J Inf Process Syst, Vol.13, No.5, pp.1089~1102, October 2017
https://doi.org/10.3745/JIPS.04.0019

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

Test Set Generation for Pairwise Testing Using Genetic Algorithms

Sangeeta Sabharwal* and Manuj Aggarwal*

Abstract

In software systems, it has been observed that a fault is often caused by an interaction between a small number of input parameters. Even for moderately sized software systems, exhaustive testing is practically impossible to achieve. This is either due to time or cost constraints. Combinatorial (*t*-way) testing provides a technique to select a subset of exhaustive test cases covering all of the *t*-way interactions, without much of a loss to the fault detection capability. In this paper, an approach is proposed to generate 2-way (pairwise) test sets using genetic algorithms. The performance of the algorithm is improved by creating an initial solution using the overlap coefficient (a similarity matrix). Two mutation strategies have also been modified to improve their efficiency. Furthermore, the mutation operator is improved by using a combination of three mutation strategies. A comparative survey of the techniques to generate *t*-way test sets using genetic algorithms was also conducted. It has been shown experimentally that the proposed approach generates faster results by achieving higher percentage coverage in a fewer number of generations. Additionally, the size of the mixed covering arrays was reduced in one of the six benchmark problems examined.

Keywords

Combinatorial Testing, Genetic Algorithm, Mixed Covering Arrays, Pairwise Testing, Test Set, *t*-way Testing

1. Introduction

Software testing is an important technique for producing reliable software systems and to maintain quality control. More than 50% of software development resources are being spent on testing [1]. For a complex system, which has n input parameters where each parameter can take single discrete v values, the exhaustive testing would require different v^n input cases. It has been observed that the number of necessary test cases grows exponentially as the number of input parameters increases [2]. Due to time and resource constraints, exhaustive testing is practically impossible to achieve and a technique that selects a subset of exhaustive test cases without causing much of a loss to the fault detection capability of the system is required. Combinatorial testing provides the solution.

Combinatorial testing is based on the fact that many faults can be exposed by interactions involving only a few input parameters. It creates tests by selecting values for input parameters and then combining these values, such that every combination of values of any t parameters are covered by at

Manuscript received May 23, 2014; first revision September 17, 2014; accepted October 28, 2014; onlinefirst August 17, 2015.
Corresponding Author: Manuj Aggarwal (mmanuj.aggarwal@gmail.com)
* Dept. of Computer Science and IT, Netaji Subhas Institute of Technology, Delhi, India (ssab63@gmail.com, mmanuj.aggarwal@gmail.com)

least one test case. Here, t is referred to as the strength of coverage and usually takes a small value [2]. The notion of t-way testing can substantially reduce the number of tests. A system with 20 parameters where each parameter can have 10 possible values requires 10^{20} tests for exhaustive testing; but only 180 tests for pairwise testing. Empirical studies have shown that t-way testing can effectively detect faults in various types of applications [3].

For the generation of test cases from parameter combinations, various combinatorial objects are used, such as Latin squares [4], orthogonal arrays [5], covering arrays [5,6], etc. Out of these, covering arrays (CAs) are the most widely used. A CA (N, k, S, t) is an $N\times k$ matrix that takes values from a finite set S of s symbols such that each $N\times t$ sub-matrix contains each possible t-tuple at least once, where N denotes the number of test cases, k is the number of input parameters, and s is the number of values each parameter can take. A mixed level covering array (MCA) (N, t, k, $v_1v_2....v_k$) is an $N\times k$ matrix where each column c_i, for $1\leq i\leq k$, can take values from the set (0, 1, ...v_i-1), such that each $N\times t$ sub-matrix contains each possible t-tuple at least once. Here, N denotes the number of test cases, k is the number of input parameters, and v_i is the number of values the parameter c_i can take when $1\leq i\leq k$. The matrix can also be represented as MCA (N, t, $s_1^{k1}s_2^{k2}.....s_p^{kp}$), which implies $k1$ parameters can have s_1 values, $k2$ parameters can have s_2 values, and so on. Thus, it is a matrix of $N\times k$ elements where each $N\times t$ sub-matrix contains at least one occurrence of each t-tuple corresponding to the columns and $k=\sum_{i=1}^{p}(k_i)$ [1].

A genetic algorithm (GA) is an evolutionary algorithm that has emerged as a practical, robust, optimization technique and a search method. GA is inspired by the way species evolve in nature via natural selection of the survival of the fittest individual [7]. The initial population is a set of possible candidate solutions (individual or chromosome) for the problem. The fitness value of an individual is calculated using the fitness function. A GA uses three operators, namely selection, crossover, and mutation, to improve the solution. A GA stops when stopping criteria is satisfied (i.e., either the solution is found or the maximum number of generations has taken place). The basic algorithm for a GA is presented in Fig. 1.

```
Initialise (population)
Evaluate (population)
While (stopping criteria not satisfied)
{
        Selection (population)
        Crossover (population)
        Mutation (population)
        Replacement (population)
        Evaluation (population)
}
```

Fig. 1. Basic algorithm of genetic algorithm.

In this paper, an approach for improving the performance of the GA is proposed. This proposed approach is capable of covering all of the pairwise interactions of the input set in a lesser number of generations. The performance of the GA is improved by improving the initial population and mutation function. A comparative review of the techniques to generate CAs using the GA is presented. Experimental results and a comparison of the proposed approach with an existing approach are also presented. The rest of the paper is organized as follows: Section 2 gives a brief overview of the work that has been done in this field. Section 3 describes the proposed approach to generate the optimal solution in a

lesser number of generations. Section 4 presents the experimental results and shows the effectiveness of the proposed approach. Section 5 concludes the paper and directions for future work are discussed.

2. Related Work

In the literature on combinatorial testing, several effective methods exist for constructing (near) optimal CAs. They are broadly classified into: mathematical methods, recursive methods, greedy methods, metaheuristic methods, and hybrid methods. Mathematical methods use mathematical functions to construct orthogonal arrays for test set generation [8]. Recursive methods require combinatorial objects of specific dimensions to construct larger instances of combinatorial objects. Greedy algorithms include the one-row-at-a-time approach and the In-Parameter-Order (IPO) algorithm. In the one-row-at-a-time approach, a single row of the array is constructed at each step until all t-sets have been covered. Examples include an automatic efficient test generator (AETG) [9] and the density-based greedy algorithm [10]. The IPO algorithm generates all t-sets for the first t factors and then incrementally expands the solution, both horizontally and vertically, until the array is complete [11]. Metaheuristic algorithms include tabu search [6], simulated annealing [12], ant colony optimization (ACO) [13], particle swarm optimization (PSO) [14,15], GA [16-22], Harmony Search [23,24], etc. The hybrid approach combines different methods to efficiently generate a small test suite. The mathematical method can be combined with metaheuristics, or greedy methods can be combined with mathematical methods to generate a test suite [25]. Next, GA approaches for t-way testing are discussed.

Ghazi [16] has proposed a technique for generating pairwise test configurations. The fitness function used for evaluating a chromosome is calculated as the number of distinct pairs covered by the chromosome divided by the total number of possible pairwise interaction configurations. However, the experimental data considered is simple. Moreover, the author has not discussed mutation and the crossover operators used in the algorithm.

McCaffrey [17,22] has proposed a genetic algorithm for pairwise test sets (GAPTS) technique for pairwise test case generation. For chromosome representation, integer array encoding is used. The fitness function defined is the total number of distinct pairs captured by the individual chromosome. Various GA parameter values defined are: population size, 20; selection method, roulette wheel selection; crossover, single crossover point; mutation rate, 0.001, etc. GAPTS uses a form of elitism, in which the individual with the highest fitness value in the population is immune from removal in each generation. For immigration, an individual with randomly generated chromosomes is inserted into the population after every 1,000 generations. The results obtained are comparable to or better than the other five algorithms examined. However, the GAPTS program required more time to generate pairwise test sets than other algorithms examined and the time to produce results increases with an increase in the size of input.

Flores and Cheon [18] developed an open source tool called PWiseGen for generating pairwise test sets. It is configurable, extensible, and reusable. The chromosome encoding used is integer array encoding. The authors have defined two fitness functions of counting the number of different pairs and penalizing for repeated pairs. The selection method used is roulette wheel selection. The crossover variants defined are the single crossover point, single random crossover point, multiple crossover point,

and multiple random crossover point. The authors defined smart mutations, namely similarity mutation, value occurrence mutation, and pair occurrence mutation. As per the experimental data, PWiseGen shows comparable results with other approaches.

Table 1. Comparison of different test case generation approaches using GA

Reference	*t*-way	Fitness function	GA parameters	Main features	Experimental results
Ghazi and Ahmed [16]	Pairwise test cases	Number of distinct pairs covered by the chromosome divided by total number of possible pairs		Proposed that GA can be used to generate *t*-way test sets.	Experimental data considered is simple.
McCaffrey [17,22]	Pairwise test cases	Total number of distinct pairs captured by the individual	Population size 20, roulette wheel selection method, single crossover point, mutation rate 0.001, elitism, immigration	Technique: GAPTS Chromosome representation: integer array encoding	Required more time to generate test sets. Takes longer time to produce results as input size increases.
Flores and Cheon [18]	Pairwise test sets	Counting the number of different pairs, and penalising for repeated pairs.	Roulette wheel selection method Crossover variants: single, single random, multiple, and multiple random crossover point	Developed an open source tool called PWiseGen Chromosome encoding: integer array encoding Introduced smart mutations	Execution time is not considered.
Yalan et al. [19]				Systematically examines the impact of and interactions between GA's five configurable parameters.	Elitism, lengthier evaluation process and creating fewer mutated individuals lead to better CA. Values of parameters considered are discrete.
Shiba et al. [20]	Test cases for t=2 and t=3	Number of new *t*-way combinations that are covered by the test case.	Tournament selection method, elitism, uniform crossover and mutation Stopping criteria: when the total number of generated candidate tests exceeds a given number.	Defined stagnation condition to escape from local minima. Compaction algorithm to reduce the number of test cases.	*t*-way test sets generated are not always optimal.
Bansal et al. [21]	Pairwise test cases	Counting the number of different pairs	Crossover strategy: crossover points are selected by identifying portion of chromosome covering least number of distinct pairs.	Initial solution: Hamming distance	Results obtained are fitter as compared to existing approach.

GA=genetic algorithm, GAPTS=genetic algorithm for pairwise test sets, CA=covering array.

Yalan et al. [19] designed three classes of experiments (i.e., pairwise, base choice and hill climbing) to systematically examine the impact of and interactions among the GA's five configurable parameters (population size, number of generations, crossover probability, mutation probability, and GA variants). It was observed that the selection methods (random selection of chromosomes and selecting inferior and eliminating superior chromosomes) with elitism generate better configurations. Employing a lengthier evaluation process improves the solution. Creating fewer mutated individuals leads to better CA. However, the authors have considered discrete values for configuration parameters.

Shiba et al. [20] proposed a test generation algorithm for combinatorial testing based on the GA. The fitness function calculates the number of new *t*-way combinations that are covered by the test case. Stopping criteria is achieved when the total number of generated candidate tests exceeds a given number. The selection operator used is tournament selection. An elite strategy is used to retain the best chromosome in the population at each generation. Uniform crossover and mutation are performed. In order to escape from local minima, the authors have defined stagnation condition that if there is no improvement in the solution for a specified number of generations, then massive mutation is applied to each position of every chromosome. A compaction algorithm is also proposed to improve the size of *t*-way test set. The authors conducted some experiments to show the effectiveness of the proposed algorithms. However, the *t*-way test sets generated are not always optimal.

Bansal et al. [21] proposed an approach to generate pairwise test cases using the GA. The authors generated the initial solution using the Hamming distance. A new crossover strategy that selects crossover points by identifying portions of the chromosome covering the least number of distinct pairs was also proposed. A comparative study of the different *t*-way test set generation approaches using GAs is summarized in Table 1.

As can be concluded from this literature survey, various approaches for *t*-way test set generation using the GA have been proposed. Most of the approaches have generated (near) optimal test sets. However, the time required to generate the *t*-way test set is considerably large. Thus, an approach that generates optimal t-way test sets in a fewer number of generations is required. In PWiseGen [18], apart from simple mutations, mutations specific to pairwise testing have also been defined. Hence, we aim to define a GA that is customized to two-way testing and investigate its efficiency.

3. GA Based Pairwise Test Set Generation (Proposed Approach)

GAs have been used to solve a large set of problems in artificial intelligence, including test set generation for pairwise testing. The basic GA improves the solution for a given problem by using operators such as crossover, selection, and mutation.

The key features of the proposed approach are as follows. 1) It provides a method to create an improved initial solution. 2) It offers a method that improves the value occurrences mutation. 3) It also provides a method to improve the pair occurrences mutation. 4) It facilitates to use a mixture of mutation methods.

The terminology used is defined as follows: A MCA (N, t, $s_1^{k1}s_2^{k2}.....s_p^{kp}$) represents the test set M of $N\times k$ size, where each column P_j ($1\leq j\leq k$) represents an input parameter and each row is a test case. An element $m_{i,j}$ in matrix M can take a value from the set {0,1,...V_j-1} where, V_j is the possible values the j^{th}

parameter can take. Thus, the parameter P_j can have a value from the set S_j={0,1,.....V_j-1}, where, V_j is the cardinality of the set S_j (1≤ j≤ k), N is the total number of test cases, and k is the total number of input parameters.

3.1 Creating an Initial Population

In the case of combinatorial testing, a candidate solution (chromosome) is represented as a sequence of test cases, where the count of test cases represents the CA number and the size of the test case is equal to the total number of input parameters [18]. A test case contains one value corresponding to each input parameter. For chromosome representation, integer array encoding is used as suggested in [18]. In a GA an initial solution can be generated randomly [18] or by using techniques such as Hamming distance [21], Euclidean distance [6], etc.

In our proposed approach, an overlap coefficient (a similarity measure) is used to generate the initial solution. An overlap coefficient measures the overlap between two sets and is defined as the set of intersections divided by the smaller size of the two sets [26,27].

$$Overlap(X,Y) = \frac{|\{X\} \bigcap \{Y\}|}{\min(|X|,|Y|)} \tag{1}$$

where, X and Y are sets. The algorithm for creating the initial population is described in Fig. 2.

Begin
Generate a random test case.
Add that random test case to the test set.
While test set size is not achieved
{
- Randomly generate P test cases (tc_1, tc_2, ...tc_P), where P =(test set size)/3.
- From the generated test case, select the test case having minimum overlap coefficient with the test set already created.
- Add it to test set.

}
End

Fig. 2. Algorithm for creating initial population.

First, a random test case is generated and added to the test set. Then, an iterative process is followed in which P candidate test cases {tc_1, tc_2,.....tc_p} are generated, where the value of P is one third of the test set size. Each tc_i, for 1≤ i≤ P, is compared with the test set already generated to calculate the overlap coefficient using Eq. (1) where:

X= set of pairs covered by the test set already generated

Y= set of pairs covered by the test case tc_i, where, 1≤ i ≤ P.

The test case having a minimum overlap coefficient with the test set already generated is added to the test set. The iterative process is repeated until the size of the test set already generated is less than N, where N is the test set size to be achieved. The generated solution is better in terms of fitness value (i.e., covers a greater number of distinct pairs).

3.2 Value Occurrences Mutation

Flores and Cheon [18] defined the value occurrences mutation as a mutation strategy that replaces a duplicate value of a parameter in an individual (chromosome) with a missing value of that parameter. An attempt has been made to improve the efficiency of the value occurrences mutation. The algorithm is described in Fig. 3.

- Begin
- Identify the value of Min_Value_Count for each parameter using (2).
- Identify a parameter's value whose occurrence is less than Min_Value_Count for that parameter.
- Identify a value for the identified parameter whose occurrence is highest among the possible values for the parameter.
- Replace the highest occurring value with a value having occurrence less than Min_Value_Count for that parameter.
- End

Fig. 3. Algorithm for improved value occurrences mutation.

For each parameter P_j ($1 \leq j \leq k$), a Min_Value_Count$_j$ value is calculated, where:

$$Min\_Value\_Count_j = max\ \{\{V_1, V_2 \ldots\ldots V_k\} - V_j\} \text{ for } 1 \leq j \leq k \tag{2}$$

It calculates the minimum number of times each value of a parameter must occur in order to cover all of the value pairs with different values of other parameters. For each parameter P_j ($1 \leq j \leq k$), it calculates the Min_Value_Count$_j$ by considering all the parameters P_1, P_2,.....P_k, excluding parameter P_j. Then, for the remaining k-1 parameters, their corresponding V_m value is identified, where V_m is the number of possible values for parameter P_m ($1 \leq m \leq k$). The highest numeric V_m value is assigned to the Min_Value_Count$_j$. Thus, for each parameter, the Min_Value_Count is calculated. It then identifies a value of parameter P_j, whose occurrence is less than the Min_Value_Count$_j$ for that parameter. Then, for that parameter, the highest occurring value is identified and replaced with the value whose occurrence is less than the Min_Value_Count for that parameter.

For example, for the problem of MCA (N, 2, $4^1 3^2 2^1$), parameter P_1 can have four values (a, b, c, d), parameter P_2 can have three values (e, f, g), parameter P_3 can have three values (h, i, j), and parameter P_4 can have two values (k, l). Thus, the values of V_1, V_2, V_3 and V_4 are 4, 3, 3, and 2, respectively. Using Eq. (2), the Min_Value_Count for P_1 is 3, and for P_2, P_3, and P_4 is 4. Thus, each value of P_1 must occur at least three times in order to form all possible pairs with three different values of P_2 and P_3 each.

Table 2 shows the MCA (N, 2, $4^1 3^2 2^1$). Here, each value of P_1 occurs at least three times and each value of P_2, P_3 and P_4 occurs at least four times. This is in conformation with our calculation of the Min_Value_Count value. While generating the solution, if an intermediate matrix contains three occurrences of "*e*" and five occurrences of "*f*," then value occurrences mutation first computes the Min_Value_Count for each parameter. Then, it identifies the value "*e*," as its occurrences are less than the Min_Value_Count for parameter P_2. Then for parameter P_2 it identifies a value whose occurrences are greater than the Min_Value_Count and is highest, which in this case is value "*f*." Finally, it replaces one occurrence of "*f*" with "*e*."

Table 2. Mixed covering array for (N, 2, $4^1 3^2 2^1$)

P_1	P_2	P_3	P_4
a	e	h	k
a	f	i	l
a	g	j	k
b	e	i	l
b	f	h	k
b	g	j	l
c	f	j	k
c	e	h	l
c	g	i	k
d	e	j	l
d	f	i	k
d	g	h	l

The value occurrences mutation improves the solution for the first few generations. When all of the occurrences of values are greater than or equal to their corresponding Min_Value_Count, then the mutation does not change the solution. Furthermore, an attempt has been made to improve the solution by changing the structure of the solution. The algorithm is described in Fig. 4.

- Begin
- Identify an uncovered pair ($value_i$, $value_j$) where $value_i \in S_i$ of parameter P_i and $value_j \in S_j$ of parameter P_j, and $1 \leq i < j \leq k$.
- Identify a test case tc_m ($1 \leq m \leq N$) where value of P_i is $value_i$.
- Identify another test case tc_n ($1 \leq n \leq N$) where value of P_j is $value_j$.
- Interchange the genes of tc_m and tc_n from gene location i+1 till location j.
- End

Fig. 4. Algorithm for improving structure of the solution in value occurrences mutation.

An uncovered pair ($value_i$, $value_j$) (i.e., a value pair not covered in the test set is identified). Here, $value_i \in S_i$ of parameter P_i and $value_j \in S_j$ of parameter P_j, and $1 \leq i < j \leq k$, (i.e., in the test case P_i) appears before P_j. A test case tc_m ($1 \leq m \leq N$) is identified where the value of P_i is $value_i$. Then, another test case tc_n ($1 \leq n \leq N$) is identified where value of P_j is $value_j$. The genes of tc_m and tc_n are interchanged from gene location i+1 till location j. Thereby, the resultant test set covers the pair ($value_i$, $value_j$) without there being much of a loss to the existing covered pairs.

3.3 Pair Occurrences Mutation

Flores and Cheon [18] have defined the pair occurrences mutation as a mutation strategy that replaces a duplicate pair in an individual with a missing pair of the same parameters. We have attempted to improve the pair occurrences mutation. The algorithm is presented in Fig. 5.

The shortcoming of the existing approach is the loss of the distinct pairs already covered, as the values that are being replaced might be contributing to the distinct pairs covered. Our proposed method attempts to reduce the loss of the existing distinct pairs covered. All of the values in the test set that are not covering a distinct pair have been identified. A value pair ($value_i$, $value_j$) having no occurrence in the test set was then identified. Here, $value_i \in S_i$ of parameter P_i and $value_j \in S_j$ of parameter P_j, and $1 \leq$

$\{i, j\} \leq k$ and $i \neq j$. Then a test case tc_l ($1 \leq l \leq N$) is identified in which either or both of the values corresponding to parameters P_i and P_j are not covering a distinct pair. Then, the value corresponding to parameters P_i and P_j in the test case tc_l is replaced with the $value_i$ and $value_j$, respectively.

- Begin
- Identify all the values in the test set which are not covering a distinct pair
- Select an uncovered pair ($value_i$, $value_j$), where $value_i \in S_i$ of parameter P_i and $value_j \in S_j$ of parameter P_j, and $1 \leq \{i, j\} \leq k$ and $i \neq j$.
- Identify a test case in which at least one of the values for parameter P_i and P_j are not covering a distinct pair.
- Replace the value corresponding to parameter P_i and P_j in the test case with $value_i$ and $value_j$ respectively.
- End

Fig. 5. Algorithm for improved pair occurrences mutation.

For example, for the problem of MCA (N, 2, $4^1 3^2 2^1$), parameter P_1 can have four values (a, b, c, d), parameter P_2 can have three values (e, f, g), parameter P_3 can have three values (h, i, j), and parameter P_4 can have two values (k, l). The total pairs to be covered are 53. To demonstrate our method, consider the test set given in Fig. 6. First, all of the values in the test set that are not covering a distinct pair are identified. The evaluation starts from test case tc_1 and continues from left to right (i.e., first, "a" in tc_1 is evaluated and then "e" in tc_1 and it continues until "k" in tc_7). It is observed in tc_6 that the value "l" does not contribute to the distinct pairs covered by the test set, as the pair "bl" is already covered by tc_2, "gl" by tc_3, and "jl" by tc_3. Similarly, "a" and "e" in tc_7 also do not contribute a distinct pair. Thus, value "l" in tc_6 and values "a" and "e" in tc_7 are identified. Next, for the given test set, uncovered pairs are identified. These are *af*, *ag*, *aj*, *be*, *bh*, *bk*, *ce*, *cf*, *ch*, *ck*, *ci*, *df*, *dg*, *di*, *dj*, *dl*, *ej*, *fh*, *fj*, *fk*, *gh*, *gi*, *gk*, *hl*, and *jk*. From these uncovered pairs, a pair is selected at random. The cases explained below may arise.

Aehk	*bfil*	*cgjl*	*dehk*	*aeil*	*bgjl*	*aeik*
tc_1	tc_2	tc_3	tc_4	tc_5	tc_6	tc_7

Fig. 6. Test set for the problem of mixed level covering array (N, 2, $4^1 3^2 2^1$).

Case 1: The uncovered pair "*fh*" is selected. Here, "*f*" belongs to parameter P_2 and "*h*" belongs to parameter P_3. The next step is to identify a test case in which at least one of the values for parameters P_2 and P_3 are not covering a distinct pair. As can be seen, no test case exists for which the values for P_2 and P_3 are marked. For tc_7 the value of P_2 is marked. Thus, the values of P_2 and P_3 are replaced in tc_7 by "*f*" and "*h*," respectively.

Case 2: Uncovered pair "*cf*" is selected. Here "*c*" belongs to parameter P_1 and "*f*" belongs to parameter P_2. The next step is to identify a test case in which at least one of the values for parameters P_1 and P_2 are not covering a distinct pair. As can be seen, the tc_7 values of P_1 and P_2 are marked. Thus, the values of P_1 and P_2 are replaced in tc_7 by "*c*" and "*f*," respectively.

Thus, the proposed approach helps to reduce the loss of pairs already covered by the test set.

3.4 Use of Mixture of Mutation Methods

In the proposed approach, a combination of three mutation methods, namely simple mutation, value occurrences mutation, and pair occurrences mutation, is used. In simple mutation, the gene to be replaced is randomly selected and its value is replaced with a valid random value for that gene location. For the first ¼ of the maximum possible generations, the value occurrences mutation is used. For the initial generations, where the initial solution may have missed certain values from the solution, the value occurrences mutation improves the results. Afterwards, until ¾ of the maximum possible generations, mutation method to be used is simple mutation is used. At that stage, in most of the cases, 98% coverage (in terms of total pairs) is achieved, leaving only a few uncovered pairs. Then, pair occurrences mutation that selectively finds an uncovered pair and tries to fix it in the test set without there being much of a loss to the existing pairs covered is used.

4. Experimental Results

The proposed approach has been implemented using the open source tool of PWiseGen [18]. It is implemented in Java programming language and generates test cases for pairwise testing using GA. A series of experiments were conducted in order to empirically evaluate the effectiveness of the proposed approach. The performance was evaluated using the following six benchmark problems of different sizes: 3^4, 3^{13}, $4^{15}3^{17}2^{29}$, $4^{1}3^{39}2^{35}$, 2^{100}, and 10^{20}, where a^b means "b" input parameters have "a" distinct possible values. These six benchmark problems were input to PWiseGen with an existing approach, as described in [18], and to our improved approach. The effectiveness of both of the approaches were measured in terms of percentage coverage (pairs covered out of total pairs) versus the generation number. An approach that achieves higher percentage coverage in a fewer number of generations is considered to be better. In order to analyze the effect of the initial solution and the mutation strategy, we kept other factors, such as population size, mutation rate, crossover strategy, etc. as the same for both of the approaches. Each approach was executed 30 times for each benchmark problem. The results obtained were averaged and plotted on a graph where the x-axis represents the generation number and the y-axis represents the percentage coverage. The results are summarized in Fig. 7. As can be analyzed from the graphs, our proposed approach achieves higher coverage in a fewer number of generations in all of the cases. Table 3 represents the size of the test set obtained for the six benchmark problems. It does so by using the proposed approach and existing approaches of the pairwise test set generation in the following work by Flores and Cheon (PWiseGen) [18], McCaffrey (GAPTS) [17], and Shiba et al. [20]; and other approaches such as AETG [9], which generates test sets using the greedy approach; by Chen et al. [14] using PSO; and Shiba et al. [20] using ACO. As can be seen, GA approaches give comparable results to other approaches taken under consideration. However, for Problem 6, AETG [9] generates better results. On examining GA approaches, it is observed that the proposed approach generates comparable results to other approaches using GA. Moreover, for the problem of size $4^{1}3^{39}2^{35}$, the size of the MCA has been reduced from 26 to 25 using the proposed approach.

Table 3. Results of the performance of proposed approach compared with the existing approaches of Pairwise test set generation for the six benchmark problems

	Total pairs	Proposed approach	PWiseGen [18]	GAPTS [17]	GA [20]	AETG [9]	ACO [20]	PSO [14]
Problem 1 (3^4)	54	9	9	9	9	9	9	9
Problem 2 (3^{13})	702	15	15	15	17	17	17	18
Problem 3 ($4^{15}3^{17}2^{29}$)	14026	34	34	35	37	41	37	38
Problem 4 ($4^{1}3^{39}2^{35}$)	17987	**25**	26	27	27	28	27	27
Problem 5 (2^{100})	19800	10	10	10	12	10	13	13
Problem 6 (10^{20})	19000	220	220	196	227	194	225	213

GAPTS=genetic algorithm for pairwise test sets, GA=genetic algorithm, AETG=automatic efficient test generator, ACO=ant colony optimization, PSO=particle swarm optimization.

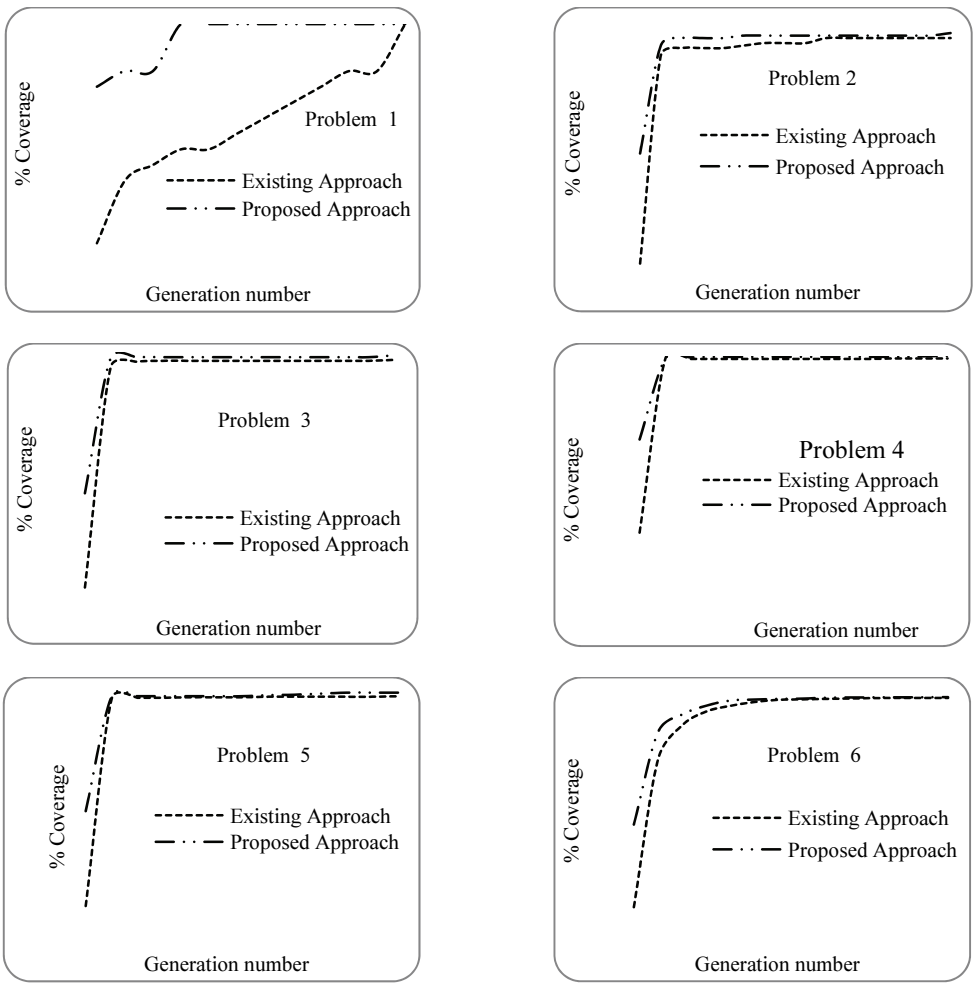

Fig. 7. Comparison of the proposed approach and existing approach for the six benchmark problems.

5. Conclusion

In this paper, a survey of the techniques generating *t*-way test sets from a GA has been conducted. It has been observed that the GA generates (near) optimal test sets. Mutation methods that are customized to pairwise testing have also been proposed by the researchers. We were motivated by these mutation methods, as we believe that a customized mutation method will generate results in a fewer number of generations, as compared to the simple mutation method. We have proposed algorithms that can improve these mutation methods, namely value occurrences mutation and pair occurrences mutation. Moreover we improved the performance of the algorithm by creating an input solution using a similarity matrix—the overlap coefficient. Lastly, rather than using a single mutation method, we used a combination of three mutation methods to mutate the solution. It can be concluded from the results that the proposed approach generates solutions with higher percentage coverage in a fewer number of generations. Moreover, the size of MCA (N, 2, 75, $4^{1}3^{39}2^{35}$) was reduced from 26 to 25 using the proposed approach. As for future work, we will further investigate the efficiency of the GA after customizing it to *t*-way testing. Moreover, we will extend our approach for the generation of higher strength *t*-way test cases.

References

[1] A.P. Mathur, *Foundations of Software Testing*, 1st ed. Delhi, India: Pearson Education, 2008, pp. 309-315.

[2] 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, 1997.

[3] Combinatorial Testing, http://www.combinatorialtesting.com/.

[4] A. Hartman, "Software and hardware testing using combinatorial covering suites," in *Graph Theory, Combinatorics and Algorithms.* New York, NY: Springer, 2005, pp. 237-266.

[5] G. Sherwood, "On the construction of orthogonal arrays and covering arrays using permutation groups," 2004; http://testcover.com/pub/background/cover.htm.

[6] L. Gonzalez-Hernandez, N. Rangel-Valdez, and J. Torres-Jimenez, "Construction of mixed covering arrays of strengths 2 through 6 using a tabu search approach," *Discrete Mathematics, Algorithms and Applications*, vol. 4, no. 3, 2012.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, 1st ed. Reading, MA: Addison-Wesley, 1989, pp. 1-22.

[8] K. A. Bush, "Orthogonal arrays of index unity," *Annals of Mathematical Statistics,* vol. 23, no. 3, pp. 426-434, 1952.

[9] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton, "The combinatorial design approach to automatic test generation," *IEEE Software,* vol. 13, no. 5, pp. 83–88, 1996.

[10] R. C. Bryce, C.J. Colbourn, and M.B. Cohen, "A framework of greedy methods for constructing interaction test suites," in *Proceedings of the 27th International Conference on Software Engineerin*, New York, 2005, pp. 146–155.

[11] J. D. McCaffrey, "Generation of pairwise test sets using a genetic algorithm," in *Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC '99)*, Seattle, WA, 2009, pp. 626-631.

[12] Y. Lei and K. C. Tai, "In-parameter-order: a test generation strategy for pairwise testing," in *Proceedings of 3rd IEEE International High-Assurance Systems Engineering Symposium*, Washington, DC, 1998, pp. 254-261.

[13] B. J. Garvin, M. B. Cohen, and M. B. Dwyer, "An improved meta-heuristic search for constrained interaction testing," in *Proceedings of the 1st International Symposium on Search Based Software Engineering*, Windsor, UK, 2009, pp. 13-22.

[14] X. Chen, Q. Gu, X. Zhang, and D. Chen, "Building prioritized pairwise interaction test suites with ant colony optimization," in *Proceedings of IEEE 9th International Conference on Quality Software (QSIC'09)*, Jeju, Korea, 2009, pp. 347-352.

[15] X. Chen, Q. Gu, J. Qi, and D. Chen, "Applying particle swarm optimization to pairwise testing," in *Proceedings of IEEE 34th Annual Computer Software and Applications Conference (COMPSAC)*, Seoul, Korea, 2010, pp. 107-116.

[16] B. S. Ahmed, K. Z. Zamli, and C. P. Lim, "Application of particle swarm optimization to uniform and variable strength covering array construction," *Applied Soft Computing*, vol. 12, no. 4, pp. 1330-1347, 2012.

[17] S. A. Ghazi and M. A. Ahmed, "Pair-wise test coverage using genetic algorithms," in *Proceedings of the 2003 Congress on Evolutionary Computation (CEC'03)*, Canberra, Australia, 2003, pp. 1420-1424.

[18] J. D. McCaffrey, "An empirical study of pairwise test set generation using a genetic algorithm," in *Proceedings of 7th International Conference on Information Technology: New Generations (ITNG)*, Las Vegas, NV, 2010, pp. 992-997.

[19] P. Flores and Y. Cheon, "PWiseGen: generating test cases for pairwise testing using genetic algorithms," in *Proceedings of 2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, Shanghai, China, 2011, pp. 747-752.

[20] L. Yalan, C. Nie, J. M. Kauffman, G. M. Kapfhammer, and H. Leung, "Empirically identifying the best genetic algorithm for covering array generation," presented at 3rd International Symposium on Search Based Software Engineering, Szeged, Hungary, September 10-12, 2011.

[21] T. Shiba, T. Tsuchiya, and T. Kikuno, "Using artificial life techniques to generate test cases for combinatorial testing," in *Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)*, Hong Kong, 2004, pp. 72-77.

[22] P. Bansal, S. Sabharwal, S. Malik, V. Arora, and V. Kumar, "An approach to test set generation for pair-wise testing using genetic algorithms," in *Proceedings 5th International Symposium on Search Based Software Engineering (SSBSE2013)*, St. Petersburg, Russia, 2013, pp. 294-299.

[23] J. D. McCaffrey, "Generation of pairwise test sets using a genetic algorithm," in *Proceedings of 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC'99)*, Seattle, WA, 2009, pp. 626-631.

[24] A. R. A. Alsewari and K. Z. Zamli, "Design and implementation of a harmony-search-based variable-strength t-way testing strategy with constraints support," *Journal of Information and Software Technology*, vol. 54, no. 6, pp. 553-568, 2012.

[25] A. R. A. Alsewari and K. Z. Zamli, "A harmony search based pairwise sampling strategy for combinatorial testing," *International Journal of the Physical Sciences*, vol. 7, no. 7, pp. 1062-1072, 2012.

[26] M. B. Cohen, C. J. Colbourn, and A. C. Ling, "Constructing strength three covering arrays with augmented annealing," *Discrete Mathematics*, vol. 308, no. 13, pp. 2709-2722, 2008.

[27] Overlap coefficient, http://en.wikipedia.org/wiki/Overlap_coefficient.

Sangeeta Sabharwal http://orcid.org/0000-0003-4693-213X

She is currently working as Professor and Head of the Department for Information Technology at Netaji Subhas Institute of Technology in Delhi, India. Her areas of interest are Software Engineering, Metamodeling, Object Oriented Analysis, Software Testing, and Data warehousing. Currently she is actively involved in applying different Soft Computing Techniques to different areas of software engineering, mainly in the area of testing. She has published papers in several international journals and conferences. She has also written a book on software engineering. Numbers of students are pursuing their Ph.D. under her guidance.

Manuj Aggarwal http://orcid.org/0000-0003-1650-8629

He obtained B.Tech. degree in Computer Science Engineering from University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi and M.Tech. degree in Information Systems from Netaji Subhas Institute of Technology (NSIT), Delhi. Currently, he is pursuing Ph.D. from NSIT, Delhi. His areas of interests are Software Engineering, Software Testing, Object Oriented Modeling and Technology, Programming Languages, Model-based Testing, Domain Specific Modeling and Applications of Soft Computing in Software Testing.