Optimization of Test Data By Flower Pollination Approach In Structural Testing

Monika Thakur[1], Sanjay[2]

Student, Dept. of Computer Science & Engg., Himachal Pradesh Technical University, Hamirpur, India

Assistant Professor, Dept. of Computer Science & Engg., Himachal Pradesh Technical University, Hamirpur, India

Abstract— In this thesis work on test case convergence of test case optimization by Ant colony approach and flower pollination algorithm on eight different programs but as our result analysis FPA P-value goes to more than 80% and success rate 75% . it show that in test cases Flower pollination algorithm.

I. INTRODUCTION

With information about the idea of the thing or organization under test to give partners the software testing is examination coordinated. To empower the business to recognize and grasp the perils of software use the software testing can in like manner give an objective, self-ruling point of view of the software . With the point of finding software bugs (mistakes or distinctive imperfections) the test methodology join the route toward executing a program or application, and watching that the software thing is fit for use. To survey no less than one properties of interest the software testing incorporates the execution of a software part or system portion.

As a rule, these properties show the degree to which the part or framework under test:

- Meets the necessities that guided its outline and advancement,
- Responds accurately to a wide range of data sources,
- Performs its capacities inside a worthy time,
- Is adequately usable,
- Can be introduced and keep running in its proposed surroundings, and
- Achieves the general outcome its partner's yearning.

For even fundamental software parts is essentially boundless as the amount of possible test, for the open time and resources all the software testing uses some framework to pick tests that are feasible. Hence, with the objective of finding software bugs (botches or distinctive defects) the software testing conventionally (however not exclusively) attempts to execute a program or application. When one bug is settled the work of testing is an iterative methodology, more significant bugs, can even make new ones, or it can illuminate other. About the idea of software and risk of its failure to customers or benefactors, the software testing can give objective, independent information. At the point when executable software (paying little mind to the likelihood that to a limited extent whole) exist the software testing can be coordinated. The general way to deal with software progression frequently chooses when and how testing is driven. For example, in an arranged strategy, most testing occurs after system requirements have been described and after that realized in testable activities. Alternately, under an Agile approach, essentials, programming, and testing are often done at the same time.

Software Testing Methods:

White-box testing: White-box testing (generally called clear box testing, glass box testing, direct box testing and fundamental testing, by seeing the source code) tests internal structures or workings of a program, instead of the helpfulness exhibited to the end-customer. In white-box testing an internal viewpoint of the system, and what's more programming capacities, are used to design test cases. The analyzer picks inputs to hone courses through the code and determine the fitting yields. This is for all intents and purposes equal to testing centers in a circuit, e.g. in-circuit testing (ICT).

Black-box testing:

Black-box testing sees the item as a "black box", taking a gander at usefulness with no data of inside utilization, without seeing the source code. The analyzers are quite recently aware of what the item ought to do, not how it does it. Black-box testing methods include: fairness distributing, regard examination, all-sets testing, state move tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing and specification-based testing.

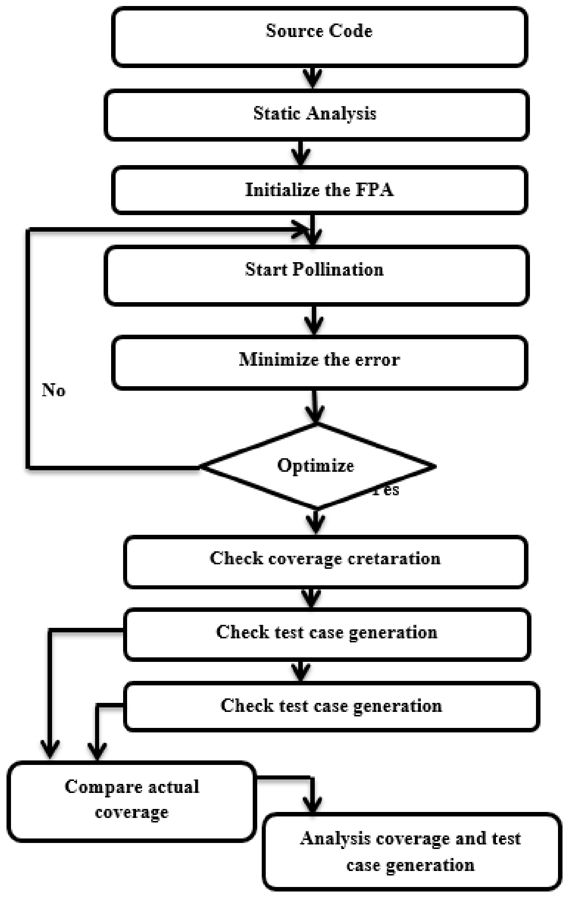
Visual testing:

The purpose of visual testing is to outfit engineers with the ability to review what was happening at the reason for programming dissatisfaction by displaying the data to such an extent that the fashioner can without a lot of an extend find the information she or he requires, and the information is imparted clearly.

II. LITARATURE REVIEW

In [2] the fundamental ACO algorithm is transformed into discrete form to produce test information for structural testing. To start with, the specialized guide of joining the adjusted ACO algorithm and test process together is presented. Keeping in mind the end goal to enhance algorithm's searching capacity and produce more differing test inputs, a few techniques, for example, nearby exchange, worldwide exchange and pheromone refresh are characterized and connected. In [3] Search-Based Software Testing is the utilization of a meta-heuristic upgrading search method, for example, a Genetic Algorithm, to mechanize or halfway robotize a testing undertaking; for instance the programmed era of test information. Key to the optimization procedure is an issue particular wellness work. The part of the wellness work is to control the search to great arrangements from a possibly unending search space, inside a commonsense time confine. Work on Search-Based Software Testing goes back to 1976, with enthusiasm for the region starting to assemble pace in the 1990s. All the more as of late there has been a blast of the measure of work. In [4] to close the crevice by exploring the two perspectives in regards to the banquet and cutoff points of test computerization. The scholarly perspectives are considered with an efficient writing audit while the experts sees are evaluated with an overview, where it got reactions from 115 software experts. In [5] Crowdsourcing middle people assume a key part in crowdsourcing activities as they guarantee the association between the crowdsourcing organizations and the group. Be that as it may, the issue of how crowdsourcing go-betweens oversee crowdsourcing activities and the related difficulties has not been addresses by explore yet. It address these issues by directing a contextual analysis with a German start-up crowdsourcing mediator called testCloud that offers software testing administrations for organizations planning to halfway or completely outsource their testing exercises to a specific group. In [6] to show and examine (some of) the best software testing research performed since the year 2000. It is without a doubt troublesome, if certainly feasible, to condense in any entire way very nearly 15 years of research and refer to every single important paper in a generally short report, for example, this one. They in this manner did not endeavor to cover every important subject and endeavors, but instead concentrated on those that our partners and we thought were especially applicable, had just had an extensive effect, or appeared to probably have affect in the (close) future. In [7] gives a complete review of ebb and flow ways to deal with the prophet issue and an examination of patterns in this imperative zone of programming testing exploration and practice. In [8] the meaning of cloud testing was gotten from the idea of cloud computing. It dissected the inquiries of which programming testing undertakings can do the cloud testing, why do clouds testing, how to do cloud testing. This paper was an examination for the future programming testing strategies. In [9] presents a way to deal with robotize the era of such test-information. The test-information era depends on the utilization of a dynamic streamlining based scan for the required test-information. A similar approach can be summed up to take care of other test-information era issues. In [10] Genetic algorithms have been used to make test sets thusly by means of looking through the space of the item for sensible regards to satisfy a predefined testing measure. These criteria have been set by the necessities for test educational accumulation sufficiency of structural testing, for instance, obtaining full branch scope and controlling the amount of emphasess of an unexpected circle. In [11] starts with a concise prologue to transformation investigation. It at that point takes the peruser on a guided voyage through Mothra, underscoring how it connects with the analyzer. At that point it give a short exchange of Mothra's interior outline. Next, it talked about some real issues with utilizing change examination and talk about conceivable arrangements. It closed by introducing an answer for one of these issues another technique for naturally creating transformation satisfactory test information.

III. METHODOLOGY


Source Code
Static Analysis
Initialize the FPA
Start Pollination
Minimize the error
No
Optimize
Yes
Check coverage cretaration
Check test case generation
Check test case generation
Compare actual coverage
Analysis coverage and test case generation

Step1: Take the source code.

Step2: In this step, do the static analysis of source code.

Step3: Initialized the FPA after the analysis.

Step4: When FPA is initialized pollination is started.

Step5: Error is minimized in this step.

Step6: Optimized the code.

Step7: If code is optimized the check coverage cretaration, otherwise go to step4.

Step8: Check test case generation.

Step9: When test cases are checked then compare actual coverage.

Step10: In final step, Analysis coverage and test case generation.

Flower Pollination Algorithm:

```
min or max objective f(y), y = (y1, y2 , . . . , yd )
Initialize b flower or pollen gametes populace with irregular solutions
Distinguish the best solution (g*) in the underlying populace
Express a switch probability p [0, 1]
**While** (t < Max Generation)
for i = 1 : n (all n blooms in the populace)
**if** rand < p,
Draw a (d-dimensional) step vector L from a Levy distribution
Global pollination by means of $y_i^{t+1} = y^i t + L(g^* - y_i^t)$
else
Draw from a uniform distribution in [0,1]
Do local pollination by means of $y_i^{t+1} = y_i^t + (y_j^t - y_k^t)$
 end if
Evaluate new solutions
On the off chance that new solutions are better, refresh them in populace
end for Find current best solution
end while
Yield the best solution acquired
```

IV. RESULTS

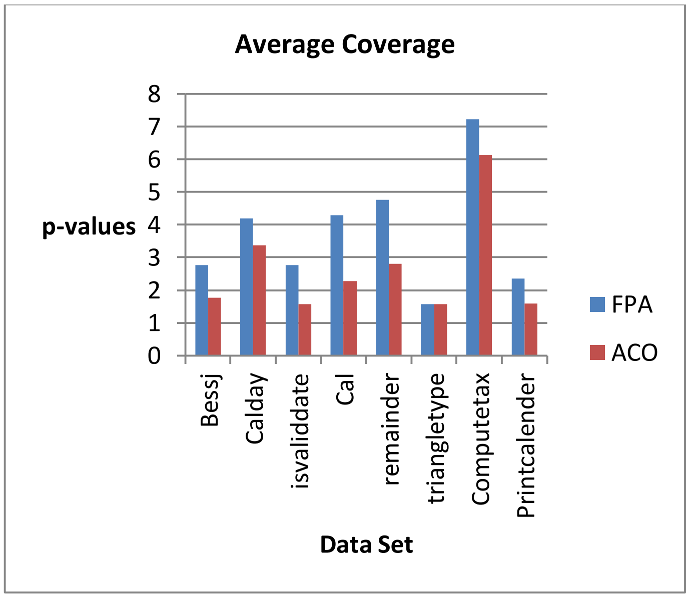
Table 1: Average coverage of Flower pollination algorithm

	Average Coverage	FPA
Data set Name	X-Y(%)	p-values
Bessj	1.34	2.76E-03
Calday	1.05	4.27E-03
Isvaliddate	3.05	2.76E-34
Cal	2.5	4.29E-41
Remainder	1.9	4.76E-21
Triangletype	2.12	1.57E+00
Computetax	5.62	7.23E-28
Printcalender	6.72	2.35E-105

Table 2: Average coverage of Ant Colony algorithm

	Average Coverage	ACO
Data set Name	X-Y(%)	p-values
Bessj	0.48	1.76E-08
Calday	0.03	3.37E-11
Isvaliddate	2.3	1.58E-48
Cal	0.73	2.29E-41
Remainder	0.15	2.86E-23
triangletype	0.12	1.57E-01
Computetax	4.78	6.13E-65
Printcalender	5.54	1.59E-130

Graph 1: Comparison graph between the average coverage (p-value) of FPA and ACO



Graph 2: Comparison graph between the average coverage (X-Y) of FPA and ACO

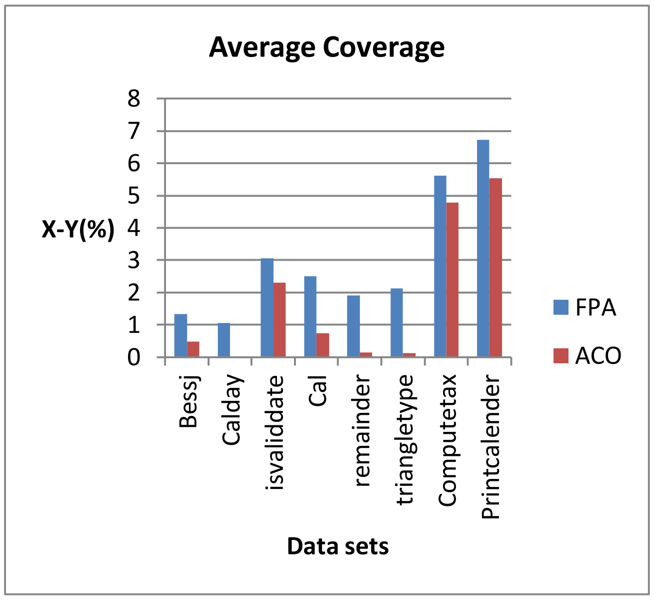


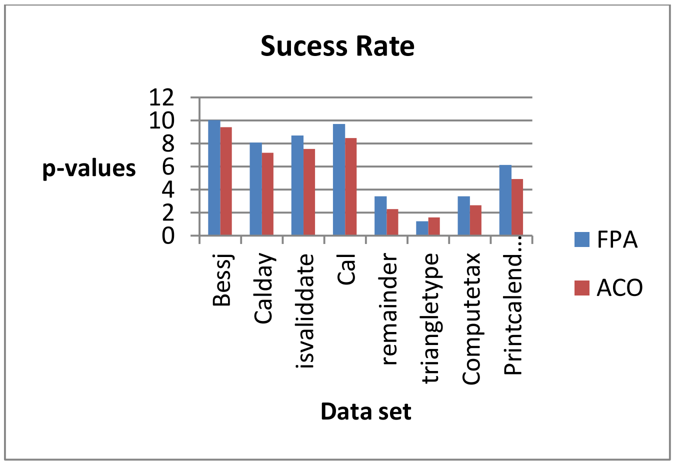
Table 3: Success rate of FPA

	Success Rate	
Data set Name	**X-Y(%)**	**p-values**
Bessj	3.5	1.04E-05
Calday	2.04	8.11E-04
isvaliddate	4.9	8.70E-20
Cal	5.7	9.68E-25
remainder	3.2	3.42E-15
triangletype	1.85	1.24E+00
Computetax	10.2	3.44E-05
Printcalender	81.02	6.1342-124

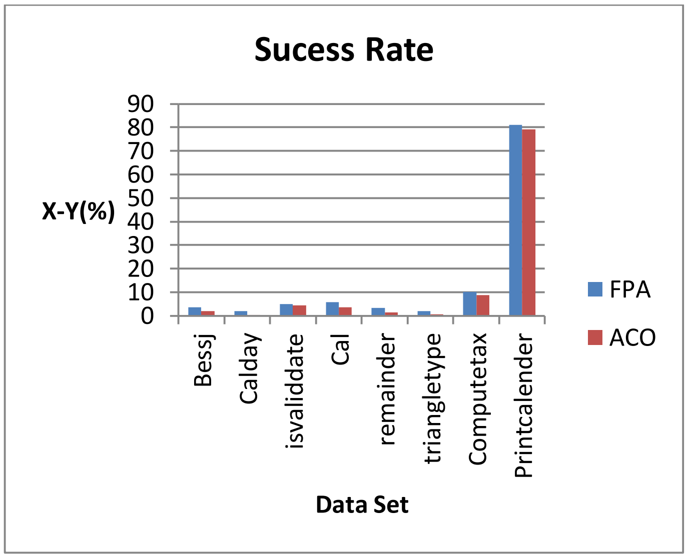
Table 4: Success rate of ACO

	Success Rate	
Data set Name	**X-Y(%)**	**p-values**
Bessj	2	9.41E-06
Calday	0.4	7.21E-12
isvaliddate	4.5	7.51E-45
Cal	3.5	8.48E-49
remainder	1.4	2.32E-26
triangletype	0.6	1.57E-01
Computetax	8.7	2.63E-35
Printcalender	79.1	4.9586-154

Graph 3: Comparison between success rate (p-value) of FPA and ACO



Graph 4: Comparison between success rate (X-Y) of FPA and ACO



V. REFERENCES

[1]. https://en.wikipedia.org/wiki/Software_testing.

[2]. Mao, Chengying, et al. "Adapting ant colony optimization to generate test data for software structural testing." *Swarm and Evolutionary Computation* 20 (2015): 23-36.

[3]. McMinn, Phil. "Search-based software testing: Past, present and future." *Software testing, verification and validation workshops (icstw), 2011 ieee fourth international conference on*. IEEE, 2011.

[4]. Rafi, Dudekula Mohammad, et al. "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey." *Proceedings of the 7th International Workshop on Automation of Software Test*. IEEE Press, 2012.

[5]. Zogaj, Shkodran, Ulrich Bretschneider, and Jan Marco Leimeister. "Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary." *Journal of Business Economics* 84.3 (2014): 375-405.

[6]. Orso, Alessandro, and Gregg Rothermel. "Software testing: a research travelogue (2000–2014)." *Proceedings of the on Future of Software Engineering*. ACM, 2014.

[7]. Harman, Mark, et al. "A comprehensive survey of trends in oracles for software testing." *University of Sheffield, Department of Computer Science, Tech. Rep. CS-13-01* (2013).

[8]. Jun, Wang, and Fanpeng Meng. "Software testing based on cloud computing." *Internet Computing & Information Services (ICICIS), 2011 International Conference on*. IEEE, 2011.

[9]. Tracey, Nigel, et al. "An automated framework for structural test-data generation." *Automated Software Engineering, 1998. Proceedings. 13th IEEE International Conference on*. IEEE, 1998.

[10]. Jones, Bryan F., H-H. Sthamer, and David E. Eyres. "Automatic structural testing using genetic algorithms." *Software Engineering Journal* 11.5 (1996): 299-306.