International Journal of Research Studies in Computer Science and Engineering (IJRSCSE)
Volume 2, Issue 5, May 2015, PP 29-37
ISSN 2349-4840 (Print) & ISSN 2349-4859 (Online)
www.arcjournals.org

Constructing T-Way Test Cases Using Genetic Algorithms

M.Lakshmi Prasad [1], S.Susmitha [2], C.Sai Sharanya[3], B.Vishnu Praneeth[4]

[1]NBKR Institute of Science & Technology
[1]Department of Computer Science & Engineering
[1]Nellore, AP, India
[2, 3, 4] NBKR Institute of Science & Technology
[2, 3, 4] Department of Computer Science & Engineering
[2, 3, 4] Nellore, AP, India
[1]prasad.hinduniv@gmail.com, [2]susmitha107@gmail.com, [3]sharanya073@gmail.com, [4]vishnupraneethreddy@gmail.com

Abstract: *there has been a developing pattern to create programming utilizing distinctive parts. Along these lines the expense of the product diminishes and the designer has the capacity finish the framework productively. The parts code could possibly be obvious to the designer. Testing, for this situation, requires the improvement of an arrangement of test setups that can be connected on the product. In any case, for programming that contains an extensive number of parts, it is infeasible to test every last test arrangement inside of the constrained testing spending plan and time. In this paper we propose a CA-based system that recognizes an arrangement of test Configurations that are required to amplify pair-wise scope, with the imperative that the quantity of test setups is predefined. Pairwise testing is a combinatorial procedure used to decrease the quantity of experiment inputs to a framework in circumstances where comprehensive testing with every conceivable data is unrealistic or restrictively costly. Given an arrangement of information parameters where every parameter can tackle one of a discrete arrangement of qualities, a pairwise test set comprises of a gathering of vectors which catches every single conceivable blend of sets of parameter qualities. The era of negligible pairwise test sets has been indicated to be a NP-complete issue and there have been a few deterministic calculations distributed. This paper introduces the aftereffects of an examination of creating pairwise test sets utilizing a hereditary calculation. Contrasted and distributed results for deterministic pairwise test set era calculations, the hereditary calculation methodology delivered test sets which were equivalent or better regarding test set size in 39 out of 40 cases. In any case, the hereditary calculation methodology obliged longer preparing time than deterministic methodologies in all cases. The outcomes show that the era of pairwise test sets utilizing a hereditary calculation is conceivable, and recommend that the methodology may be functional and valuable in certain testing situation*

Keywords: *component; formatting; style; styling; insert (key words)*

1. INTRODUCTION

Pairwise testing is an effective, combinatorial testing technique that, for each pair of input parameters to a software system, tests all possible combinations of these parameters. It is based on the observation that most software errors are caused by interactions of at most two factors such as input values. Its test suite is much smaller than that of exhaustive testing yet still very effective in finding errors. However, one problem of pairwise testing is that finding the least number of test cases has been proven to be an NP-complete problem. This means that an efficient way to find an optimal solution is not known and that the time required for finding a minimum number of test cases grows rapidly when the numbers of parameters and possible values increase.

Pairwise testing is a combinatorial testing technique in which every pair of input parameters of software is tested. It is regarded as a reasonable cost-benefit compromise among combinatorial testing methods; it can be performed much faster than exhaustive testing that tests all combinations of all input parameters, and is more effective than less exhaustive methods that fail to exercise all possible pairs of input parameters. The reasoning behind pairwise testing is that the majority of software errors are caused by a single input parameter or a combination of two input parameters. Pairwise testing thus requires that each pair of input parameter values be captured at least by one test case. As an example, let us consider software that takes three input parameters, say *x*, *y*, and *z*.

If each parameter can have three different values, then there will be 27 different pairs: (*x*1, *y*1), (*x*1, *y*2),.,., (*y*3, *z*3). A test case (*x*1, *y*3, *z*2), for example, captures three of these 27 pairs: (*x*1, *y*3), (*x*1, *z*2), and (*y*3, *z*3). By selecting test cases judiciously, all pairs of input parameters can be exercised with a minimum number of test cases; e.g., a set of nine test cases can capture all 27 pairs of three parameters, each with three different values.

A genetic algorithm is a technique that simulates the natural process of evolution. It was discovered as a useful tool for dealing with search and optimization-related problems and is known to be effective for finding solutions for problems with a huge search space and complexity. In a genetic algorithm, a population of candidate solutions, called *individuals*, to a problem evolves toward better solutions. The evolution is governed by so-called genetic operators such as mutation and crossover that select and modify individuals to form a new population. In general, a fitter individual has a better chance to survive and prevail in a population.

Genetic algorithms use biological models to emulate the process of evolution, where a population is made of a set of possible solutions called *individuals*]. The search starts with an initial population of which individuals are typically generated randomly. The population is evolved into a new generation by applying operations inspired by genetics and natural selection, such as selection, crossover, and mutation. This evolution process is repeated until a solution is found in the population or a certain stopping condition, e.g., the maximum number of iterations, is met. The search is guided by a fitness function that calculates the fitness values of the individuals in the population in that the fitter ones have a better chance to survive and thus evolve into the next generation. The effectiveness of a genetic algorithm is thus determined in part by the quality of its fitness function. For an algorithm to be considered to be genetic, it should at least have the following key elements.

Chromosome encoding. This is a way to represent a possible solution. A chromosome consists of genes representing a feature of an individual, and the possible values for a gene are called alleles. For example, the eye color feature of a person is a gene, and the alleles for the gene could be black, brown, blue, and green. The combination of genes in a chromosome is what defines an individual's set of features, and its encoding can vary widely depending on the specific problem to be solved.

Fitness function. This is a means to measure each individual's potential. It determines how good an individual is amongst all the others. The fitness value—calculated by a fitness function and associated with each individual— is the element used to determine which individuals have more opportunities to prevail in a population.

Genetic operations. These are the rule for evolution, as they are applied to the individuals of a population to facilitate their evolutions. The most common genetic operations are (a) *selection* that selects individuals for reproduction, (b) *crossover* that combines the genes of two parents and generates two new children, (c) *mutation* that modifies the genes of individuals randomly, and (d)*replacement* that defines the rules of replacing existing individuals in a population with the newly created individuals.

1.1 History

Pairwise 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, we find the technique to be over promoted and poorly understood. Knowledge of the weaknesses of the pairwise testing technique, or of any testing technique, is essential if we are to apply the technique wisely.

Different test generation strategies have been published for pairwise testing. One strategy starts with an empty test set and adds one test at a time. To generate a new test, the strategy produces a number of possible candidate tests according to a greedy algorithm and then selects one that covers the most uncovered pairs. Another approach to generating a pairwise test set is to use orthogonal arrays. The original method of orthogonal arrays requires that all parameters have the same number of values and that each pair of values be covered the same number of times. The first requirement can be relaxed by adding *don't care* values for missing values. But the use of *don't care* values creates extra tests. The second requirement is considered unnecessary for software testing and also creates extra tests for pairwise testing.

1.2 Motivation

Pairwise testing (or 2-way testing) is a specification based testing criterion, which requires that for each pair of input parameters of a system, every combination of valid values of these two parameters be covered by at least one test case. Empirical results show that pairwise testing is practical and effective for various types of software systems. By seeing the graph we have found that there are 75% of errors can be covered by applying pair wise testing. Hence pair wise testing can be used in web applications.

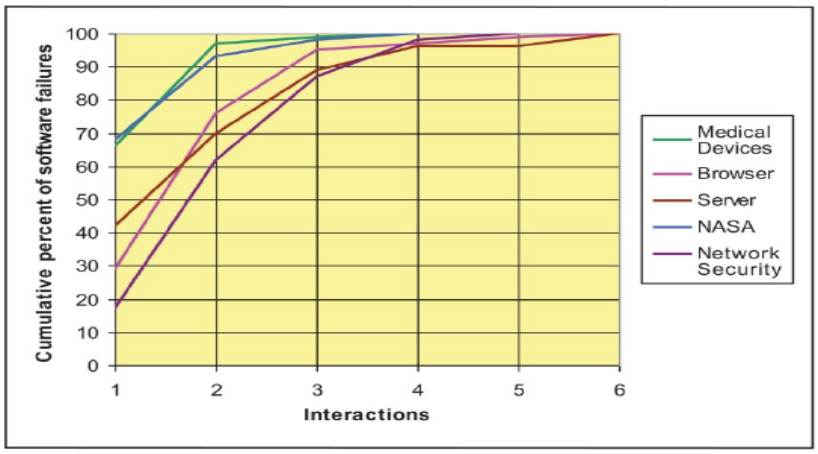

Figure 1. Percentage of failures triggered by *t*-way interactions

2. LITERATURE SURVEY

T. Shiba et al, [226] had used artificial life techniques to generate test cases for combinatorial testing. Combinatorial testing is a specification-based testing criterion, which requires that for each t-way combination of input parameters of a system, every combination of valid values of these t parameters be covered by at least one test case. Their approach is motivated by the observation that in many applications a significant number of faults are caused by interactions of a smaller number of parameters. They had proposed a new test generation algorithms for combinatorial testing based on two artificial life techniques: a genetic algorithm (GA) and an Ant Colony Algorithm (ACA).

Bestoun S. Ahmed, et al, [93] had applied the Particle Swarm Optimization strategy to uniform and variable strength covering array construction.

B.S. Ahmed and K.Z. Zamli [145] had proposed PSTG-a t-way strategy adopting particle swarm optimization. As an activity to ensure quality and conformance, testing is one of the most important activities in any software or hardware product development cycle. Often, the challenge in testing is that the system may support a wide range of configurations. Ideally, it is desirable to test all of these configurations exhaustively. However, exhaustive testing is practically impossible due to time and resource limitations. To address this issue, there is a need for a sampling strategy that can select a subset of inputs as test data from an inherently large search space. Recent findings demonstrate that t-way interaction testing strategies based on artificial intelligence (i.e. where t indicates interaction strength) have been successful to obtain a near optimal solution resulting into smaller test set to be considered. Motivated by such findings, they have developed a new test generation strategy, called Particle Swarm Test Generator (PSTG). They had discussed the design of PSTG and demonstrate their preliminary test size reduction results against other competing t-way strategies including IPOG, WHITCH, Jenny, TConfig, and TVG.

A. Calvagna, G. Pappalardo and E. Tramontana, [243] had proposed a novel approach to effective parallel computing of t-wise covering arrays. They had presented a novel parallel technique to compute t-wise covering arrays. The massive computational work, implied by the considered task when large configuration spaces are modeled, is distributed over a scalable set of parallel computing resources by means of an MPI-compliant algorithm. Due to NP-completeness of the covering array problem, existing research on combinatorial generation algorithms commonly assumes this computation task as strictly sequential. Conversely, basing on inherent combinatorial properties, we show that it is possible to scatter the overall workload into several and independent processing sub-tasks, and then collect all outcomes into a global solution whose size is still comparable to that of a

sequentially computed solution. Their reported results show that in this way significant speed-up is achieved on the computation times with respect to the sequential computation of the same task.

Mohammed I. Younis and Kamal Z. Zamli, [233] had presented the MC-MIPOG- A Parallel t-Way Test Generation Strategy for Multicore Systems. Combinatorial testing has been an active research area in recent years. One challenge in this area is dealing with the combinatorial explosion problem, which typically requires a very expensive computational process to find a good test set that covers all the combinations for a given interaction strength (t). Parallelization can be an effective approach to manage this computational cost, that is, by taking advantage of the recent advancement of multicore architectures. In line with such alluring prospects, their work presents a new deterministic strategy, called multicore modified input parameter order (MC-MIPOG) based on an earlier strategy, input parameter order generalized (IPOG). Unlike its predecessor strategy, MCMIPOG adopts a novel approach by removing control and data dependency to permit the harnessing of multicore systems. Experiments are undertaken to demonstrate speedup gain and to compare the proposed strategy with other strategies, including IPOG. The overall results demonstrate that MC-MIPOG outperforms most existing strategies (IPOG, IPOF, IPOF2, IPOG-D, ITCH, TConfig, Jenny, and TVG) in terms of test size within acceptable execution time. Unlike most strategies, MC-MIPOG is also capable of supporting high interaction strengths of $t > 6$.

B.S. Ahmed and K.Z. Zamli, [146] had proposed T-Way Test Data Generation Strategy Based on Particle Swarm Optimization. Due to market demands, software has grown tremendously in size and functionalities over the years. As side effects of such growth, there tend to be more and more unwanted interaction between software and system parameters. These unwanted interactions can sometimes lead to nasty and difficult bugs to detect. In order to address these issues, t-way strategies (i.e. where t indicates interaction strength) are helpful to generate a set of test cases (i.e. to form a complete suite) that cover the required interaction strength as least once from a typically large space of possible test values. They had highlighted a new t-way strategy based on Particle Swarm Optimization, called PSTG. Preliminary results demonstrated that PSTG compares well against other existing t-way strategies.

Bestoun S. Ahmed, Kamal Z. Zamli and Chee Peng Lim [147] had constructed a t-way interaction test suite using the particle swarm optimization approach.

Yu Lei et al, [201] had proposed a IPOG- a general strategy for t-way software testing. Most existing work on t-way testing has focused on 2-way (or pairwise) testing, which aims to detect faults caused by interactions between any two parameters. However, faults can also be caused by interactions involving more than two parameters. They had generalized an existing strategy, called In-Parameter-Order (IPO), from pairwise testing to t-way testing. A major challenge of their generalization effort is dealing with the combinatorial growth in the number of combinations of parameter values. They had described a t-way testing tool, called FireEye, and discuss design decisions that are made to enable an efficient implementation of the generalized IPO strategy. They also report several experiments that are designed to evaluate the effectiveness of FireEye.

3. EXISTING PHASE

Combination strategies are a class of test-case selection methods where test cases are identified by choosing "interesting" values1, and then combining those values of test object parameters. The values are selected based on some combinatorial strategy. Some combination strategies are based on techniques from experimental design.

This section first explains the different coverage criteria, normally associated with combination strategies and then briefly describes the combination strategies that were identified in the literature. The combination strategies have been organized into different classes based on the amount of randomness of the algorithm and according to how the test suites are created. Figure 2.1 shows an overview of the classification scheme. The combination strategies labeled non-deterministic all depend to some degree on randomness. A property of these combination strategies is that the same input parameter model may lead to different test suites. The simplest non-deterministic combination strategy is pure random selection of test cases. The group of non-deterministic combination strategies also includes two heuristic methods, CATS and AETG.

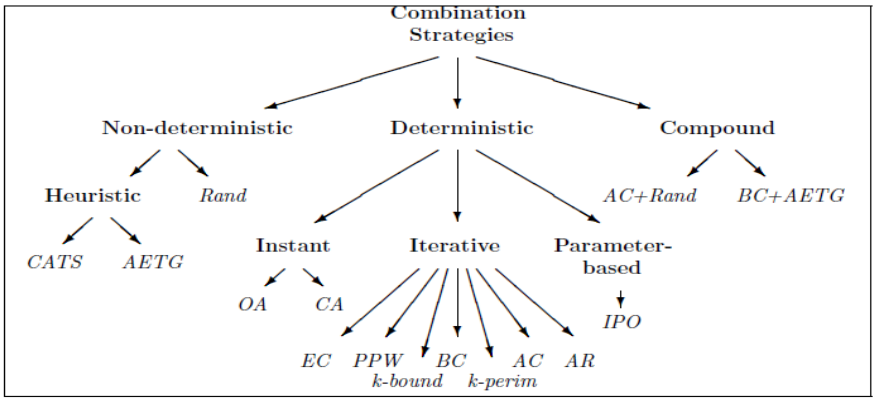

Figure2. *Classification Scheme for Combination Strategies*

The deterministic combination strategies group is further divided into three subgroups, instant, iterative, and parameter-based. All of these combination strategies will always produce the same result from a specific input parameter model. The two instant combination strategies, Orthogonal Arrays (OA) and Covering Arrays (CA), produce the complete test suite directly. The largest group of combination strategies is iterative. They share the property that the algorithms generate one test case at a time and add it to the test suite.

Each Choice (EC), Partly Pair-Wise (PPW), Base Choice (BC), All Combinations (AC), and Anti-random (AR) all belong to the iterative combination strategies. The parameter-based combination strategy, In Parameter Order (IPO), starts by creating a test suite for a subset of the parameters in the input parameter model. Then one parameter at a time is added and the test cases in the test suite are modified to cover the new parameter. Completely new test cases may also need to be added.

Like many test-case selection methods, combination strategies are based on coverage.. The following subsections define the coverage criteria satisfied by combination strategies are included.

Each-used (also known as 1-wise) coverage is the simplest coverage criterion. 100% each-used coverage requires that every interesting value of every parameter is included in at least one test case in the test suite. 100% Pair-wise (also known as 2-wise) coverage requires that every possible pair of interesting values of any two parameters are included in some test case. Note that the same test case may cover more than one unique pair of values.

A natural extension of pair-wise (2-wise) coverage is t-wise coverage, which requires every possible combination of interesting values of t parameters be included in some test case in the test suit, t-wise coverage is formally defined. A special case of t-wise coverage is N -wise coverage, where N is the number of parameters of the test object. N -wise coverage requires all possible combinations of all interesting values of the N parameters be included in the test suite.

The each-used, pair-wise, t-wise, and N -wise coverage criteria are purely combinatorial and do not use any semantic information. More coverage criteria can be defined by using semantic information. Cohen et al. indicate that valid and error parameter values should be treated differently with respect to coverage. Normal values lie within the bounds of normal operation of the test object, and error values lie outside of the normal operating range. Often, an error value will result in some kind of error message and the termination of the execution. To avoid one error value masking another author suggests that only one error value of any parameter should be included in each test case. This observation was also made and explained in an experiment also.

4. PROPOSED SYSTEM

Our proposed approach uses Genetic algorithm to generate the configuration sets and based on fitness of each set we can select a set that can serve as S. There are two objectives for the test configuration coverage problem.

1) To generate a set of configurations, S, that can cover all the pair-wise interactions between the components.
2) To minimize |S|, i.e., to minimize the number of configurations that can cover all interaction elements.

Chromosome structure Recall from the previous sections that we have a number of parameters, p. and each parameter may a number of equivalence classes of values v. Each chromosome T sub set of C consists of a number of configurations, where T is the set of all possible test configurations. Each configuration (in the chromosome) is in the form of {v... v_p} having one value for each parameter.

Fitness function The fitness function used for evaluating a chromosome C is calculated as the number of distinct pair-wise interaction configurations covered by all of the chromosome's configurations, divided by the total number of possible pair-wise interaction configurations $|\Phi_2|$, where Φ_2 is the set of all possible pair-wise interaction configurations.

For example, assume that there is a chromosome having two configurations for three parameters: {{1, 2, and 2}, {l, 1, 2}}, let's suppose each of the parameters can take two possible values, namely 1 or 2. In this case, the set of pair-wise interaction configurations covered by the chromosome in hand is calculated as follows.

N_1 = {{l, 2, X}, {l, X, 2}, {X, 2, 2}}

N_2 = {{1, 1, X}, {1, X, 2}, {X, 1, 2}}.

Where Ni is the set of distinct pair-wise interaction elements covered by configuration i.

Accordingly, the overall number of distinct pair-wise configurations covered by the chromosome = 3+3-1=5 and

Φ = {{l, 1, X}, {1, 2, X}, {2, 1, X), {2, 2, X), {1, X, 1}, {1, X, 2),{2, X, 1}, {2, X, 2}, {X, 1, 1}, {X, 1, 2}, {X. 2, 1}, {X, 2, 2} },i.e., Φ_2 = 12.

Accordingly, the fitness of the chromosome {{1, 2, 2}, {l, 1, 2}} for this particular case is 5/12 = 0.42. While the chromosome {{l, 2, 2}, {l, l, l}} would have a fitness = 0.5.

Consider a system which has n input parameters where each parameter can take on a single, discrete value. In many situations exhaustive testing of all possible combinations of input values is not feasible. For example, if n = 20 input parameters, where each parameter can be assigned one of 10 values, there are 1020 different input sets. If tests can be executed at a rate of 1,000 cases per second, a test run would require 1017 seconds, or roughly 3 billion years, to complete. Even when the total number of test case combinations is small, exhaustive testing may not be possible if each test case is expensive. Pairwise testing is a combinatorial technique which selects a subset of all possible test case input combinations.

A pairwise test set consists of a collection of test vectors which captures all possible combinations of pairs of input parameter values. In informal terms, for two parameters p0 and p1, and any valid values v0 for p0 and v1 for p1, there is a test vector in which p0 has the value v0 and p1 has the value v1. The concept is best illustrated by example. Suppose a system has four parameters, p0, p1, p2, and p3. Further, suppose that parameter p0 can accept one of two possible values, {a0, and a1}. And suppose the possible values for parameters p1, p2, and p3 are {b0, b1, b2, b3}, {c0, c1, c2}, and {d0, d1} respectively. For this situation there are a total of 2 * 4 * 3 * 2 = 48 combinations of input values. For example, one arbitrary test vector is {a0, b2, c1, d0}. Additionally, for this situation there are a total of 44 pairs of input values:

{a0, b0}, {a0, b1}, {a0, b2}, {a0, b3}, {a0, c0}, {a0, c1}, {a0, c2}, {a0, d0}, {a0, d1}, {a1, b0}, {a1, b1}, {a1, b2}, {a1, b3}, {a1, c0}, {a1, c1}, {a1, c2}, {a1, d0}, {a1, d1}, {b0, c0}, {b0, c1}, {b0, c2}, {b0, d0}, {b0, d1}, {b1, c0}, {b1, c1}, {b1, c2}, {b1, d0}, {b1, d1}, {b2, c0}, {b2, c1}, {b2, c2}, {b2, d0}, {b2, d1}, {b3, c0}, {b3, c1}, {b3, c2}, {b3, d0}, {b3, d1}, {c0, d0}, {c0, d1}, {c1, d0}, {c1, d1}, {c2, d0}, {c2, d1}.

A pairwise test set for this scenario consists of a collection of test vectors which capture all input pairs. For example, the following test set of 12 test vectors captures all 44 possible pairs of input values:

0: a0 b0 c0 d0 1: a1 b0 c1 d1 2: a1 b1 c2 d0

3: a0 b2 c2 d1 4: a1 b3 c0 d1 5: a0 b1 c1 d0

6: a1 b2 c0 d0 7: a0 b3 c1 d0 8: a0 b0 c2 d0

9: a0 b1 c0 d1 10: a0 b2 c1 d0 11: a0 b3 c2 d0

Because the intent of pairwise testing is to reduce the number of test cases, smaller test set sizes are better than larger test set sizes. The fundamental notion behind pairwise testing is the premise that most software faults result from either single-value inputs or by an interaction between pairs of input values. Generating minimal size pairwise test sets is an NP Complete problem. One approach to pairwise test set generation is the use of orthogonal arrays. Another approach is the use of an iterative technique which employs a greedy algorithm to construct a test set one vector at a time until all possible pairs are captured. A third approach is to generate a test set for the first two parameters, and then iteratively extend the test set to account for each remaining parameter. A comprehensive review of the research literature on pairwise test set generation techniques yielded a single paper which explored the use of a genetic algorithm.

That our project presented the results of a feasibility study performed on a single input set. However, the input set was small (four parameters, each of which could take on one of three values) and the resulting pairwise test set size was non-optimal (10 test vectors rather than 9 vectors). Additionally, the study did not compare the effectiveness of the approach with other techniques. This project extends that feasibility study and demonstrates the use of a genetic algorithm to generate pairwise test sets. The technique is referred to as GAPTS (Genetic Algorithm for Pairwise Test Sets) generation. The GAPTS algorithm was executed against seven benchmark input sets, and the GAPTS results were compared with the results produced by five other pairwise test set generation 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.

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

($p1$….., pn) = selectParents(P)

($c1$……. cn) = crossover ($p1$… pn);

Step 5: **if** (mutate?) **then**

$c1$ = mutate ($c1$);

$c2$ = mutate ($c2$);

end

Step 6: $C = C \cup \{c1; c2\}$

end

Step 7: **if** (immigration?) **then**

I = createImmigrants();

end

Step 8: P = updatePopulation(P, $C \cup I$)

Step 9: $i = i + 1$**end**

End

5. EXPERIMENTAL RESULTS

The GAPTS algorithm is compared with other deterministic algorithms which are shown in Table1. Using published results as guidelines, for a given input set an initial test set size was supplied to the GAPTS algorithm. We have implemented a tool called GAPTS based on ANNs algorithm. Experiments have been conducted by considering different systems which can be defined using different parametric values. The parametric values considered for different systems have shown below

S1: 4 (12-value parameters), **S2**: 4 (11-value parameters), **S3**: 13 (3-value parameters), **S4**: 61 parameters (15 (4- value parameters), 17 (3- value parameters), 29 (2- value parameters)), **S5**: 75 parameters (1 (4- value parameters), 39 (3- value parameters), 35 (2- value parameters)), **S6**: 100 (2-value parameters), **S7**: 20 (10- value parameters).

The numbers of test cases generated for different systems which can be defined using various parametric combinations are shown in the table 2. It can be seen from the above table that the number of test cases generated by ANN-PTCG are minimal considering any of the system configurations.

Table2. *Comparison of Different Strategies*

System	S1	S2	S3	S4	S5	S6	S7
AETG	n/a	9	15	41	28	10	194
PICT	12	13	20	38	31	16	216
QICT	12	11	22	42	34	16	219
All Pairs	12	10	22	41	30	16	664
Pair Test	n/a	9	19	36	29	15	218
GAPTS	12	9	15	35	27	10	196

From the above table, it is seen that the number of test cases generated by our technique is quiet minimal for considering any of the system configurations.

6. CONCLUSION

Hence we concluded that a genetic algorithm test generation strategy was proposed to generate the optimal test cases and improve the quality of pair wise testing result. We have implemented this test generation algorithm and have shown some empirical results. When used properly, pair wise test set generation is an important technique that can help you produce better software systems.

The GA strategy was presented in this paper can be easily extended for multi-way testing. We are investigating possible improvements of algorithm without increasing time complexity.

REFERENCES

[1] T. Shiba, T. Tsuchiya and T. Kikuno, "**Using artificial life techniques to generate test cases for combinatorial testing**", *COMPSAC, Proceedings of the 28th Annual International Computer Software and Applications Conference*, pp. no. 72-77 vol.1, 28-30 Sept. 2004

[2] Bestoun S. Ahmed, Kamal Z. Zamli, and Chee Peng Lim, "**Application of Particle Swarm Optimization to uniform and variable strength covering array construction,"** Application Software Comput., Vol.12, pp.no.1330-1347, April 2012.

[3] B.S. Ahmed and K.Z. Zamli, "**PSTG: A T-Way Strategy Adopting Particle Swarm Optimization**," *AMS, 2010 Fourth Asia International Conference on Mathematical/Analytical Modeling and Computer Simulation*, pp.no.1-5, 26-28 May 2010.

[4] A. Calvagna, G. Pappalardo and E. Tramontana, "**A Novel Approach to Effective Parallel Computing of t-Wise Covering Arrays**", *WETICE, 2012 IEEE 21st International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. no. 149-153, 25-27 June 2012.

[5] Mohammed I. Younis and Kamal Z. Zamli, "**MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems**", Proceedings of the ETRI Journal, Volume 32, Number 1, February 2010.

[6] B.S. Ahmed and K.Z. Zamli, "**T-Way Test Data Generation Strategy Based on Particle Swarm Optimization**,", *2010 Second International Conference on Computer Research and Development*, pp.93-97, 7-10 May 2010.

[7] Bestoun S. Ahmed, Kamal Z. Zamli and Chee Peng Lim ,"**Constructing A T-Way Interaction Test Suite Using The Particle Swarm Optimization Approach,"** ICIC, *International Journal of Innovative Computing, Information and Control* Vol.8, No.1, pp.no.431-451, January 2012

[8] Yu Lei, Raghu Kacker, D. Richard Kuhn, Vadim Okun, James Lawrence, "**IPOG: A General Strategy for T-Way Software Testing**", Proceeding of: 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems, 2010.