Advances in Computer Science and Information Technology (ACSIT)
Print ISSN: 2393-9907; Online ISSN: 2393-9915; Volume 1, Number 2; November, 2014 pp. 61-64

http://www.krishisanskriti.org/acsit.html

Applications of Meta Heuristic Search Algorithms in Software Testing: An Investigation into Recent Trends

Abhishek Pandey[1], Dr. Soumya Banerjee[2] and Dr. G. Sahoo[3]

[1]Assistant professor, UPES, Dehradun
[2]CSE Associate Professor, BIT Mesra Ranchi
[3]CSE Professor, CSE BIT Mesra Ranchi
E-mail: [1]apandey@ddn.upes.ac.in, [2]soumyabanerjee@bitmesra.ac.in, [3]gsahoo@bitmesra.ac.in

Abstract: *Software engineering is concerned to develop software that is economical as well as efficient in its applications. This is a sub discipline of computer science and engineering. There are various phases in software development life cycle. Software testing constitutes of around 50% of the total cost of the software development. At the same time the branch or decision coverage has to be maximized. This paper gives an insight into the recent trends of the applications of search based techniques to generate Applications of meta heuristic techniques in the field of software testing phases originates a new field of research known as the search based software testing The sub field of software engineering that is concerned with the application of search based techniques into the field of software engineering is known as search based software engineering(SBSE).*

Keywords: *Search based software engineering, meta heuristic techniques, software testing*

1. INTRODUCTION

Automatic structural testing and test data generation has been the area of interest in the research community since 1970s. In this decade two approaches of the problem emerged. The first one came to be known as symbolic execution [1] and the method that reformulated the problem as an optimization problem, which latter known as search based approach. For a give path through a program, symbolic execution involves the construction of the path condition. The path condition is the set of constraints in terms of the input variables. It describes when the selected paths executed. The path conditions can be set of linear constraints or nonlinear constraints or the combinations of both. In case of nonlinear constraints it becomes very complex tasks to solve the constraints. Latter in the concolictesting[2] some problems associated with symbolic execution has been alleviated combining both symbolic execution and concrete testing.The application of evolutionary algorithms to test data generation is often referred to as evolutionary testing in the literature .the first reported work on this topic is that of the Xanthakis et al [3]. Up until this point, work on structural test data generation had largely focused on finding input data for specific paths or individual structures with programs, such as branches or statements. Initially, however, techniques using Genetic Algorithms took slightly different ways. Different techniques applying Evolutionary Algorithms to structural test data generation can be categorized on the basis of objective function construction. Coverage-Oriented Approaches reward individuals on the basis of covered program structures. In the work of Roper [4], an individual is rewarded on the basis of the number of structures executed in accordance with the coverage criterion. Under this scheme, however, the search tends to reward individuals that execute the longest paths through the test object . Guidance is not given for structures that are unlikely to be covered by chance, for example deeplynestedstructures, or branch predicates that are only true when an input variable has to be a specific value from a large domain.

2. LITERATURE REVIEW

The work of Watkins [5] attempts to obtain full path coverage for programs. The objective function penalises individuals that follow already covered paths, by assigning a value that is the inverse of the number of times the path has already been executed during the search. The direction of the search, therefore, is under constant adaptation. However, the penalizations of covered paths, in itself, provides little guidance to the discovery of new, previously unfound paths. The results show that in comparison with Random Testing, the Genetic Algorithm approach required an order of magnitude fewer tests to achieve path coverage for two experimental programs. However, both of these programs are of a simple nature, containing no loops. Furthermore, the input domains were artificially restricted for the search. In general, the problem with coverage-oriented approaches is the lack of

guidance provided for structures which are only executed with values from a small portion of the overall input domain. Therefore, it is difficult to expect full coverage with these techniques for any non-trivial program.Structure-Oriented Approaches follow similar lines to the earlier work of Korel,and take a `divide and conquer' approach to obtaining full coverage. A separate search is undertaken for each uncovered structure required by the coverage criterion. Structure-oriented techniques differ in the type of information used by the objective function. These can be categorised as either Branch- Distance-Oriented, Control-Oriented, or Combined approaches.Branch-Distance-Oriented approaches exploit information from branch predicates, in a similar style to earlier work by Miller and Spooner, and later Korel. In the work of Xanthakis et al. [3], Genetic Algorithms are employed to generate test data for structures not covered by random search. A path is chosen, and the relevant branch predicates are extracted from the program. The Genetic Algorithm is then used to find input data that satisfies all the branch predicates at once, with the objective function summing branch distance values. However, this scheme suffers from similar problems suffered by the work of Miller and Spooner. Furthermore, the need to select a path is a burden on the tester. Inthe work of Jones et al. [6] for obtaining branch coverage, a path does not need to be selected. The objective function is simply formed from the branch distance of the required branch. However, no guidance is provided so that the branch is actually reached within the program structure in the first place. McGraw et al. [7] alleviate this problem for condition coverage, by delaying an attempt to satisfy a condition within a branching expression until previous individuals have been already found which reach the branching node in question. The initial generation for the target condition is then seeded with these individuals. Thisscheme, however, is inefficient if test data is required for the coverage of one, specific condition. The earlier work of Korel had already removed the need for the tester to select a path. Since new test data considered by the search had to conform to the successful sub-path already found, explicit control-oriented information regarding the target did not need to be included in the objective function. However, such rigid constraints increase the chances of the search becoming via the objective function. This is the problem addressed by Control-Oriented approaches. With Control-Oriented approaches, the objective function considers the branching nodes that need to be executed in some desired way in order to bring about execution of the desired structure. The approach of Jones et al. [11] to loop testing falls into this category. Here, the objective function is simply the difference between the actual and desired number of iterations. In the work of Pargas et al. [13], for statement and branch coverage, the control dependence graph of the test object is used. The sequence of control dependent nodes is identified for each structure. These are the branching nodes that must be executed with a specific outcome in order for the structure to be reached. The objective value of an individual is simply assigned as the number of control dependent nodes executed as intended. Recall that the branch leading away from the target at a control dependent node is identified as a critical branch in Korel's work. The measure used by Pargas et al. is therefore equivalent to the number of critical branches successfully avoided by the individual.

3. APPLICATIONS OF METAHEURISTIC SEARCH ALGORITHMS IN SOFTWARE TESTING

In the work of Watkins[10], Roper[9], Pargas et.al[13] the fitness of an individual is determined on the basis of the coverage measured of the associated test data. i.e test data set that covered more program branches than others are assigned more fitness values. Whereas Watkins,Roper measure the coverage acquired by the test datum on the basis of the control flow graph. Pargas et al.use the control dependence graph of the test object for this purpose. One drawback of their approach is exclusive use of coverage as fitness criteria. Therefore search will mainly be directed at the execution of a few long and easily accessible program paths therefore making it more difficult to attain the complete coverage. Different objective functions in different approaches:

The work of Tracey [7] builds on previous work which used Simulated Annealing. The strategy for combining both techniques is as follows. The control dependent nodes for the target structure are identified. If an individual takes a critical branch from one of these nodes, a distance calculation is performed using the branch predicate of the required, alternative branch. This is computed using the functions of Table (and Table for and and or logical connectives). Tracey then uses the number of successfully executed control dependent nodes to scale branch distance values. Let branch dist be the branch distance calculation performed at the branching node where a critical branch was taken.

The formula used by Tracey for computing the objective function is:

Executed/ dependent × br _distance

Wegener et al. [8] map branch distance values branch dist logarithmically into the range [0, 1] (call this m_branchdist). The minimizing objective function is zero if the target structure is executed, otherwise, the objective value is computed as:

(dependent -executed - 1) +m_ branch dist

The (dependent –executed-1) sub-calculation is referred to as the approximation level or perhaps more appropriately, the approach level attained by the individual.

In simple form the objective value can be calculated as the following:

Approach level + branch distance

4. SEARCH BASED TESTING TOOLS

Austin and cute testing tools [9] are based on the search based and symbolic execution based testing tools respectively. Concolic testing[10] originates in the seminal work of Godefroid et al. on Directed Random Testing . It formulates the test data generation problem as one of finding a solution to a constraint satisfaction problem, the constraints of which are produced by concolic execution of the program under test. Concolicexecution[2] combines symbolic [1] and concrete execution. Concrete execution drives the symbolic exploration of a program, and dynamic variable values obtained by real program execution can be used to simplify path constraints produced by symbolic execution. Search based testing [11] formulates the test data adequacy criteria as objective functions, which can be optimized using Search Based Software Engineering [12]. The search space is the space of possible inputs to the program under test. The objective function captures the particular test adequacy criterion of interest. Concolic testing builds on the ideas of symbolic execution. For a given path through a program, symbolic execution involves constructing a path condition; a system of constraints in terms of the input variables that describe when the path will be executed. The path condition can easily become unsolvable, however, if it contains expressions that cannot be handled by constraint solvers. This is often the case with floating-point variables,or non-linear constraints. Concolic testing can alleviate some of the problems of non-linearity by combining concrete execution with symbolic execution. The idea is to simplify the path condition by substituting sub-expressions with concrete values, obtained by actual dynamic executions. This substitution process can remove some of the non-linear sub-expressions in a path condition making them amenable to a constraint solver. The term concolic was coined by Sen et al. [2] in their work introducing the CUTE tool, which is based upon similar principles.CUTE attempts to execute all feasible program paths, using a depth-first strategy. The first path executed is that, which is traversed with all zero or random inputs as described above.

The next path to be attempted is the previous path, but taking the alternative branch at the last decision statement executed in the path. The new path condition is therefore the same

as the previous path condition, but with the last constraint negated, allowing for substitution of sub-expressions in the new path condition with sensible concrete values (as in the example above). For programs with unbounded loops, CUTE may keep unfolding the body of the loop infinitely many times, as there may be an infinite number of paths. The CUTE tool is therefore equipped with a parameter which places a limit on the depth of the depth-first path unfolding strategy.AUSTIN is a tool for generating branch adequate test data for C programs. AUSTIN does not attempt to execute specific paths in order to cover a target branch; the path taken up to a branch is an emergent property of the search process. The objective function used by AUSTIN was introduced by Wegener et al. [13] for the Daimler Evolutionary Testing System. It evaluates an input against a target branch using two metrics; the approach level and the branch distance. The approach level records how many nodes on which the branch is control dependent, were not executed by a particular input. The fewer control dependent nodes executed, the ‘further away’ the input is from executing the branch in control flow terms. The branch distance is computed using the condition of the decision statement at which the flow of control diverted away from the current ‘target’ branch.

5. MODELLING THE TEST PROBLEM AS A MULTI OBJECTIVE PROBLEM

The multi objective approach to test data generation problem[14] emerged recently which aims to reformulate the test data generation problem as a multi objective optimization problem. Recent advances in multi objective optimization is clear success toward this scheme. More recently the test data generation problem has been attacked with the memeticalgorithms[15,16]. A Memetic Algorithm (MA) hybridizes global and local search. The use of MAs for test generation was originally proposed by Wang and Jeng in the context of test generation for procedural code. Arcuri [17] combined a GA with hill climbing to form a MA when generating unit tests for container classes. Harman and McMinn [18] analyzed the effects of global and local search, and concluded that MAs achieve better performance than global search and local search. Baresi et al. [19] also use a hybrid evolutionary search in their TestFul test generation tool, where at the global search level a single test case aims to maximize coverage, while at the local search level the optimization targets individual branch conditions.

6. IDENTIFIED VARIOUS RESEARCH AREAS UNDER SEARCH BASED SOFTWARE TESTING

Various methods applying in the software testing problem has been demonstrated using small programs. It can be scaled to large programs also. recent developments in the field of genetic algorithm and multi objective optimization algorithms can be utilized to search based testing. Constraint satisfaction problems is another research area under the search based testing.

Various hybrid approaches along with the data mining tools can also be utilized in search based testing.

7. CONCLUSION

In this paper we have tried to bring almost all the developments in the field of search based testing. Also we

have identified some key areas of research were these approaches can be utilized. We are also working on some new models of testing were we will use program slicing and applying multi objective optimization algorithms to solve the test data generation problems as a completely new way.

REFERENCES

[1] J. C. King, Symbolic execution and program testing. Communications of the ACM, 19(7):385–394, July 1976

[2] K. Sen, D. Marinov, and G. Agha. CUTE: a concolic unit testing engine for C. In M. Wermelinger and H. Gall, editors, ESEC/SIGSOFT FSE, pages 263–272. ACM, 2005.

[3] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios. Application of genetic algorithms to software testing.In 5th International Conference on Software Engineering and its Applications, pages 625-636, Toulouse, France, 1992.

[4] M. Roper. Computer aided software testing using genetic algorithms. In 10th International Software Quality Week, San Francisco, USA, 1997.

[5] Watkins, A. E. L. (1995) A Tool for the Automatic Generation of Test Data using Genetic Algorithms. In *Proc. SoftwareQuality Conf.*, Dundee, Scotland.

[6] B. Jones, H. Sthamer, and D. Eyres. Automatic structural testing using genetic algorithms. Software Engineering Journal, 11(5):299-306, 1996.

[7] G.McGraw, C.Michael, and M.Schatz. Generating software test data by evolution. IEEE Transactions on Software Engineering, 27(12):1085-1110, 2001.

[8] K. Lakhotia, M. Harman, P. McMinn, Handling dynamic data structures in search based testing, in: GECCO ’08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM, Atlanta, GA, USA, 2008, pp. 1759–1766.

[9] P. Godefroid, N. Klarlund, and K. Sen. DART: directed automated random testing. ACM SIGPLAN Notices, 40(6):213–223, June 2005.

[10] P. McMinn, M. Harman, D. Binkley, and P. Tonella. The species per path approach to search-based test data generation. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2006), pages 13–24, Portland, Maine, USA, 2006. ACM.

[11] P. McMinn. Search-based software test data generation: A survey. Software Testing, Verification and Reliability, 14(2):105–156, 2004

[12] J. Wegener, A. Baresel, and H. Sthamer. Evolutionary test environment for automatic structural testing. Information and Software Technology, 43(14):841–854, 2001.

[13] Harman, Lakhotia K. Phil Mcminn. A multi objective approach to search based data generation, *GECCO’07*, July 7–11, 2007, London, England, United Kingdom.

[14] G. Fraser and A. Arcuri. Whole test suite generation. *IEEE* Transactions onSoftware Engineering, 39(2):276–291, 2013.

[15] A. Arcuri and X. Yao. A memetic algorithm for test data generation of object-oriented software. In *IEEE* Congress on Evolutionary Computation(CEC), pages 2048–2055, 2007.

[16] A. Arcuri and G. Fraser. On parameter tuning in search based software engineering. In International Symposium on Search Based Software Engineering (SSBSE), pages 33–47, 2011.

[17] M. Harman and P. McMinn. A theoretical and empirical study of search based testing: Local, global and hybrid search. IEEE Transactions on Software Engineering (TSE), 36(2):226–247, 2010

[18] L. Baresi, P. L. Lanzi, and M. Miraz. Testful: an evolutionary test approach for java. In *IEEE Int.Conference on Software Testing, Verification andValidation (ICST)*, pages 185–194, 2010.