Vol.5, No.1 ISSN:2349-0012

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

Hybrid Ant colony and genetic based optimistic software testing

Sri sai college of engineering and technology badhani,pathankot

Ankita,sukhbeer singh

Abstract: The review has show that the finding the best automation software testing is still an open area of research. Software testing involves much time for its testing, thus is expensive task. The comprehensive review has shown that the role of meta-heuristic techniques during software testing is ignored in most of existing research. The ant colony optimization technique suffers from poor convergence speed. The hybridization of ant colony optimization with genetic algorithm is also ignored. Therefore using the hybrid ant colony technique with genetic algorithm will be proposed in this work. The use of the proposed technique has ability to predict weather software can be used in real time or not.

Keywords: Software Testing, Software Testing techniques, genetic Algorithm

1. Introduction

Testing software in an automated way has been a goal for researchers and industry during the last few decades. Still, success has not been readily available. Some tools that are partly automated have evolved, while at the same time the methodologies for testing software has improved, thus leading to an overall improvement. However, much software testing is still performed manually and thus prone to errors, inefficient and costly.

In the past few decades the growing influences of software in all areas of industry have led to an ever-increasing demand for complex and reliable software. According to a study conducted by the National Institute of Standard & Technology, approximately 80% of the development cost is spent on identifying and correcting defects .The same study found that software bugs cost the United States economy around $59.5 billion a year, with one third of this value being attributed to the poor software testing infrastructure. However, the automation of test data generation is still a topic under research. Recently, a number of methods such as metaheuristic search, random test generation and static analysis have been used to completely automate the testing process, but the application of these tools to real software is still limited.

2 Testing Levels Based on Software Activity

Tests can be derived from requirements and specifications, design artifacts, or the source code. A different level of testing accompanies each distinct software development activity:

1. Acceptance Testing – Assess software with respect to requirements.
2. System Testing – Assess software with respect to architectural design.
3. Integration Testing – Assess software with respect to subsystem design.
4. Module Testing – Assess software with respect to detailed design.
5. Unit Testing – Assess software with respect to implementation.

3. Software Testing Techniques1

Software testing techniques is widely used in different applications using various testing strategies. Generally, software-testing techniques are classified into two categories: *static analysis* and *dynamic testing* .

In static analysis, a code reviewer reads the program source code, statement by statement, and visually follows the logical program flow by feeding an input. This type of testing is highly dependent on the reviewer's experience. Static analysis uses the program requirements and design documents for visual review. In contrast, dynamic testing techniques execute the program under test on test input data and observe its output. Usually, the term testing refers to just dynamic testing.

The following subsections give a brief background on these two testing categories.

- **Static Analysis**

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

For years, the majority of the programmers assumed that the programs are written solely for machine execution and are not intended to be read by human, and that the only way to test a program is by executing it on a machine. This manner began to change in the early 1970s, because of the Weinberg's work on "The Psychology of Computer Programming". Weinberg provided a convinced argument for why programs should be read by people and indicated that this could be an effective error detection process. Experience has shown that static analysis [4], a.k.a. non-computer-based or human testing, methods are quite effective in finding errors. Static analysis methods are meant to be applied during the period that is between the code completion and the beginning of the execution-based testing. Typical static analysis methods are code inspections, code walkthroughs, desk checking, and code reviews. Code inspections and walkthroughs are the two primary static analysis methods and they have a lot in common. Inspections and walkthroughs involve the reading or visual inspection of a program by a team of people. Both methods involve some preparatory works by the participants. The climax is a meeting of the minds, i.e. brainstorming, in a conference-like gathering held by the participants. The objective of the meeting is to find errors, but not to find solutions to the errors, i.e. to test but not to debug.

i)Code Inspections

Code inspection is a set of procedures and error-detection techniques for group code reading. Most discussions of code inspections focus on the procedures, forms to be filled out, and so on. During the inspection session, two activities are conducted: code narration and code examination. Code is read statement by statement and analyze with respect to a checklist of historically common programming errors [4] (e.g. data reference, data-declaration, computation, comparison, control-flow, input/output, interface).

ii) Code Walkthroughs

The initial procedure is identical to that of the inspection process. The difference, however, is in that rather than simply reading the program or using error checklists,one of the participants designated as a tester comes to the meeting with a small set of paper test cases that represent sets of input and expected output for the tested program or module. During the meeting, each test case is mentally executed, i.e. the test data are walked through the logic of the program. The state of the program, i.e. the values of the variables, is monitored on paper or a blackboard .

Definitely, the test cases must be simple in nature and few in number, because people execute programs at a much slower rate than a machine. Thus, the test cases themselves do not play a critical role; rather, they serve as a vehicle for getting started and for questioning the programmer about his or her logic and assumptions. In most walkthroughs, more errors are found during the process of questioning the programmer than are found directly by the test cases themselves.

iii) Desk Checking

Desk checking can be seen as a one-person inspection or walkthrough; a person reads a program, checks it with respect to an error list, and/or walks test data through it.

There are three main reasons that desk checking, for most people, is relatively unproductive: completely undisciplined process, the principle that people are generally ineffective in testing their programs, and no competition like in the teamwork.

iv) Code Reviews

Code review is a technique for evaluating anonymous programs in terms of their overall quality, maintainability, extensibility, usability, and clarity. The purpose of the review is to provide programmer assessment.

- **Dynamic Testing**

Dynamic testing techniques execute the program under test on test input data and observe its output .Usually , the term testing refers to dynamic testing.

There are two types of dynamic testing: black-box and white-box .

White-box testing is concerned with the degree to which test cases exercise or cover the logical flow of the program.

Black-box testing, on the other hand, tests the functionalities of software regardless of its internal structure, a.k.a. functional or specification–based testing.

The following subsections give a brief background on these two types of dynamic testing.

i)White Box Testing

White-box testing is more widely applied. It is also called logic-coverage testing or structural testing, because it sees the structure of the program .The objective of white box testing is to exercise of the different logic structures and flows in the program.

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

There are several White Box (Structural) testing criteria :

- ***Statement testing*:** Every statement in the software under test has to be executed at least once during testing. A more extensive and stronger strategy is branch testing.The advantage of this metrics is that it can be applied directly to object code and does not required processing source code. The disadvantage of statement coverage is that it is insensitive to some control structures.
- ***Branch testing:*** Branch coverage is a stronger criterion than statement coverage. It includes statement coverage since every statement is executed if every branch in a program is exercised once. However, some errors can only be detected if the statements and branches are executed in a certain order, which leads to path testing. This metric has the advantage of simplicity without the problems of statement coverage. A disadvantage is that this metric ignores branches within Boolean expressions which occur due to short-circuit operators.
- ***Condition coverage:*** In this case, one writes enough test cases such that each condition in a decision takes on all possible outcomes at least once. Although the condition coverage criterion appears, at first glance, to satisfy the decision coverage criterion, it does not always do so. If the decision IF (A AND B) is being tested, the condition coverage criterion would require one to write two test cases – A is TRUE, B is FALSE, and A is FALSE, B is TRUE – but this would not cause the THEN clause of the IF statement to execute.
- ***Path testing:*** In path testing every possible path in the software under test is executed; this increases the probability of error detection and is a stronger method than both statement and branch testing. A path through software can be described as the conjunction of predicates in relation to the software's input variables. However, path testing is generally considered impractical because a program with loop statements can have an infinite number of paths. A path is said to be 'feasible', when there exists an input for which the path is traversed during program execution, otherwise the path is unfeasible. Path coverage has the advantage of requiring very thorough testing.

ii) Black Box Testing

Black-box testing, a.k.a. functional or specification-based testing, tests the functionalities of software against its specification, regardless of its structure. There are four types of black-box testing: Equivalence Partitioning, Boundary-Value Analysis, Cause-Effect Graphing, and Error Guessing .

1. ***Equivalence partitioning*** partitions the input domain of a program into a finite number of equivalence classes such that one can reasonably assume (but, of course, not be sure absolutely) that a test of a representative value of each class is equivalent to a test of any other value within the corresponding class.

That is, if one test case in an equivalence class detects an error, all other test cases in the equivalence class would be expected to find the same error. Conversely, if a test case did not detect an error, we would expect that no other test cases in the equivalence class would find an error. This is with the exception that a subset of the equivalence class falls within another equivalence class, since equivalence classes may overlap one another. The equivalence partitioning concept maybe applied to white-box testing as well.

2. ***Boundary value analysis*** is those situations directly on, above, and under the edges of input equivalence classes and output equivalence classes. Experience shows that test cases that explore boundary conditions have a higher payoff than test cases that do not. One weakness of boundary-value analysis and equivalence partitioning is that they do not explore combinations of input data

as decision/condition coverage does in white-box testing.

3. ***A cause-effect graphing*** came to tackle this problem. It is a formal language into which a natural-language specification is translated, which also points out incompleteness and ambiguities in the specification. The graph is actually a digital-logic circuit (a combinatorial logic network), but rather than using standard electronics notation, a somewhat simpler notation is used. The idea is pretty similar with decision/condition coverage in the white box testing, while boundary-value analysis and equivalence partitioning are similar with condition

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

coverage criterion in white box testing. Thus, using similar analogy in white box testing, cause-effect graphing outperforms boundary-value analysis and equivalence partitioning.

4. ***Error guessing*** is largely an intuitive and ad-hoc process, whose procedure is difficult to formalize. This technique needs an expertise that is able to smell out errors. The basic idea is to enumerate a list of possible errors or errorprone situations and then write test cases based on the list.

4. Genetic Algorithm

"A genetic Algorithm is an iterative procedure maintaining a population of structures that are candidate solutions to specific domain challenges. During each temporal increment (called a generation), the structures in the current population are rated for their effectiveness as domain solutions, and on the basis of these evaluations, a new population of candidate solutions is formed using specific genetic operators such as reproduction, crossover, and mutation."

Genetic Algorithm can be defined as: "They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some of the innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old; an occasional new part is tried for good measure. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance."

5. Literature Survey

Dhananjay Thiruvady et al.(2014)[1]Contemporary motor vehicles have increasing numbers of automated functions to augment the safety and comfort of a car. The automotive industry has to incorporate increasing numbers of processing units in the structure of cars to run the software that provides these functionalities. The software components often need access to sensors or mechanical devices which they are designed to operate. The result is a network of hardware units which can accommodate a limited number of software programs, each of which has to be assigned to a hardware unit. They find that despite the large number of constraints, ACO on its own is the most effective method providing good solutions by also exploring infeasible regions

Yong Chen et al.(2008)[2] Automatic path-oriented test data generation is an undecidable problem and genetic algorithm (GA) has been used to test data generation since 1992. In favor of MATLAB, a multi-population genetic algorithm (MPGA) was implemented, which selects individuals for free migration based on their fitness values. Applying MPGA to generating path-oriented test data generation is a new and meaningful attempt.

A. Bouchachia et al.(2007)[3] This paper aims at incorporating immune operators in genetic algorithms as an advanced method for solving the problem of test data generation. The new proposed hybrid algorithm is called immune genetic algorithm (IGA). A full description of this algorithm is presented before investigating its application in the context of software test data generation using some benchmark programs. Moreover, the algorithm is compared with other evolutionary algorithms.

C.C. Michael et al.(2001)[4] discusses the use of genetic algorithms (GAs) for automatic software test data generation. This research extends previous work on dynamic test data generation where the problem of test data generation is reduced to one of minimizing a function. In our work, the function is minimized by using one of two genetic algorithms in place of the local minimization techniques used in earlier research. They describe the implementation of our GA-based system and examine the effectiveness of this approach on a number of programs, one of which is significantly larger than those for which results have previously been reported in the literature.

D.J.Berndt et al.(2004)[5] Highly complex and interconnected systems may suffer from intermittent or transient failures that are particularly difficult to diagnose. This research focuses on the use of genetic algorithms for automatically generating large volumes of software test cases. In particular, the paper explores two fundamental strategies for improving the performance of genetic algorithm test case breeding for high volume testing. The first strategy seeks to avoid evaluating test cases against the real target system by using oracles or models. The

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

second strategy involves improving the more costly components of genetic algorithms, such as fitness function calculations.

Xiajiong Shen et al.(2009)[6] A kind of software test data automated generation method based on genetic algorithm and tabu search algorithm is proposed. Having both local search capabilities of tabu search algorithm and global search capability of genetic algorithm, this tabu genetic algorithm combines tabu search algorithm with genetic algorithm. The experiment results show that the tabu genetic algorithm with tabu search as mutation operator is effective on generating test cases and its optimizing performance is superior to the simple genetic algorithm.

E.Diaz et al.(2004)[7] Software testing is an expensive and difficult process which need much time. For this reason, the existence of tools that allow to decrease this effort is very important. This tool automatically generates test cases in order to obtain branch coverage in software testing from a source code. They describe the modules of the tool and present the result they have obtained compared the needed time to generate the test cases with manual instrumentation and the needed time with an automatic process

Ruilian Zhao et al.[2003][8] shows that partition testing strategies are relatively ineffective in detecting faults related to small shifts in input domain boundary. They present an innovative software testing approach based on input domain analysis of specifications and programs, and propose the principle and procedure of boundary test case selection in functional domain and operational domain. To automatically determine the operational domain of a program, the ADSOD system is prototyped. The system supports not only the determination of input domain of integer and real data types, but also non-numeric data types such as characters and enumerated types. It consists of several modules in finding illegal values of input variables with respect to specific expressions.

6. RESEARCH METHODOLOGY

The various steps required to automatically test the software before its release.

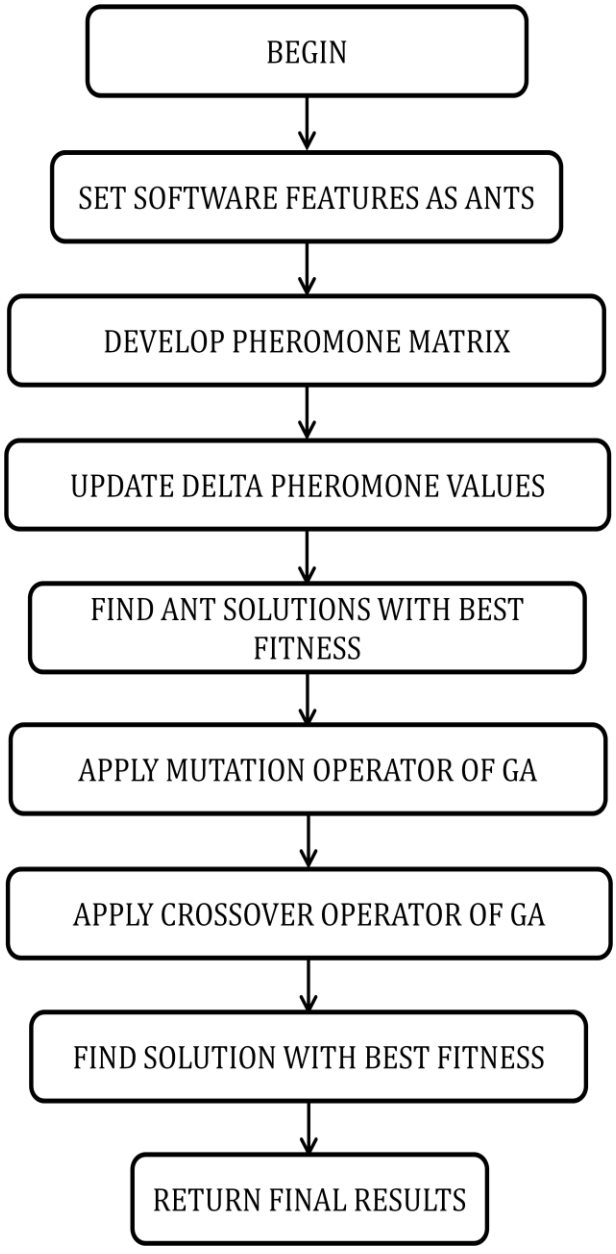

Figure 1: Fow chart of proposed methodology

7. RESULT AND DISCUSSION

7.1 Experimental Setup

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

Proposed approach has been designed and implemented in WEKA. The Hybrid approach is compared with various algorithms. The WEKA is a high-performance language in simple and easy-to-use environment to compute, visualize and programming to express the problems and solutions in mathematical form, developed by Mathworks. WEKA uses various languages C, C++, Java, FORTRAN and Python. It is supported on windows, Unix and Macintosh. The various algorithms have been applied on the given dataset. Parameters are used for making comparisons between existing algorithms and proposed algorithm. Hybrid approach provides the better results than all algorithms when compared with them. The proposed algorithm is compared with different algorithms, which are Decision Stump, AdaBoostM1, Stacking, ASC, IBK and I/P Mapped Classifer.

7.2 Performance Analysis

This major section depicts the values of various parameters such as true positive rate, false positive error, precision, recall, f-measure, ROC area.

TP (True Positive): It denotes records that are predicted as true and they were actually true.

FN (False Negative): It denotes records that are predicted as false and actually they were true.

FP (False Positive): It denotes the records that are predicted as true actually they were false.

TN (True Negative): It denotes the records that are predicted as false.

Table 1: TP & FP Rate Comparison Table

Algorithm	TP Rate	FP Rate
Decision Stump	0.855	0.855
ADABOOSTMI	0.953	0.201
IBK	0.829	0.643
ASC	0.883	0.455
STACKING	0.855	0.855
I/PMAPPED CLASSIFIER	0.855	0.855
Existing	0.867	0.595
Proposed	0.956	0.221

This comparison table proves that the proposed work results are much better than the existing algorithms which is as shown in fig 2.

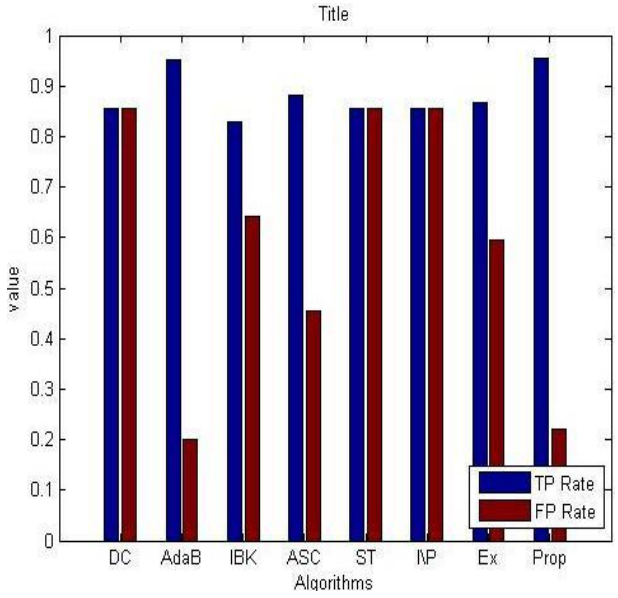

Figure 2: TP & FP Rate Graph

Precision: Precision is the part of retrieved instances that are appropriate. It is measure of exactness.

$$\text{Precision} = \frac{TN}{(TN + FP)}$$

Recall: Recall is the fraction of related instances that are retrieve it is measure of completeness.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Table 2: Precision & Recall Comparison Table

Algorithm	TP Rate	FP Rate
Decision Stump	0.731	0.855
ADABOOSTMI	0.951	0.953
IBK	0.807	0.829
ASC	0.873	0.883
STACKING	0.731	0.855
I/PMAPPED CLASSIFIER	0.731	0.855
Existing	0.846	0.867
Proposed	0.955	0.956

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

This comparison table proves that the proposed work results are much better than the existing algorithms which is as shown in fig 3.

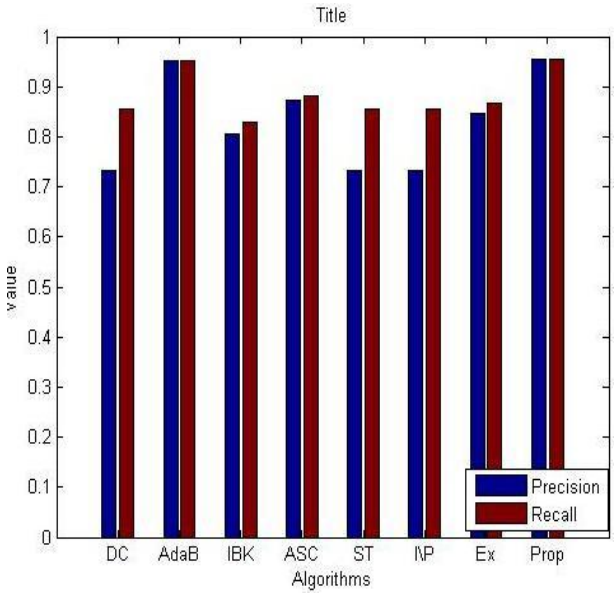

Figure 3: Precision & Recall Graph

F Measure: it combines and balance recall and precision. It is the compromise between recall and precision. When both are high

$$\text{F measure} = 2 * \frac{\text{precision.recall}}{\text{precision} + \text{recall}}$$

F measure is high. High value of F measure indicates better is the algorithm.

Table 2: F-Measure & ROC Comparison Table

Algorithm	TP Rate	FP Rate
Decision Stump	0.788	0.498
ADABOOSTMI	0.951	0.878
IBK	0.816	0.589
ASC	0.877	0.728
STACKING	0.789	0.5
I/PMAPPED CLASSIFIER	0.788	0.498
Existing	0.849	0.857
Proposed	0.953	0.914

This comparison table proves that the proposed work results are much better than the existing algorithms which is as shown in fig 4.

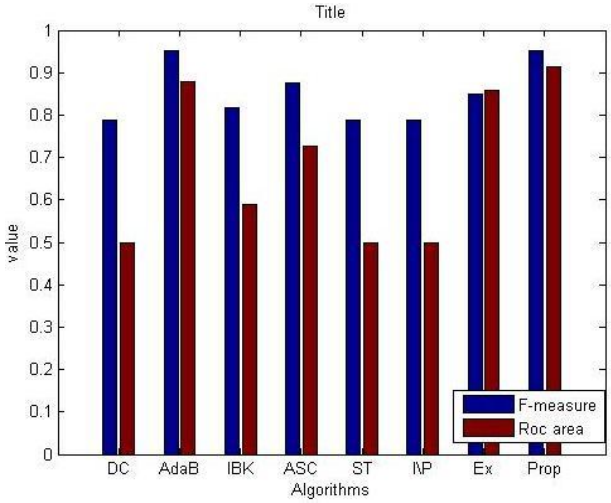

Figure 4: F-Measure & ROC Graph

Conclusion:

In this paper, we have evaluated the performance of ant colony optimization based software testing. Then propose a hybrid ant colony optimization with genetic algorithm to effectively test the software before its release. Further in this paper, we have compared the existing and proposed technique using following metrics such as true positive rate, false positive error, precision, recall, f-measure, ROC area. The proposed work results are much better than the existing results.

References

[1] Thiruvady, Dhananjay, et al."Constraint programming and ant colony system for the component deployment problem.", 14th international conference on computational science, vol. 29, pp. 1937-1947, 2014

[2] Y. Chen and Y. Zhong, "Automatic Path-Oriented Test Data Generation Using a Multi-population Genetic Algorithm," *2008 Fourth International Conference on Natural Computation*, Jinan, 2008, pp. 566-570.

[3] A. Bouchachia, "An Immune Genetic Algorithm for Software Test Data

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

Generation," *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, Kaiserlautern, 2007, pp. 84-89.

[4] Michael, Christoph C., Gary McGraw, and Michael A. Schatz. "Generating software test data by evolution." IEEE transactions on software engineering 27.12 (2001): 1085-1110.

[5] Berndt, Donald J., and Alison Watkins. "Investigating the performance of genetic algorithm-based software test case generation." High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on. IEEE, 2004.

[6] Shen, Xiajiong, et al. "Automatic generation of test case based on GATS algorithm." Granular Computing, 2009, GRC'09. IEEE International Conference on. IEEE, 2009.

[7] Díaz, Eugenia, Javier Tuya, and Raquel Blanco. "A modular tool for automated coverage in software testing." Software Technology and Engineering Practice, 2003. Eleventh Annual International Workshop on. IEEE, 2003.

[8] Zhao, Ruilian, Michael R. Lyu, and Yinghua Min. "A new software testing approach based on domain analysis of specifications and programs."*Software Reliability Engineering, 2003. ISSRE 2003. 14th International Symposium on*. IEEE, 2003.

[9] Ng, S. P., et al. "A preliminary survey on software testing practices in Australia." Software Engineering Conference, 2004. Proceedings. 2004 Australian. IEEE, 2004.

[10] Abu, Ghaffari, Joao W. Cangussu, and Janos Turi. "A quantitative learning model for software test process." Proceedings of the 38th Annual Hawaii International Conference on System Sciences. IEEE, 2005.

[11] Bai, Xiaoqing, Chiou Peng Lam, and Huaizhong Li. "An approach to generate the thin-threads from the UML diagrams." Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International. IEEE, 2004.

[12] Liu, Zhenyu, Ning Gu, and Genxing Yang. "An automate test case generation approach: using match technique." The Fifth International Conference on Computer and Information Technology (CIT'05). IEEE, 2005.

[13] Chang-Ai, Sun, et al. "Architecture framework for software test tool." Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Asia 2000. Proceedings. 36th International Conference on. IEEE, 2000.

[14] Last, Mark, and Menahem Friedman. "Automated detection of injected faults in a differential equation solver." High Assurance Systems Engineering, 2004. Proceedings. Eighth IEEE International Symposium on. IEEE, 2004.