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

Ankita
SSCET Badhani,Pathankot
Research scholar
CSE Department

Sukhbeer Singh
SSCET Badhani,Pathankot
Research scholar
CSE Department

ABSTRACT

The evaluation has show that the obtaining the very best automation application testing remains an open area of research. Software testing requires enough time because of its testing, ergo is costly task. The detailed evaluation has shown that the role of meta-heuristic methods throughout application testing is ignored in most of current research. The ant colony optimization process suffers from bad convergence speed. Therefore utilizing the hybrid ant colony process with genetic algorithm will be proposed in that work. The use of the proposed process has power to anticipate climate application can be utilized in real time or not.

Keywords
Software Testing, Software Testing techniques, genetic Algorithm.

1. INTRODUCTION

In recent ages the rising influences of application in most regions of business have led to an ever-increasing demand for complicated and trusted software. In accordance with a study done by the National Institute of Common & Technology, approximately 80% of the development price is used on identifying and repairing problems.

The same study discovered that application insects price the United Claims economy around $59.5 thousand a year, with one third of this value being related to the indegent application screening infrastructure. However, the automation of test knowledge generation continues to be a topic under research. Recently, several practices such as for example metaheuristic research, arbitrary test generation and static analysis have been used to completely automate the screening method, but the application of the instruments to real application continues to be limited.

2. SOFTWARE TESTING TECHNIQUES

Software screening techniques is widely found in various applications using various screening strategies.
Usually, software-testing techniques are classified into two groups: *static analysis* and *dynamic analysis.*
In static analysis, a code customer reads the programsource rule, record by record, and successfully follows the rational plan movement byfeeding an input. This kind of screening is extremely determined by the reviewer'sexperience. Static analysis employs the program needs and design documents forvisual review.
In comparison, active screening techniques execute the program under teston test feedback knowledge and see their output. Frequently, the word screening describes justdynamic testing.
The following subsections give a brief history on those two screening categories.

- **Static Evaluation**

For decades, many the programmers assumed that the applications are written exclusively for machine performance and aren't intended to be read by human, and that the only path to test a course is by executing it on a machine. Static analysis practices are meant to be applied during the period that is involving the rule completion and the start of the execution-based testing. Normal static analysis practices are rule inspections, rule walkthroughs, workplace examining, and rule reviews.
Rule inspections and walkthroughs are the 2 primary static analysis practices and they have a great deal in common. Inspections and walkthroughs involve the examining or aesthetic inspection of a course by a group of people. Both practices involve some preparatory functions by the participants. The climax is a conference of the thoughts, i.e. brainstorming, in a conference-like gathering presented by the participants. The aim of the conference is to find errors, but not to find solutions to the errors, i.e. to test but not to debug.

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

i) Code Walkthroughs

The original treatment is identical to that of the examination process. The huge difference, however, is in that rather than reading the program or applying error checklists,one of many individuals specified as a tester involves the meeting with a small set of paper check cases that represent pieces of insight and expected output for the tried program or module. Through the meeting, each check case is emotionally executed, i.e. the check knowledge are stepped through the reason of the program. The state of the program, i.e. the prices of the variables, is monitored in some recoverable format or perhaps a blackboard.

ii) Desk Checking

Desk checking is visible as a one-person examination or walkthrough; an individual says a course, checks it with respect to a mistake number, and/or hikes check knowledge through it.

There are three significant reasons that desk examining, for most of us, is fairly unproductive: completely undisciplined process, the principle that folks are usually inadequate in screening their applications, and no opposition like in the teamwork.

iii) Code Reviews

Code review is a strategy for assessing confidential applications in terms of their overall quality, maintainability, extensibility, simplicity, and clarity. The goal of the review is to offer engineer assessment.

- **Dynamic Evaluation**

Dynamic testing methods perform the program under check on check insight knowledge and view its output .Usually , the definition of screening refers to active testing.

There are two types of active screening: black-box and white-box.

White-box screening is worried with their education to which check cases exercise or cover the logical movement of the program.

Black-box screening, on the other give, checks the functionalities of computer software regardless of its central design, a.k.a. functional or specification–centered testing.

The next subsections offer a short background on those two types of active testing.

i)White Box Testing

White-box testing is more commonly applied. It is also known as logic-coverage screening or architectural screening, since it considers the design of the program .The objective of white package screening is always to exercise of the different reason structures and moves in the program.

ii) Black Box Testing

Black-box testing functional or specification-based screening, checks the functionalities of computer software against its specification, regardless of its structure. There are four types of black-box screening: Equivalence Dividing, Boundary-Value Evaluation, Cause-Effect Graphing, and Error Guessing.

3. GENETIC ALGORITHM

"A genetic Algorithm is definitely an iterative treatment sustaining a population of structures which can be choice methods to unique domain challenges. During each temporal increment (called a generation), the structures in today's population are rated because of their effectiveness as domain alternatives, and on the foundation of the evaluations, a fresh population of choice alternatives is shaped applying unique genetic operators such as for instance replica, crossover, and mutation."

Genetic Algorithm could be described as: "They combine survival of the fittest among sequence structures with a organized however randomized data change to make a search algorithm with some of the progressive style of human search. Atlanta divorce attorneys technology, a fresh set of artificial animals (strings) is established applying bits and items of the fittest of the previous; an unexpected new portion is attempted permanently measure. While randomized, genetic formulas are no easy random walk. They effortlessly use historical data to speculate on new research factors with estimated improved performance."

4. LITERATURE SURVEY

Xiaoqing Bai et al.(2004)[1] Application testing plays an essential position in assuring computer software qualityA scenario-based business design needs to be personally made or labor-intensive organization analysis needs to be personally moved out in order to extract the thin-threads from a computer software system. They propose an computerized way of right produce thin-threads from the UML artifacts. The produced thin-threads can be utilized to produce and to prioritize the test instances for scenario-based computer software testing.

Zhenyu Liu et al.(2005)[2] The report reports the principle of test case sell technique and put ahead an instantly test case technology approach. The used test case is represented in a way like the classification of computer software component. And then your software of test case is described utilising the related method of the information of facet in used computer software component. Centered on these technique, an automate method of test case is presented. The technology of test case technology could improve test performance and qualification.

Sun Chang-Ai et al.(2000)[3] necessity of research on an integrated platform for component-based computer software test tools is analyzed, based on the analysis of the original style of computer software test tools and the characteristics of such tools. They examine the architectural design of this incorporated platform and come up with a reference model for the platform, which is founded on components and which may be suitable for CORBA. Moreover, the technology for employing an energetic setup centered on different customers'needs can also be introduced

M.Last et al.(2004)[4] show the use of a data mining approach, called Info-Fuzzy System (IFN), which could instantly induce reasonable dependencies from performance data of a reliable computer software edition, develop a set of non-redundant test instances, and recognize bad outcomes in new, perhaps bad releases of the same system. The planned method is put on the Unstructured Mesh Finite Aspect Solver (UMFES) which is a normal finite factor plan for solving 2D elliptic partial differential equations.

Zhi Quan Zhou et al.(2007)[5] As pc systems are permeating our culture in everyday life and are performing an increasing quantity of critical projects, research in computer software testing and analysis is becoming of paramount importance. Although they are currently not able to prove plan correctness for real-world programs, rigorous computer software growth operations in conjunction with testing gives people with confidence in the quality of software.

Hong Zhu et al.(2007)[6] In computer software growth practice, testing records for around 50% of whole growth efforts. It is crucial to lessen the fee and improve the effectiveness of computer software testing by automating the testing process, which includes several testing connected activities applying different practices and methods.

A.Watkins et al.(2004)[7] emphasize that current testing practices are inferior, and that supporting minimize computer software insects and errors is an essential area of research with a considerable payoff. The objectives of the report are to explore the use of genetic formulas for testing complex distributed systems, along with to produce a framework or terminology of important environmental features that characterize complex systems failures.

5. RESEARCH METHODOLOGY

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

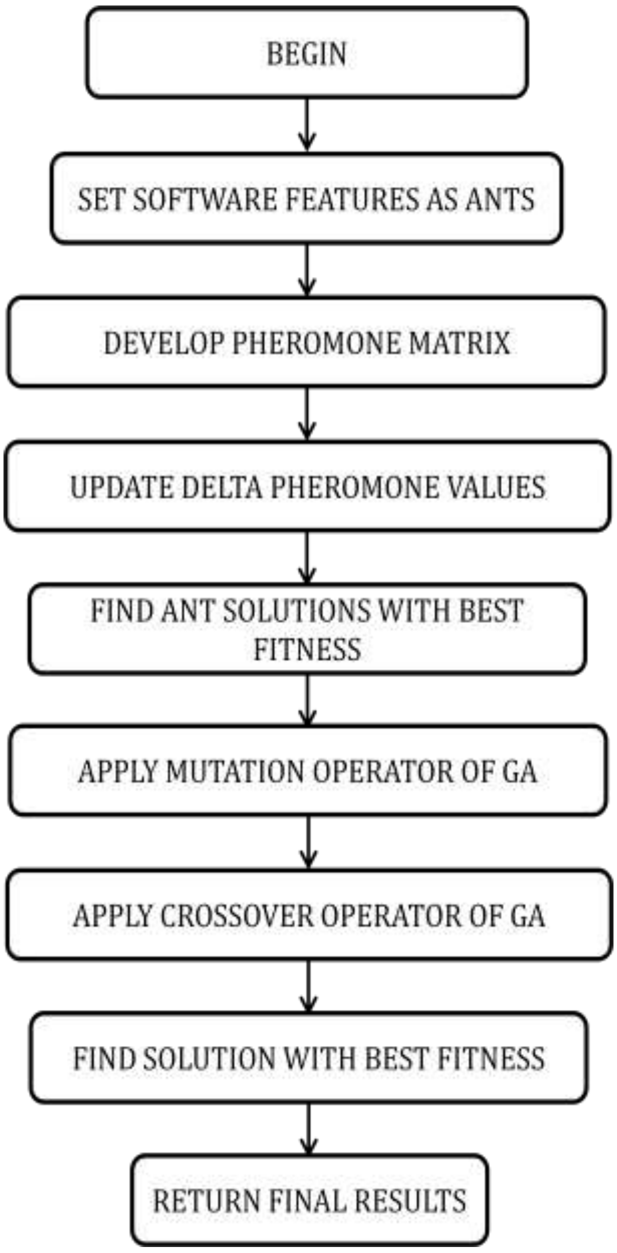
BEGIN
SET SOFTWARE FEATURES AS ANTS
DEVELOP PHEROMONE MATRIX
UPDATE DELTA PHEROMONE VALUES
FIND ANT SOLUTIONS WITH BEST FITNESS
APPLY MUTATION OPERATOR OF GA
APPLY CROSSOVER OPERATOR OF GA
FIND SOLUTION WITH BEST FITNESS
RETURN FINAL RESULTS

Figure 1: Fow chart of proposed methodology

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

INNOVATIVE RESEARCH ORGANISATION

International Journal of Advance Research in Education, Technology & Management

(Scholarly Peer Review Publishing System)

6. RESULT AND DISCUSSION

6.1 Experimental Setup

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.

6.2 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.
FP (False Positive): It denotes the records that are predicted as true actually they were false.

Table 1: TP & FP Rate 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

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

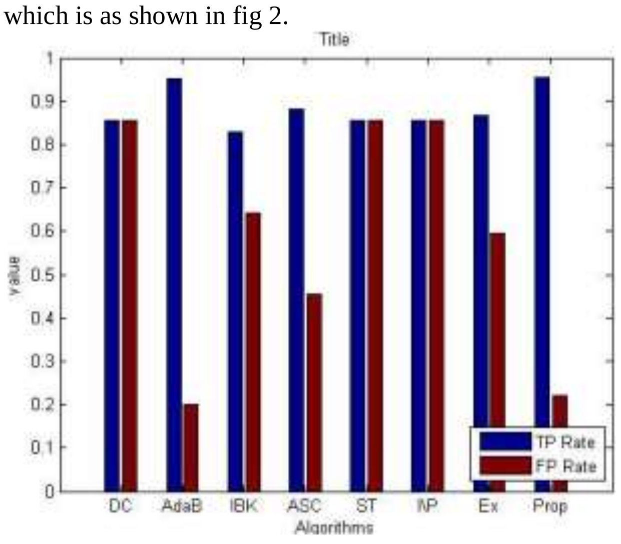

Figure 2: TP & FP Rate Graph

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 and false positive error. The proposed work results are much better than the existing results.

7. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] Zhou, Zhi Quan, Bernhard Scholz, and Giovanni Denaro. "Automated software testing and analysis: Techniques, practices and tools." System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on. IEEE, 2007.

[6] Zhu, Hong, W. Eric Wong, and Amit Paradkar. "Automation of Software Test--Report on the Second Interional Workshop AST 2007." Companion to the proceedings of the 29th

International Conference on Software Engineering. IEEE Computer Society, 2007.

[7] Watkins, Alison, et al. "Breeding software test cases for complex systems." System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on. IEEE, 2004.

[8] Berndt, Donald, et al. "Breeding software test cases with genetic algorithms." System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on. IEEE, 2003.

[9] Murphy, John A., and David Coppit. "Random Generation of Test Inputs for Implicitly Defined Subdomains." Proceedings of the Second International Workshop on Automation of Software Test. IEEE Computer Society, 2007.

[10] Elbaum, Sebastian, et al. "Bug hunt: Making early software testing lessons engaging and affordable." *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007.

[11] Taipale, Ossi, Kari Smolander, and H. Kalviainen. "Factors affecting software testing time schedule." Australian Software Engineering Conference (ASWEC'06). IEEE, 2006.

[12] Cai, Kai-Yuan, Yong-Chao Li, and Ke Liu. "How to test software for optimal software reliability assessment." Quality Software, 2003. Proceedings. Third International Conference on. IEEE, 2003.

[13] Cangussu, Joao W. "Modeling and controlling the software test process." Proceedings of the 23rd International Conference on Software Engineering. IEEE Computer Society, 2001.

[14] Torres, Dante, Daniel Leitao, and Flavia Barros. "Motorola specnl: A hybrid system to generate nl descriptions from test case specifications." 2006 Sixth International Conference on Hybrid Intelligent Systems (HIS'06). IEEE, 2006.

[15] Hyunsook Do and G. Rothermel, "On the Use of Mutation Faults in Empirical Assessments of Test Case Prioritization Techniques," in *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 733-752, Sept. 2006.

[16] W. Masri, A. Podgurski and D. Leon, "An Empirical Study of Test Case Filtering Techniques Based on Exercising Information Flows," in *IEEE Transactions on Software Engineering*, vol. 33, no. 7, pp. 454-477, July 2007.

[17] S. Elbaum and M. Diep, "Profiling deployed software: assessing strategies and testing opportunities," in *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 312-327, April 2005