# Use of Evolutionary Techniques for Symbolic Execution Based Testing

Anjali Kapoor[#1], Mohit kumar[*2]

[#1]Research Scholar, CSE Deptt, RIMT-IET, India

[#2]Associate Professor, CSE Deptt, RIMT-IET, India

***Abstract*— - Evolutionary methods when used as a test data generator optimize the given input (usually called test case) according to a selected test coverage criterion encoded as a fitness function. Basically, the genetic algorithms and other Evolutionary techniques are based on pure random search. However, these algorithms adapt to the given problem. In the last decade lot of evolution based metaheuristic techniques are applied for searching software errors. This survey paper presents the work applying computational evolutionary methods in structural software testing based test data generation.**

***Keywords*— Evolutionary Techniques, Symbolic testing.**

## I. INTRODUCTION

Soft computing based techniques such as evolutionary algorithms, fuzzy modeling, neural networks and swarm intelligence based algorithms have found several applications in software engineering domain. Some of these applications are cost/effort estimation for better resource utilization and allocation in software development [6, 42], software evolution for hassle free design [3], automated software testing for cost cutting and making software more reliable [29], module clustering for effective maintenance [31] and software project management activities such as resource scheduling [2] and selection of optimum development teams [49]. Considerable effort has been put in by various researchers to apply soft computing in the area of software engineering [18, 26, 35]. One of the most intensively applied areas of soft computing in software engineering is test data generation, which is categorized as NP-hard [52] or NP-complete [25] problem due to requirement of enormous efforts in finding the data out of large search space satisfying the complex and non-linear search objectives.

Testing is both technically and economically vital for high quality software production. Nearly 50% of the expenses in software development has been estimated to be spent on testing. Much of the testing is done manually or using other labour intensive methods. One of the most costly activities in software testing, where lot of scope of automation, is test data generation. To develop efficient, cost effective, and automatic test data for software testing remains a major goal in testing. Structural-oriented test methods which define test cases on the basis of internal program structure are widely used. Evolutionary testing is a promising approach for automation of structural test case design, which search test data that fulfill given structural test criteria. Several researchers have used different techniques for automated test data generation.

In the last decade lot of evolution based metaheuristic techniques are applied for searching software errors. In this paper, we review the work applying computational evolutionary methods in structural software testing. Software testing is most cumbersome and costly but an unavoidable activity in software development process [5]. Although several researchers have underlined the importance of non-execution based software testing [16, 23, 38] but program execution based testing is still the most desirable and reliable method for identifying bugs and errors in software [21]. In order to perform the execution based testing, generation of test cases is of prime importance, as the testing success depends upon the efficacy and completeness of the test cases. Incomplete and poor test cases may not be able to detect hidden faults in software; consequently providing a false notion of security and reliability of software leading to loss of millions of dollars.

Although manual generation of test cases is relatively easy but is a slow and costly process. Automatic generation of test cases can save time and resources. At the same time, it is also free from human biases and doesn't require special team of testers other than the developers. Despite having so many benefits, automated test case generation is not so easy because it requires intelligence of human mind to identify the non-linearity and discreteness in test inputs' search space. For improving the quality of automation and fulfilling the requirements of test case generation, many researchers have explored new soft computing based techniques such as genetic algorithm, simulated annealing, tabu search, ant colony optimization, particle swarm optimization, memetic algorithms etc. to fulfill testing requirement and to generate suitable test cases automatically [14, 28].

## II. Software Testing:

Testing is a process in which software is executed with an objective to find out bugs and errors ]. Secondary objective of testing is to establish confidence of user, developer and customer that software is error free. Testing can also be defined as a process of executing a program with the intent of finding errors. Testing process comprises following activities:

1. First, inputs (test cases) are selected.

2. Second, software under test is executed using these inputs for fault(s) identification.
3. Third step is of isolation and resolution of these fault(s).

When a software product is treated as a black box for generation of test cases, it is called black box testing. Its internal structure is irrelevant. This technique of testing is also termed testing to specifications, data-driven, functional or input/output-driven testing. In a realistic application there will be dozens of factors generating huge number of different test cases making technique computationally infeasible. Such combinatorial explosions make black box testing infeasible for automatic testing without using some heuristic. Another fault-based testing technique which is less used is introduced by DeMillo et al. [11] is called mutation analysis. This is based on the assumption that a program will be well tested if all simple faults are detected and removed. Simple faults are introduced into the program by mutation operators. Each change or mutation created by a mutation operator is encoded in a mutant program. A mutant is killed by a test case that causes it to produce incorrect output. A test case that kills a mutant is considered to be effective at finding faults in the program, and the mutants it kills are not executed against later test cases. The goal of mutation is to find test cases that kill all non-equivalent mutants; a test set that does so is adequate relative to mutation.

The other extreme is to focus on the internal structure of a software product to select test cases. Other names for this strategy are glass box, logic-driven or path-oriented testing. The most common form of white box testing requires that each possible path through the code is executed at least once. Structural testing is infested by several problems such as inability of identification of infeasible paths, inability of handling of internal variables, loops and arrays where index depends on input variables and dynamic data structure such as pointers, lists etc. So, there is a strong need of a search algorithm which besides having good search capability should be able to some of these testing problems. Testing is execution of software with the intent of finding errors [32]. It requires searching the inputs' domains for such values which can invoke different output(s) or execute different component(s) making it necessary for employing an efficient search algorithm for test data generation for fulfilling testing objectives (criteria). Structural testing is infested by several problems such as inability of identification of infeasible paths, inability of handling of internal variables, loops and arrays where index depends on input variables and dynamic data structure such as pointers, lists etc. So, there is a strong need of a search algorithm which besides having good search capability should be able to some of these testing problems.

## III. Path Based Software Test Data generation

For software testing purpose, as solution lies in searching inputs, every possible set of inputs represent the global population in search algorithm and selected inputs from this global set are represented by individuals in the population. Suitability of the individuals can be assessed by following a testing criterion for which a unique fitness function has to be defined. In structural testing, these criteria can be anything from all-statement-execution to all-path-coverage [15].

The path testing method involves generation of test data for a target feasible path in such a way that on executing program, it covers all branches on that path. To cover a particular branch, the condition(s) at branch node must be satisfied by the test data, which directs the control flow of program to the next branch of the path. A path may contain several branches and in order to execute that path, all these branch-conditions must be evaluated true by the test data. Consequently, problem of path testing can be formulated simply as constraint satisfaction problem which should be analyzed and solved with the help of some search method by generating inputs in such a way that can satisfy all the branch constraints on the path. A valid test case is generated, which should execute the particular path by satisfying all of the boolean expressions included in that path.

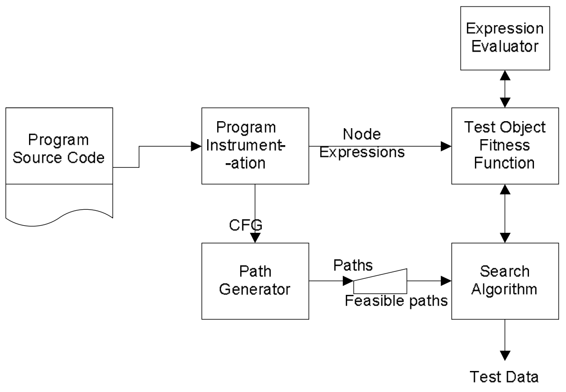

Figure 1. An Overview of a Test Data Generator[9]

Figure 1 shows the different building blocks of a path based automatic test data generator. First test object source code is fed to program instrumentation for CFG and node expressions generation. Subsequently CFG is used to generate all possible paths which are filtered manually for feasible path in order to become input to search algorithm. Node expressions include branch node predicates as well as non-branch node statements which are used to evaluate candidate solutions in test object fitness functions.

In [57], P.R. Srivastava and Tai have presented a method for optimizing software testing efficiency by identifying the most critical path clusters in a program. The SUT is converted into a CFG. Weights are assigned to the edges of the CFG by applying 80-20 rule. 80 percentage of weight of incoming credit is given to loops and branches and the remaining 20 percentage of incoming credit is given to the edges in sequential path. The summation of weights along the edges comprising a path determines criticality of path. Higher the summation more critical is path and therefore must be tested before other paths. In this way by identifying most critical paths that must be tested first, testing efficiency is increased.

Another test generation approach proposed by P.R Srivastava is based on path coverage testing [56]. The test data is generated for Resource Request algorithm using Ant Colony Optimization algorithm (ACO) and GA. Resource request algorithm is deadlock avoidance algorithm used for resource allocation by operating system to the processes in execution cycle [58]. The ACO algorithm is inspired from behaviour of real ants where ants find closest possible route to a food source or destination. The ants generate chemical substance called pheromones which helps ants to follow the path. The pheromone content increases as more ants follow the trail. The possible paths of CFG are generated having maximum number of nodes. Using ACO, optimized path ensuring safety sequence in resource request algorithm is generated covering all edges of CFG. Using GA, suitable test data set is generated which covers the need for each process. The backbone of genetic process is the fitness function which counts number of times a particular data enters and continues the resource request algorithm. Higher the value of count, higher is chances of avoiding a deadlock. The test data with higher values of count is taken and genetic crossover and mutation is applied to yield better results. Simultaneously, poor test data is removed each time.

## IV. AUTOMATIC GENERATION OF TEST CASES

The success of a test data generation method largely depends upon the efficiency of its search technique. There are several classifications of path based testing methodologies. Some follow traditional methods while others use heuristic for test data generation. Broadly speaking test data generation techniques can be divided into three categories; random based, numerical methods and metaheuristic methods.

### A. Random Testing Based Technique

Random testing based search algorithms in which random values are generated from domains of inputs and program is executed using these values. If these inputs are able to satisfy the testing criterion then they form a test case. The random testing is being reported satisfactory by Duran and Ntafos [13], but Myers [32] viewed random testing as a worst case of program testing.. In recent times Mayer and Schneckenburger [27] empirically investigated different flavours of adaptive random testing and concluded that distance based random testing and restricted random testing are the best methods for this class of testing techniques. Although, in general, random testing is a simple, cost-effective and unbiased technique, but researchers do not find it suitable for large and complex programs, especially where input domain size is large. Demillo *et al* [11] also used random testing for identifying seeded faults in programs.

### B. Algorithmic Techniques

Another class of search methods used in testing is algorithmic such as numerical maximization techniques which were used by Miller and Spooner to optimize the test data [30], alternate variable method which was employed by Korel for its dynamic test data generator TESTGEN [22]. In TESTGEN author has used alternate variable method which works in two phases First, an input variable is selected and its value is changed in small steps just to find out the direction in which variable minimizes the branch function. This is called exploratory search. Once the direction of search is known then pattern search is taken in large steps to find the value of the variable in consideration for satisfying or minimizing the branch function. If selected value of the variable fails to decrease the branch function then steps of the pattern search are decreased successively before exploring other variables for minimization purpose.

Another method in this class is domain reduction procedure proposed and used by Demillo *et al* for its fault based test data generator named as GODZILA [10]. It involves successively splitting of the domains of input variables which helps in reducing the search space (input domain) for solving a constraint system. Considering one predicate at a time on target path in control flow graph (CFG), it reduces the domains of input variables step by step until all the constraints on complete test path is satisfied. The backtracking approach is used when a constraint system becomes infeasible due to poor choice of division points in domains for reduction at some of the previous steps. Although many novel techniques such as identification of undesirable variables, finding optimum order of consideration of input variables, use of binary search algorithm and expression handling technique were used for improving the performance of algorithmic search methods some but a plenty of manual and time consuming analysis is required for making these algorithmic employable. Moreover these search methods are slow and ineffective. These also lack global search capabilities, which is a necessary requirement for software testing where objective-functions are very complex.

### C. Metaheuristic Techniques

The most successful search algorithm class for test data generation is based on optimization techniques such as genetic algorithm (GA), simulated annealing (SA), tabu search, ant colony optimization (ACO), particle swarm optimization (PSO), memetic algorithm. A detail and up to date survey on use of these techniques for software testing is given in McMinn [28] .

To apply search techniques for generation of test data first we need to transform, the testing objective into search problem. For this, a mechanism should be evolved to encode the testing problem in search algorithm and next should be to assess the suitability of solutions produced by search technique to guide the individuals for exploring search space i.e fitness function need to be defined.

A popular class of search algorithms is population based iterative techniques where the term population denotes group of individuals as fish schools, birds' flocks and insect colonies like bee, GA population and ant colonies. These individual competes for better efficiency by adjusting their attribute values. Generally due to huge size of search space, practical considerations limit the size of population in search algorithms. Each member of population is called an individual or a probable solution which is evaluated for its fitness so that new and better individual(s) may be generated. This process is iterated in algorithm till the search stopping criterion is met. General algorithm for population based search algorithm is represented in figure 2.

1. Generate initial population comprising multiple individuals
2. Evaluate each individual following a criterion specific to problem
3. Generating next population by using current population based on their fitness.
4. Go to step 2 until stopping criterion is met

**Figure 2: Population based search algorithm**

For software testing purpose, a solution lies in searching inputs. Every possible set of inputs represent the global population in search algorithm and selected inputs from this global set are represented by individuals in the population. Suitability of the individuals can be assessed by following a testing criterion for which a unique fitness function has to be defined.

GA first time used by Xanthakis [51] is he most popular and intensively pursued techniques for software testing. Pargas *et al* [34], Wagener *et al* [48], Watkins [47], Ropar [39], Lin and Yeh [24], Michael *et al.*[29] are some of the pioneers researchers whose works are reported frequently in the testing literature. Pargas proposed a GA based testing technique where number of executed control dependent nodes of the target node decides the fitness of solutions. Michael *et al* proposed GADGET (Genetic Algorithm Data Generation Tool) to generate test cases for large C and C++ programs by using condition decision coverage metrics.

Watkins and Ropar used coverage based criteria for assessing the fitness of individuals in their GA based test generator. Wagener *et al* logarithmized the objective function to provide better guidance for its GA based test case generator. Lin and Yeh used hamming distance based metric in objective function of their GA program to identify the similarity and distance between actual path and already selected target path in dynamic testing.

Another metaheuristic method used for test case generation is SA in which process of cooling of a material simulates the change in energy level with time or iterations Tracey [44, 45]. Authors in [44,45] used a hybrid objective function which includes both concepts; branch distance and number of executed control-dependent-nodes based on the analogy of cooling process to construct test cases for safety critical system.. The steady state in energy symbolizes the convergence of solution.

Another interesting technique used by Diaz *et al* [12] is based on tabu search which maintains a search list also called as tabu list. The Tabu search uses neighbourhood information and backtracking for solving local optima. Two cost functions are defined to intensify and diversify the search mechanism. These cost functions are similar to the functions used by Wegner *et al* in which individuals are penalised for taking wrong path while executing the program. Penalty is fixed on the basis of error value produced by an individual in the effort of satisfying a branch constraint.

Another important technique of soft computing is PSO, which has been applied successfully to a wide variety of search and optimization problems [8, 20]. Windisch *et al* [50] have reported the application of this swarm intelligence based technique for test data generation for dynamic testing. They have conducted experiments to prove the usefulness and utility of search algorithm towards test case generation. Authors in [9] proposed another PSO based algorithm for automatic test case generation activity using symbolic testing. The approach has been validated and compared with GA on very small problems and was found to be a promising alternate to the test case generation.

Another recent search algorithm in swarm intelligence category is artificial bee colony (ABC) algorithm which simulates the honey bees' working toward food foraging and nectar gathering system optimization. The technique has been successfully employed on scores of engineering applications such as internet server allocation[33] pattern recognition [36] , job scheduling [7], data clustering [35] etc. In [54] authors have used this algorithm to find out test cases. Results are said to be encouraging.

Huaizhong and Lam [19] proposed an ACO approach to automate test data generation for state based software testing. Ayari *et al* [4] proposed an evolutionary approach based on ACO to reduce the cost of test data generation in the context of mutation testing. This ACO based approach is enhanced by a probability density estimation technique in order to better guide the search for continuous input parameters.

Another recent search algorithm in soft computing category is Big Bang Big Crunch (BBBC) algorithm, which simulates the energy stabilization in the universe. Singh and kumar [55] presents a Big Bang Big Crunch concept based search algorithm for automatic generation of structural software tests. Test cases are symbolically generated by measuring fitness of individuals with the help of branch distance based objective function. Evaluation of the test generator was performed using ten real world programs. Some of these programs had large ranges for input variables. Results show that the new technique is a reasonable alternative for test data generation, but doesn't perform very well for large inputs and where constraints are having many equality constraints.

In [54] authors evaluated and compared different metaheuristic algorithms namely Genetic Algorithm (GA), Artificial Bee Algorithm (ABC), Particle Swarm Optimization (PSO) and Big Bang Big Crunch (BBBC) algorithms to identify suitable testing techniques [2, 3, 5, 11, 13]. In this work authors also identify key factors, which are responsible for software testing efforts by making a correlation between the characteristics of programs and their respective efforts made in test data generation.

## II. Conclusions

Genetic algorithm based optimization methods in the software testing area have reported to provide good performance which is vetted by a large number of research activities in the testing field using GA. Other methods such as PSO, SA have also been said to give better performance than GA by some of the researchers. Other optimization algorithm have also been used but only for namesake and investigative purpose. Despite positive and encouraging results, nearly all the techniques have used small and trivial programs as experimental objects. This makes scalability and applicability of these techniques for larger programs a real research concern in software testing.

.

## References

[1] Ahmed MA, Hermadi I. GA-based multiple paths test data generator. *Computers and Operations Research* (2007),(article in press)

[2] Alba E and Chicano F, Software Project Management with GAs, Information Sciences, 177(11) 2007 pp.2380-2401

[3] Amoui M, Mirarab S, Ansari A and Lucas C, A Genetic Algorithm Approach to Design Evolution using Design Pattern Transformation, International Journal of Information Technology and Intelligent Computing 1(2, 2006 pp. 235-244.

[4] Ayari K, Bouktif S and Antoniol G, Automatic Mutation Test Input Data Generation via Ant Colony, *GECCO'07*, July 7–11, 2007, London, England, United Kingdom.

[5] Beizer B. *Software testing techniques*. 2nd ed., Dreamtech publication New Delhi. 1990.

[6] Burgess CJ and Lefley M, Can Genetic Programming Improve Software Effort Estimation? A Comparative Evaluation, Information & Software Technology, 43(14), 2001, pp. 863-873

[7] Chong C.S., Low M.Y.H., Sivakumar A.I., Gay K.L., A Bee Colony Optimization Algorithm to Job Shop Scheduling, *Proceedings of the 37th Winter Simulation*, Monterey, California, 1954-1961,2006.

[8] Clow B, and White T, An evolutionary race: A comparison of genetic algorithms and particle swarm optimization for training neural networks. *In Proceedings of the International Conference on Artificial Intelligence, IC-AI '04*, Volume 2, pages 582–588. CSREA Press, 2004.

[9] Dahiya SS, Chhabra JK and Kumar S, Application of Particle Swarm Optimization Algorithm to Symbolic Software Testing, *IISN 2010*, To be held in ISTK, Kalawad on 24-27 February 2010. (Communicated for publication)

[10] Demillo R. A., and Offutt A. J., Constraint-based automatic test data generation, *IEEE transaction on Software engineering*. Vol.17, No.9, September, 1991 pp. 900-910

[11] DeMillo, R.A., Lipton R.J., and Sayward F.G., "Hints on Test Data Selection: Help for the Practicing Programmer," *IEEE Computer*, Vol. II, No. 4, pp. 34-41, 1978.

[12] Díaz E, Javier T, Raquel B, José JD. A tabu search algorithm for structural software testing. *Computers and Operations Research* (2007), doi:10.1016/j.cor.2007.01.009

[13] Duran JW, Ntafos AS Report On Random Testing, *international Conference on Software engineering Proceedings of the 5th international conference on Software engineering* 1981, San Diego, California, United States *March 09 - 12, 1981*

[14] Edvardsson J. A survey on automatic test data generation In *Proceedings of the second conference on computer science and engineering*, Linkoping: ESCEL; October 1999; 21–28.

[15] Frankl PG, Weyuker EJ. An Applicable Family of Data Flow Testing Criteria. *IEEE Transaction On Software Engineering*. 1988; 14(10):1483-1498.

[16] Gilb T, Graham D. *Software Inspection*. Addison-Wesley 1993

[17] Goldberg DE. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[18] Harman M and Jones BF, Search-based Software Engineering, *Information & Software Technology*, 43(14) 2001, pp. 833-839

[19] Huaizhong LI, LAM Peng C. An Ant Colony Optimization Approach to Test Sequence Generation for State based Software Testing, *Proceedings of the Fifth International IEEE Conference on Quality Software (QSIC'05)* 2005.

[20] Jones K. O. Comparison of genetic algorithm and particle swarm optimization. *In Proceedings of the International Conference on Computer Systems and Technologies*, 2005.

[21] Jorgenson P. *Software Testing: A Craftman's Approach*, 2nd edition CRC Press, Inc. Boca Raton, FL, USA, 2001.

[22] Korel B. Automated software test data generation. *IEEE transaction on software engineering*, 1990; 16(8):870-879.

[23] Laitenberger, O. and DeBaud, J. An encompassing life cycle-centric survey of software inspection. *J. Syst. Soft. 50* (2000), 5–31.

[24] Lin JC,Yeh PL. Automatic test data generation for path testing using GAs. *Information Sciences* 2001; 131:47–64.

[25] Mansour N, Salame M. Data generation for path testing. *Software Quality Journal* 2004; 12:121–136.

[26] Mantere T and Alander JT, Evolutionary Software Engineering, A Review, *Applied Soft Computing*, 5(3) 2005, pp. 315-331

[27] Mayer J, Schneckenburger C, An Empirical Analysis and Comparison of Random Testing Techniques, *ISESE'06*, September 21–22, 2006, Rio de Janeiro, Brazil pp. 105-114.

[28] McMinn P. Search-based Software Test Data Generation: A Survey. *Software Testing, Verification and Reliability* June 2004; 14(2):105-156.

[29] Michael C, McGraw G, Schatz M. Generating software test data by evolution. *IEEE Transactions on Software Engineering* 2001; 27(12):1085–1110.

[30] Miller W, Spooner D. Automatic generation of floating-point test data. *IEEE Transactions on Software Engineering* 1976; 2(3):223-226.

[31] Mitchell BS and Mancoridis S, On the Automatic Modularization of Software Systems using the Bunch Tool, IEEE Transactions on Software Engineering, 32(3), 2006, pp. 193-208

[32] Myers GJ. *The art of software testing*. New York: Wiley; 1979

[33] Nakrani S., Tovey C, On Honey Bees and Dynamic Allocation in an Internet Server Colony, *Proceedings of 2nd International Workshop on the Mathematics and Algorithms of Social Insects*, Atlanta, Georgia, USA, 2003.

[34] Pargas RP, Harrold MJ, Peck R. Test-data generation using genetic algorithms. Journal *of Software Testing, Verification and Reliability* 1999; 9(4):263–82.

[35] Pedrycz W, Computational Intelligence as an Emerging Paradigm of Software Engineering, *Proceedings of the 14th International ACM Conference on Software Engineering and Knowledge Engineering (SEKE '02)*, 2002, pp. 7-14

[36] Pham D.T., Otri S., Afify A., Mahmuddin M., and Al-Jabbouli H. Data clustering using the Bees Algorithm, in 40th CIRP International Seminar on ManufacturingSystems. 2007: Liverpool.

[37] Pham D.T., Otri S., Ghanbarzadeh A., Kog E., Application of the Bees Algorithm to the Training of Learning Vector Quantisation Networks for Control Chart Pattern Recognition, *ICTTA'06 Information and Communication Technologies*, 1624-1629, 2006b.

[38] Porter A, Sey A, Votta L. A review of software inspections. *Technical Report: CS-TR-3552*, University of Maryland at College Park College Park, MD, USA, 1995

[39] Roper M. Computer aided software testing using genetic algorithms. In 10th *International Software Quality Week*, San Francisco, USA, 1997.

[40] Schmickl T., Thenius R., Crailsheim K., Simulating Swarm Intelligence in Honey Bees: Foraging in Differently Fluctuating Environments, *GECCO'05*, Washington, DC, USA, 273-274,2005.

[41] Seeley T.D., *The Wisdom of the Hive*, Harvard University Press, Cambridge, MA, 1995.

[42] Shukla KK, Neuro-Genetic Prediction of Software Development Effort, Information and Software Technology, 42(10) 2000, pp. 701-713

[43] Thayer, R.A., M. Lipow, and E.C. Nelson, *Software Reliability*, North-Holland, Amsterdam, 1978.

[44] Tracey N, Clark J, Mander K, and McDermid J. An automated framework for structural test-data generation. In *Proceedings of the International Conference on Automated Software Engineering*, 1998; 285-288.

[45] Tracey N. A Search-Based Automated Test-Data Generation Framework for Safety Critical Software. *PhD thesis*, University of York, 2000.

[46] Watkins A, Hufnagel E. M. Evolutionary test data generation: a comparison of fitness functions. *Software Practice & Experience* 2006; **36**:95–116

[47] Watkins AL. The automatic generation of test data using genetic algorithms. In *The fourth software quality conference* 1995; 2:300–309.

[48] Wegener J, Baresel A, Sthamer H., "Evolutionary test environment for automatic structural testing". Information and Software Technology 43, 841–54, 2001;

[49] Wen F and Lin C, Multistage Human Resource Allocation for Software Development by Multi-objective Genetic Algorithm, The Open Applied Mathematics Journal, 2, 2008, pp. 95-103

[50] Windisch A, Wappler S and Wegener J, Applying Particle Swarm Optimization to Software Testing, *Proceedings of the 2007 conference on Genetic and evolutionary computation GECCO'07*, July 7–11, 2007, London, England, United Kingdom.

[51] Xanthakis S, Ellis C, Skourlas C, Gall AL, Katsikas S, Karapoulios K. Application of genetic algorithms to software testing. In *The fifth international conference on software engineering* 1992; 625–36.

[52] Yuan Z. A Search-Based Framework for Automatic Test-Set Generation for MATLAB/Simulink Models. *PhD Thesis*, University of York Department of Computer Science, December 2005.

[53] Surender Singh and Parveen Kumar "Empirical Evaluation of Metaheuristic Approaches for Symbolic Execution based Automated Test Generation" *International Journal of Information Technology and Knowledge Management (ISSN: 0973-4414) July-December 2012, Volume 5, No. 2, pp. 489-493* (Impact Factor 0.47)

[54] Surender Singh Dahiya, Jitender kumar Chhabra and Shakti Kumar "Application of Artificial Bee Colony Algorithm to Software Testing", *In the proceeding of 21st Australian Software Engineering Conference*, Auckland, New Zealand April 2010.

[55] Surender Singh and Parveen Kumar "Application of Big Bang Big Crunch Algorithm to Software Testing" *International Journal of Computer Science and Communication Vol. 3, No. 1, January-June 2012, pp. 259-262*(Impact Factor 0.48)

[56] Praveen Ranjan Srivastava et. al., "Generation of test data using Meta heuristic approach" IEEE, 2008, pp.19 - 21.

[57] Praveen Ranjan Srivastava and Tai-hoon Kim, "Application of genetic algorithm in software testing" International Journal of software Engineering and its Applications, 3(4), 2009, pp.87 – 96.

[58] Jose Carlos et. al., "A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object oriented software", AST' 08. ACM, 2008,