AUTOMATIC TEST DATA GENERATION BY PARALLEL BIG BANG-BIG CRUNCH

Naveen Shaillu [1]
Research Scholar
Baddi University of Emerging Science and Technology, Baddi, India
Dr. Amar Singh [2]
Baddi University of Emerging Science and Technology, Baddi, India
Rajesh Chaudhary [3]
Baddi University of Emerging Science and Technology, Baddi, India

Abstract: *Software testing is a method of realizing a program with a plan to discover the software bugs. Software testing is complex and expensive process, usually consuming minimum 35% of total development time along with 50% of the total development cost. The main stage of software testing is automated test data generation (ATDG).This paper presents Parallel Big Bang –Big crunch soft computing approach for automatic test data generation. The performance of proposed approach was tested on four real world programs. Therefore, ultimately results show that PBB-BC based approach for automatic test data generation outperforms BB-BC based approach.*

Keywords: Software Testing, Test Data, Automatic Test Data Generation, BB-BC, PBB-BC

I. Introduction

Software testing is a component that is costing lots of money in software development and maintenance [1]. In software testing, test data generation is labour –intensive component. Test data is a documented data that is basically used to test the software program. Test data generation is a technique that helps to find program inputs which specify the testing requirement. Automated test data generation is a technique which helps to make software development and maintenance process cost and time effective process. Because it reduces time and cost require for software development and maintenance. There are two techniques which are used for automated test data generation such as functional and Structural testing. Structural testing are further divided into sub categories which includes different kind of approaches and implementation methods such as Random approach, Path-oriented approach, Goal-oriented approach, Static method, Dynamic method, Hybrid method, Intelligent Approach [2].

Structural Testing: These techniques are based on the core structure of a program, where test cases are selected so that structural components are covered. Examples of structural testing are statement testing, decision testing, path testing, condition testing, dataflow testing, structured testing, and domain testing [3]. Additional structural techniques could be based on different approaches and implementation methods.

Functional Testing: This type of testing is used where test cases are derived from the program's specification and it deals with the functionality of the software. Examples of these techniques are equivalence partitioning, boundary value analysis, random testing and functional analysis-based testing [3].

Random Approach: Random test data generation approach is basically consists of producing inputs at an unsystematic way until a useful input is not found. This approach is rapid and simple but might be a poor selection with complex programs and with complex adequacy standards [4].

Path oriented Approach: Path oriented approach is usual approach used for creating CFG (control-flow graph). In this method, graph is generated initially and by using the graph a particular path is selected. By using this technique such as symbolic assessment test data is generated for that path in the end [5, 6]. While traversing the path in its place of actual values execution variables are used in symbolic. When producing paths/graphs that method will face the difficulties during traversing test data through branches and predicates (infeasible path problem), meanwhile although difficulty of data types.

Goal-oriented Approach: In this approach test-data is chosen from the existing pool of candidate test data to implement the certain goal, such as a statement, no matter how the path is taken [5]. This approach includes two basic steps: to recognize a set of statements (respective branches) the covering of which involves covering the measure; to produce input test data that perform every chosen statement (respective branch) [6]. Two distinctive approaches, 'Assertion-based' and 'Chaining approach' are acknowledged as goal oriented. In the first case assertions are introduced and then resolved while in the second case data requirement investigation is carried out. Usually the goal-oriented approach considers problems of goal selection and choice of adequate test data.

Intelligent Approach: According to Pargas et al [5] the intelligent test-data generation approach frequently depend on sophisticated study of the code to guide the examine for new test data. However in the author's view this method can be lengthy up to the intelligent investigation of program requirement as well. With the planned lengthy capability this approach will lie between functional and structural testing.

Static Methods: These are the testing methods implemented for study and testing of system illustrations such as the requirement documents, design diagrams and the software source code, either manually or automatically, without truly executing the software [7, 8]. In additional words, the static methods do not involve the software under test to be performed. They typically use symbolic implementation to achieve constraints on input variables for the precise test principle. Outcomes to these controls symbolize the test-data [9]. Implementing a program symbolically means that instead of using actual values, 'variable substitution' is used.

Dynamic Method: In this method, Instead of using variable backup, these methods execute the software under test with some, feasibly randomly particular input [6]. By detecting the program flow the system can handle whether the projected path was taken or not. In case of an undesirable response the system backtracks to the node where the flow took the inappropriate path. Using different types of search methods the flow can then be changed by control the input in a way so that the accessible branch is taken.

Hybrid Methods: As the name suggests, a hybrid method is the combination of static and dynamic methods. This combination could be in the form of side-by-side use, probably resolving some tasks by using static methods and some by using the dynamic methods, or by both methods.

Soft computing is a word applied to an arena with in computer science to find out the results to computationally –hard tasks such as the solution of NP-hard problems for which a precise solution should not be derived in polynomial time. There be present a number of techniques for producing test cases like fuzzy logic, finite state machine, soft computing, genetic programing and evolutionary computation [10].

This paper presents path–oriented approach for automatic test data generation that uses a PBB-BC algorithm, which is guided by the control flow graph of the program, to search for test data to satisfy test requirements. Figure 1 (initially drawn by [11]) shows the architecture of an automated test data generator using path-oriented approach, in easy and simple format.

As shown in Figure 1 the program analyzer, through control flow graph and data dependence graph, analyze the software under test (SUT).After that, path selector identifies set of paths to satisfy selected testing criterion and at the end the test data generator generates test data accordingly [12].

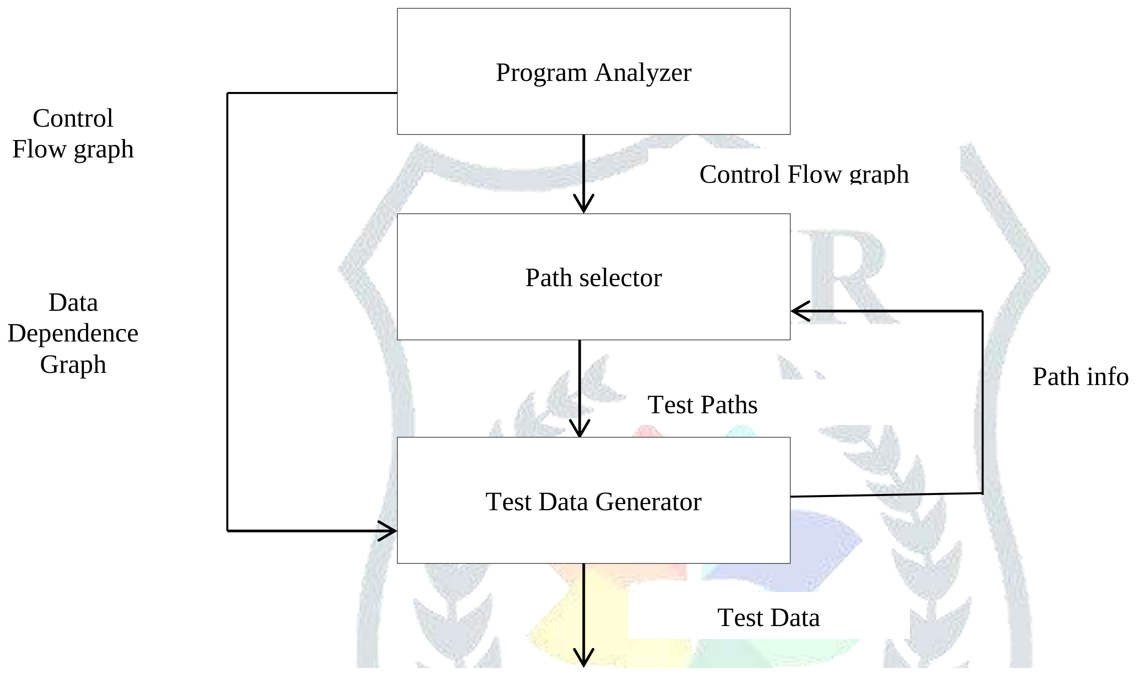

Fig.1 Architecture of a test data generator using path oriented approach

This paper contains six sections. Section-II presents Big Bang-Big Crunch (BB-BC) approach for Automatic Test Data generation. Section-III presents Parallel Big Bang Big Crunch approach. Section –IV presents proposed approach for Automatic Test Data Generation i.e. (Parallel Big Bang-Big crunch). Section-V presents simulation and results. Section-VI presents conclusion.

II. BIG BANG BIG CRUNCH(BB-BC) BASED TEST DATA GENERATION

Big Bang-Big Crunch algorithm is a population based meta-heuristic algorithm. This algorithm based on the theories that are responsible for evolution of the eternity hence known as the Big Bang-Big Crunch theory. The BB-BC theory is proposed by Erol and Eksin [Erol2006]. It consists of two different phases in which first phase named as Big Bang phase and second phase named as Big Crunch phase.

In first phase, candidate solutions are arbitrarily scattered over the search space and in the second phase, arbitrarily scattered elements are drawn into an organized fashion. The BB-BC optimization method creates arbitrary points in the first phase and reduces these points into only one point in the second phase. The second phase has a convergence operator that has several inputs but only one output, and this convergence operator is known as the "centre of mass", since the only one output has been derived by calculating the centre of mass[13].

BB-BC Search Algorithm workflow for test data generation:

1. Initialize number of candidate solutions i.e. MC and selected MC randomly from within the search space.
2. The fitness function values can be calculated for all the candidate solutions.
3. Using the centre of mass equation we find the best candidate that is excellent in its group.

$$y_c = \frac{\sum_{i=1}^{MC} \frac{1}{f^i} y_i}{\sum_{i=1}^{MC} \frac{1}{f^i}} \qquad (1)$$

Here

y_c= position of the centre of mass;

y_i= position of candidate i;

f^i= fitness function value of candidate i;

4 After choosing elite, choose novel elite around the centre of mass by adding or subtracting an ordinary arbitrary number whose value reduce as the iteration elapse. This can be calculated by:

$$y^{new}=y^{c}+\frac{m(rand)}{k} \qquad (2)$$

Where y^c stands for centre of mass, *m* is the upper bound of the constraint, rand is an ordinary random number and *k* is the kth iteration of the algorithm. Then new point y^{new} is upper and lower limit.
5. Return to Step 2 until stopping criteria (maximum iteration/best solution) has been found.
6. Stop.

III. **Parallel BIG BANG BIG CRUNCH (PBB-BC)**

The BB-BC algorithm is based upon single population whereas the search ability of the BB-BC algorithm has been improved by making it a multi-population algorithm and this multi-population algorithm known as Parallel Big Bang Big Crunch (PBB-BC algorithms).The algorithm further renamed as PB3C[14][15].
1. Initialize a set of N populations each of MC candidate solutions, selected randomly from within the search space.
2. For each population calculate the fitness function values of all the candidate solutions.
3. For each population for guiding the new search calculate centre of mass using Equation (1).

$$y_c=\frac{\sum_{i=1}^{MC}\frac{1}{f^i}y_i}{\sum_{i=1}^{MC}\frac{1}{f^i}} \qquad (1)$$

Where
y_c= position of the Centre of mass;
y_i= position of candidate i;
f^i= fitness function value of candidate i;
For excellent candidate from where we can choose as the centre of mass in place of Equation (1). By using that way we shall get “N” local best elite l_{elite} for each of the “N” populations.
4. Calculate global best (g_{elite}) elite from the selected “N” local best l_{elite} elite, based upon the fitness values,
5. With a given probability change a gene of l_{elite} elite with the corresponding gene of g_{elite} elite.
6. For each population find new elite around the centre of mass by surplus or subtracting a common random number from where value reduce as the iterations elapse. This can be calculate as:

$$y^{new}=y^{c}+\frac{m(rand)}{k} \qquad (2)$$

Where y^c used for centre of mass, *m* stands for upper bound of the constraint, rand is a common random number and *k* is the kth iteration of the algorithm. Then new point y^{new} is upper and lower limit.
7. Return to Step 2 until stopping standards has been found.
8. Stop.

IV. **PARALLEL BIG BANG BIG CRUNCH (PBB-BC) BASED TEST DATA GENERATION**

In this section we can use PB3C algorithm for automatic test data generation. PB3C is a multipopulation based meta-heuristics technique. Basically this approach is an enhancement of BB-BC algorithm. In this algorithm we can take N number of population rather than single population.
Firstly we can randomly generate the random N population with multiple test data. After that we can define the value of iteration i.e. k. if the value of k is one then we can use initial populations otherwise we can use previous solution as population.
After that we can calculate fitness of each test data for N population. Fitness of each test data for N population based upon paths which are covered by that test data. The test data which cover more path having more fitness value. Later on we can calculate the N numbers of local best test data for N population. After that we can select one global best test data from these N local best test data and the Global best test data is that local best test data which have greater fitness value.
After half iteration i.e. k we can move local best test data towards global best test data. For this purpose we can generate a random number between 0 and 1.If the number which is generated is greater than 0.5 than that corresponding gene of local best test data replace with correspondence gene of global best test data. That test data now become global best test data.
After that we can generate a new test data by adding or subtracting a number from the global test data. The number which is added or subtract from the global best data is generated randomly between two small numbers. The number which is smaller in both numbers is known as lower limit and number which is greater is known as upper limits.
After that the new test data is generated that test data is act as inputs for next iteration. After all iteration we have best set of test data for maximum path coverage.
The control flow graph of PBB-BC shown in fig.1

PBB-BC ALGORITHM FOR TEST DATA GENERATION:

```
Begin
/* Big Bang Phase */
Create M populations each of size MC test data randomly;
/* End of Big Bang Phase */
While not SC /* SC is a Stopping criterion */
```

```
/* Big Crunch Phase */
For i = 1: M
    Compute the fitness value of all the test data of ith population;
    Best fit test data can be choose according to path coverage.
    Select local best test data lbest (i) for ith population;
End
From amongst "M" lbest Test data select the globally best gbest Test data;
For i =1: M
    With a given possibility exchange a gene of lbest (i) with the corresponding
    gene of global best gbest test data
End
/* End of Big Crunch Phase */
/* Big Bang Phase */
Compute new test data by adding or subtracting a normal random number whose value decreases as the iterations elapse using
/* End of Big Bang Phase */
End while
End
```

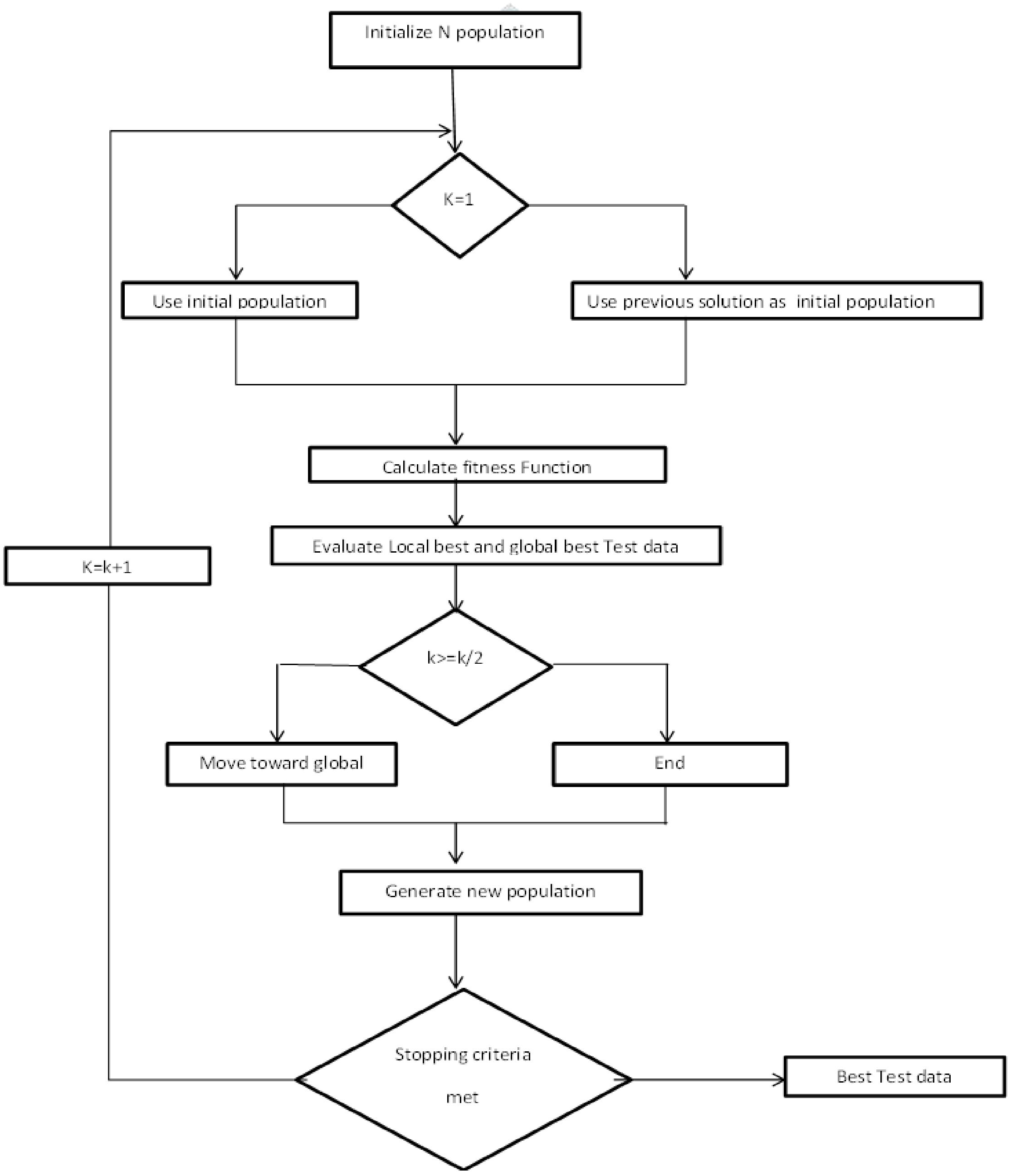

Fig. 1 Flow chart Of PBBBC for Test data generation

Fig. 4.1 Test Data Generation using PBB-BC

V. SIMULATION AND RESULTS

The proposed approach tested on four real world problems. The objective of the research was to produce test data automatically from the corresponding CFG using the standard BB-BC algorithm. The CFG of programs are manually designed by corresponding source code and all feasible paths are also identified manually. The PBB-BC algorithm is executed using the MATLAB programming environment. The performance of the algorithms is measured using Average Percentage Coverage (APC) metrics.

We have chosen 4 real world programs for test data generation activity. Some of these are frequently used by researchers.

1. Is prime (ISPRIME) in this program we can test an integer for its primness.

2. Triangle classifier (TC) this program is mostly used for testing of test data generation. In this program we can give 3 input as 3 sides of a triangle and then decides whether these sides form a triangle or not if yes, then of what type.

3. Quadratic equation program (QUAD) in this program we can check that whether equation is quadratic or not if yes, then finds the roots of a quadratic equation. This program also tests equation for linearity or infeasibility.

4. Car Brake Controller System (CBCS) in this program according to speed and distance we can identify there is any requirement of brake to stop the car or not if yes, then which type of brake required at that place to stop car on particular location.

CFG of all programs shown below from fig. 3 to fig.6:

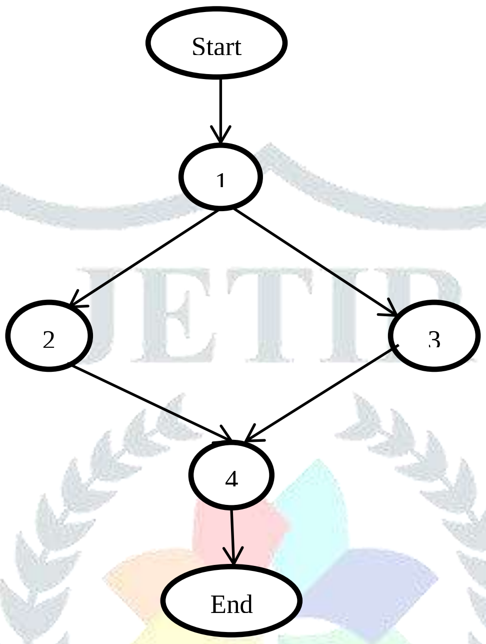

Fig. 3 CFG of ISPRIME

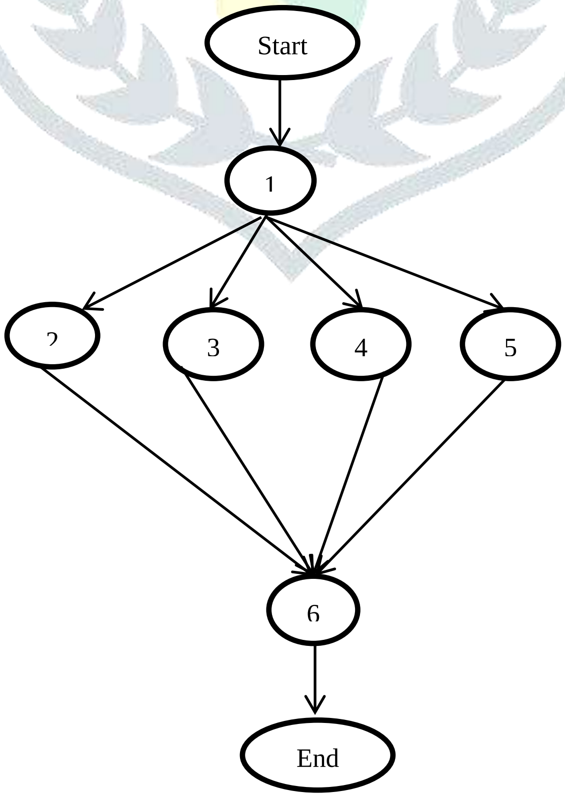

Fig.4 CFG of TC

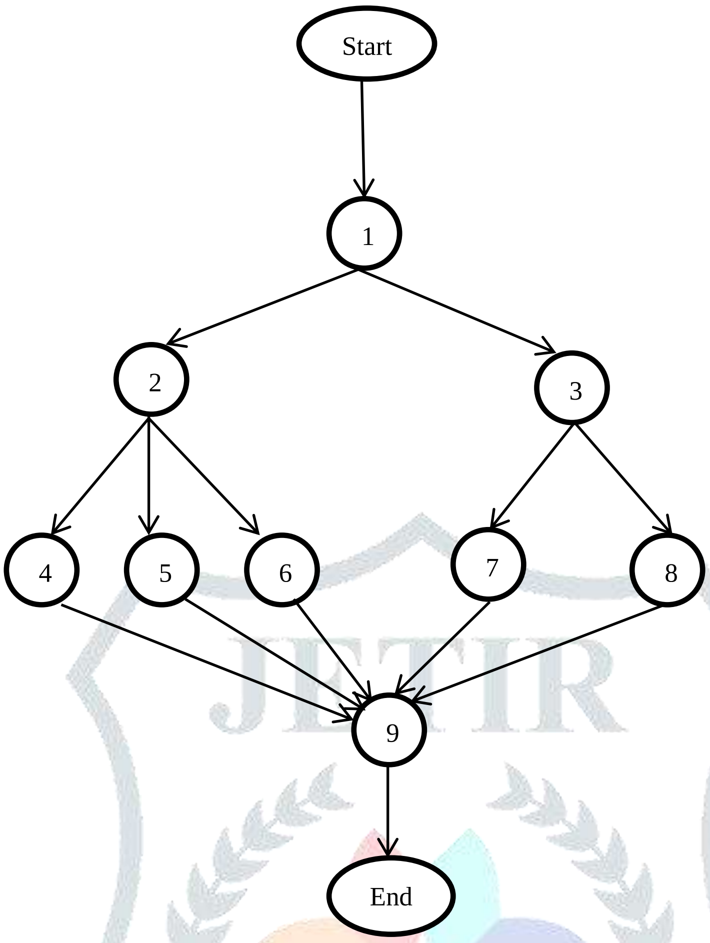

Fig. 5 CFG of QUAD

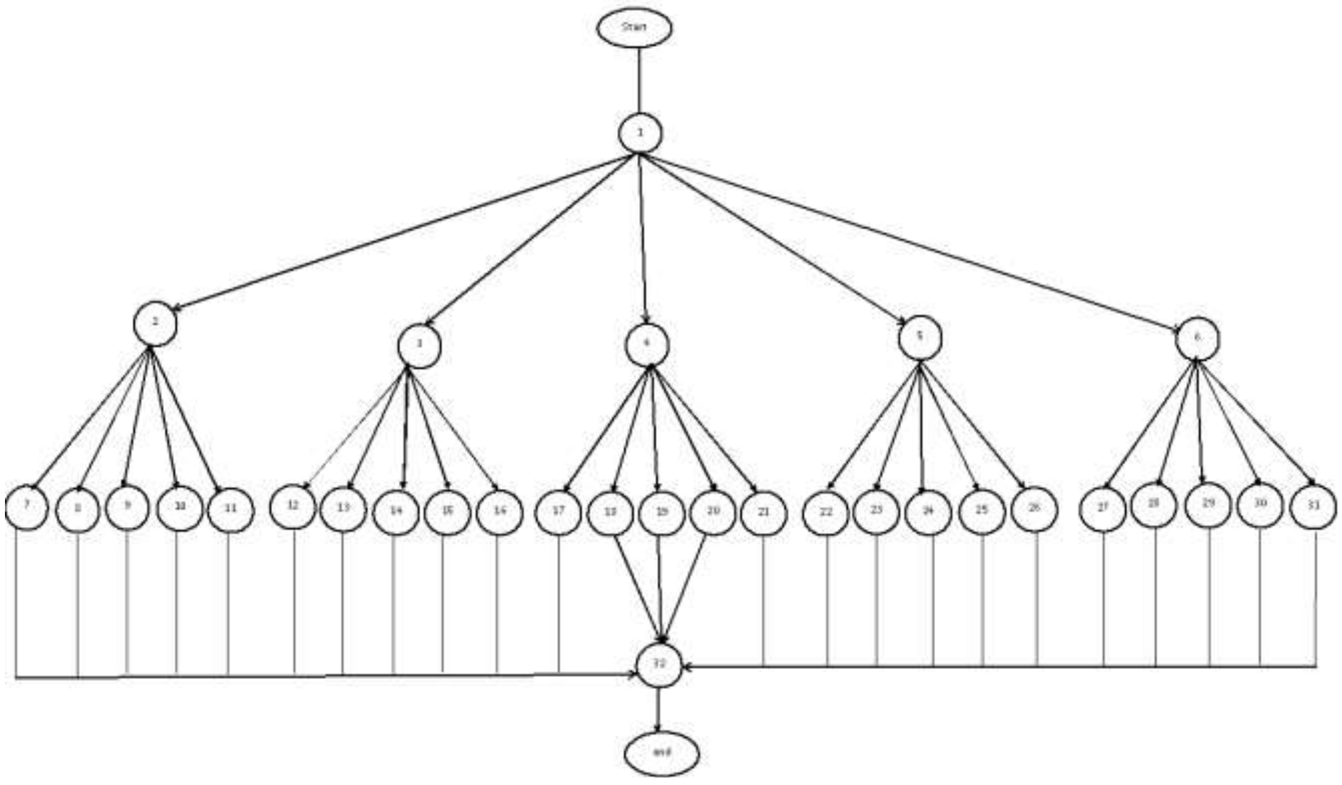

Fig. 6 CFG of CBCS

For each program, we can conducted 20 trails the average percentage coverage of each program shown in table 1

Table 1 Average percentage coverage for all progrms

Name of Program	APC by BB-BC	APC by PBB-BC
ISPRIME	85%	100%
QUAD	90%	100%
TC	85%	100%
CBCS	80%	95.6%

Table1 shows that the average percentage coverage for ISPRIME by BB-BC is 85% and by PBB-BC is 100% . The average percentage coverage for QUAD by BB-BC is 90% and by PBB-BC is 100%. The average percentage coverage for TC by BB-BC is 85% and by PBB-BC is 100%. The average percentage coverage for CBCS by BB-BC is 80% and by PBB-BC is 95.6%.
Experimental results of each program by both approaches shown in fig.7 in the form of chart

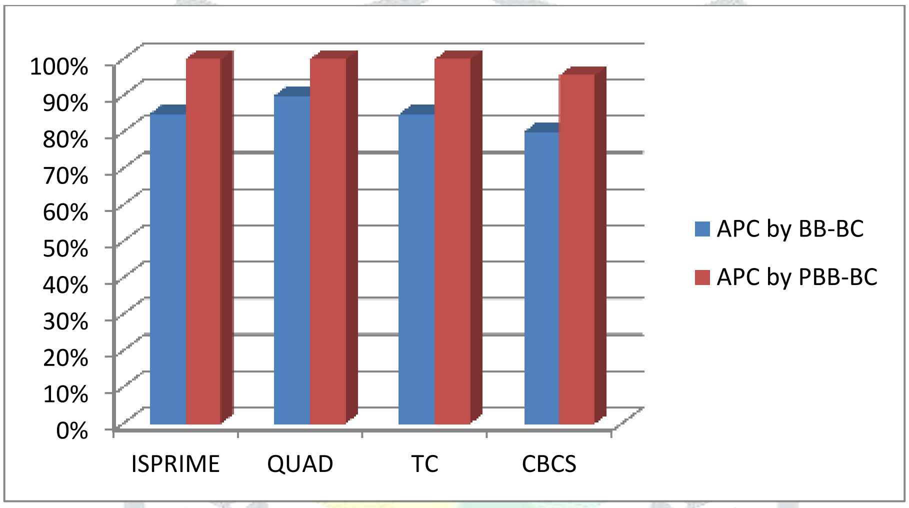

Fig.7 APC of all programs

VI. CONCULSION

Test data generation is most challenging issue in software testing. In this paper we proposed PB3C based technique for automatic test data generation. From the simulation and results we can conclude that PBB-BC approach gives better results than BB-BC approach for automatic test data generation. So, PBB-BC is best alternative for automatic test data generation.

REFERENCES

[1].C.Ghezzi, M.Jazayeri, and D.Mandrioli. Fundamental of software engineer Prentice Hall, Englewood cliffs, NJ, 1991.
[2].Shahid Mahmood. A Systematic Review of Automated Test Data Generation Techniques. 2007.
[3].Mansour, Nashat; Salame, Miran. Data Generation for Path Testing. Springer,2004.
[4].Bogdan Korel. Automated Test Data Generation for Programs with Procedures. In proceedings of the 1996 ACM SIGSOFT international symposium on Software testing and analysis '96, 209-215/96, Vol.21, No 03, ISSTA 1996.
[5].Pargas Roy P., Harrold Mary Jean, Peck Robert R. Test-data generation using genetic algorithms. Software Testing, Verification and Reliability 9, 263–282, 1999.
[6].Akos Hajnal and Istvan Forgacs. An Applicable Test Data Generation Algorithm for Domain Errors. In proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis ISSTA98; Vol. 23, No. 2, 63-72/98, ISSTA 1998.
[7].Arand Gotlieb, Bernard Botella, Michel Ruether. Automatic Test Data Generation using Constraint Solving Techniques. In proceedings of the 1998 ACM SIGSOFT international symposium on Software testing and analysis98; Vol. 23, No. 2, 53-62/98, ISSTA 1998.
[8].Huey-Der Chu, John E. Dobson, and I-Chiang Liu. FAST: a framework for automating statistics-based testing. Software Quality Journal 6; 13–36, 1997.
[9]. I. Sommerville. *Software Engineering*, 5th edn (Addison-Wesley, Wokingham, 1996).
[10]. Nargis Akhter, Dr. Amar singh. Literature Review Of Different Test Case generation Approaches, international Journal of creative research and thought vol.6 Issue 2, 2018.
[11]. Jon Edvardsson. A Survey on Automatic Test Data Generation. In *Proceedings of the Second Conference on Computer Science and Engineering in Linkoping*, pages 21-28. ECSEL, October 1999.
[12]. Shahid Mahmood. A Systematic Review of Automated Test Data Generation Techniques. *October 2007.*

[13]. Preeti, Kompal Ahuja. An Evolutionary Algorithm for Automated Test Data Generation for Software Programs. International journal for research in applied science and engineering technology Vol. 2 Issue III, March 2014.
[14].Shakti kumar, Sukhbir Singh Walia, Amar Singh. Parallel Big Bang-Big Crunch Algorithm. International Journal of Advanced Computing, vol.46, Issue.3, September 2013.
[15]. Shakti Kumar, Amar Singh and Sukhbir Walia. Parallel Big Bang–Big Crunch Global Optimization Algorithm: Performance and its Applications to routing in WMNs.Springer2018.