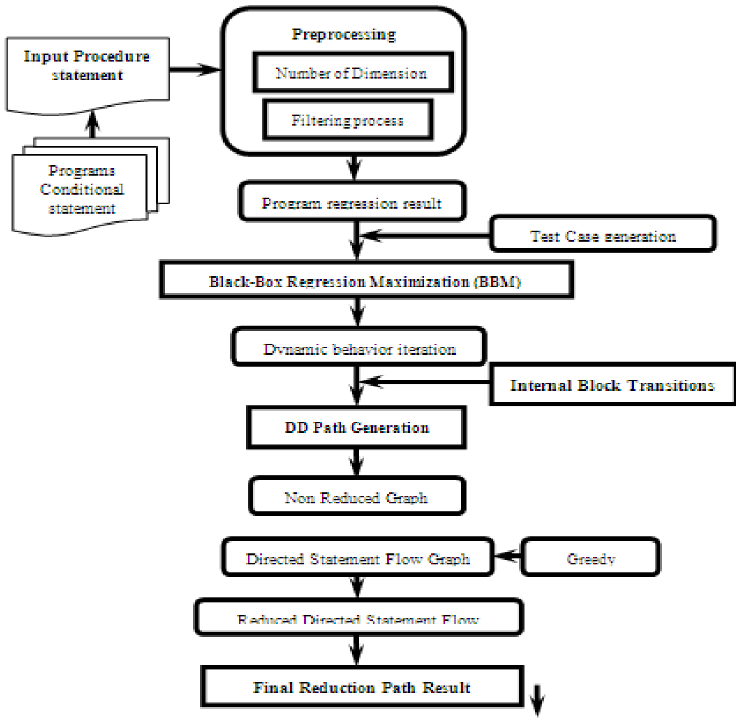

Input Procedure statement
Preprocessing
Number of Dimension
Filtering process
Programs Conditional statement
Program regression result
Test Case generation
Black-Box Regression Maximization (BBM)
Dynamic behavior iteration
Internal Block Transitions
DD Path Generation
Non Reduced Graph
Directed Statement Flow Graph
Greedy
Reduced Directed Statement Flow
Final Reduction Path Result

Fig. 3:Proposed System Architecture

In figure 3, input procedure with conditional statement is taken as testing process. The first process is called preprocessing methods namely scale of the procedure and removing empty spaces with help of filtering method. After that test case generation is linked into the regression result. The BBM algorithm performs partitioning the statement according to the test case. In that each variable and statement is separated in to individual blocks. The internal block transitions execute a test case in sequence of internal program states into separate blocks. In that each blocks is called DD path non-reduced graph generation. The DSFG mapping with greedy search algorithm involves reducing the non-reduced graph with small number of blocks.

A. *Preprocessing*

The preprocessing work to “clean” the data by filling in missing values, smoothing noisy data, identifying or removing anomalies, and resolving inconsistencies. If users believe the data are dirty, they are unlikely to trust the results of any data that has been applied. Furthermore, unwanted data can cause confusion for the mining procedure, resulting in unreliable output. Although most mining routines have some procedures for dealing with incomplete or noisy data, they are not always robust. Instead, they may concentrate on avoiding over fitting the data to the function being modeled. In this process, the dimension of the program is calculated (i.e, number of lines in the procedure or program) and the filtering process is to estimate the number node corresponds to a block of sequential statements. To identify the usage at a node, a node is associated with the data identified for the statements that correspond to that node.

*Example:*Scanf (x) into scanf(x)

B. *Black-Box Regression Maximization (BBM) Algorithm*

The block-box regression maximization prediction provides for clustering in the multiple regressions setting in which you have a dependent variable *Y* and one or more independent variables, the *X*'s. The algorithm partitions the data into two or more clusters and performs an individual multiple regression on the data within each cluster. The BBM program spectrum may characterize a program's behavior statically; a program spectrum is usually used in characterizing dynamic behavior exhibited by the execution of a test or multiple tests. One of the earliest proposed program spectra are Decision-to-Decision (DD) path. Path spectra are represented by the executed paths in a program. There are variants of path spectra depending on whether to use the complete paths or partial paths (loop-free intra procedural paths) as well as whether to track the frequency of path occurrences.

Algorithm : Consider a procedure with independent features $\{X'_i\}^j_{i=1}$ and an multiple regression class Y . To keep the notation simple, to transform the features $\{X'_i\}^j_{i=1}$into DD path features$\{X_i\}^n_{i=1}$, i.e., $\{X_i\}^n_{i=1}$Binarize($\{X'_i\}^j_{i=1}$). The inputs to the *BBM* algorithm are the preprocessed features $\{X_i\}^n_{i=1,}$ the corresponding labels Y and a pre-specified condition α.

Algorithm1: *Black-Box Regression Maximization (BBM) Algorithm*

```
Input:Procedure statement (features $X_1$, $X_2$,…$X_n$ and label $Y$ ), α
Output: Path Generation
for i = 1 to procedure length do
Select the independent feature $X^*$, which gives the maximum α -divergence criterion
if (number_of_lines<proce_length) then
Process independent variable(i.e, variable datatypes)
Add a path node to the independent variable
ifcondition_statement is achieved then
Add a path to the corresponding node
else
Add a path to the next node
end if
Else if(proce_length is achieved) then
Stop growing.
End if
Partition the training data into two paths, based on the value of $X^*$
endfor
```

C. *Internal Block Transitions*

The internal block transitions execution of a program can be considered as a sequence of internal program states. Each internal block state comprises the program's in-scope variables and their values at a particular execution point. Each program execution unit (in the granularity of statement, block, code fragment, function, or component) receives an internal program state and then produces a new one. The program execution points can be the entry and exit of a user-function execution when the program execution units are those code fragments separated by user-function call sites. Program output statements (usually output of I/O operations) can appear within any of those program execution units.Since it is relatively expensive in practice to capture all internal program states between the executions of program statements, we focus on internal program states in the granularity of user functions, instead of statements.

Example

```
(1) scanf(x);
(2) scanf(y);
(3) r=x%y;
(4) if[r==0] goto (8);   [Internal block transitions]
(5) x=y;
(6) y=r;
(7) goto (3);
(8) printf(y);
(9) stop;
```

D. *Mathematical Model for Greedy Search*

The mathematical model for Greedy Search with Directed Statement Flow Graph (DSFG) mapping algorithm shows global prior probabilities for Combinatorial Testing process. Let T_i be an input source code with n number of lines ($T_{i1}, T_{i2,\ldots,} T_{in}$) where T_i = (i = 1,2, …, n). The equation (1) initialize the undirected graph for the input test file and find the path is defined as follows:

$$G = \{V, E\} \quad \text{eqn. (1)}$$

The Graph edge weights w_e is defined as,

$$\text{weight } (T) = \sum_{e \in E'} w_e \quad \text{eqn. (2)}$$

The Greedy search feature represents the n^{th} combinations value of conditional test defined as,

$$GS = \max_{S \subseteq \{1,2,\ldots,n\}} \{G(s) | W(S) \le C\} \quad \text{eqn. (3)}$$

where, $w_1, \ldots, w_n \in N$ be weights, and let $C \in N$ be a weight. For each S ⊆ {1, …,n} let G(S) = eqn. (2)

Algorithm: Identifying Interactions with Greedy Search algorithm –Data Flow Techniques

1. Read the Input Test case features from the input file T = {Parameter, Constraints and Test set }
2. Train the input test features in N number of combinations to covariance set
3. Read the test program procedure file for identifying the interactions testing
4. Convert the input procedure file to Directed Statement Flow Graph (DSFG)
5. Find the Conditional statement and goto procedure using Greedy search method.

6. Sorting the DSFG graph in ascending order
7. For each test conditions finding the reduced DSFG to the point in candidate test set
    a. Determine the each code entry element weight position
    b. Search conditions set on to program that are active in the conditional delimiters statement
    c. To map the conditional points according to the exact line number.
    d. Greedy search method is to search the exact paths to connected the graph.
    e. Calculate the Execution Time is Elapsed Time = (End time – Start Time) / 1000
    f.

4. RESULT AND DISCUSSION

The research work results describe a preliminary experimental evaluation of the Identifying Interactions for Combinatorial Testing using Data Flow Technique using Graph based Greedy Search algorithms results described in figure 4. The research work implemented the proposed algorithms in JAVA and compared them against a few other prominent Combinatorial Testing techniques. The proposed system performed all the experiments on a Windows machine with a dual core 2.8 Ghz processor and 2 GB of RAM. In this experiment study, we compare the performance of Computational Time of the testing source code.

$$\text{Computational Time} = \frac{Proces\ End\ time}{Process\ Start\ time} * 1000 \quad \text{eqn.(3)}$$

Table 1 describes the performance of the system using metrics such as Time metrics. Computational time metrics is analyzed in terms of combinatorial testing and Graph path generation in figure 2. Computational Time is defined as

Table 2: Comparison of Computational Time

Methods	1	2	3	4	5	6
DERIVING INTERACTIONS FOR COMBINATORIAL TESTING	1.8	0.99	2.40	1.14	0.78	1.5
GREEDY DECISION PATH GRAPH	0.5	0.9	0.2	0.45	0.68	0.14

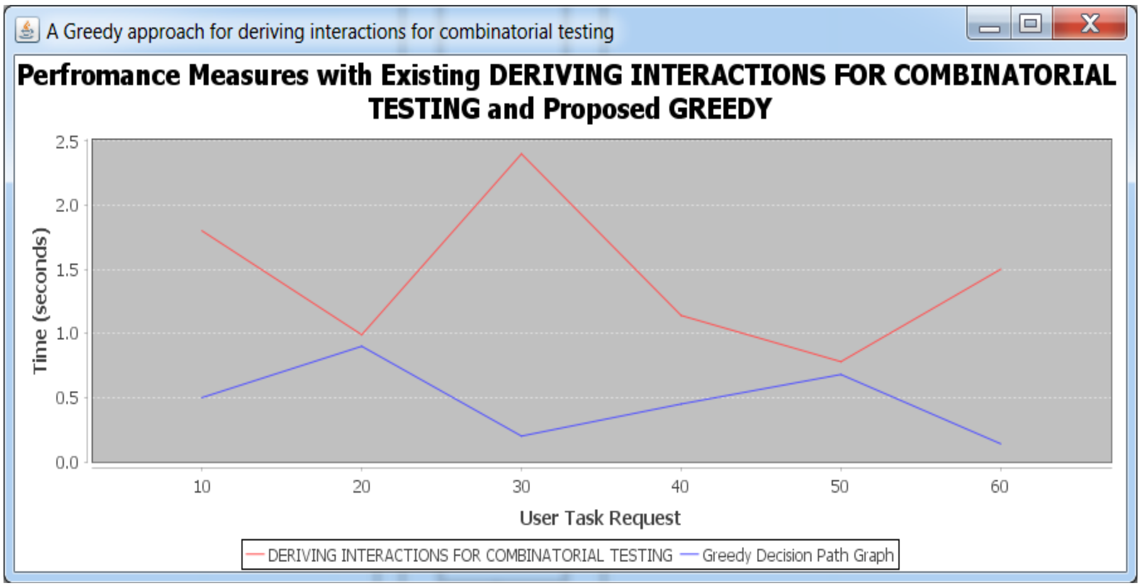
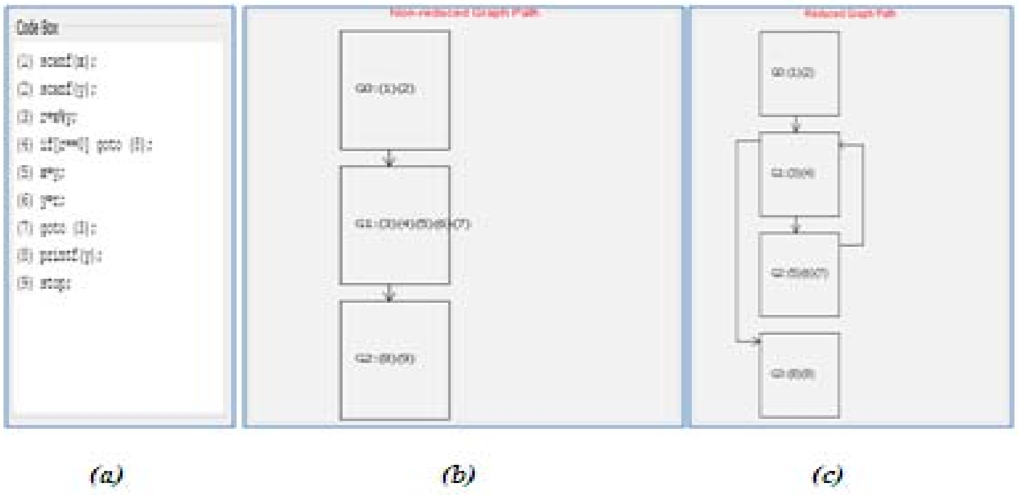


Fig.5: Combinatorial testing and Graph path generation



```
Code Box
(1) scanf(x);
(2) scanf(y);
(3) z=x%y;
(4) if(z==0) goto (8);
(5) x=y;
(6) y=z;
(7) goto (3);
(8) printf(y);
(9) stop;
```

Fig.5: Geedy decision path graph results. (a) Input procedure conditiona statement; (b) Non-reduced graph path; (c) Final Reduced Graph path result.

5. CONCLUSION

In this paper presentsthe Black-Box Regression Maximization (BBM) Algorithm approach with Density-based Spatial Clustering Algorithm (DSC) with Greedy Search optimization algorithm focuses on combinatorial testing and proactively exposes behavioral deviations by checking inside block transitions. In this method presents new approach of BBM with Internal block transitions to measure the dissimilarity statements in large program data. To identify specific faults, an adaptive testing rule repeatedly constructs and tests configurations in order to determine, for each interaction of interest, whether it is faulty or not. In order to perform such testing in a procedure environment, it is imperative that testing results can be combined from block transitions.

The further work enhancedand expandedfor the automation of Decision-to-Decision path with Density-based Spatial Clustering Algorithm (DSC)for black-box regression testing using Greedy Search optimization algorithm.

REFERENCES

[1] R.S. Pressman, Software Engineering: A Practitioner's Approach, third ed., McGraw Hill, New York, 1992.

[2] P.Aditya Mathur, Foundations of Software Testing, 2/e, Pearson Education, India, 2008.

[3] G.J. Myers, C. Sandler, T. Badgett, The Art of Software Testing, third ed., Wiley, New York, NY, USA, 2011.

[4] D.R. Chicago Kuhn, R.N. Kacker, Y. Lei, Practical Combinatorial Testing, NIST Special Publication, 2010. 800-142.

[5] D.M. Cohen, S.R. Dalal, J. Parelius, G.C. Patton, The combinatorial design approach to automatic test generation, IEEE Softw. 13 (5) (1996) 83–88.

[6] S. Varshney, M. Mehrotra, Search-based test data generator for data-flow dependencies using dominance concepts, branch distance and elitism, Arabian J. Sci. Eng. (2015) 1–29.

[7] S. Sabharwal, M. Aggarwal, Identifying interactions for combinatorial testing using data flow techniques, in: SIGSOFT, vol. 39, issue 6, ACM, New York, NY, 2014.

[8] C. Nie, H. Leung, A survey of combinatorial testing, ACM Comput. Surv. J. 43 (2) (2011) 11:1–11:29.

[9] L.G. Hernandez, N.G. Valdez, J.T. Jimenez, Construction of mixed covering arrays of strengths 2 through 6 using a tabu search approach, Discrete Math.: Algorithms Appl. 4 (3) (2012) 1–20.

[10] M.B. Cohen, C.J. Colbourn, A.C.H. Ling, Augmenting simulated annealing to build interaction test suites, in: Proc. of the 14th International Symposium on Software Reliability Engineering (ISSRE'03), IEEE Computer Society, Los Alamitos, CA, 2003, pp. 394–405.

[11] P.Aditya Mathur, Foundations of Software Testing, 2/e, Pearson Education, India, 2008.