Journal of Computer Science 9 (8): 972-980, 2013
ISSN: 1549-3636

doi:10.3844/jcssp.2013.972.980 Published Online 9 (8) 2013 (http://www.thescipub.com/jcs.toc)

OPTIMIZATION OF TEST CASES BY PRIORITIZATION

[1]T. Prem Jacob and [2]T. Ravi

[1]Department of CSE, Sathyabama University, Chennai, India
[2]Department of CSE, Srinivasa Institute of Engineering and Technology, Chennai, India

Received 2013-06-10, Revised 2013-06-12; Accepted 2013-07-04

ABSTRACT

Regression testing is testing the software in order to make sure that the modification made on the program lines does not affect the other parts of the software, it is in maintenance phase and accounts for 80% of the maintenance cost and thus optimizing regression testing is one of the prime motives of software testers. Here we take the advantage of selecting test case information available in regression testing and prioritize them based on the number of modified lines covered by the test case, the test case which covers the most number of modified lines has the highest priority and is executed first and the one with the least coverage of modified lines has the lowest priority and is executed last provided deadline time is not reached, thus even if the testing is not finished we will have covered maximum modified lines, the prioritization of the test cases are done using the genetic algorithm, the genetic algorithm takes test case information from regression testing as input and produces a sequence of test case to be executed such that the maximum number of modified code is covered.

Keywords: Regression Testing, Test Case, Genetic Algorithm, Test Suite

1. INTRODUCTION

Software testing requires resources and consumes 30-50% of the total cost of development. It is impractical to repeatedly test the software by executing a complete set of test cases under resource constraints (Zhong, 2008). Because of these reason researches have considered various methods for reducing the cost of regression testing, this includes test case minimization and regression test selection, test suite minimization techniques lower cost by reducing a test suite to a minimal subset that maintains equivalent coverage of the original test suite with respect to a particular test adequacy criterion, regression test selection method reduces the cost of regression testing by selecting an appropriate subset of the existing test suite based on information about the program, modified version (Jacob and Ravi, 2013a). Test suite minimization methods and Regression test selection, however, can have drawbacks (Smith, 2009). For example, although some empirical evidence indicates that, in certain cases, there is little or no loss in the ability of a minimized test suite to reveal faults in comparison to its non-minimized original other empirical evidence shows that the fault detection capabilities of test suites can be severely compromised by minimization (Sampath, 2008). Because test case prioritization techniques do not themselves discard test cases, they can avoid the drawbacks that can occur when regression test selection and test suite minimization discard test cases (Islam, 2012). Alternatively, in cases where the discarding of test cases is acceptable, test case prioritization can be used in conjunction with regression test selection or test suite minimization techniques to prioritize the test cases in the selected or minimized test suite (Kapfhammer, 2007).

2. RELATED WORK

Huang (2010) has proposed a cost cognizant test case prioritization technique based on the use of historic records and genetic algorithm. They run a controlled experiment to evaluate the proposed technique's

Corresponding Author: T. Prem Jacob, Department of CSE, Sathyabama University, Chennai, India

effectiveness. This technique however does not take care of the test cases similarity. Sabharwal (2011) has proposed a technique for prioritization test case scenarios derived from activity diagram using the concept of basic information flow metric and genetic algorithm. Sabharwal (2011) has generated prioritized test case in static testing using genetic algorithm. They have applied a similar approach as to prioritize test case scenarios derived from source code in static testing. Andrews and Sasikala (2012) has applied genetic algorithm for randomized unit testing to figure out the best suitable test cases. Mohsen FallahRad has applied common genetic and bacteriological algorithm for optimizing testing data in mutation testing.

3. PROBLEM DEFINITION

Prioritization (orderings) of T and f are a function that, applied to any such ordering, yields an award value to that ordering. For simplicity and without loss of generality, the definition assumes that higher award values are preferable to lower ones. For given T, a test suite, PT, the set of permutations of T and f, a function from PT to the real number. Our aim is to find T'∈ PT such that:

$$(\forall T'')\,(T''\in PT')$$
$$(T'' \neq T')\;\; [f(T') \geq f(T'')]$$

To measure the success of a prioritization technique in meeting the goal, we must describe the goal quantitatively. Depending upon the choice of f, the test case prioritization problem may be intractable. It is also possible to integrate test case prioritization with regression test selection or test suite minimization techniques (Jacob and Ravi, 2013b). Alternatively, we might prioritize test cases in terms of their increasing cost-per-coverage of features listed in a requirements specification. We restrict our attention, focusing on general test case prioritization in application to regression testing, independent of regression test selection and test suite minimization (Canessane and Srinivasan, 2013; Andrews and Sasikala, 2012).

4. GENETIC ALORITHM

Genetic Algorithms (GAs) are search methods based on principles of natural selection and genetics. GAs encodes the decision variables of a search problem into finite-length strings of alphabets of certain cardinality. The strings which are candidate solutions to the search problem are referred to as chromosomes, the alphabets are referred to as genes and the values of genes are called alleles (Sabharwal, 2011). Unlike traditional search methods, genetic algorithms rely on a population of candidate solutions. Once the problem is encoded in a chromosomal manner and a fitness measure for discriminating good solutions from bad ones has been chosen, we can start to evolve solutions to the search problem using the following steps.

4.1. Initialization

The initial population of candidate solutions is usually generated randomly across the search space.

4.2. Evaluation

Once the population is initialized the fitness values of the candidate solutions are evaluated.

4.3. Selection

Selection allocates more copies of those solutions with higher fitness values and imposes the survival-of-the-fittest mechanism on the candidate solutions.

4.4. Recombination

Recombination combines parts of two or more parental solutions to create new, possibly better solutions (i.e., offspring).

4.5. Mutation

While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution.

4.6. Replacement

The offspring population created by selection, recombination and mutation replaces the original parental population. Repeat steps from evolution to replacement until a terminating condition is met.

5. PROPOSED METHODOLOGY

Genetic algorithm is stochastic search technique, which is based on the idea of selection of the fittest chromosome. Fitness of the chromosome can be defined by a suitable objective function. Genetic algorithm carry out a multidimensional search by maintaining population of potential user, random methods consisting of a combination of iterative search methods and simple random search methods can find a solution for a given problem. The steps of genetic algorithm are.

Table 1. Test case execution history

Test case ID	A	B	C	Expected output	Execution history
T1	30	20	40	Obtuse angle triangle	8,9,10,11,12,13
T2	30	20	40	Obtuse angle triangle	8,9,10,11,12,13,14,15,16,17
T3	30	20	40	Obtuse angle triangle	10,11,12,13
T4	30	20	40	Obtuse angle triangle	10,11,12,13,14,15,16,20,21,22
T5	30	20	40	Obtuse angle triangle	12,13,14,15,16,20,21,22
T6	30	20	40		22,23,24,25,28
T7	30	20	40	Obtuse angle triangle	5,6,7,8,9,10,11,12,13,14,15,16, 20,21,15, 16,20,21,35
T8	-	-	-		
T9	30	20	40		5,6,7,8,9,10,11,12,13,14,15,16, 20,21,15,16,20,12,35
T10	30	20	40		18,19,20,21,35
T11	30	20	40	Obtuse angle triangle	24,25
T12	30	20	40	Obtuse angle triangle	15,16,20,21

5.1. Generate Population

Initially population is randomly selected and encoded. Each chromosome represents the possible solution of the problem.

5.2. Evaluate the Fitness

Fitness of the chromosome can be defined by the objective function. This objective function generates a real number from the input chromosome. Based on this number two or more chromosome can be compared.

5.3. Apply Selection

In general the selection is depending on the fitness value of the chromosome. The chromosome with higher or lower value will be selected based on the problem definition.

5.4. Apply Crossover and Mutation

Parents are chosen and randomly combined. This technique for generating random chromosome is called crossover.

6. TEST CASE OPTIMIZATION USING GA

Let's say a program has test case suite T, now if one can make modification in the program p, suppose modified program is P', so in order to test program P' one can generate a prioritize sequence of test cases from test case suite T, on the basis of the line of code modified (Binkley and College, 1997).

6.1. Fitness Function

The following fitness function will be used.

Fitness value (F) = Σ {order * (number of modified lines covered by test cases)}.

6.2. Crossover

Here one can use one point cross over with crossover probability Pc = 0.33.

Crossover Probability = Fitness Function of Chromosomes/$\sum$Fitness Function.

6.3. Mutation

Here we will use mutation probability Pm = 0.2. It means that 20% of the genes will be muted within a chromosome. **Table 1** tells us which test case covers which line code. This is helpful later on when we know the number of modified lines, we can compare the number of modified lines with above information and sort out which test case covers most modified lines of code (Sastry, 2007). Assume that lines 5, 8, 10, 15, 20, 23, 28, 35 are modified and the modified lines of code covered by each test case are shown in the **Table 2**. It shows the test cases which does not at all cover modified lines of code though they cover lines. We limit only to prioritize the test cases based on number of modified lines a test case covers are shown in the **Table 3**.

Now we apply genetic algorithm, on this data, generate random number without repetition and put it in the following column, these pattern of random number would represent chromosomes and we would have chromosomes, e1, e2, … and so on and then we find the fitness of each chromosomes, find probability, perform selection and recommend which chromosomes to be taken into the popula-tion. Based on the random number we came to know that the first random number recommends the chromosome1 which is represented as:

Table 2. Test case code coverage

Statement	Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6	Test case 7	Test case 8	Test case 9	Test case 10	Test case 11	Test case 12
5							X		X			
6							X		X			
7							X		X			
8	X	X					X		X			
9	X	X					X		X			
10	X	X	X	X			X		X			
11	X	X	X	X			X		X			
12	X	X	X	X	X		X		X			
13	X	X	X	X	X		X		X			
14		X		X	X		X		X			
15		X		X	X		X		X			X
16		X		X	X							X
17		X										
18										X		
19										X		
20				X	X		X		X	X		X
21				X	X				X	X		X
22				X	X	X						
23						X						
24						X						
25						X					X	
26											X	
27												
28						X						
29												
30												
31												
32												
33												
34												
35							X		X	X		

Table 3. Number of modified lines covered by the test case

Test case	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12
Number of modified lines	2	4	1	3	2	2	5	2	4	1	0	2

Table 4. Using genetic algorithm on the same data

Chromosomes	Fitness value	Normalized value	Cumulative probability	Selection of random numbers	Recommendation
T1->T2-> T3-> T4-> T5->T6->T7-> T8->T9->T10->T11-> T11-> T12	196	196/573 = 0.342	0.342	0.3	Chromosomes e1
T2->T4->T6->T8->T10->T12-> T1->T3->T5-> T7->T9->T11	189	189/573 = 0.329	0.671	0.4	Chromosomes e2
T5->T6->T8->T9->T12->T1-> T7->T11->T2-> T3->T4->T10	188	188/573 = 0.328	1.000	0.2	Chromosomes e1

(T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12)

Because the selected random number lies between 0-0.342. Second random number recommends the chromosome 2 which is represented as:

(T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11)

Because the random number lies between 0.342-0.671. The third random number recommends the chromosome 1 which is represented as:

(T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12)

Because the selected random number lies between 0-0.342. So now we have the following member in our mating pool:

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12
T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11
T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

Now we will apply the one point crossover on these chromosomes and will generate the new off springs:

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12
T2→T4→T6→T8→T10→T12→T1→T3→T5→T7→T9→T11
T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T11→T12

When we apply one point crossover to the selected population then we get these offspring's:

T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12
T2→T4→T6→T8→T10→T12→T1→T9→T11→T3→T5→T7
T1→T2→T3→T4→T5→T6→T7→T9→T11→T8→T10→T12

Suppose if the crossover probability is 0.3 then we select 2 chromosomes from the offspring and one from the parents based on the fitness function value. This process is repeated certain fixed number of iterations, on repeating this procedure multiple times, we will get the nearly optimum solution are shown in the **Table 4**.

7. STEP BY STEP PROCEDURE FOR GENETIC ALGORITHM

7.1. GA Initialization

In this module sample population is initialized. It is generated randomly. Population is a collection of chromosomes. Each chromosome consists of genes in it. Here order is the priority of the test case, if the test case is to be executed first then the order of the test case will be n, where n is the number of test case, NML is number of lines modified. E1, E2,.. are the chromosomes, to generate this random pattern we use rand() present in stdlib of c language, if "K" the random number generated it should satisfy this condition $K \leq N$, the other condition is that the number should not repeat, thus if we calculate the total number of possibilities then one will have to calculate the value of N X (N-1) X (N-2) X (N-3)….1 this value will be very large if N is large, thus genetic algorithm would much optimize the load of find such a possibilities.

7.2. GA Evaluation

Once the population is initialized, the fitness values of the candidate solutions are evaluated. This is where we attempt to identify the most successful members of the population and typically we accomplish this using a fitness function (Guillaumier, 2003):

Order	NML	E1	E2	E3	E4	E5
12	2	5	9	.	.	.
11	4	4	4	.	.	.
10	6	8	2	.	.	.
9	7	9	10	.	.	.
8	6	1	5	.	.	.
7	1	2	11	.	.	.
6	0	10	12	.	.	.

The fitness calculation is done for each chromosome using the following formula:

$$\text{fitness vlaue of each chromosomes} = \sum_{1}^{n} \text{Order} \times \text{NML}$$

Here one can find the order and number of modified lines of each test cases in a test case pattern present in a chromosomes, gives the fitness value of a particular chromosomes. Here for instance if one takes the first chromosome e1, then one has test case 5 scheduled to be executed first, test case 4 comes second thus, for first test case We take the value 5 and index it in the array of matrix, this gives as the order and number of the particular test case in column one and two, we find the product of order and number of modified line test case 5 and it comes out to be 48 as 8×6 then one can proceed with test case 4 it comes out to be 63 and then we add 48+63, this process continues till then end of all the test cases finally we get the fitness of chromosomes e1 and we calculate for e1-e5.

7.3. GA Selection

In the selection process typically we call the fitness function to identify the individuals that we use to create the next generation. We calculate the probability and cumulative probability of the population by the formula:

$$\text{Probablilty of a chromosome} = \frac{\text{Fitness of the particular chromosomes}}{\sum(\text{fitness of all chromosomes})}$$

$$\text{Cumulative probability of ith chromosome} = \sum_{1}^{ith} \text{probability}$$

After finding the cumulative probability, one use roulette wheel technique to find the parents, so that one can perform crossover and mutation operation.

7.4. GA Crossover

Recombination combines parts of two or more parental solutions to create new, possibly better solutions. Consider that the following two chromosomes (e1, e2) were selected to be the fittest amongst the five chromosomes. The execution sequence of these two chromosomes:

E2 T8 T7 T3 T5 T1 T12 T4 T11 T6 T10 T2 T9
E4 T3 T4 T1 T10 T12 T11 T9 T8 T7 T2 T5 T6

In one point cross over one generates the a random number smaller than the number of test cases, then one can take that random number of point of crossover, we calculate the cross over probability:

E2 T8 T7 T3 T5 T1 T12 T4 T11 T7 T2 T5 T6
E4 T3 T4 T1 T10 T12 T11 T9 T8 T6 T10 T2 T9

7.5. GA Mutation

While recombination operates on two or more parental chromosomes, mutation locally but randomly modifies a solution. Considering the below chromosomes where cross over is already performed and suppose the mutation probability is 0.16 then one can generate two random numbers and then brings changes about those structure, if 3 and 8 are then number generated then the above chromosomes becomes. The structure that is at the index 3, index 8 that are swapped as a process of mutation, it is believed to improve the fitness if mutation is done once in certain iteration and not all:

E2	E4
T8	T3
T7	T4
T11	T8
T5	T10
T1	T12
T12	T11
T4	T9
T3	T1
T7	T6
T2	10
T5	T2
T6	T9

8. PSEUDO-CODE FOR GENETIC ALGORITHM

```
Begin
T<-0
Initialise P(t)
while (not termination condition)
    Evaluate P(t)
    Select P(t+1) from P(t)
    Crossover P(t+1)
    Mutate P(t+1)
    t<-t+1
end while
end procedure
```

8.1. Evaluation Operation

Test info is an array that stores all the necessary information of a test case represents the chromosomes. Fitness is variable that stores fitness value of chromosomes. Fitar is an array that stores the fitness value of each chromosome:

```
Fitness<-0
Order starts from number of test cases
for (each number of test case)
TID<-testinfo[i][e]
Fitness<-fitness+ (order*testinfo[TID-1][1])
Order decremented by one
Put the fitness value in fitar;
increment j
end while
```

8.2. Selection Operation

for(number of chromosomes times)

```
calculate the probability for each chromosomes;
sum of probability of each chromosomes;
calculate the cumulative probability of;
end for
for (two parents)
do
until
generate a random number;
check where the number lies in roulette wheel
Convert the generated number i.e., between 0-1
for(the number of chromosomes times)
if(number lies in-between 0 ,cumulative probability)
break;
else if(check where it lies in cumulative probability)
break; end for
Set Check true;
for (j<-0;j<i; increment j)
if (if number is already used)
set check to false
break; //no need to check other elements of crom[]
end while if  check is not true
end for
```

8.3. Crossover Operation

```
for(the number of test case times)
if( until the point of cross over)
new matrix first column = elements of selected chromosomes
end if
else
do
until
initialize the n;
if(n crosses the number of test cases)
then set n to zero
set check true;
for(j from 0 to current index)
if(current chromosomes already in the new matrix)
set check to false
break;
end if
end for
end while if check is not true
new matrix first column = element of selected chromosomes
end else
end for
for(the number of test case times)
if( until the point of cross over)
new matrix second column = elements of selected chromosomes
end if
else
do until
initialize the n;
if(n crosses the number of test cases)
then set n to zero
set check true;
for(j from 0 to current index)
if(current chromosomes already in the new matrix)
set check to false
break;
end if ;end for
end while if check is not true
new matrix second column = element of selected chromosomes
end else
end for
```

8.4. Mutation Operation

```
srand(time(NULL));
generate first random number
do
set check true
generate second random number;
if(the two random numbers are same)
set check to false
break;
end while if  check is not true
Swap the execution order for first child
Swap the execution order for second child
```

9. PERFORMANCE ANALYSIS

For performance analysis we use some random chromosomes it then uses a fitness function and checks how at an average is the fitness of each chromosomes, we observe that in the beginning or otherwise called first generation are shown in the **Table 5**, at an average the fitness value of the chromosomes is very poor, in order to improve the fitness at an average it uses the genetic algorithm, its main postulate being “the survival of the fittest”, this algorithm mimics the nature and produces the best optimum solution. Amongst many operations available in the genetic algorithm cross over and mutation are the two that is implemented, the two produces a fairly good outcome. The output which is produced by the chromosome has the fitness function as in **Table 6**.

If the average fitness value of the chromosomes are found it comes out to be 190.6 fitness values. With above fitness value we search two best parents and perform cross over for fixed amount of times, for instance with five iteration we get the following output. The chromosomes fitness values are shown in **Table 7**.

Now after the implementation of genetic algorithm if we find the average fitness value of the below execution sequence the fitness value comes out to be 205.6. The best execution sequence of the chromosomes are shown in the **Table 8**.

Table 5. First generation

E1	E2	E3	E4	E5
T5	T8	T9	T3	T2
T2	T7	T1	T4	T4
T7	T3	T8	T1	T6
T8	T5	T2	T10	T8
T9	T1	T7	T12	T10
T3	T12	T3	T11	T12
T1	T4	T6	T9	T1
T10	T11	T4	T8	T3
T12	T6	T5	T7	T5
T11	T10	T12	T2	T7
T4	T2	T10	T5	T9
T6	T9	T11	T6	T11

Table 6. Fitness function

Chromosomes	E1	E2	E3	E4	E5
Fitness value	208	178	216	162	189

Table 7. Fitness values

Chromosomes	E1	E2	E3	E4	E5
Fitness value	208	216	202	206	196

Table 8. Final generation

F1	F2	F3	F4	F5
T5	T9	T9	T5	T9
T2	T1	T1	T2	T1
T7	T8	T8	T7	T8
T8	T2	T7	T8	T7
T9	T7	T12	T12	T11
T3	T3	T4	T4	T4
T1	T6	T11	T11	T12
T10	T4	T6	T6	T10
T12	T5	T10	T9	T6
T11	T12	T2	T3	T5
T4	T10	T3	T10	T2
T6	T11	T5	T1	T3

Table 9. Fitness value

Iteration	1	2	3	4	5	6
Average fitness	190.6	201.6	205	205.6	200	205.6

On performing five iterations and finding the fitness value we get the following result are shown in the **Table 9**.

Plotting graph for the above result we get the following curve, which suggest the genetic algorithm does not always guarantee the answer.

10. CONCLUSION

Here the genetic algorithm is applied on the test cases with their execution history. We used a fitness function which gives higher value if a test case covers more line of code and a test case which has higher fitness value is provide higher priority in ordered sequence. When we applied genetic algorithm a large number of time we will get a nearly optimized solution. The input given to the genetic algorithm is a set of chromosomes and the chromosomes are set of test cases with the execution history, below is an instance of chromosome:

T1→T2→T3→T4→T5→T6→T7→T8→T9→T10→T1
1→T12

We consider a random execution sequence generated by random number generator function available in stdlib library (c language) the sequence so generated becomes one chromosomes, we use five chromosomes, generates the fitness of each chromosomes and then the average fitness value is found. In the first generation the average fitness value comes out to be 190.6, we use iteration value five as a fixed terminating condition, after the fifth iteration we find that the average fitness value of the population becomes 205.6 a much better one than the first generation.

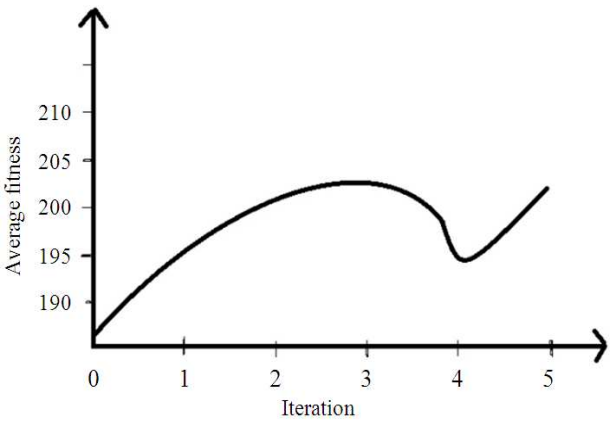

Fig. 1. Fitness plot for each iteration

This means that the final population has a set of chromosomes, whose execution sequence is nearly the best optimum solution are shown in the **Fig. 1**. We considers a random terminating value, we can perform analysis on bench mark problems and derive the terminating criteria by which we can find the least iteration value that will provide guarantee the near optimal solution.

11. REFERENCES

Andrews, J. and T. Sasikala, 2012. An effective orchestration algorithms pertained to benchmark applications. Natl. J. Adv. Comput. Manage., 3: 1-5.

Binkley, D. and L. College, 1997. Semantics guided regression test cost reduction. IEEE Trans. Soft. Eng., 23: 498-516. DOI: 10.1109/32.624306

Canessane, R.A. and S. Srinivasan, 2013. Framework for analyzing the system quality.

Guillaumier, K., 2003. Generic chromosome representation and evaluation for genetic algorithms. University of Malta.

Huang, Y., 2010. Hypergraph based visual categorization and segmentation. The State University of New Jersey.

Islam, M.M., 2012. MOTCP: A tool for the prioritization of test cases based on a sorting genetic algorithm and Latent Semantic Indexing. Proceedings of the 28th IEEE International Conference on Software Maintenance, Sept. 23-28, IEEE Xplore Press, Trento, pp: 654-657. DOI: 10.1109/ICSM.2012.6405346

Jacob, T.P. and T. Ravi, 2013a. Detecting of software source code defects using test case prioritization rules. Proceedings of the 2nd International Conference on Latest Computational Technologies, (CT' 13), London.

Jacob, T.P. and T. Ravi, 2013b. Regression testing: Tabu search technique for code coverage. Ind. J. Comput. Sci. Eng.

Kapfhammer, G.M., 2007. A comprehensive framework for testing database-centric applications. PhD Thesis, Pennsylvania State University.

Sabharwal, S., 2011. A genetic algorithm based approach for prioritization of test case scenarios in static testing. Proceedings of the 2nd International Conference on Computer and Communication Technology, Sept. 15-17, IEEE Xplore Press, Allahabad, pp: 304-309. DOI: 10.1109/ICCCT.2011.6075160

Sampath, S., 2008. Prioritizing user-session-based test cases for web applications testing. Proceedings of the 1st International Conference on Software Testing, Verification and Validation, Apr. 9-11, IEEE Xplore Press, Lillehammer, pp: 141-150. DOI: 10.1109/ICST.2008.42

Sastry, K., 2007. Genetic algorithms and genetic programming for multiscale modeling: Applications in materials science and chemistry and advances in scalability. Ph.D. Thesis, University of Illinois at Urbana-Champaign.

Smith, A.M., 2009. An empirical study of incorporating cost into test suite reduction and prioritization. Proceedings of the 24th Symposium on Applied Computing, Mar. 08-12, Honolulu, HI., pp: 461-467. DOI: 10.1145/1529282.1529382

Zhong, H., 2008. An experimental study of four typical test suite reduction techniques. Inform. Software Technol., 50: 534-546. DOI: 10.1016/j.infsof.2007.06.003