International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

DYNAMIC TEST CASE GENERATION USING RIVER FORMATION DYNAMICS ALGORITHM

Gaurav Kumar
Managing Director
Magma Research and Consultancy Pvt. Ltd.

Corporate Office
5480-81/1, Surya Tower, Second Floor
Opp. Bank of Maharashtra, Nicholson Road
Ambala Cantt., Haryana, India

ABSTRACT

Test Case Generation is deemed as one of the utmost important task during development as well as post deployment in the software engineering domain. In the classical way, the test cases can be generated using greedy methods or heuristic approaches. In the generalized methodology of manual software testing, the greedy method is used while the heuristic approaches are used in the automated test cases with the integration of specialized software suites including silk test, test director, load runner, win runner and many others. In this research work, an effective and unique metaheuristic approach named river dynamics is proposed to be used alongwith the literature analysis of existing approaches so that the effective comparison can be placed. The river formation dynamics makes use of the approach by which the rivers are formed in their path from mountains to plain locations. It is expected and proposed that the novel algorithm will give better

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

and optimized results in terms of execution time, overhead, complexity and overall performance of the test case generation.

Keywords - Software Testing, Test Case Generation, Metaheuristic based Test Case Generation.

I. INTRODUCTION

Software testing [1] is an investigation conducted to provide users with information about the quality of the product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques [2] include the process of executing a program or application with the intent of finding software bugs, errors or other issues.

It involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- Meets the requirements that guided its design and development,
- Responds correctly to all kinds of inputs,
- Performs its functions within an acceptable time,
- Sufficiently usable
- installed and run in its intended environments, and
- Achieves the general result as per the requirements.

II. REVIEW OF LITERATURE

To propose and defend the research work, a number of research papers are analyzed. Following are the excerpts from the different research work performed by number of academicians and researchers.

The work in [3] presents the integration of Search-Based Software Testing with the use of a meta-heuristic optimizing search technique, such as a Genetic Algorithm, to automate or partially automate a testing task, for example the automatic generation of test data. Key to the optimization process is a problem-specific fitness function. Here, the role of the fitness function is to guide the search to good solutions from a potentially infinite search space, within a practical time limit. Work on Search-Based Software Testing dates back to 1976, with interest in the area beginning to gather pace in the 1990s. This paper reviews past work and the current state of the art, and discusses potential future research areas and open problems that remain in the field.

In the research paper [4], the issue of applying search-based software engineering (SBSE) techniques is discussed with a multitude of different parameters that need to be chosen including the population size, selection mechanism, settings to use for dozens of other parameters. This problem not only troubles users who want to apply SBSE tools in practice, but also researchers performing experimentation how to compare algorithms that can have different parameter settings. To put the light on the problem of parameters, this work performed the largest empirical analysis on parameter tuning in SBSE to date, collecting and statistically analyzing data from more than a million experiments. As case study, this research work chose test data generation, one of the most popular problems in SBSE. The data confirm that tuning does have a critical impact on algorithmic performance, and over-fitting of parameter tuning is a dire threat to external validity of empirical analyses in SBSE. Based on this large empirical evidence, this work gives guidelines on how to handle parameter tuning.

[5] The aim of Search Based Software Engineering (SBSE) research is to move software engineering problems from human-based search to machine-based search, using a variety of techniques from the metaheuristic search, operations research and evolutionary computation paradigms. The idea is to exploit humans' creativity and machines' tenacity and reliability, rather than requiring humans to perform the more tedious, error prone and thereby costly aspects of the engineering process. SBSE can also provide insights and decision support. This work presents

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

the reader with a step-by-step guide to the application of SBSE techniques to Software Engineering. It assumes neither previous knowledge nor experience with Search Based Optimisation. The intention is that the work covers sufficient material to allow the reader to become productive in successfully applying search based optimisation to a chosen Software Engineering problem of interest.

The work in [6] Search-based techniques have been shown useful for the task of generating tests, for example in the case of object-oriented software. But, as for any meta-heuristic search, the efficiency is heavily dependent on many different factors, seeding is one such factor that may strongly influence this efficiency. In this paper, we evaluate new and typical strategies to seed the initial population as well as to seed values introduced during the search when generating tests for object-oriented code. This work report the results of a large empirical analysis carried out on 20 Java projects (for a total of 1,752 public classes). The proposed experiments show with strong statistical confidence that, even for a testing tool that is already able to achieve high coverage, the use of appropriate seeding strategies can further improve performance.

[7] Test case generation is among the most labour-intensive tasks in software testing. It also has a strong impact on the effectiveness and efficiency of software testing. For these reasons, it has been one of the most active research topics in software testing for several decades, resulting in many different approaches and tools. This paper presents an orchestrated survey of the most prominent techniques for automatic generation of software test cases, reviewed in self-standing sections. The techniques presented include: (a) structural testing using symbolic execution, (b) model-based testing, (c) combinatorial testing, (d) random testing and its variant of adaptive random testing, and (e) search-based testing. Each section is contributed by world-renowned active researchers on the technique, and briefly covers the basic ideas underlying the method, the current state of the art, a discussion of the open research problems, and a perspective of the future development of the approach. As a whole, the paper aims at giving an introductory, up-to-

date and (relatively) short overview of research in automatic test case generation, while ensuring a comprehensive and authoritative treatment.

[8] Search-based testing can automatically generate unit test suites for object oriented code, but may struggle to generate specific values necessary to cover difficult parts of the code. Dynamic symbolic execution (DSE) efficiently generates such specific values, but may struggle with complex datatypes, in particular those that require sequences of calls for construction. The solution to these problems lies in a hybrid approach that integrates the best of both worlds, but such an integration needs to adapt to the problem at hand to avoid that higher coverage in a few corner cases comes at the price of lower coverage in the general case. We have extended the Genetic Algorithm (GA) in the Evosuite unit test generator to integrate DSE in an adaptive approach where feedback from the search determines when a problem is suitable for DSE. In experiments on a set of difficult classes our adaptive hybrid approach achieved an increase in code coverage of up to 63% (11% on average); experiments on the SF100 corpus of roughly 9,000 open source classes confirm that the improvement is of practical value, and a comparison with a DSE tool on the Roops set of benchmark classes shows that the hybrid approach improves over both its constituent techniques, GA and DSE.

[9] Randomized algorithms are widely used to address many types of software engineering problems, especially in the area of software verification and validation with a strong emphasis on test automation. However, randomized algorithms are affected by chance and so require the use of appropriate statistical tests to be properly analysed in a sound manner. This paper features a systematic review regarding recent publications in 2009 and 2010 showing that, overall, empirical analyses involving randomized algorithms in software engineering tend to not properly account for the random nature of these algorithms. Many of the novel techniques presented clearly appear promising, but the lack of soundness in their empirical evaluations casts unfortunate doubts on their actual usefulness. In software engineering, although there are guidelines on how to carry out empirical analyses involving human subjects, those guidelines are

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

not directly and fully applicable to randomized algorithms. Furthermore, many of the textbooks on statistical analysis are written from the viewpoints of social and natural sciences, which present different challenges from randomized algorithms. To address the questionable overall quality of the empirical analyses reported in the systematic review, this paper provides guidelines on how to carry out and properly analyze randomized algorithms applied to solve software engineering tasks, with a particular focus on software testing, which is by far the most frequent application area of randomized algorithms within software engineering.

The work in [10] presents the Software Product Line (SPL) testing as challenging due to the potentially huge number of derivable products. To alleviate this problem, numerous contributions have been proposed to reduce the number of products to be tested while still having a good coverage. However, not much attention has been paid to the order in which the products are tested. Test case prioritization techniques reorder test cases to meet a certain performance goal. For instance, testers may wish to order their test cases in order to detect faults as soon as possible, which would translate in faster feedback and earlier fault correction. In this paper, we explore the applicability of test case prioritization techniques to SPL testing. This work proposes five different prioritization criteria based on common metrics of feature models and we compare their effectiveness in increasing the rate of early fault detection, i.e. a measure of how quickly faults are detected. The results show that different orderings of the same SPL suite may lead to significant differences in the rate of early fault detection. The research work also show that the proposed approach may contribute to accelerate the detection of faults of SPL test suites based on combinatorial testing.

[11] The software prototype model can be used for the generation of the verification test. The input stimuli, which form essential activity vectors, are selected from randomly generated ones on the base of software prototype. The essential activity vectors correspond to the terms of logical functions of output the existence of which is tested during the verification. The verification test is formed on the base of the essential activity vectors according to the defined

rules. The quality of the verification test is measured by the following parameters: the length of test, the fault coverage of the stuck-at faults, the fault coverage of the pin pair faults, and the number of the essential activity vectors. The paper presents the experimental results for the benchmark suite ISCAS'85. The value of this approach is highlighted by the fact that the selected input stimuli detect the same stuck-at faults as the initially generated test set.

The research paper [12] presents a novel approach for generating string test data for string validation routines, by harnessing the Internet. The technique uses program identifiers to construct web search queries for regular expressions that validate the format of a string type (such as an email address). It then performs further web searches for strings that match the regular expressions, producing examples of test cases that are both valid and realistic. Following this, our technique mutates the regular expressions to drive the search for invalid strings, and the production of test inputs that should be rejected by the validation routine. The paper presents the results of an empirical study evaluating our approach. The study was conducted on 24 string input validation routines collected from 10 open source projects. While dynamic symbolic execution and search-based testing approaches were only able to generate a very low number of values successfully, our approach generated values with an accuracy of 34% on average for the case of valid strings, and 99% on average for the case of invalid strings. Furthermore, whereas dynamic symbolic execution and search-based testing approaches were only capable of detecting faults in 8 routines, our approach detected faults in 17 out of the 19 validation routines known to contain implementation errors.

In [13], the key main challenge in the production and development of large and complex software projects is the cost estimation with high precision. Thus it can be said that estimating the cost of software projects play an important role in the organization productivity. With the increasing size and complexity of software projects the demand to offer new techniques to accomplish this important task increases day by day. Therefore, researchers have long attempted to provide models to fulfill this important task. The most documented algorithmic model is the

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

Constructive Cost Model (COCOMO), which was introduced in 1981 by Barry W. Boehm. But due to the lack of values for the constant parameters in this model, it cannot meet the high precision for all software projects. Nowadays, regarding the increasing researches on machine learning algorithms and the success of these studies, in this paper, we have tried to estimate the cost of software projects according to meta-heuristic algorithms. In this paper, Ant Colony Optimization (ACO) and Lorentz transformation have been used as Chaos Optimization Algorithm (COA) and NASA datasets as training and testing sets. To compare and evaluate the results of the proposed method with COCOMO model, MARE is used, and the results show a decline in MARE to 0.078%.

III. PROBLEM FORMULATION

As the domain of Software Testing and Test Case Generation is much diversified, there is lots of scope of research for the scholars and practitioners. The proposed work shall integrate the test case generation using river formation dynamic algorithm by which only the selected and optimized path will be followed depending upon the fuzzy fitness parameters.

River formation dynamics (RFD) is based on imitating how water forms rivers by eroding the ground and depositing sediments. After drops transform the landscape by increasing/decreasing the altitude of places, solutions are given in the form of paths of decreasing altitudes. Decreasing gradients are constructed, and these gradients are followed by subsequent drops to compose new gradients and reinforce the best ones. This heuristic optimization method was first presented in 2007 by Rabanal et al. The applicability of RFD to other NP-complete problems was studied in. Some other authors have also applied RFD in robot navigation or in routing protocols.

IV. PROPOSED ALGORITHMIC APPROACH

- Our research work generates the test cases using river formation dynamics.
- Classically, River is having a bulk of water that flow in a specific direction without leakage to different paths.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

- In our case, the set of best test cases shall be considered as a River.
- In natural phenomena, the water comes from multiple direction mountains and then merge into the river.
- In our research work, each test case (water) will be generated from different mathematical functions (mountains).
- This process will continue till we get the best and optimal set (path of river) of test cases (free flowing water without leakage).
- Here, Leakage of Water is related to the Overlapping and Redundant Test Cases

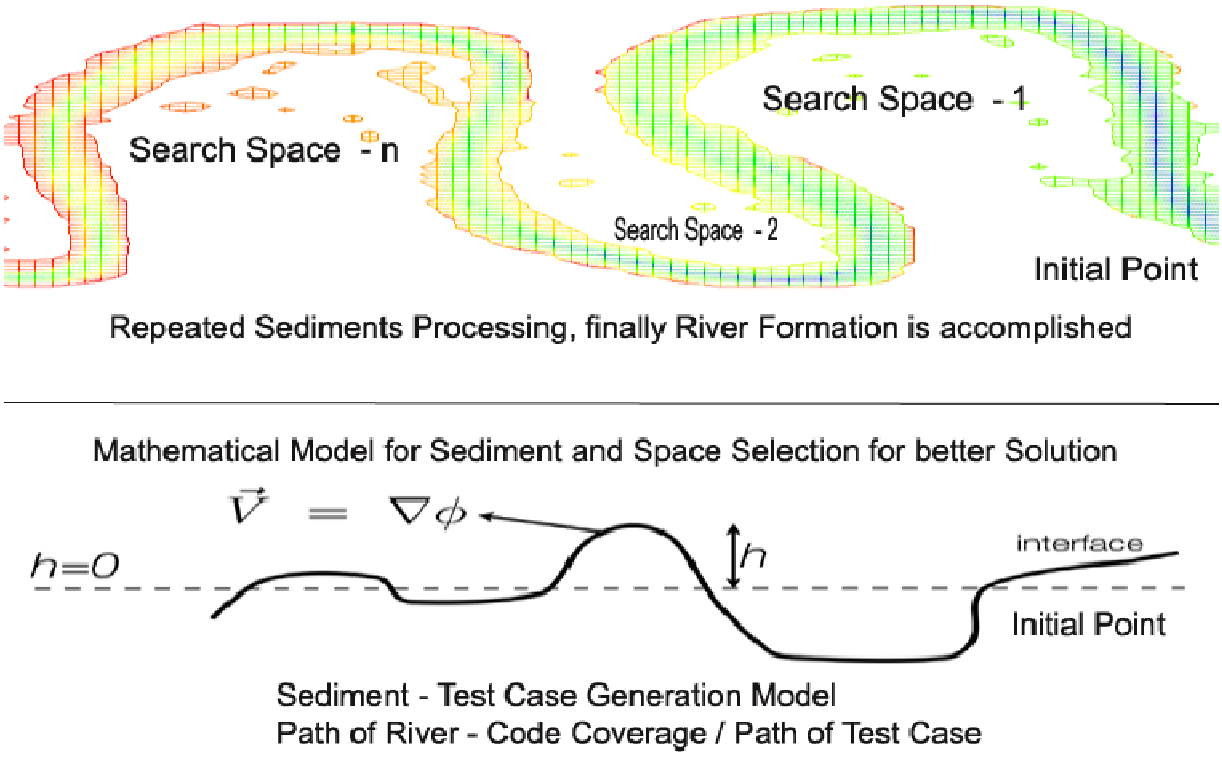

Figure 1 – Diagrammatic Representation of River Formation Dynamics

Figure 1 depicts the flow of proposed model for finding the optimal solution. Initially, there is a huge search space which is required to be shortened and refined after number of iterations. In

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

this proposed approach, the sediments are associated with the related parameters which are mandatory for the test case generation. Once the best fit sediments are linked with the test case in flow, the better and effective results can be retrieved.

In the mathematical mode, the notations are specified as follows –
h - Region of Finding the Best Fit Sediments (Parameters) from River (Global Search Space)
Δ - Diversion or Change in the Parameter Aspects using Fuzzy Mathematical Model. It is implemented to avoid the biasing and to improve the transparency.
ϕ - This variable parameter is directly proportional to Δ. It means the value will change on the basis of changes on other parameter. This approach is used to maintain the consistency in the algorithmic approach.

V. PERFORMANCE ASPECTS OF GREEDY METHOD

Greedy algorithms don't generally yield the truly ideal solutions. In such cases the greedy strategy is as often as possible the premise of a heuristic methodology. Notwithstanding for issues which can be explained precisely by a greedy algorithm, building up the strategy's accuracy may be a non-trifling procedure. It might frequently return off base results. It frequently requires next to no investment to do as such

Classical Greedy Approach	Proposed Metaheuristic / Nature Inspired Algorithmic Approach
The solution space is limited	Unlimited and foreseen solution space can be processed.
Local optimal solution is achieved	Global optima or Best Fit Optimal Solution is achieved
The search space is very less and restricted because of the currently analyzed domain	The search space is huge and in some cases can be infinite. By this approach, the best solution

	is found based on the fitness value
The implementation is non-fuzzy based.	Fuzzy modeling based implementation can be done. This methodology improves the results based on the dynamic parameters
The greedy solution in classical aspects are prone to failure because of non acceptability score in the solution set	The failure rate or non-acceptance rate is not in this approach. This scenario make the solution more robust and predictable.

VI. REASON OF FAILURE OF GREEDY ALGORITHMS

For many other problems, greedy algorithms fail to produce the optimal solution, and may even produce the unique worst possible solutions. One example is the nearest neighbor algorithm mentioned above: for each number of cities there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour.

Greedy algorithms mostly (but not always) fail to find the globally optimal solution, because they usually do not operate exhaustively on all the data. They can make commitments to certain choices too early which prevent them from finding the best overall solution later. For example, all known greedy algorithms for the graph coloring problem and all other NP-complete problems do not consistently find optimum solutions. Nevertheless, they are useful because they are quick to think up and often give good approximations to the optimum.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like dynamic programming.

International Journal of Computing and Corporate Research
ISSN (Online) : 2249-054X
Volume 5 Issue 6 November 2015
International Manuscript ID : 2249054XV5I6112015-01

Examples of such greedy algorithms are Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees, Dijkstra's algorithm for finding single-source shortest paths, and the algorithm for finding optimum Huffman trees. Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighboring node which is "closest" to the destination. The notion of a node's location (and hence "closeness") may be determined by its physical location, as in geographic routing used by ad-hoc networks. Greedy algorithms generally fail to find the globally optimal solution, as they usually do not operate profoundly on all the data. They can make commitments to certain choices too early which avoid them from finding the best solution later.

VII. LIMITATIONS OF USING HEURISTICS

1. In many cases, convergence is generally guaranteed
2. Optimality may be achieved but it is not proved
3. In many cases, they may not be able to generate a feasible solution.

Metaheuristics are said to be high level procedures which coordinate simple heuristics such as local search, to find solutions that are of better quality than those found by simple heuristics done.

VIII. PROMINENT METAHEURISTIC APPROACHES

- Tabu search [Glover, 89 et 90]
- Simulated Annealing [Kirckpatrick, 83]
- Threshold accepting [Deuck, Scheuer, 90]
- Variable neighborhood [Hansen, Mladenovi´c, 98]
- Iterated local search [Loren¸co et al, 2000]
- Genetic Algorithm, Holland 1975 [Goldberg 1989]
- Memetic Algorithm [Moscatto 1989]

- Ant Colony Optimization [Dorigo 1991]
- Scatter search, Laguna, Glover [Marty 2000]

CONCLUSION

In this paper work, the empirical and pragmatic review on the software test cases generation approaches is addressed with their relative performance and algorithmic approaches. The proposed work in this research work underlines the use and importance of metaheuristic approaches so that the overall or global optimization can be met on multiple parameters. The solutions obtained as "Optimal Solutions" is in many cases are not as per the requirements. The goal and objective of using a metaheuristics approach is to produce efficient solutions. The metaheuristics such as Simulated Annealing is one of the most popular approaches to explore in the field of optimization and it will bring wonder to the world of algorithms in future.

REFERENCES

[1] Kuhn, D. R., Wallace, D. R., & Gallo Jr, A. M. (2004). Software fault interactions and implications for software testing. *Software Engineering, IEEE Transactions on*, *30*(6), 418-421.

[2] Korel, B. (1990). Automated software test data generation. *Software Engineering, IEEE Transactions on*, *16*(8), 870-879.

[3] McMinn, P. (2011, March). Search-based software testing: Past, present and future. In *Software testing, verification and validation workshops (icstw), 2011 ieee fourth international conference on* (pp. 153-163). IEEE.

[4] Arcuri, A., & Fraser, G. (2011). On parameter tuning in search based software engineering. In *Search Based Software Engineering* (pp. 33-47). Springer Berlin Heidelberg.

[5] Harman, M., McMinn, P., De Souza, J. T., & Yoo, S. (2012). Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical Software Engineering and Verification* (pp. 1-59). Springer Berlin Heidelberg.

[6] Fraser, G., & Arcuri, A. (2012, April). The seed is strong: Seeding strategies in search-based software testing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 121-130). IEEE.

[7] Anand, S., Burke, E. K., Chen, T. Y., Clark, J., Cohen, M. B., Grieskamp, W., ... & McMinn, P. (2013). An orchestrated survey of methodologies for automated software test case generation. *Journal of Systems and Software*,*86*(8), 1978-2001.

[8] Galeotti, J. P., Fraser, G., & Arcuri, A. (2013, November). Improving search-based test suite generation with dynamic symbolic execution. In *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*(pp. 360-369). IEEE.

[9] Arcuri, A., & Briand, L. (2014). A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing, Verification and Reliability*, *24*(3), 219-250.

[10] Sánchez, A. B., Segura, S., & Ruiz-Cortés, A. (2014, March). A comparison of test case prioritization criteria for software product lines. In *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on* (pp. 41-50). IEEE.

[11] Bareiša, E., Jusas, V., Motiejūnas, K., & Šeinauskas, R. (2015). The use of a software prototype for verification test generation. *Information Technology and Control*, *37*(4).

[12] Shahbaz, M., McMinn, P., & Stevenson, M. (2015). Automatic generation of valid and invalid test data for string validation routines using web searches and regular expressions. *Science of Computer Programming*, *97*, 405-425.

[13] Dizaji, Z. A., & Gharehchopogh, F. S. (2015). A hybrid of ant colony optimization and chaos optimization algorithms approach for software cost estimation. *Indian Journal of Science and Technology*, *8*(2), 128-133.