Techniques for Automatically Generating Test Cases from the Code

Philip John
University of Tartu, Email: philipjohn007@gmail.com

Abstract

One of the most challenging tasks in software testing area is to select the test inputs. There are many tools and methods which can automatically generate test cases from the code. The main methods used are random selection, code annotations, search-based techniques and symbolic execution based techniques. The search based techniques has been used for generating different varieties of test cases. This paper provides information about various search based algorithms like hill climbing, simulated annealing and genetic algorithms. An analysis is made regarding their effectiveness and efficiency, strengths and weaknesses and their applicability. Finally, details about the cases where to apply these algorithms in order to obtain the best possible near optimal solution are provided.

1. Introduction

Software systems need thorough testing to test all its functional and non-functional requirements. Software testing is becoming increasingly important as it validates the efficiency of the system. One challenging issue in software testing field is the sufficiently large amount of effort put in creating the test scenarios that will cover all the functionalities of the system. It is seen that almost fifty percent of the total development cost is due to the effort required for testing [4]. It is therefore necessary to automate the testing process in order to reduce the development cost. Some of the key requirements for automatic test generation include quick (faster than model-checking) and more reliable than static analysis, produce and actual test cases, reduced test cases, usage of existing tools that are freely available

A possible method that can be used for testing automation is to use metaheuristic search (MHS) algorithms [5]. This interest is because test case generation problems can be expressed as search or optimization problems [1].

2. Search Based Testing

Search based testing uses a fitness function to automatically find test data by searching a test object’s input domain [2]. The fitness function combines approach level and branch distance. Branch distance, as the name highlights, is obtained by taking the values of variables where the search flow deviated from the target path. Approach level denotes the number of branch control dependent areas that were not traversed during the execution. The fitness value usually computed by adding approach level to a normalised branch distance [2].

Search-based testing tries to solve, by reformulating to search problems, software engineering problems [6]. **Random search** is a mechanism which does not require the gradient of the problem involved. They are applicable mainly on functions which are not continuous and differentiable. These methods are also called as black-box methods.

Metaheuristic search algorithms include some generic algorithms which does not provide you the exact result, but manages to find the suboptimal solutions to complex space problems [6]. The search spaces are generally quite complex during reformulating into search problems. Simple search strategies will not work under these circumstances. Instead, we need to consider better search algorithms.

There are many MHS algorithms used for search based test case generation. The main MHS algorithms include hill climbing, simulated annealing, evolutionary algorithms etc. Evolutionary algorithms and hill climbing are the main and most common algorithms used in the field of software test case generation. In real world, SBST uses more global algorithms based on the complexity. We will go into the details of these algorithms.

3. MHS Search Algorithms

MHS algorithms are algorithms that include some specific strategies that adapts itself while the search progresses. Each of these algorithms need to have some parameter settings. Some of the main MHS algorithms are explained below.

3.1 Hill Climbing

Hill climbing is a local search algorithm that starts from a single random point and is used to better a single candidate solution [7]. It searches for a neighbouring search space from one state. In case a proper candidate is not found, then the algorithm terminates. Else, the search directs itself to that point. The method got its name because of its likeliness to climbing of hills of the fitness function [7].

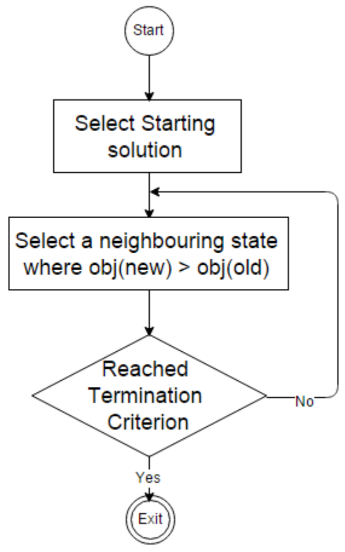
Start
Select Starting solution
Select a neighbouring state where obj(new) > obj(old)
Reached Termination Criterion
No
Yes
Exit

Figure 1: Hill Climbing Steps

The main approach that uses HC algorithm is the method used by Korel [9] which is termed as 'alternating variable method'. This method uses HC algorithm along with accelerated selection of neighbouring search points. The steps are depicted in Figure 1. It takes an input from the vector and adjusts its value in isolation from the remaining inputs in the vector and if it does yield an improved fitness, then the next input is chosen. The method continues until no modification of inputs results in a better fitness.

Hill climbing provides quick and simple results. However a well-known problem with hill climbing is that the result yielded will be sub-optimal in a local space. The obtained result may not be the actual result in the whole search space, meaning most of the cases it is not globally optimal [8]. On encountering this state, called the 'fitness invariant plateaux' [7], the search gets stuck at the peak of a hill and fail to search the neighbouring search spaces. To tackle this issue, the algorithm has to be restarted with a new random starting point many times until most of the fitness tests are exhausted.

3.2 Simulated Annealing

Simulated annealing technique, in principle, is similar to hill climbing. The difference between the two is that simulated annealing allows probabilistic acceptance of poor solutions by providing more movement within the search space. The probability of accepting a poor solution varies as the algorithm progresses. The probability *p* is calculated as $p = e^{-\frac{\delta}{t}}$, where t the temperature (a control parameter) and δ is the difference between objective values of the current and poor solution. The temperature is kept high initially in order to provide random movement within the search space. The advantage of this is that it reduces the dependency of the starting position. Based on a cooling schedule, the temperature is reduced. The cooling schedule should be such that it should not allow rapid cooling. Rapid cooling can cause the search to localize and the search can get stuck at a local optima [8].

The algorithm got its name from the annealing process in chemistry (cooling metals in heat bath). When a solid material is melted by heating, and then cooled to its solid state, the resulting material will have properties based on the speed of cooling it underwent [8].

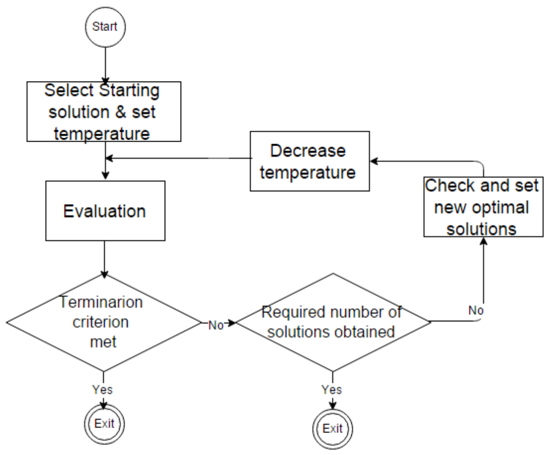
Start
Select Starting solution & set temperature
Decrease temperature
Evaluation
Check and set new optimal solutions
Terminarion criterion met
No
Required number of solutions obtained
No
Yes
Exit
Yes
Exit

Figure 2: Simulated Annealing Steps

The steps involved in this algorithm is provided in Figure 2. The first step is to select a starting solution from the search space. Set the initial temperature. Select a new state from the neighbouring structure. If the difference between the new state and the existing solution is positive, then set the solution to the current state. If not, generate a random probability value and in case it is less than $e^{-\frac{\delta}{t}}$ then set the solution to the current state. The same process continues with a decreased temperature value according to the cooling schedule.

3.3 Genetic Algorithm

Evolutionary algorithms are algorithms which aims at evolving superior candidate solutions. It is inspired by the Darwinian evolution theory. Genetic algorithm belong to this family of evolutionary algorithms. The search generates several inputs from the population making it a global search [2].

The population of candidate solutions taken into consideration for genetic algorithm are called 'individuals' or 'chromosomes' [8]. Each of the candidate solution is termed as 'genes'. The chromosome representation is in bit string format. e.g. <255,0,255> is represented as 111111110000000011111111.

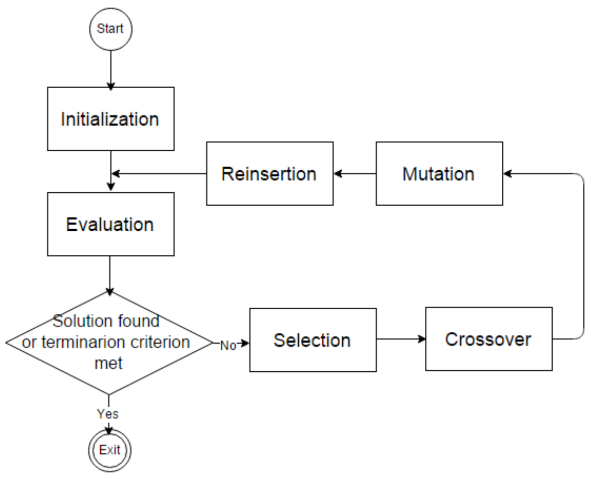

Figure 3: Genetic Algorithm Steps

The steps involved in this algorithm is provided in Figure 3. The main loop in the algorithm produces subsequent generations with the aim of producing fitter input values. Selection involves choosing the parents for crossover [7]. An optimum selection strategy has to be considered in this case. Crossover involves taking two parents to generate offspring by discrete recombination [10]. In this mechanism a gene is copied to one or more children at an even probability. In mutation, random modification are made on the children to provide diversity in the search [2]. Reinsertion involves producing new generation of individuals using the current population along with the generated offspring. Fitness evaluation is performed on the new population. A test is performed at this stage to check if a global optimum has been found, or if the search has failed. In either cases, the process is terminated.

3.4. Comparison between Various Algorithms

Mansour and Salame [11] have found that the hill climbing method is faster than genetic algorithm and simulated annealing mechanisms. But simulated annealing and genetic algorithm can cover more path. It was also reported that simulated annealing has a better performance compared to genetic algorithm. But in the study made by Xiao [12], the genetic algorithm outperformed simulated annealing when they were tested for condition-decision coverage.

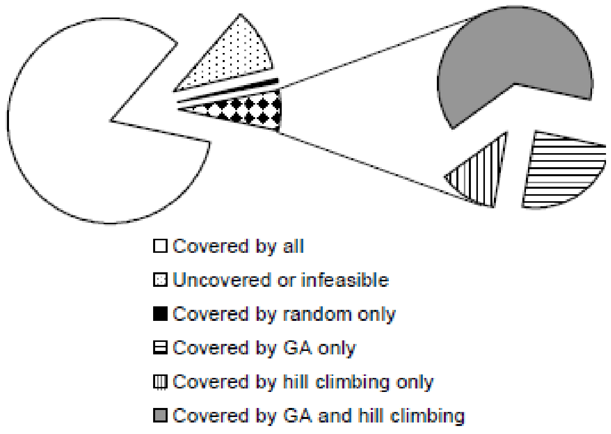

Figure 4: Coverage for various algorithms

Figure 4 shows the coverage percent for various algorithms during an experiment conducted by Harman and McMinn. The search was done on 640 branches. 532 branches were covered by all the search mechanisms. 41 branches were covered by either genetic algorithm or hill climbing which included 10 by genetic algorithm only and hill climbing alone covered 5. There were only four branches found by random search alone. The remaining 63 branches were not covered by any mechanisms.

It has been seen that genetic algorithm outperforms all other algorithms in the case of Royal Road functions [7]. These functions are developed in such a way that they isolate some key features of the fitness landscapes which are expected to be relevant for the performance of a genetic algorithm.

4. Conclusion

There has been many research conducted on search-based test case generation. Although there are many algorithms good enough to find an optima, they need not be 100% reliable. Harman and McMinn [2] comes up with a hybrid of genetic algorithm and hill climbing called Memetic algorithm and they prove that this algorithm can outperform the other search based test code generation mechanisms. Still in some problems we can see that the superior memetic algorithm can be outperformed by some simple search algorithms. Therefore we need to perform many empirical calculations on the problem and understand the type of it in order to determine the algorithm which can provide the best optimal solution to the problem.

5. References

[1] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test-case generation," *IEEE Transactions on Software Engineering (TSE)*, vol. 36, no. 6, pp. 742762, 2010.

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

[3] Kostyantyn Vorobyov and Padmanabhan Krishnan, "Combining Static Analysis and Constraint Solving for Automatic Test Case Generation," In *Proceedings of the 2012 IEEE Fifth International Conference on Software Testing, Verification and Validation (ICST '12). IEEE Computer Society, Washington, DC, USA*, 915-920, 2012.

[4] B. Beizer, "Software testing techniques", *Van Nostrand Reinhold Co*., 1990.

[5] I. H. Osman and J. P. Kelly, Metaheuristics: Theory and Applications: Kluwer Academic Publishers, 1996.

[6] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper, and M. Shepperd, "Reformulating software engineering as a search problem," IEE Software vol. 150, pp. 161-175, 2003.

[7] M. Harman and P. McMinn. "A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation". In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2007)*, pages 73–83, London, UK, 2007. ACM Press.

[8] P. McMinn, "Search-based software test data generation: A survey, " Software Testing, Verification and Reliability, vol. 14, no. 2, pp. lO5-156, 2004.

[9] B. Korel. Automated software test data generation. IEEE Transactions on Software Engineering, 16(8):870–879, 1990.

[10] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine, editors, Proc. of the First European Conference on Artificial Life, pages 245–254, Cambridge, MA, 1992. MIT Press.

[11] N. Mansour and M. Salame. Data generation for path testing. Software Quality Journal, 12(2):121–134, 2004.

[12] M. Xiao, M. El-Attar, M. Reformat, and J. Miller. Empirical evaluation of optimization algorithms when used in goal-oriented automated test data generation techniques. Empirical Software Engineering, 12(2):183–239, 2007.