CHAPTER SIX
CONCLUSION

6.1. Introduction

6.2. Conclusion

6.3. Future Work
CHAPTER SIX: CONCLUSION

6.1. Introduction

Testing is the most popular method used for enhancing the quality of software because it is simple, feasible, and provides a high cost/performance ratio. White box testing is needed to increase the probability of detecting various kinds of structural errors. It is obvious that the more we have covered in the input space, the more problems we will find and therefore we will be more confident about the quality of the software. Software testing can be very costly. Automation is a good way to cut down time and cost. Some of the advantages of automation to replace manual software testing and generation of test suites are, reduced testing time, consistent test procedures, reduced quality assurance cost, improved testing productivity and improved product quality.

The growing interest in search based software testing can be attributed to the fact that there is a need for automatic generation of test data, since it is well
known that exhaustive testing is infeasible and the fact that software test data generation is considered NP-problem.

Genetic Algorithms (GA) are computationally simple yet powerful in their search for improvement. GA-based test data generation contributes to the quality improvement as well as the reduction of software development cost.

While the Genetic algorithm has achieved some definite success, it has its limitations. Things are not so simple that in order to solve any optimizational problem, all what anyone needs to do is to represent and evaluate individual solutions. The computation of objective fitness value is non-trivial. It needs to be something a computer can do relatively faster, since thousand- even millions- of individuals need to be evaluated in the process of better and better generations.

Random test data generators make a good baseline for comparison with the other proposed generators because it is easy to implement and commonly used and reported in the literature.

The research work started with the objective planning into mind that the automatic test data generation can be optimized. The concept of using genetic
algorithms to generate test data in order to develop reliable, high quality software is present.

Different test methods and strategies are covered. Various test criteria are explained. The selection of test criterion (path coverage) used for this research work is justified. The use of GA in software testing is presented. The operators and parameters of the GA are thoroughly expounded. Characteristics, strength and limitations of the GA are also included. Further, the literature review is conducted to analyze the related work and identify various algorithms proposed to optimize the automatic test data generation problem.

The role of the objective fitness function and the feedback which it provides to the GA is also explained. Thereafter, a new fitness function for the GA is proposed. The algorithm is a modification of the SIMILARITY presented by [51]. The proposed genetic algorithm is named SMS based algorithm. Description of the proposed SMS based algorithm is presented and then the experiments and results are discussed. The obtained results are compared to random testing and other possible techniques to evaluate the proposed algorithm and to justify the hypothesis of the research work.
6.2. Conclusion

In many areas of software development the ability to achieve test coverage of code is considered as vital. Test data is often generated by hand; therefore, demand for automatic test data generator is high. The main goal of this research is to make a study of the use of GA to automate the evolution of solutions for test data generation. An algorithm for test data generation using GA and a tool implementing the approach for unit-test-level is presented. Path coverage criterion is concerned with (selected) paths and it was aptly adopted in this work since it achieves the utmost code coverage.

The main contribution of this work is the increase path coverage within limited time and computational effort.

The proposed algorithm, based on $SMS$ fitness function, is an enhancement of the SIMILARITY fitness function introduced by Lin-Yeh [51]. In any GA, the fitness function is considered the costliest component of GA. The proposed $SMS$ function quantifies the distance between two given paths. To increase path coverage within limited time and computational efforts, $SMS$ is proposed, where paths being compared are shifted to right and to left for calculating the distance.
The proposed SMS based GA accepts as input an instrumented version of the program to be tested, the list of paths to be covered, and GA parameters, population size pop-size, maximum number of generations max-gen, probabilities of crossover $P_c$ and mutation $P_m$.

### 6.2.1. Algorithm Evaluation Results

Experiments have been carried out to evaluate the effectiveness of the proposed SMS based GA compared to the Lin-Yeh’s SIMILARITY Algorithm. The capability of the proposed SMS based algorithm to compete with the SIMILARITY has been tested, since the SMS based algorithm is basically a modification of SIMILARITY. While experimenting both algorithms (SMS and SIMILARITY) on Tri-Class with small pop-size (100 individuals) and varied maximum number of generations at 10 to 50, SMS could achieve coverage within **10 runs at the most**, while no coverage achieved by SIMILARITY. With larger population of such 500 individuals, when number of generation was 50 the SMS algorithm achieved coverage with minimum and maximum number of generation equals **3 and 18 respectively**, while for SIMILARITY it was **6 and 38 respectively** which is up to **2 times** more generations. In short, the SMS algorithm was **faster** and could achieve full coverage in **only half** of generation of what is needed for SIMILARITY. The experiment also showed that both
algorithms could cover simple paths in the first generations but in equilateral path, SMS based algorithm was **twice as fast as SIMILARITY**.

The experiments also proved that the number of individuals in one generation should be large enough to maintain diversity, but not too large (small enough to avoid excessive number of tests). The proposed SMS based GA performed better, specially, with hard-to-cover paths (as in first experiment with triangle classifier program).

### 6.2.2. Studying Programs’ Complexity Effect

To study the complexity of the program affect on the performance of the proposed algorithm, proposed SMS based algorithm is used on some more experiment, and studied the effectiveness as the average frequency of coverage of the selected goal paths and efficiency as the number of generations took to cover selected goal path of SMS based algorithm on a suite of small C programs that have frequently been used as benchmark for test data generation techniques.

Random Test Data Generation was also applied on the same programs. The number of runs was permitted the same as used by genetic search. In this
way the random test generation was allowed the same computational resources that genetic search used.

The average coverage frequency of the subject programs is calculated. It is observed that the SMS proposed algorithm achieved higher results in all calculations of mean, median, Std-Dev, min and max values. For example, in tri-class program the mean, median and Std-Dev for SMS based algorithm was 11091, 11040 and 253 while for random testing it was 2649, 6258 and 89 respectively.

The results also indicate that the random testing could not distinguish simple paths from the difficult paths since there is no feedback from the program under test to the search algorithm, which results in small values of Std-Dev since it took approximately similar number of generations to cover the selected goal paths, which means, it took random testing algorithm the same number of generations to find solution in all runs with no regards to the fact that some paths are straight codes and easier to reach coverage.

When studying the effectiveness of SMS over Random Testing, it was observed that in most paths of subject programs (i.e. all the paths of Mid and Quad-equ programs, X-axis and Y-axis paths in P-Ly program, and excellent
path in Scale program), there was a considerable difference in the average of coverage frequency between SMS and random testing. It is worth mentioning that the difference between both algorithms performance was insignificant in 4 paths of the subject programs (path Origin of P-ly program fig (5-5) and paths V.good, Good and Pass in the Scale program fig (5-6)). The most difficult path in the subject programs was Equilateral triangle in Tri-Class. The average coverage frequency of the Equilateral triangle path in Tri-class program using SMS was 5.2, while Random testing achieved average of only 0.2 fig (5-7).

The results have indicated that the proposed GA algorithm requires around 30% less iterations and works more effectively than random testing. The poor performance of random generation is attributed to the complexity of the program code and the increased difficulty of path coverage. Although, some results suggests that random test generation might be valuable, our results indicate that this value may decrease considerably for complex programs.

The result of the study of efficiency of the proposed algorithm shows that the generation number at which the goal path was covered over 10 run average, with pop-size 500 individuals for max-gen 20 iterations.
In all programs except Tri-Class, SMS based algorithm achieved 100% path coverage in the initial population and Random test achieved similar results because the code of these programs did not involve nested conditions, they were easily satisfied and most of the code in programs is straight line.

It is noted that random testing failed to cover equilateral triangle path (observed in table (5-11)) where an input of three equal integers is required, whereas SMS based algorithm achieved coverage of that path in average of approximately 10 generations only out of the 500 generations allowed.

The results confirm that random test data generation was outperformed by SMS based GA even on small programs with low complexity. More complex programs are even greater problem for random test data generators. Though random test data generators make a good base line for comparison, because it is easy to implement and commonly used.

6.2.3. Studying GAs’ Parameters

The experiments investigates the effect of GA parameters on the performance of the proposed SMS based algorithm for path testing. The performance of different GA parameters such as crossover strategies, mutation rates and parent selection methods has been taken into consideration to study.
Initially, the effect of selection method is studied. The selection of parents for reproduction was either random or according to fitness (roulette wheel) method. The average of test data generated to cover the goal paths in the program over 10 runs was calculated. The population was 500 individuals for 50 generations/iterations with 1.0 probability of single-point crossover and mutation rate $P_m = 0.05$. The experiments with *Tri-Class* program showed that when the generation of the next population based on roulette wheel, the fitness function determines the probability of selection allowing those with high fitness more chance of offspring in the next generation in comparison of their less fit companions. This means that selection parent for reproduction according to their fitness proved better results than random selection of parents.

When repeating the same comparison using same parameters in the above experiment except that double-point crossover was used to generate offspring instead of single point crossover. In the triangle classifier program the path equilateral triangle was covered 168 times using selection according to fitness while it was covered only 0.9 times using random selection. The obtained results confirm again that the selection of parent according to fitness value is more successful than random method.
Investigating the mutation parameter, experiments on Tri-Class were conducted using two mutation rates. The first is 0.05 which is the reciprocal of the chromosome length \( \frac{1}{24} \) and the second was a smaller value of (0.005) which approximately is the reciprocal of generation population size. Equilateral Triangle has coverage frequency of 8148 using \( P_m=0.005 \) and frequency of 3.8 using \( P_m=0.05 \). The best mutation rate for the Tri-Class program turns to be 0.005, which means that on the average, one mutation every 200\(^{th}\) gene.

In fact, larger rates for \( P_m \) were tested but were found disruptive while much lower rates were less effective, which indicates that mutation rate \( (P_m) \) is better adjusted with program at hand.

After the best mutation rate for Tri- Class found out, one more interesting study is done. Comparison of SMS based GA performance in both single-point crossover and double-point crossover is conducted. The other parameters remain as same as before, i.e. 500 individuals for 50 generations and \( P_m=0.005 \). The average number of data generated to cover paths for the four types of triangle in double-point crossover was larger than the single-point crossover. For example, the Equilateral Triangle had coverage frequency of 168.2 using double-point crossover and 8.3 using single-point crossover. This is
because in double-point crossover there are good chances to double exchange the good genes between two fit parents to generate a yet better offspring. In the case of Tri-Class program, Comparison between the two crossover operators favors double-point crossover. Single-point-crossover may only change one input variable at a time, but double-point crossover may change two input variables as per Fig (3-6), which increase the effectiveness of the search. In general, double-point crossover explores more of the domain than single-point crossover. This gives the indication that the strength of GA techniques exists in crossover operation.

6.2.4. Studying the Search Progress in the Proposed Algorithm

For studying the GA progress from generation to generation, the experiment with Tri-Class and Max-Min programs showed that SMS-based GA performs better from one generation to the next to cover selected paths even with difficult paths like Equilateral triangle in Tri-Class program. It is observed by Fig (5-13.a) that invalid and scalene triangle types were easily covered with large amount of good test data to cover them in the 1st to the 3rd generation (and then the average is approximately 400 test cases in each generation) while it took SMS about 10 generation to improve and give an adequate number of good test data to cover Isosceles triangle path which is approximately 150 test cases in each generation. The most difficult path to be covered in the program
is the equilateral triangle. The SMS based algorithm shows improvement after about 30 generations/iterations since it is very difficult to generate test cases with 3 identical numbers to cover the path. And for Random testing the progress is not clear in the case of invalid and scalene triangles paths despite, they are comparatively easy to cover, while the progress is poor and there is almost no progress in generating test data for the difficult paths such as isosceles and equilateral triangles (observed by fig (5-13.b).

In both programs Max-Min and Tri-Class, the average number of good test data generated is less than the average generated using SMS-based GA, in all the selected goal paths which is near the half.

The comparison with random testing showed that when generation of the next population is based on selecting those individuals with high fitness, this gives the chance to generate a better offspring than of random testing which had a poor or no progress in the generation of test data and generated much less number of good test data.

6.2.5. Studying GA Performance with Data Types

The study, also, investigated the capability of the proposed GA to test programs with different data types (integers, floats and characters) and complex data structure (arrays and, loops).
The program used was the \((\text{Max-Min})\). The results showed the comparison of using array of integer numbers and then when the array is of floating numbers. The results showed that GA with \(\text{SMS}\) as objective fitness function \textbf{performs well with both integer and float numbers}. In other words, the GA performance with data in floating number format is as good as the performance with data in integer format whether the number was in single type or array type. Results were approximately similar with different array sizes (e.g. 5, 10, 15, etc.). The population size was 500 individual with 10 generation and \(P_c=1.0\) single-point crossover, \(P_m=0.05\), over 10 runs.

To allow more confidence in the approach and results, the obtained results were compared with random testing using similar parameters. The \(\text{SMS}\) based GA algorithm has outperformed the random generation. The average coverage frequency is higher in GA, even when repeating experiments with \textbf{different array sizes} (e.g. 5, 10, 15, etc.).

The results indicated that different data types have \textbf{no influence} in the performance of \(\text{SMS}\) as long as testing tool knew what kind of data structure and variables were used. The investigation confirms that GA with \(\text{SMS}\) as fitness function \textbf{has no problem} in generating test data to achieve full coverage for input of integers, \textbf{floats or characters} forms.
6.2.6. Loop Testing

Finally, the effectiveness of GA test data generation using \((SMS)\) as a fitness function in loop testing was investigated. Test harness was developed to apply a loop testing criterion which was to iterate a loop zero, one, two, more than two times. The \(SMS\) based algorithm was experimented on the liner search program \((L-Search)\).

The best results achieved with only 60 population size, and only 100 iterations with \(P_m=0.05\) and \(P_c=1.0\) single-point crossover. The proposed algorithm achieved \textbf{100\% success} rate in all loop (zero, 1, 2, more than 2) iterations.

The average tests for each loop are also obtained, \textit{e.g.} 13.8 coverage frequencies for zero loops. And for testing the same programs randomly, the same number of population size and iterations were allowed. Although the average tests generated by random testing are \textbf{higher than} \(SMS\), \(SMS\) achieved \textbf{100\% success rate} in all runs where random test achieved \textbf{50\% to 90\% only}.

The results indicated that \(SMS\) based algorithm is \textbf{more efficient} in loop testing than random. For example, \textbf{(8.3 generations)} are required on average to
obtain full coverage in the case of zero-loop, and (7.4) in the case of 1-loop iteration while in random testing it was (15.1) and (11.9) respectively.

The same experiment is conducted using array of character elements instead of integers. Results showed that SMS based algorithm is more efficient in loop testing than random testing. The average number of generations needed to get full path coverage in random testing is higher than SMS (e.g. 4.8 for SMS and 19.7 for Random). In general, the SMS based algorithm have no problem generating test data to achieve full path coverage for loop testing with input of integer or character form.

The results confirm that the proposed algorithm is capable of generating and optimizing test data in order to traverse and execute a predefined number of loop iterations. Overall results is that the GA have no problem generating test data to achieve path coverage for loop testing, and above all that, the proposed technique does not introduce any additional calculation overhead.

It is concluded that the good performance of SMS based GA depends on a fitness value that provides the necessary feedback for the GA to generate better test data from one generation to the next.
6.3. Future Work

The proposed work in the thesis provides an opportunity to pave the way for number of possibilities of researches in various directions. The aspirant researcher can undertake investigative approach in order to launch into in areas of similar studies in the future. The researcher is profoundly enthusiastic about continuing research should be carried out with vigorous enthusiasm and devoted interests.

At the stage of test generation, where fitness value comes to zero (new similarity were not detected) the uniform cross over (see section 3.2.3.1) can be used instead of single-point crossover. It is known that uniform crossover is highly disruptive. It mixes bits in test data. In other word, it encourages exploration.

Trials for the full automatic instrumentation and paths extraction of the programs under test (PUT) should be started. Though this has proved to be difficult and time demanding.

The proposed approach is scalable to larger programs. Larger programs are likely to involve larger number of paths. It is also intended to increase the
size of the problem (programs with more paths, and also including loops) and other fitness functions would be expected.

Object oriented programs (OOP) and programs with procedures and subroutines are the next targets to be considered.

The field of GA is developing, and new advanced techniques like self adoption concept, niching theory, penalties, multi-parent crossover… etc, are rapidly rising. More work can be done to evaluate the effectiveness of these techniques in the path testing domain.

Path test generation technique used in this research work can be merged with other techniques, such as data flow testing [59] to improve the test process of programs.

Finally, another possibility is to compare the algorithm with other exhaustive, search techniques to see if co-operation among different search algorithms is feasible.