Comparative Study of BBBC and PSO Algorithms for Automated Test Data Generation for Softwares

Shubham Kumar Shandil[1] Ashok Kumar[2]
[1, 2] Department of Computer Science & Engineering, MMEC,
[1, 2] M.M. University, Mullana, India

Abstract— This paper compares metaheuristic search algorithms namely Particle Swarm Optimization and Big Bang Big Crunch algorithms for automated test case generation using path testing criterion. For input generation symbolic execution method has been used in which first, target path is selected from Control Flow Graph of Software under Test and then inputs are generated using search algorithms which can evaluate path predicate corresponding to the path which is evaluated true. We have experimented on two real world and bench marked programs Triangle Classifier and Line-Rectangle Classifier showing the applicability of these techniques in genuine testing environment. The algorithm is implemented using MATLAB programming environment. The performance of the algorithms is measured using average test cases generated per path and average percentage coverage metrics. The PSO algorithm has proved its mettle by generating test data for small as well as large domain but BBBC algorithm failed to generate data for complex path having equality constraint especially in larger domain.

Keywords: - Software Testing, Particle Swarm Optimization, Big Bang Big Crunch, Symbolic Execution

I. INTRODUCTION

Quality software has to undergo complex, labor-intensive, time consuming and costly testing process before gaining the confidence of users, developers and managers of being a reliable and failure free product. Testing is not just revealing the errors in software but an indispensable activity before making software operational Test case generation is central and most important step because testing adequacy depends on the efficiency of test cases. If test cases are not able to detect faults then no conclusion can be drawn about the quality of the software, as the poor test cases can also lead to no fault generation. Test cases can be generated in several ways: automated, semi-automated or manual. Manual test case generation is relatively easy but it is a slow and costly process. Automated test case generation can save time and testing resources but they don't have intelligence of human mind to identify the non- linearity and discreteness in test inputs' search space. For improving the quality of automation and fulfilling the requirements of test case generation, many researchers have explored new soft computing based techniques such as Tabu Search, Simulated Annealing, Genetic Algorithm, Ant Colony Optimization, Big Bang Big Crunch Particle Swarm Optimization, Memetic Algorithms, etc. to fulfill testing requirement and to generate suitable test cases automatically [1,2].

II. METHODOLOGY

A. *Software testing as a Search Problem*

Testing is execution of software with the intent of finding errors. It requires searching the input domains for such values which can invoke different outputs or execute different components making it necessary for employing an efficient search algorithm for test data generation for fulfilling testing objectives. Search techniques are applied for test case generation by transforming testing objective into search problem. Two components are essential for a problem which is to be modeled as search target. First a mechanism should be used through which the problem is converted into search algorithm and second component is assessment of the suitability of solutions produced by search technique to guide the individuals for exploring search space. For software testing purpose, as solution lies in searching inputs, every possible set of inputs represent the global population in search algorithm and selected inputs from this global set are represented by individuals in the population. We have chosen the all-path coverage [3] criterion for our experimentation. The path testing method involves generation of test case for a target feasible path in such a way that on executing program, it covers all branches on that path by satisfying all the conditions at branch nodes of a specific path. Therefore, the problem of path testing can be viewed as constraint satisfaction problem. Figure 1 shows the different building blocks of a path based automatic test data generator.

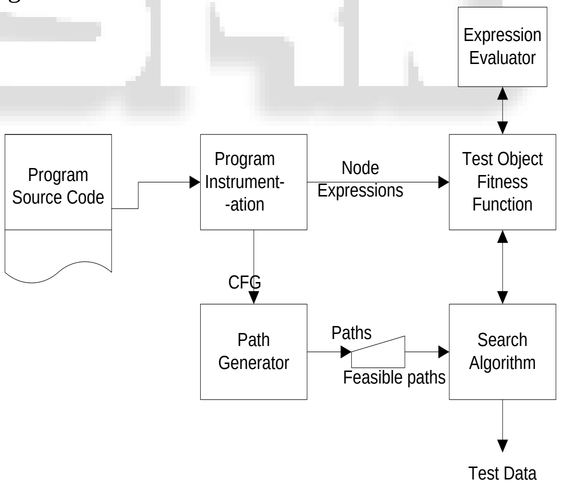

Fig 1: Automatic Symbolic Path Test Data Generator

B. *Big Bang Big Crunch Algorithm:*

The Big Bang and Big Crunch theory is introduced by Erol and Eksin [4], which is based upon the analogy of universe evolution where two phase of evolution is represented by expansion (Big Bang) & contraction (Big crunch). This algorithm has a high convergence speed and low computational time. In fact, the Big Bang phase dissipates energy and produces disorder and randomness. In the Big Crunch phase, randomly distributed particles are arranged into an order by way of a converging operator called "center of mass". The Big Bang–Big Crunch phases are followed alternatively until randomness within the search space during the Big Bang becomes smaller and smaller and finally leading to a solution. Below is given the algorithm for the BBBC algorithm in steps [6].

1. Create random population of solution.
2. Evaluate Solutions.
3. The individual which is fittest can be selected as the center of mass.
4. Calculate new individuals around the center of mass by subtracting or adding a normal random number whose value decreases as the iterations elapse.
5. Continue the algorithm until a predefined stopping criterion has been met.

C. *Particle Swarm Optimization:*

PSO is a population-based, biologically inspired algorithm which applies to concept of social interaction to problem solving where each individual is referred to as particle and represents a candidate solution. Each particle in PSO flies through the search space with an adaptable velocity that is dynamically modified according to its own flying experience and also flying experience of other particles using the following equations:

$v_{id}^{t+1} = w \times v_{id}^{t} + r_1 \times c_1 \times \left(p_d^{g} - x_{id}^{t}\right) + r_2 \times c_2 \times \left(p_{id}^{l} - x_{id}^{t}\right)$---------------------------- (1)

$x_{id}^{t+1} = x_{id}^{t} + v_{id}^{t+1}$------------------------- (2)

where

- v_{id}^{t+1} is a velocity vector at t+1 time for i particle in dth dimension
- x_{id}^{t+1} position vector at t+1 time for ith particle in d dimension
- $r_{1,} r_2$ are random number generators.
- c_1 and c_2 are are learning rates governing the cognition and social components.
- p_{id}^{l} represents the particle with best p-fitness.
- w is the inertia factor that dynamically adjusts the velocities of particles gradually focusing the PSO into a local search.

The overall workflow of PSO can be illustrated in the following steps:

1. Initialize the particle population by randomly ssigning locations (X-vector for each particle) and velocities (V-vector with random or zero velocities - in our case it is initialized with zero vector).
2. Evaluate the fitness of the individual particle and record the best fitness P$_{best}$ for each particle till now and update P-vector related to each P$_{best..}$
3. Also find out the individuals highest fitness G$_{best}$ and record corresponding position p$_{g.}$
4. Modify velocities based on P$_{best}$ and G$_{best}$ position using eq1.
5. Update the particles position using eq2.
6. Terminate if the condition is met
7. Go to Step 2

D. *Fitness Function*

In path testing approach a candidate solution (also called an individual) is used to evaluate constraint system of the target path. In order to traverse a feasible path, the control must satisfy the entire constraints, which falls on that particular path. For our experiments, we have used symbolic execution technique of structural testing. Therefore, for each path a compound predicate (CP) is made by 'anding' each branch predicate of the path. The compound predicate (CP) must be evaluated to true by a candidate solution in turn to become a valid test case. If CP is not evaluated to be true by an individual then all the constraints of a particular path are broken up in distinct predicates (DP). A distinct predicate is the one, which contains only one operator (a constraint with modulus operator is exception). Each DP is evaluated by taking values of its operands from candidate solution. If it is evaluated to be true then no penalty is imposed to candidate, otherwise candidate is penalized on the basis of branch distance concept rules as shown in table 1 which is also recommended by Watkins et al [5] for static structural static testing.

Table 1. Branch Predicate based Fitness Function

Violated Predicate	**Penalty to be imposed in case predicate is not satisfied**
A < B	A –B + ζ
A <= B	A – B
A > B	B – A + ζ
A >= B	B-A
A = B	Abs(A-B)
A ≠ B	ζ – abs(A – B)

After this integrated fitness due to whole of CP is determined by adding penalty values of two DPs, if they are connected by a conditional '&' operator. If two distinct predicates (DPs) are connected by a conditional '||' operator then minimum penalties of two DPs is considered for the evaluation of whole CP fitness. If integrated fitness is zero then CP is called evaluated or satisfied by the individual whose values are replaced in CP and search process for particular path is terminated otherwise search is allowed to proceed further.

III. Experimental Setup

All experiments are performed in the MATLAB framework. In these experiments, main aim is to fine tune the parameters of PSO and BBBC algorithm to prove the usefulness and utility of algorithm for test case generation. We have taken two test objects: triangle classifier (TC) and line-rectangle classifier (LRC) programs which are benchmark programs used frequently in testing literature. For these two test objects, test data is generated from input variables by taking different domain; one very large of the size of the order of 10^{7} and one small with a size of order of 10^{3} for each path and experiment is conducted 100 times for averaging results. In each attempt, the PSO and BBBC are iterated for 100 generations for each of 10 runs. In each run except of the first run, 1st generation population is seeded with the best solution from the previous run. Total number of real encoded individuals in each population is 30.. If a solution is not found within all runs that generates total 30,000 invalid test cases then it is declared that the test case generation process has failed for that particular attempt. This value has been obtained by multiplying total number of runs, generations and number of individual in each population. In random test generator also each attempt generates 30,000 invalid test cases before declaring it a failure. The performance of algorithms is evaluated using two

parameters**:** Average percentage coverage (APC) and Average test case generation per path (ATCPP). The APC is used to measure effectiveness of test case generation process and efficiency of process is measured by the ATCPP. A good test data generation process will try to give 100% APC with less number of ATCPP. For each test object, we have measured average test case generation per path (ATCPP) and average percentage coverage (APC). ATCPP tells the level of effort a search algorithm has to make for test data generation and is taken by sum of test case
generated for all paths divided by number of feasible paths. APC tells about the efficiency of test data generator and is calculated as fraction of paths covered. High figure of APC and low figure of ATCPP is desirable.
Detail characteristics of these test objects are given in table 2.

Table 2. Test Object characteristics

Name of Program	**TC**	**LRC**
Lines of Code	35	56
Cyclomatic Complexity	07	19
Number of Decision Nodes	06	18
Highest Nesting Level	05	12
Total Paths in CFG	07	19
Feasible Paths	07	17

IV. Results And Discussions

Table 3 presents the experimental results on two most standard benchmark programs regularly used in testing research. These are triangle classifier and line rectangle classifier programs. Average test cases generation per path (ATCPP) and Average percentage coverage with PSO and BBBC algorithms are shown in table 3.

Table 3. ATCPP and APC with PSO and BBBC algorithms

		Triangle Classifier		**Line Rectangle Classifier**	
	Input Range	**ATCPP**	**APC**	**ATCPP**	**APC**
PSO Method	**-10^3 to $+10^3$**	305	100%	105	100%
	-10^7 to $+10^7$	1020	100%	376	100%
BBBC Method	**-10^3 to $+10^3$**	5372	81.4%	202	100%
	-10^7 to $+10^7$	11725	48%	3429	87%

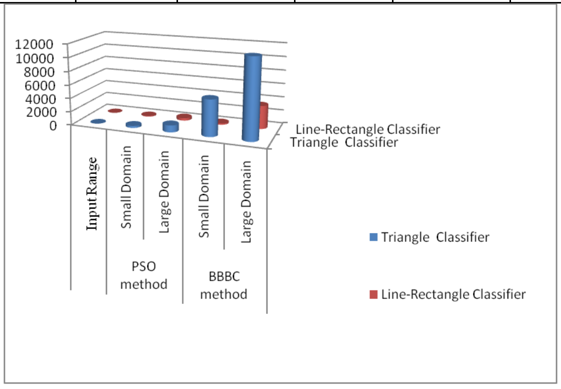

Fig 2: ATCPP of TC and LRC Programs

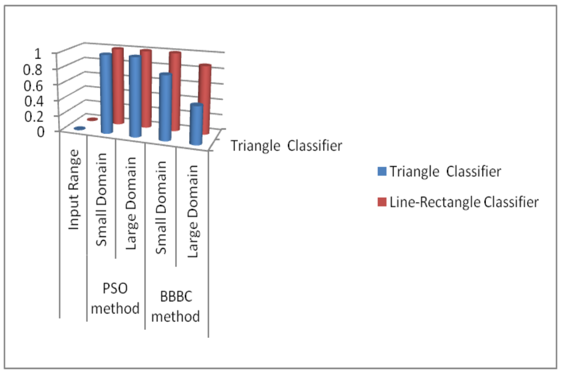

Fig 3: APC of TC and LRC Programs

If we analyze the result we can clearly find out that the PSO has generated test data regularly without failing while the BBBC has failed to generate test cases where large domain is taken. It also fails to generate test cases for such paths where equality constraints are present.

V. Conclusion

The PSO method gives encouraging result for both of test objects. The performance of the PSO method in small as well as in large domains shows that it has both type of search capabilities; local as well as global for fulfilling testing requirements. Hence, it has been observed that it is ideally suited for test case generation problem. But the BBBC method has completely failed to find data for equality constraint.

We also observed that Metaheuristic Techniques exhibits domain dependency in test case generation, as they have to generate huge number of ATCPP in larger domain.
We also observed that the testing efforts of these techniques which we have measured as ATCPP largely depend on the type of path constraints encountered by the search algorithm.

References

[1] Edvardsson J. A survey on automatic test data generation In Proceedings of the second conference on computer science and engineering, Linkoping: ESCEL; October 1999; 21–28.
[2] McMinn P. Search-based Software Test Data Generation: A Survey. Software Testing, Verification and Reliability June 2004; 14(2):105-156.
[3] Amoui M, Mirarab S, Ansari A and Lucas C, A Genetic Algorithm Approach to Design Evolution using Design Pattern Transformation, International Journal of Information Technology and Intelligent Computing 1(2, 2006 pp. 235-244.
[4] Erol OK and, Eksin I, A new optimization method: Big Bang-Big Crunch, Advances in Engineering Software, 37, 2006, 106-111.
[5] Watkins A, Hufnagel E. M. Evolutionary test data generation: a comparison of fitness functions. Software Practice & Experience 2006; 36:95–116
[6] Surender Singh, Parvin Kumar, "Application of Big Bang Big Crunch Algorithm to Software Testing", International Journal of Computer Science and Communication Vol. 3, No. 1, January-June 2012, pp. 259-262