Comparative Analysis of BBBC and GSA Algorithms for Automated Test Data Generation Using Symbolic Execution Method

Gurbaksh Singh[1] **R.P Garg**[2]
[1,2]Department of Computer Science & Engineering
[1,2]MMEC, M.M. University, Mullana, India

Abstract— In recent years various heuristic optimization methods have been developed. in this paper compares two new meta-heuristic techniques namely Gravitational Search Algorithm (GSA) and Big-Bang Big-Crunch (BBBC) algorithms for automatic test case generation using path testing criterion in symbolic execution environment. We have experimented on two real world and benchmarked programs Triangle Classifier (TC) and Line-Rectangle Classifier (LRC) showing the applicability of these techniques in genuine testing environment. Result show that both algorithms perform well in comparison to random testing but fail to generate test cases where input domain size is large.

Keywords: Gravitational search algorithm, Big Bang Big Crunch, Symbolic Execution, Software Testing.

I. Introduction

In order to save precious development resources, the software testing process requires automation of test data generation. The manual generation of test cases is relatively easy but it is a slow and costly process. The automation is also helpful in increasing the quality of test data by generating unbiased, effective and efficient test cases. Highly non-linear structure of software presents a formidable task to search algorithms for finding optimal and efficient test data from a complex, discontinuous, non-linear inputs' search space. Despite having so many benefits, automated test case generation is not so easy because it requires intelligence of human mind to identify the non-linearity and discreteness in test inputs' search space.

For such environment, the search algorithm must have both types of search capabilities; local as well as global.. For improving the quality of automation and fulfilling the requirements of test case generation, many researchers have explored several metaheuristic techniques such as genetic algorithm, simulated annealing, tabu search, ant colony optimization, particle swarm optimization, memetic algorithms etc. to fulfill testing requirement and to generate suitable test cases automatically [1]. Hence soft computing techniques can be useful to handle such complexities. Although Big Bang Big Crunch(BBBC) has been successfully employed on score of engineering application such as mechanical engineering and civil engineering, computer engineering. but gravitational search algorithm(GSA) applicability in testing domain is still unexplored.

II. Methodology

A. Software testing as a Search Problem

Software Testing is the Process of assessing the functionality and correctness of a problem through execution or analysis of the program. In structural testing, these criteria can be anything from all-statement-execution to all-path-coverage [2]. We have chosen the all-path coverage criterion for our experimentation because one, it is the hardest to follow and second, in true sense, it is the real representative of structural testing. Very few test data generators have followed this criterion. The path testing method involves generation of test data for a target feasible path in such a way that on executing program, it covers all branches on that path. To cover a particular branch, the condition(s) at branch node must be satisfied by the test data, which directs the control flow of program to the next branch of the path. A path may contain several branches and in order to execute that path, all these branch-conditions must be evaluated true by the test data. Consequently, problem of path testing can be formulated simply as constraint satisfaction problem which should be analyzed and solved with the help of some search method by generating inputs in such a way that can satisfy all the branch constraints on the path. A valid test case is generated, which should execute the particular path by satisfying all of the boolean expressions included in that path. First test object source code is fed to program instrumentation for CFG and node expressions generation. Subsequently CFG is used to generate all possible paths which are filtered manually for feasible path in order to become input to search algorithm. Node expressions include branch node predicates as well as non-branch node statements which are used to evaluate candidate solutions in test object fitness functions.

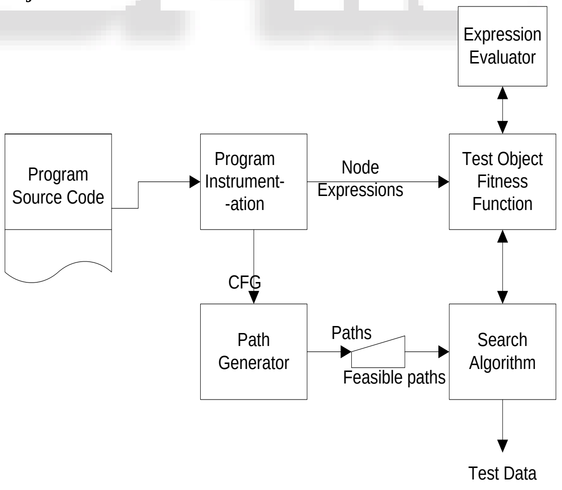

Fig. 1: Automatic Symbolic Path Test Data Generator

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 low computational time and high convergence speed. In fact, the Big Bang phase dissipates energy and produces disorder and randomness. In the Big Crunch phase, randomly distributed particles (which form the solution when represented in a problem) are arranged into an order by way of a convergence operator "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.

- Create random population of solution.
- Evaluate Solutions.
- The fittest individual can be selected as the center of mass.
- Calculate new candidates around the center of mass by adding or subtracting a normal random number whose value decreases as the iterations elapse.
- The algorithm continues until predefined stopping criteria has been met.

C. Gravitational Search Algorithm:

GSA is another latest search algorithm developed by Rashedi et.al in 2009. The GSA is based on the Newtonian law of gravity and the law of motion [3]. Here agents are considered as objects and their performance are based on its masses. All the objects attach to each other by a force called gravitational force and this force causes the global movement off all objects to object with heavier masses. The gravitational force between two particles is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. The heavy masses will correspond to good solutions and move more slowly and conversely light masses correspond to poor solutions and move towards heavy masses much faster.

F = G(t).(ma.mp/r2)………..(i)

a= F/m……………………..(ii)

Where;

F is a gravitational force

G(t) is a gravitational constant which is changing during the course of time

Ma is a active mass which represent the strength of gravitational field due to its mass.

Mp is passive mass which represent the strengths of an object interaction with gravitational field

(R2) is a distance squared

a is acceleration and m is a mass.

Following steps illustrate the overall optimization scheme of GSA:

- Step 1. Search space identification.
- Step 2. Generate initial population between minimum and maximum values.
- Step 3.Fitness evaluation of agents.
- Step 4. Update G(t), best(t), worst(t) and Mi(t) for i =1,2,. . .,m.
- Step 5.Calculation of the total force in different directions.
- Step 6.Calculation of acceleration and velocity.
- Step 7.Updating agents' position.
- Step 8. Repeat step 3 to step 7 until the stop criteria is reached.
- Step 9. Stop

D. Fitness Function

In our experiment, we have used symbolic execution technique of static structural testing. In path testing approach a candidate solution (also called an individual) is used to evaluate constraint system of the target path.So, corresponding to each path a compound predicate (CP) is made by 'anding' each branch predicate of the path. The 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. Ifit is evaluated to be true then no penalty is imposed to candidate solution, otherwise candidate solution 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 'and' operator. If two DPs are connected by a conditional 'or' 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

Both algorithms are implemented in the MATLAB framework. GATBX toolbox is used for the GSA and BBBC algorithm implementation. In these experimentations, main aim is to fine tune the parameters of GSA and BBBC algorithm to prove the usefulness and utility of algorithm for test case generation concept. We have taken two test objects; triangle classifier and line-rectangle classifier programs which are benchmark programs used frequently in testing literature. For triangle classifier and line-rectangle classifier programs, test data is generated from input variables by taking different domain; one very large of the size of the order of 107 and one small with a size of order of 103 for each path and experiment is conducted 100 times for averaging results. In each attempt, the GSA 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. This is done to check premature convergence of the population. Total number of real encoded individuals in each population is 30. Twenty percent of individuals in each population are taken from previous population. 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. An invalid test case is single individual or particle in GSA and BBBC algorithm in population, which does not qualify to become a test case. 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.

Table. 2: Test Object characteristics

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

IV. RESULTS AND DISCUSSIONS

We have experimented on two most standard benchmark programs regularly used in testing research [6, 7]. These are triangle classifier and rectangle classifier programs. We have experimented on two most standard benchmark programs regularly used in testing research [6, 7]. These are triangle classifier and rectangle classifier programs. Table 3 lists the results of test cases generation using GSA and BBBC methods applied on two programs. Third row in this table shows the result for small domain and fourth one is for larger domain. Second and fourth column register invalid average test cases per path (ATCPP) for triangle and line-rectangle problem respectively. Third and fifth columns show average percentage coverage (APC) of all paths for both programs in 100 attempts of test case generation for each path.

Table. 3: (ATCPP) and Average percentage coverage with GSA and BBBC algorithms

		Triangle Classifier		Line-Rectangle Classifier	
	Input Range	ATCPP	APC	ATCPP	APC
GSA method	-10^3 to $+10^3$	7829	83%	1529	100%
	-10^7 to $+10^7$	9723	52%	6798	84%
BBBC method	-10^3 to $+10^3$	9279	81.4%	2020	100%
	-10^7 to $+10^7$	11725	48%	3429	87%

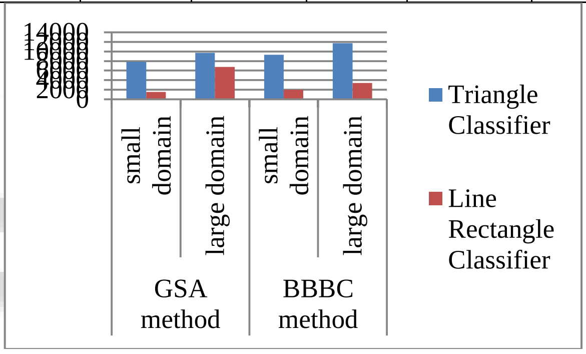

Fig. 2: ATCPP of both programs

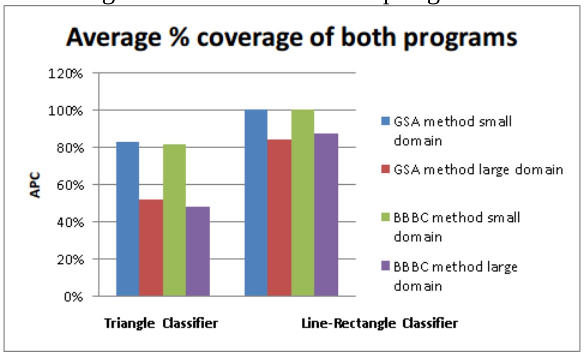

Fig. 3**:** Average % coverage of both Programs

If we analyze the result we can clearly find out that the both programs successfully generated test data for small domain but not able to generate test data for larger domain. In analysis It was also found that they fail to generate test cases for such paths where equality constraints are present.

V. CONCLUSION

The GSA and BBBC methods have given encouraging results for both of test objects in small domain however these both fail to generate test cases for larger domains and for such path which have equality constraints. During experimentation we also observed that metaheuristic techniques exhibits domain dependency in test case generation, as these have to generate huge number of ATCPP In larger domain.

Another observation, we have made during experimentations that 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] [Edvardsson1999] 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] [Frankl1988] Frankl PG, Weyuker EJ. An Applicable Family of Data Flow Testing Criteria. IEEE Transaction On Software Engineering. 1988; 14(10):1483-1498
[3] [Rashedi2009] EsmatRashedi, Hossein Nezamabadi-pour, SaeidSaryazdi, GSA: A Gravitational Search Algorithm, Information Sciences 179 (2009) 2232–2248
[4] [Erol2006]Erol O. K., Eksin I., 2006. A new optimization method: Big Bang-Big Crunch, Advances in Engineering Software, 37, 106-111
[5] [Watkins2006] Watkins A, Hufnagel E. M. Evolutionary test data generation: a comparison of fitness functions. Software Practice & Experience 2006; 36:95–116
[6] [Díaz2007] Díaz E, Javier T, Raquel B, José JD. A tabu search algorithm for structural software testing.Computers and Operations Research, 2007
[7] [Lin2001] Lin JC, Yeh PL. Automatic test data generation for path testing using GAs. Information Sciences 2001; 131:47–64