Review of
Applications of combining Search-based Test Creation and Execution mechanism with other techniques.

Nikhil Anand
North Carolina State University
Raleigh, North Carolina
nanand2@ncsu.edu

Nikhil Satish Pai
North Carolina State University
Raleigh, North Carolina
npai@ncsu.edu

ABSTRACT:

In this report we review the applications and advantages of Integrating search based testing techniques with other testing techniques and mechanisms like Constraint based testing, Evolutionary testing and symbolic executions. We have reviewed various highly cited literature proceedings published in this domain over the last decade and have tried to analyze and elucidate the need and advantages of improving search-based testing techniques, be that test case, data generation or test execution, by integrating various approaches to improve the overall code coverage.

Keywords:

Constraint Based Testing, Search Based Testing, Symbolic Execution, Search based software engineering.

1. INTRODUCTION

Software engineering involves problems associated with the competing constraints and tradeoffs in requirements. Software engineering techniques are concerned with generating near optimal solutions or solutions within an accepted tolerance making meta-heuristic search based optimization techniques applicable.

Search Based software Engineering is a technique wherein the software engineering problem is mapped onto a search problem. Since the problem is treated as a search problem, search techniques are used to solve the given problem. Some of the advantages offered by this technique are large search traversal, low computational complexity and the absence of known optimal algorithms. The field focuses on moving software engineering problems from human to computer based search making the human effort focus on guiding the search than performing the search.

Search Based software techniques are used in multiple stages of software engineering like generation of automatic test cases, Bounded Verification, Reconstruction of Core dumps, testing interactive GUI application etc. There are different algorithms developed to traverse the search space of the problem to obtain an optimal solution. We have compared and contrasted some of the highly cited papers

from the Automated software engineering literature and tracked the evolution of these algorithms.

This paper reviews some of the highly cited literature proceedings published in Search based software engineering (SBSE) domain over the last decade and have tried to analyze and elucidate the need and advantages of improving search-based testing techniques. We mainly focus on how inclusion of some other techniques with SBSE testing improves the overall test coverage or performance. In each section below we have tried to draw out that papers proclamation and added our views to those.

2. MOTIVATION

Search-Based testing techniques have various strong and flexible attributes which makes it highly sought after technique for testing in today's automated software engineering world. It is highly scaleable and can handle any code and test criterion, provided the search is appropriately guided, thus, making this technique highly dependent on the search heuristics. In these scenarios the search reaches optimal solutions at random and we don't always get the best possible coverage. This approach has its strength and weakness, these weaknesses could be complemented by other testing techniques. For example take constraint-based testing which uses the modern constraint solvers which are not dependent on search heuristics. Another suggested approach could be integrating user-feedback into the genetic algorithm applied to generate tests and provide guidance to search to move towards the optimal path.

We believe a fusion of different techniques would be a great fit for the ever-evolving field of automated software engineering. Providing the best of all the test mechanisms available to a given application based on its requirements and needs would help improving the test code coverage by a great deal and making it less prone to bugs. Such an amalgamation of technique would not only help in generation of a better test input data but also help in early detection of defects, thus, improving the quality of the software product being developed.

3. SAMPLING PROCEDURE

We started with papers published in 2011 from the automated software engineering literature which were highly cited. The methodology was to move backwards to get a historical context and move forwards to get a context of the newer developments in the field.

We were interested in the field of automated test generators hence we chose the paper "Combining Search-Based and Constraint based testing" by Jan Malburg and Gordon Fraser. Starting with four papers cited by these authors we moved backwards to get a context on the growth of the problem. We next reviewed three other papers from the future which cited the initial paper. This gave us an idea of how the problem evolved.

This paper presents a high level summary of all the reviewed publications.

4. LITERATURE SURVEY AND KEY IDEAS

Most of today's automated test generators are based on meta-heuristic algorithms or constraint solvers. Both these approaches have their advantages and drawbacks. We have outlined some of the techniques below:

Search Based Testing:

Search based techniques use meta-heuristic search techniques like Genetic Algorithms to automate testing tasks like test case generation, achieving high structural coverage etc.

The dominating operation in the optimization process is the fitness function which guides the search towards the best solution (heaven) from an exponentially large search space.

The simplest algorithms in search based testing are genetic algorithms where a initial population set is evolved towards satisfying any chosen coverage criterion. The population is mutated in each iteration.

Search based techniques are effective in handling any code and test criterion and scale well with the complexity of the problem. The success of the search based testing depends on the guidance provided; if there are local minima then the algorithm may get stuck here not producing a globally optimal solution. A second problem will depend on the landscape of the problem if all neighbors have the same fitness level and there will be no guidance but if the heuristic does not provide sufficient guidance like in the case of the flag problem.

These problems were highlighted in paper [1] and a new method integrating search based and constraint based testing was proposed. A Genetic Algorithm is used in mutating the solutions and instead of the traditional mutations, boolean path conditions are considered which help in achieving higher branch coverage in-turn increasing the efficiency of the search. This integrated approach is easier to implement on existing methods. The method has been tested on Java Path Finder.

Constraint Based Testing:

Constraint Based Testing involves generating test cases for programs by reducing the problem to verifying if a particular constraint in the system is satisfiable or not. For example if a particular branch of the code is reachable or not.

Constraint based problems are used to solve constraints generated by symbolic execution. Static symbolic execution involves extracting all the constraints along a given branch of the code and using a constraint solver to determine the solution to these constraints.

Jam Malburg and Gordon Fraser in their paper "Combining Search Based and Constraint Based testing" note that constraint based testing will be able to solve a very few

set of problem domains efficiently. Hence proposes a Genetic Algorithm mutation operator which consider branch path conditions and negate conditions to achieve better coverage.

R. Sharma et al in their paper involve construction of complex data structures by comparing constraint based and sequence based approaches. In a sequence based test generation, the final output is produced by a sequence of operations applied on a initial state. The results showed that constraint based testing would help in generating more complex data structures like Red Black Tree, AVL Trees etc.

K. Inkumsah and T. Xie in their paper proposed a framework called Evacon to integrate evolutionary testing and symbolic execution to solve for complex arguments like those present in Object oriented programs. This does not solve for nested predicates.

M. Staats and C. S. Pasareanu propose a new methodology to speedup symbolic execution. Symbolic execution algorithms do not scale well as the complexity of the programs increase. The algorithm proposes simple static partitioning of the symbolic execution tree for parallelizing symbolic execution allowing for distribution of symbolic execution and decrease in the time needed to process the tree.

Dynamic Symbolic Execution:

Dynamic symbolic execution is a constraint based testing technique where the search is initialized with a random value and as the traversal of the code is done branching conditions are evaluated and state updated when their values are changed. One of these constraints is negated to determine a new unexplored path. When the values cannot be handled symbolically, concrete values are used. This is also called concolic testing.

Kevin Salvesen et al. use dynamic symbolic execution for generating test case inputs on Graphical User Interfaces. Traditional search based techniques rely on randomized values for text boxes to generate a complex series of events for testing but since the user inputs specific values DSE is well suited for this. The paper presents a novel approach where in search based algorithm is used for the generation of complex input event sequences and DSE is used to generate the inputs.

Semi Automatic Search Based Testing:

Search based software engineering involves reducing the human effort in testing by using meta heuristic algorithms. But in cases where the fitness functions do not provide high coverage the results generated will not be optimal. Paper [6] defines a hybrid approach wherein a human tester will help in providing guidance to a search algorithm when the fitness function gets stagnated in a local minima. This technique is applied to object oriented programs where the regular branch traversal is not sufficient to test all the

complex system of method calls which are possible.

The algorithm begins as regular Search based technique which mutates an initial random solution to generate newer children. If the children produced after evolution are better, then the algorithm continues. But if the algorithm is stuck in a local minima or maxima a external user input is considered to proceed further with the mutation.
<<IMAge ???>>

The drawback is that the tester must know the code base well and must be experienced
in testing, also because of the human element the modification done may worsen the output.

5. RELATED WORK

In this Section we list out some other related literature proceedings and tools which were also referenced with the main the paper to improve the understanding regarding the topic in question.

An existing evolutionary tool was combined with a Dynamic Symbolic Execution by Inkumsah and Xie. Lakhotia et al. worked on combining floating points constraints with search based techniques; but this approach has been generalized for other constraints in this paper. Majumdar and Sen interleaved DSE with random search; however in this combination only one technique is used at a time.

Generation of complex data inputs for better coverage using symbolic execution was introduced by Khurshid et al. Extension of the aforementioned work was done by Pasareanu where they worked on model checking (using JPF) and abstraction on container data structures. d'Amorim et al. compared random generation and symbolic execution.

Khurshid et al. introduced the Alloy Annotation Language to propose a translation similar to this paper. Vaziri et al. propose a set of rules to be applied along the translation to a SATformula in order to profit from properties of functional relations. Saturn is also a SATbased static analysis tool for C. It uses as its main techniques a slicing algorithm and function summaries.

In order to generate test data cheaply, randomized processes are used by DART ,Randoop and Jartege. Search based test generation uses genetic algorithms to find both method arguments and method sequences for the program under test but suffers from the path problem. This paper addresses the path problem by means of leveraging symbolic execution and constraint solving.

Xiao et al talks about approach to report problems that a testing tool encounters during test generation to the user. They call out external method calls and object creation as the main problems in test generation based on dynamic symbolic execution.
Fraser and Arcuri talks about a study on an unbiased sample pointing that in practice environmental dependencies like file access are a major inhibitor towards high code coverage and object creation and complex string-based calculations can also be for search-based testing.

BARAD is a GUI testing framework for applications written in Java with the Standard

Widget Toolkit (SWT). GAZOO is a fully automated GUI testing tool for NET applications. Both rely on DSE to find input data for event sequences. COLLIDER targets Android applications. It uses concolic execution to build symbolic summaries of the GUI's event handlers.

6. COMMENTARY

The following are the tools which were either used or referenced for analysis in the surveyed literature:

Evosuite: Is a tool to automatically generate unit tests for Object Oriented Programming languages like Java. It utilizes a hybrid approach to generate and optimize test suites to satisfy a coverage criteria.

TACO: Stands for Translation of Annotated Code, an open source program analysis tool to verify the compliance of a Java program to the Java Modelling Language specification. The verification done is bounded by a user specified input.

JPF: Java Pathfinder is an explicit state software model checker for Java bytecode. It is a Java virtual machine which executes the program in all theoretically possible ways checking for property violations like deadlocks etc along all possible execution paths. The entire path to the error is reported.

jCUTE: Java Concolic Unit testing engine. It automatically generates unit test cases for Java. The execution combines symbolic execution and constraint solving to traverse the entire code.

Korat: Tool to generate test inputs for automated testing of Java programs. The inputs are structural (like linked lists) and must satisfy complex invariants specified for the data structure.

EXSYST: Test generation tool designed for user interfaces and just focus on the source code.

DataSet used for SF100 Benchmark were randomly selected top 100 projects from SourceForge to provide a well defined statistical benchmark for open source projects

7. AREA OF IMPROVEMENT

Yury Pavlov and Gordon Fraser in their paper use the SF100 benchmark dataset, in most papers reviewed, specially the ones where object oriented programs were tested, feasibility of the algorithms suggested is based on empirical results drawn from a limited data set which may be biased, In most cases this was highlighted in the threats to validity section. There is a need to perform testing on large benchmarks like SF100 to validate the working of the algorithm.

There are multiple experiments where "magic weights" were used to set values or parameters, exhaustive testing will be needed to determine the best values for these parameters.

Most of the testing was done on datasets in a controlled testing environment, more real world scenarios need to be considered while testing the approach.

8.CONCLUSION AND FUTURE WORK

We have studied the growth of algorithms over a period of about 8 years, in the beginning where all the algorithms were individually applied on the data sets to a multiple papers where hybrid algorithms which were combinations of different ones like symbolic execution, search based testing etc.

We have also seen the evolution of techniques used on traditional java programs to generate test cases for interactive GUI based applications. Traditionally the focus has been on completely automating the process of software testing but in some cases human intervention is done to provide a better guidance. More research is needed in these areas to further our understanding.

Matt Staats et al use Parallel architectures in scaling symbolic execution algorithms for automated test generators, with newer advances in hardware technologies like multi-core processors and distributed systems the process of solving heuristics becomes more challenging, more research is needed in this field to utilize these technologies.

Most of the algorithms focused only on Java; further testing is needed in other programming languages like C, C++ etc which are primarily used on the systems side. There are multiple challenges faced by Operating systems which have a very high complexity wherein meta heuristic techniques can be applied.

Current studies are performed considering on parameters like branch coverage etc, the feedback from the users is not being considered in the testing process. Newer techniques need to be evolved; for the same Jeremias Rößler suggests one such methodology in his paper.

With the trend of research in all the reviewed paper, we could conclude that there are few aspects of software testing that would require more attention in future research. Like working towards improving the technique for code coverage, automating bug detection process process and incorporating the SBE as a core concept of the software development life cycle.

9. ACKNOWLEDGEMENT

We thank Prof. Timothy Menzies for giving us an opportunity to enhance our knowledge in Artificial Intelligence and its applications in automated software engineering.

10. REFERENCE

1. Jan Malburg and Gordon Fraser. 2011. *Combining search-based and constraint-based testing*. In Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE '11). IEEE Computer Society,

Washington, DC, USA, 436-439.DOI=10.1109/ASE.2011.6100092

2. R. Sharma, M. Gligoric, V. Jagannath, and D. Marinov, "*A Comparison of Constraint-Based and Sequence-Based Generation of Complex Input Data Structures*," 2nd Workshop on Constraints in Software Testing, Verification and Analysis, Apr. 2010.
3. Juan Pablo Galeotti, Nicolas Rosner, Carlos G. L. Pombo, and Marcelo F. Frias. 2010. *Analysis of invariants for efficient bounded verification*. In Proceedings of the ACM International Symposium on Software Testing and Analysis (ISSTA'10). 25–36.
4. K. Inkumsah and T. Xie, *"Improving Structural Testing of Object Oriented Programs via Integrating Evolutionary Testing and Symbolic Execution,"* Proc. IEEE/ACM Int'l Conf.Automated Software Eng., pp. 297-306, 2008
5. M. Staats and C. S. Pasareanu. *Parallel symbolic execution for structural test generation.* In Proceedings of the Nineteenth International Symposium on Software Testing and Analysis (ISSTA 2010), pages 183–194, 2010.
6. Y. Pavlov and G. Fraser, "*Semi-automatic Search-Based Test Generation,*" in 5th International Workshop on Search-Based Software Testing (SBST'12) at ICST'12, 2012, pp. 777-784.
7. Jeremias Rossler, Andreas Zeller, Gordon Fraser, Cristian Zamfir, George Candea, *"Reconstructing Core Dumps*", ICST , 2013, 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST 2013), 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation (ICST 2013) 2013, pp. 114123, doi:10.1109/ICST.2013.18.
8. Kevin Salvesen, Juan P. Galeotti, Florian Gross, Gordon Fraser, and Andreas Zeller. 2015. *Using dynamic symbolic execution to generate inputs in search-based GUI testing*. In Proceedings of the Eighth International Workshop on Search-Based Software Testing (SBST '15). IEEE Press, Piscataway, NJ, USA,32-35.
9. M. Harman, *"The Current State and Future of Search Based Software Engineering,"* Proc. Int',l Conf. Future of Software Eng. 2007, pp. 342-357, 2007.
10. P. McMinn, *"Search-Based Software Testing: Past, Present and Future,"* The 4th International Workshop on Search-Based Software Testing (SBST'11), in conjunction with the 4th IEEE International Conference on Software Testing (ICST'11), 2011, pp. 153-16
11. S. Bardin, B. Botella, F. Dadeau, F. Charreteur, A. Gotlieb, B. Marre, C. Michel, M. Rueher, and N. Williams, "*Constraint-based software testing,"* Journée du GDR-GPL, vol. 9, 2009.
12. M. Harman, L. Hu, R. Hierons, J. Wegener, H. Sthamer, A. Baresel and M. Roper, *"Testability transformation,"* IEEE Trans. Softw. Eng., vol. 30, pp. 3–16, January 2004.
13. Gordon Fraser and Andrea Arcuri. 2011. *EvoSuite: automatic test suite generation for object-oriented software.* In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering (ESEC/FSE '11). ACM, New York, NY, USA, 416-419. DOI=http://dx.doi.org.prox.lib.ncsu.edu/10.1145/2025113.2025179
14. Jeremias Rößler. 2012. How helpful are automated debugging tools?. In *Proceedings of the First International Workshop on User Evaluation for Software Engineering Researchers* (USER '12). IEEE Press, Piscataway, NJ, USA, 13-16.
15. K. Inkumsah and T. Xie, *"Improving structural testing of object-oriented programs via integrating evolutionary testing and symbolic execution,"* in *Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE'08)*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 297–306.
16. K. Lakhotia, N. Tillmann, M. Harman, and J. de Halleux, *"FloPSy - search-based floating point constraint solving for symbolic execution,"* in *22nd IFIP International Conference on Testing Software and Systems*, ser. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2010, pp. 142–157.
17. R. Majumdar and K. Sen, *"Hybrid concolic testing,"* in Proceedings of the 29th International Conference on Software Engineering (ICSE'07). Washington, DC, USA: IEEE Computer Society, 2007, pp. 416–426.
18. S. Khurshid, C. Pasareanu, and W. Visser, "*Generalized symbolic execution for model checking and testing,*" in Proc. 9th International Conference on Tools and Algorithms for

Construction and Analysis of Systems (TACAS), 2003.

19. W. Visser, C. S. Pasareanu, and R. Pelanek, "*Test input generation for red-black trees using abstraction.*" in Proc. 20th International Conference on Automated Software Engineering (ASE), 2005
20. M. d'Amorim, C. Pacheco, T. Xie, D. Marinov, and M. D. Ernst, "*An empirical comparison of automated generation and classification techniques for object-oriented unit testing,*" in Proc. 21st International Conference on Automated Software Engineering (ASE), 2006.
21. Khurshid, S., Marinov, D., Jackson, D., *An analyzable annotation language*. In OOPSLA 2002, pp. 231245
22. Vaziri, M., Jackson, D., *Checking Properties of HeapManipulating Procedures with a Constraint Solver*, in TACAS 2003, pp. 505520
23. Xie, Y., Aiken, A., *Saturn: A scalable framework for error detection using Boolean satisfiability*. in ACM TOPLAS, 29(3): (2007).
24. P. Godefroid, N. Klarlund, and K. Sen, "*DART: Directed automated random testing,*" in Proc. ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, 2005, pp. 213–223
25. C. Pacheco and M. D. Ernst, "*Randoop: Feedback-directed random testing for Java*," in Companion to ACM SIGPLAN Conference on Object Oriented Programming Systems and Applications Companion, 2007, pp. 815–816.
26. C. Oriat, "*Jartege: A tool for random generation of unit tests for java classes,"* .
27. X. Xiao, T. Xie, N. Tillmann, and J. de Halleux. *Precise identification of problems for structural test generation.* In Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, pages 611–620, New York, NY, USA, 2011. ACM.
28. G. Fraser and A. Arcuri. *Sound empirical evidence in software testing.* In ACM/IEEE International Conference on Software Engineering (ICSE), 2012. To appear.