International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 6, June 2015, PP 19-26
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

The Survey on Artificial Life Techniques for Generating the Test Cases for Combinatorial Testing

Lakshmi Prasad Mudarakola [1]
Dept. of Computer Science & Engineering,
NBKRIST, Nellore, AP, India.
prasad.hinduniv@gmail.com

M.Padmaja [2]
Dept. of Computer Science and Engineering,
SKUniversity, Anathapur, AP, India.
padmaja.ruthwika@gmail.com

Abstract: *Software testing has faced many intractable problems: for real world programs, the number of possible input combinations can exceed the number of atoms in the ocean, so as a practical matter it is impossible to show through that the program works correctly for all inputs. Combinatorial testing offers a solution. Combinatorial testing of software analyzes interactions among variables using a very small number of tests. It can help to detect the problem early in the testing life cycle. Artificial Life techniques can dramatically change our ability to solve a host of problems in applied science and engineering; many search techniques have been developed and applied successfully in many fields. In this paper we had shown different variants from existing search algorithms: Genetic Algorithm, Particle Swarm Optimization and Ant Colony Algorithm, Bee colony Optimization, Simulate Annealing. Combinatorial testing can use a small number of test cases to test systems while preserving fault detection ability. However, the complexity of test case generation problem for combinatorial testing is NP-complete. The efficiency and complexity of this testing method have attracted many researchers from the area of combinatorics and software engineering. We believe that these search techniques can be further improved by fine-tuning their configuration and used in broad ranges of area.*

Keywords: *Genetic Algorithm; Ant Colony Algorithm; Particle Swarm Optimization; Covering Array; Combinatorial Testing;*

1. INTRODUCTION

Combinatorial testing is a black-box technique that could dramatically reduce the number of tests as it is a highly efficient technique to detect software faults. The method generally followed in combinatorial testing is to derive test cases from input domain of the system under test. But, when the input domain is larger and the output domain is much smaller, it is preferable to go for testing the output domain either exhaustively or as much as possible.

Combinatorial testing is a specification based sampling technique that provides a systematic way to select combinations of program inputs or features for testing. It is an effective testing technique to test hardware/software that reveals failures in a given system based on input or output combinations. It has been applied over the years to test input data, configurations, web forms, protocols, graphical user interfaces, software product lines etc. The pair wise testing can detect possible t-way combinatorial interactions for t=2, 3, 4, 5, 6 or more. Classification of combinatorial techniques is shown in figure 1.

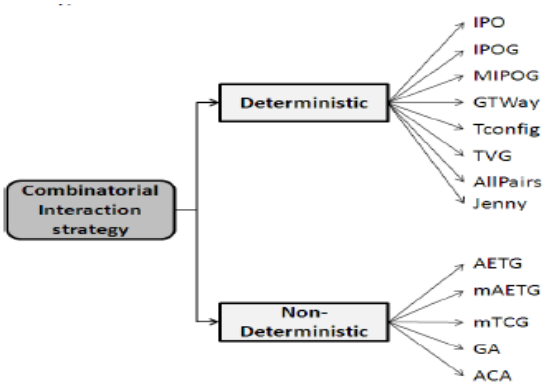

Fig.1. *Classification of combinatorial testing.*

Combinatorial testing is used to detect problems early in the testing life cycle. The test requirements for this testing are either configurations or input parameters. Test planning and design is also needed for implementing this testing. Combinatorial testing plays an important role in generating the test cases for different applications. This type of testing plays a vital in role in test case prioritization also. Audit trail and certification is also important one. The classification of combinatorial testing is as shown above.

2. SURVEY OF LITERATURE

Pair wise testing is a wildly popular approach to combinatorial testing problems. The number of articles and textbooks covering the topic continues to grow, as do the number of commercial and academic courses that teach the technique. Despite the technique's popularity and its reputation as a best practice, old techniques to be over promoted and poorly understood. Knowledge of the weaknesses of the pairwise testing technique, or of any testing technique, is essential to apply the technique wisely. A wide variety of different strategies and implementations for generating pair wise test sets have been published.

According to Yu Lei and K.C. Tai [1] propose a test generation strategy, called in-parameter-order (or IPO), for pairwise testing. The IPO strategy allows the use of local optimization techniques for test generation and the reuse of existing tests when a system is extended with new parameters or new values of existing parameters. They present practical, IPO-based test generation algorithms. They describe the implementation of an IPO-based test generation tool and show some empirical results.

Xiang Chen, Qing Gu, Xin Zhang and Daoxu Chen [2] adopt ant colony optimization (ACO) to build this prioritized pairwise interaction test suite (PITS). The biased covering array is proposed and the Weighted Density Algorithm (WDA) is developed. In their research, they propose four concrete test generation algorithms based on Ant System, Ant System with Elitist, Ant Colony System and Max-Min Ant System respectively. They also implement these algorithms and apply them to two typical inputs and report experimental results. The results show the effectiveness of these algorithms.

Kewen Li and Zhixia Yang [3] propose ant colony arithmetic, which is a new way to solve the pair-wise test data generating question. It can generate fewer test cases which can cover more pair combinations, and can solve questions with fast calculate speed. The method can get the goal of optimizing in the process of regression test. The result shows that the method is feasible.

J.D. McCaffrey [4] presents the results of generating pairwise test sets using a simulated bee colony algorithm. Compared to published results for seven benchmark problems, the simulated bee colony approach produced test sets which were comparable or better in terms of size for all seven problems. However, the simulated bee colony approach required significantly longer generation time than deterministic approaches in all cases. The results demonstrate that the generation of pairwise test sets using a simulated bee colony algorithm is possible, and suggest that the approach may be useful in testing scenarios where pairwise test set data will be reused.

Xiang Chen, Qing Gu, Jingxian Qi and Daoxu Chen [5] apply particle swarm optimization (PSO), a kind of meta-heuristic search technique, to pairwise testing (i.e. a special case of combinatorial testing aiming to cover all the pairwise combinations). To systematically build pairwise test suites, two different PSO based algorithms is proposed. To successfully apply PSO to cover more uncovered pairwise combinations in this construction process, a detailed description is provided on how to formulate the search space, define the fitness function and set some heuristic settings. Final empirical results show the effectiveness and efficiency of their approach.

Mohammed I. Younis, Kamal Z. Zamli and Nor Ashidi Mat Isa [6] propose an efficient pairwise strategy for generating pairwise combinatorial test set using artificial parameters and values, termed RA and ORA, that can systematically minimize the pairwise test set generated from higher order test parameters to lower order ones. Their paper demonstrates and compares the results against existing strategies including IRPS, IPO, GA, ACA, Jenny and All Pairs. The Results show that ORA and RA are performs well, and gives a minimal number of the test set than other published result and available tools.

S.A. Ghazi and M.A. Ahmed [7] propose a GA-based technique that identifies a set of test configurations that are expected to maximize pair-wise coverage, with the constraint that the number of test configurations is predefined. Although the paper primarily focuses on the interaction between

software components, the idea can be applied to single code component testing. Some experiments are also performed using their proposed approach. The results were promising.

S. Khatun, K.F. Rabbi, C.Y. Yaakub and M.F.J. Klaib [8] propose an effective random search based pairwise test data generation algorithm named R2Way to optimize the number of test cases. The algorithm is able to support both uniform and non-uniform values effectively with performance better than the existing algorithms/tools in terms of number of generated test cases and time consumption.

James D. McCaffrey [9] presented the results of an investigated the generated pairwise test sets using a genetic algorithm. Compared with published results for deterministic pairwise test set generation algorithms, the genetic algorithm approach produced test sets which were comparable or better in terms of test set size in 39 out of 40 cases. However, the genetic algorithm approach required longer processing time than deterministic approaches in all cases.

J.D. McCaffrey [10] describes the results of an investigation of pairwise test set generation using a genetic algorithm. The results illustrate that generation of pairwise test sets using a genetic algorithm is possible, and suggest that the technique may be both practical and useful in certain software testing situations.

Pedro Flores, Yoonsik Cheon [11] formulated the problem of finding a pairwise test set as a search problem and applies a genetic algorithm to solve it. They also describe an open source tool called PWiseGen for generating pairwise test sets. PWiseGen produces competitive results compared with existing pairwise testing tools. Besides, it provides a framework and a research platform for generating pairwise test sets using genetic algorithms.

Priti Bansal, Sangeeta Sabharwal, Shreya Malik, Vikhyat Arora, and Vineet Kumar [12] present a method to generate initial population using hamming distance and an algorithm to find crossover points for combining individuals selected for reproduction. They apply genetic algorithm, a meta-heuristic search algorithm, to find an optimal solution to the pair-wise test set generation problem. They describe the implementation of the proposed approach by extending an open source tool PWiseGen and evaluate the effectiveness of the proposed approach. Empirical results indicate that their approach can generate test sets with higher fitness level by covering more pairs of input parameter values

Manisha Patil, P.J. Nikumbh [13] formulate the problem of finding a pair-wise test set as a search problem and apply a search technique "simulated annealing" to solve it. Their empirical results concluded that the optimal solution for their problem statement is to find the maximum number of different pairs which are the best test set which we capture. The key contribution of their work includes simulating annealing algorithm approach for generating the best test case set. A software for web based and semantic based application is developed in software engineering for determining minimized test cases.

3. PICTURE OF DIFFERENT ARTIFICIAL LIFE TECHNIQUES

A. Partical Swarm Optimization

Particle swarm optimization (PSO) is a population-based stochastic approach for solving continuous and discrete optimization problems. In particle swarm optimization, simple software agents, called *particles*, move in the search space of an optimization problem. The position of a particle represents a candidate solution to the optimization problem at hand. Each particle searches for better positions in the search space by changing its velocity according to rules originally inspired by behavioral models of bird flocking.

Particle swarm optimization belongs to the class of swarm intelligence techniques that are used to solve optimization problems. Particle swarm optimization (PSO) is a computational method that optimizes a problem by iteratively trying to improve a candidate solution with regard to a given measure of quality. PSO optimizes a problem by having a population of candidate solutions, here dubbed particles, and moving these particles around in the search-space according to simple mathematical formulae over the particle's position and velocity. Each particle's movement is influenced by its local best known position but, is also guided toward the best known positions in the search-space, which are updated as better positions are found by other particles. This is expected to move the swarm toward the best solutions.

PSO is a meta-heuristic as it makes few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, meta-heuristics such as PSO do not guarantee an optimal solution is ever found. More specifically, PSO does not use the gradient of the problem being optimized, which means PSO does not require that the optimization problem be differentiable as is required by classic optimization methods such as gradient descent and quasi-newton methods. PSO can therefore also be used on optimization problems that are partially irregular, noisy, change over time, etc. Strategy for generating the combinatorial test cases Particle Swarm Optimization is as shown below.

Particle Swarm Optimization Strategy

```
1: Input: Parameters' values, strength of coverage t;
2: Output: A test case;
3: Let Ps be a set of all not covered combinations of
   parameter values;
4: Generate Ps;
5: Let Ts be a set of candidate tests;
6: While Ps is not empty do {    //termination-condition
7:    Randomly initialize particles Xi(t) and velocities
      Vi(t);
8:    For a specific number of iterations do {
9:       Evaluate Xi(t) for its T-interaction element
         coverage with Ps;
10:      if Xi(t) covered maximum interaction element in
         PS{
11:         Add Xi(t) to final test suit TS;
12:         Remove Xi(t) from Ps;
13:         continue;
14:         }
15:      else {
16:         Choose best coverage particle to be lBest;
17:         Calculate Vi(t+1) according to lBest;
18:         Move Xi(t) to Xi(t+1) according to Vi(t+1);
19:         }
20:      Evaluate Xi(t+1);
21:      If lBest(t+1) cover bigger T-interaction
         elements;
22:      lBest=lBest(t+1);
23:      }//End for
24:   Let gBest be the best test case found;
25:   gBest = lBest(t+1);
26:   Add gBest to the test set Ts;
27:   Remove those combinations in Ps that covered by;
28: }//End while
```

B. Genetic Algorithms

Genetic Algorithms (GAs) are a class of computational procedures inspired by biological evolution. GAs encode a potential solution to a specific problem using a simple chromosome-like data structure and then apply operators modeled after genetic recombination and mutation to these structures in a way that is designed to preserve essential information. GAs maintains a population of individuals each of which consists of a chromosome/solution and a fitness value which measures how well the individual's chromosome solves the problem. Individuals with high fitness values are selected to serve as the basis for producing offspring solutions. Individuals with low fitness values are removed from the population of solutions and replaced by offspring solutions.

Genetic algorithms are typically used to solve maximization and minimization problems that are combinatorial complex and which do not lend themselves to standard algorithmic techniques. In pseudo code, one typical form of a GA is: set generation := 0 initialize population while (generation < maxGenerations) evaluate population fitness values sort population based on fitness if (optimal solution exists) break select high-fitness individuals produce offspring stochastically mutate offspring replace low-fitness individuals end while return best individual. There are many variations of the basic algorithm structure which are possible.

Genetic algorithms merely provide a basic framework for solving a problem and the implementation of a specific genetic algorithm which solves a specific problem requires several design decisions.

Some of the major design decisions include the following. First, a chromosome representation of a solution to the target problem must be designed. Second, a fitness function which measures how well a chromosome solves the target problem must be constructed. Third, stochastic algorithms to implement genetic crossover and mutation must be designed. Additional GA design parameters include selection of the population size, a method for determining which chromosome-solutions are selected for reproduction, and a method for determining which chromosome solutions are selected for removal from the population. The strategy for generating the combinatorial test cases is shown below.

Genetic Algorithm

Begin
Step 1: *P* = initializePopulation()
Step 2: i = 0
Step 3: **while** (*i* < MAX_GEN && !has Solution(*P*))
do
calculateFitness(P)
Step 4: C = ∅ **while** (|*C*| < NUM_CROSSOVER)
do
(*p*1….., *pn*) = selectParents(P)
(*c*1……. *cn*) = crossover (*p*1… *pn*);
Step 5: **if** (mutate?) **then**
*c*1 = mutate (*c*1);
*c*2 = mutate (*c*2);
end

Step 6: C = C ∪{c1; c2}
end
Step 7: **if** (immigration?) **then**
I = createImmigrants();
end
Step 8: P = updatePopulation(*P, C ∪ I*)
Step 9: i = i + 1**end**
End

C. Ant Colony Optimization

The ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs. This algorithm is a member of the ant colony algorithms family, in swarm intelligence methods, and it constitutes some meta-heuristic optimizations. Initially proposed by Marco Dorigo in 1992 in his PhD thesis,[1][2] the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

ACO is a new heuristic method to solve complex optimization questions, which based on simulation and evolution of populations. The ant colony arithmetic is a random search one, which gets optimal solution through evolving candidate ones that composes ant colony. The arithmetic is composed of many ants, which search solutions independency in the candidate solution space. Pheromone is left over by ants on the solutions and can be apperceived by other ants. Ants are adapted to choose the high-concentration path and the collectivity behavior reflects a phenomenon of information positive feedback: the more ants passed the path, the bigger probability the late comers choose. The ants, as the distributed intelligent body, can search optimal solutions according to the directions of artificial pheromone trail and can make use of the heuristic information based on the questions. Additional, there are two important mechanisms: pheromone volatilization and background behavior. Forgetting is an advanced intelligent behavior, as a kind of which, because the pheromone will volatilize along with the time, the pheromone will volatilize along with time to avoid the solving process running into local optimal solution. The background behavior includes neighbor search process and collection of local information of questions. ACO was formalized into a meta-heuristic search technique by M. Dorigo and his co-workers in 1992.

Now ACO was widely applied for solving many combinatorial optimization problems, such as traveling salesman problem, scheduling problems, and vehicle routing problems. The ACO can be summarized as follows. Ants start searching for a test by initially randomly choosing tests. For each test chosen by an ant, the quality of the test (i.e. the incremental benefit) is evaluated. Then the ant deposits pheromone trail, i.e., marks with pheromone the levels forming the test. The quantity of deposed pheromone is proportional to the test quality. As in nature, artificial ants tend to follow pheromone trail. This indirect communication between ants progressively promotes the test covering more incremental benefits.

Ant Colony optimization Strategy

Input: tests generated
//**step1**: initialize pheromone trails
$t \leftarrow 0$, $NC \leftarrow 0$ //t is the time counter, NC is the iteration counter
bestTest←null // store best-so-far test
bestIncreBenefits←0.0 //store the incremental benefit of best-so-far test
for $\forall e_{ij} \in E$ **do**
  $\tau_{i,j}(t) \leftarrow C$; $\Delta\tau_{i,j}(t) \leftarrow 0$
  Compute local heuristic $\eta_{i,j}(t)$
end for
Put m ants in node f_i
//**step2**: for each ant, construct a solution (i.e. a valid test)
for factor $f_i = f_i$ to f_k **do**
  for i=1 to m **do**
    ant i moves to the next node according to **edge selection rule**
  end for
end for
//**step3**: Compute pheromone to be updated
for i=1 to m **do**
  Get the test $test_i$ created by ant i
  Store the incremental benefit of $test_i$ to *tmpBen*
  if *tmpBen* > *bestIncreBenefits* **then**
    bestTest ← $test_i$; *bestIncreBenefits* ← *tmpBen*
  end if
  for $\forall e_{ij} \in$ E **do**
    for l=1 to m **do**
      Compute $\Delta\tau_{i,j}^{l}(t)$ deposited by ant l according to **pheromone**
update rule
      $\Delta\tau_{i,j}(t) \leftarrow \Delta\tau_{i,j}(t) + \Delta\tau_{i,j}^{l}(t)$
    end for
  end for
//**step4** update pheromone value
for $\forall e_{ij} \in E$ **do**
  $\tau_{i,j}(t+k) \leftarrow (1-\rho)\tau_{i,j}(t) + \Delta\tau_{i,j}(t)$; $\Delta\tau_{i,j}(t) \leftarrow 0$
end for
$t \leftarrow t + k$; $NC \leftarrow NC + 1$
for i =1 to m **do**
  ant i moves from node *End* to node f_i
end for
//**step5** stop criterion
if $NC > NC_{max}$ **then**
  Output *bestTest*
else
  goto step 2
end if

D. Bee Colony Optimization Strategy

The Bees Algorithm is a population-based search algorithm which was developed in 2005.[1] It mimics the food foraging behavior of honey bee colonies. In its basic version the algorithm performs a kind of neighborhood search combined with global search, and can be used for both combinatorial optimization and continuous optimization. The only condition for the application of the Bees Algorithm is that some measure of topological distance between the solutions is defined. The effectiveness and specific abilities of the Bees Algorithm have been proven in a number of studies.

Common honey bees such as Apis mellifera take on different roles within their colony over time. Young bees nurse larvae, construct and repair the hive, guard the entrance to the hive, and so on. Mature bees typically become foragers. Foraging bees typically occupy one of three roles: active forgers, scout foragers, and inactive foragers. Active foraging bees travel to a food source, gather food, and return to the hive. Roughly 10% of foraging bees in a hive are employed as scouts. These scout bees investigate the area surrounding the hive, often a region of up to 50 square miles, looking for attractive new food sources. At any given time some foraging bees are inactive. These inactive

foraging bees wait near the hive entrance. When active foragers and scouts return to the hive, depending on the quality of the food source they are returning from, they may perform a waggle dance to an audience of inactive foraging bees. This waggle dance is believed to convey information to the inactive foragers about the location and quality of the associated food source. Inactive foragers receive this food source information from the waggle dance and may become active foragers. In general, an active foraging bee continues gathering food from a particular food source until that food source is exhausted, at which time the bee becomes an inactive forager.

Algorithms inspired by the behavior of natural systems have been studied for decades. These algorithms are sometimes called meta-heuristic algorithms because they provide a high-level framework which can be adapted to solve optimization, search, and related problems, as opposed to providing a stringent set of guidelines for solving a particular problem. A review of the literature on algorithms inspired by bee behavior suggests that the topic is evolving and that there is no consensus on a single descriptive title for meta-heuristics based on bee behavior. Algorithm names in the literature include Bee System, BeeHive, Virtual Bee Algorithm, Bee Swarm Optimization, Bee Colony Optimization, Artificial Bee Colony, and Bees Algorithm

There are many ways to map honey bee foraging behavior to an algorithm which solves a specific optimization problem.

Each Bee object has an Act method which encapsulates the core SBC algorithm. In pseudo-code the logic for an active foraging bee is:

if (role == Active)
(leave hive, go to food source)
examine a neighbor food source
if (quality >= current quality)
current memory := neighbor location
numberVisits := 0
else
numberVisits := numberVisits + 1
endif
(return to hive)
if (numberVisits == maxNumberVisits)
role := Inactive
numberVisits := 0
else
perform waggle dance to hive
endif
endif

4. CONCLUSION

Using combinatorial methods for either configuration or input parameter testing can help make testing more effective at an overall lower cost. We concluded that this survey paper highlights the generation of test cases for combinatorial testing using artificial techniques. Our survey will be useful for the researchers and industry persons can easily navigate through the research undertaken in chosen field of interest so that they pursue research further.

REFERENCES

[1]. Yu Lei and K.C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," Proceedings of Third IEEE International High-Assurance Systems Engineering Symposium, 1998, pp. no. 254-261, 13-14 Nov 1998.

[2]. Xiang Chen, Qing Gu, Xin Zhang and Daoxu Chen, "Building Prioritized Pairwise Interaction Test Suites with Ant Colony Optimization," *QSIC, 9th International Conference on Quality Software*, pp.no 347-352, 24-25 Aug. 2009

[3]. Kewen Li and Zhixia Yang, "Generating Method of Pair-Wise Covering Test Data Based on ACO", *ETT and GRS, 2008 IEEE International Workshop on Geoscience and Remote Sensing*

and International Workshop on Education Technology and Training, vol.2, pp. no. 776-779, 21-22 Dec. 2008.

[4]. J.D. McCaffrey, "Generation of pairwise test sets using a simulated bee colony algorithm," *IRI '09, IEEE International Conference on Information Reuse & Integration*, pp.no. 115-119, 10-12 Aug. 2009.

[5]. Xiang Chen, Qing Gu, Jingxian Qi and Daoxu Chen, "Applying Particle Swarm Optimization to Pairwise Testing", *COMPSAC, 2010 IEEE 34th Annual Computer Software and Applications Conference*, pp. no. 107-116, 19-23 July 2010.

[6]. M.I. Younis, K.Z. Zamli and N.A.M Isa, "Generating Pairwise Combinatorial Test Set Using Artificial Parameters and Values", ITSim, *2008 IEEE International Symposium*, pp. no. 1-8, 26-28 Aug 2008.

[7]. S.A. Ghazi and M.A. Ahmed, "Pair-wise test coverage using genetic algorithms", *CEC, The 2003 Congress on Evolutionary Computation*, Vol. 2, pp. no. 1420-1424, 8-12 Dec. 2003.

[8]. S. Khatun, K.F. Rabbi, C.Y. Yaakub and M.F.J. Klaib "A Random search based effective algorithm for pairwise test data generation", INECCE*, 2011 IEEE International Conference on Electrical, Control and Computer Engineering*, pp. no.293-297, 21-22 June 2011.

[9]. James D. McCaffrey, "Generation of Pairwise Test Sets using a Genetic Algorithm", 33rd Annual IEEE International Computer Software and Applications Conference, 2009.

[10]. J.D. McCaffrey, "An Empirical Study of Pairwise Test Set Generation Using a Genetic Algorithm", *ITNG, 2010 IEEE Seventh International Conference on Information Technology: New Generations*, pp. no. 992-997, 12-14 April 2010.

[11]. Pedro Flores, Yoonsik Cheon, "PWiseGen: Generating Test Cases for Pairwise Testing Using Genetic Algorithms", IEEE International Conference on Computer Science and Automation Engineering (CSAE), June 2011.

[12]. Priti Bansal, Sangeeta Sabharwal, Shreya Malik, Vikhyat Arora, and Vineet Kumar, "An Approach to Test Set Generation for Pair-Wise Testing Using Genetic Algorithms," Search Based Software Engineering Vol.8084, pp.no.294-299, 2013.

[13]. Manisha Patil and P.J. Nikumbh, "Pair-wise Testing Using Simulated Annealing", 2212-0173 © 2012 Published by Elsevier Ltd.

[14]. J.D. McCaffrey, Software Testing: Fundamental Principles and Essential Knowledge, BookSurge Publishing, Charleston, SC, 2009.

[15]. R. Mandl, "Orthogonal Latin Squares: An Application of Experiment Design to Compiler Testing", *Communications of the ACM*, vol. 28, no. 10, pp. 1054-1058, 1985.

[16]. D.M. Cohen, S.R. Dalal, M.L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design", *IEEE Transactions on Software Engineering*, vol. 23, no. 7, pp. 437-443, July 1997.