Optimization of Software Testing Technique using Novel Genetic Algorithm

Sudha Katkuri[1], Prof. P. Premchand[2]

[1]Research Scholar, Dept of Computer Science, Rayalaseema University, Kurnool, Andhra Pradesh, India
[2]Professor, Dept of Computer science and Engineering, UCE, Osmania University, Hyderabad, Telangana, India

DOI: 10.29322/IJSRP.10.04.2020.p10081
http://dx.doi.org/10.29322/IJSRP.10.04.2020.p10081

Abstract

Software Testing is a process performed to maintain software quality. Thus, the goal of testing is systematically and stepwise detection of different classes of errors within a minimum amount of time and also with a much less amount of effort. Software testing is also an important component of software quality assurance (SQA), and a number of software organizations are spending up to 40% of their resources on testing.

Software testing is a very broad area, which involves many other technical and non-technical areas, such as specification, design and implementation, maintenance, process and management issues in software engineering. The study focuses on the state of the art in testing techniques, as well as the latest techniques which represents the future direction in this domain.
Numerous genetic algorithms have been developed and implemented for optimizing the testing techniques.

In this research work, a novel genetic algorithm (GA) which combines Cuckoo search and Particle Swarm Optimization (PSO) is designed and implemented to optimize the testing technique.

For implementing this novel genetic algorithm (GA), the Booth's and Hump Camel functions are used to calculate global best evaluation. The global or final best evaluation is calculated based on the number of particles and epochs.

Key Words: Particle Swarm Optimization (PSO), Genetic Algorithm (GA), Cuckoo Search Algorithm, White Box Testing.

I. INTRODUCTION

Testing [1] is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. Testing [2] is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements.

1.1. Objectives of Software Testing

- A good test case is one that has a probability of finding an as yet undiscovered error.
- A good test is not redundant.
- A successful test is one that uncovers a yet undiscovered error [3].
- A good test should be "best of breed".
- A good test should neither be too simple nor too complex.
- To check if the system does what it is expected to do.
- To check if the system is "Fit for purpose".
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- Executing a program with the intent of finding an *error*.

Different phases of software testing life cycle are represented in the

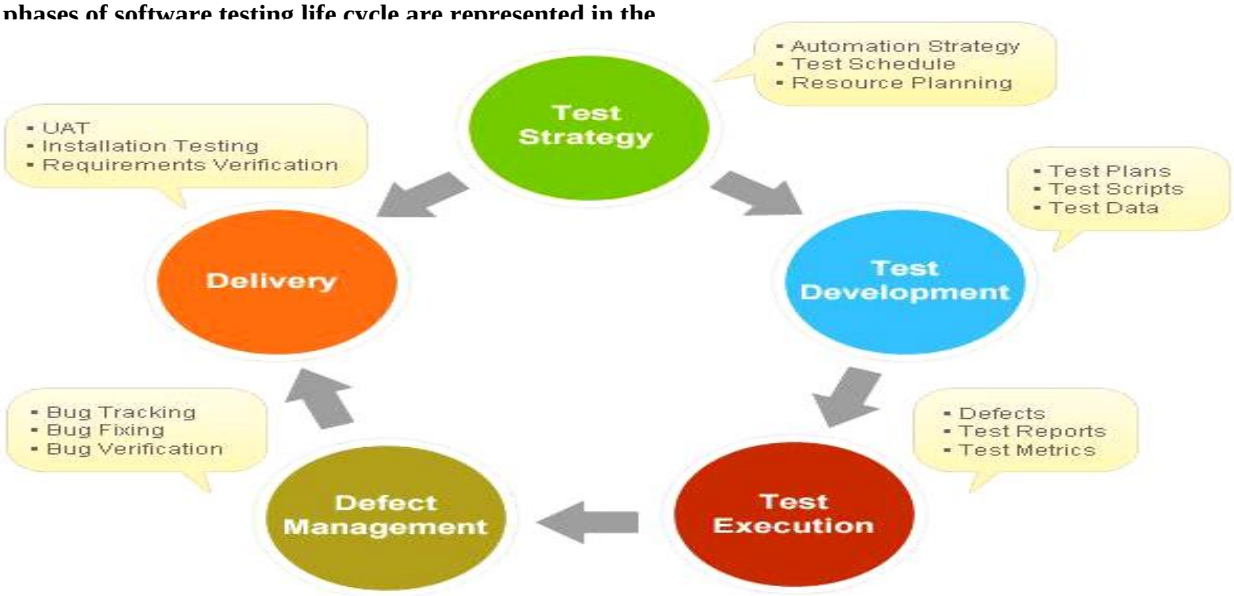

figure1.

Fig.1.:Testing activities

II. OBJECTIVES AND AIM

2.1. Objectives and aims of the research

The overall aim of this research project is to design a new Genetic Algorithm (GA) [6] that is cuckoo bird search algorithm, implement and investigate the effectiveness of Genetic Algorithm (GA) with regard to random testing and to automatically generate test data to traverse all branches of software. The objectives of the research activity can be defined as follows:

- The furtherance of basic knowledge required to develop new techniques[7] for automatic testing.
- To assess the feasibility of using GA to automatically generate test data for a variety of data type variables and complex data structures for software testing.
- To analyze the performance of GAs under various circumstances e.g. large systems.
- Comparison of the effectiveness of GAs with pure random testing for software developed in Java
- The automatic testing of complex software procedures.
- Analysis of the test data adequacy using mutation testing.

The performance of GAs in automatically generating test data for small procedures is assessed and analyzed. A library of GA is developed and then applied to larger systems.
The efficiency of GAs in generating test data is compared to random testing with regard to the number of test data sets generated and the CPU time required.

This research project presents a system for the generation of test data for software written in Java. The problem of test data generation is formed and solved completely as a numerical optimization problem using Genetic Algorithm and structural testing techniques.

Software testing is about searching and generating certain test data in a domain to satisfy the test criteria. Since GAs are an established search and optimization process. The basic aim of this research is to generate test sets which will traverse all branches in a given procedure under test.

Testing criteria

The criterion of testing [12] in this thesis is branch testing, the aim is to develop a test system to exercise every branch of the software under test. In order to generate the required test data for branch testing Genetic Algorithm, PSO and Cuckoo search are used. Flow chart for genetic Algorithm process is represented in figure 2.

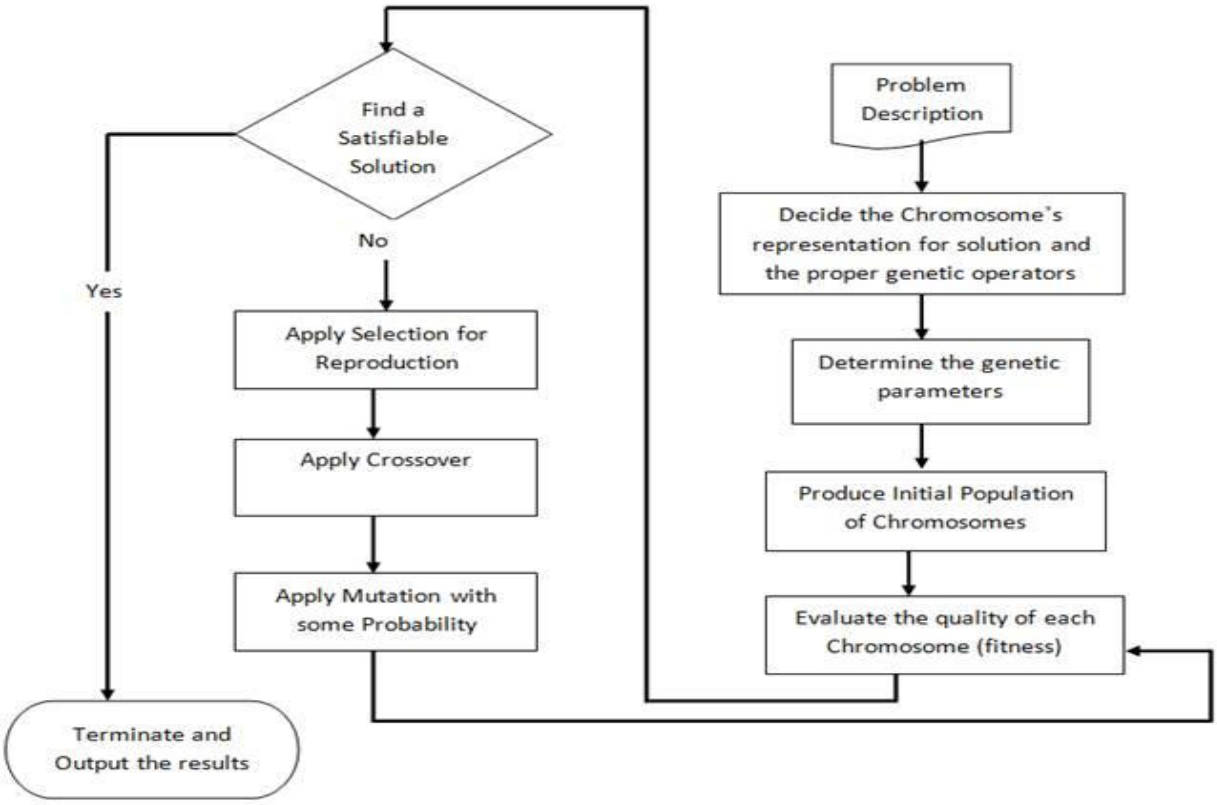

Fig. 2. Flow chart for genetic Algorithm Process

III. Methodology in this research work

In this work, The Particle Swarm Optimization (PSO), Cuckoo search and Genetic Alogrithm (GA) are used for designing and Implementing a Novel Genetic algorithm for Optimization of Testing Technique.

Structural testing can be done by the method of data flow testing or path testing [8]. Path testing comprises generating a set of paths that will cover every branch in the program and discover the set of test cases that will execute every path in this set of program path. In data flow testing, the emphasis is on the points at which variables obtain values and the points at which these values are used up.

In the area of structural or white-box testing [9], evolutionary algorithms can assist in finding test cases which cover the code base under test to maximum extent. The aim is to execute the code under test with as many different input parameters as possible, in order to maximize the chance of detecting errors in the code.

The actual aim of this work is to offer better optimization approach, which is introduced in the test case by using Genetic Algorithm [10]. Optimization approach adapted with different layer tasks to inspect. This research also provides a survey to determine better quality testing process within the time. In this study we analyze how test cases can be optimized and gives best solution. Evolutionary tests generally are a very good growing methodology associated with routinely bringing in high quality analyzes information. The actual evolutionary algorithms are now put on inside many correct living problems.

In this work, we proposed a new approach to optimize the software testing techniques [11] by test case suite reduction. The proposed technique is based on concepts of PSO, Cuckoo search and GA. The technique selects the set of test case from the available test suite that will cover all the faults detected earlier in minimum execution time. Here particles are used as agents who explore the minimum set of test cases. Half of the particles will initially start foraging with randomly selected test cases. Now particles will add new test cases on the explored path if adding of a test case increases its fault detection capacity.

Population based search protocol can turn out to be Particle swarm Optimization (PSO). PSO was designed on the basis of the social presentation of birds in a flock. Each particle flies in the search space with a velocity varied by its own flying memory and its companion's flying experience in PSO. Each particle has its major function value which is settled on by a fitness function.

3.1.Particle Swarm Optimization

PSO is an optimization seeks technique based on population. It is an innovative branch of evolutionary computation and the particle swarm can be seen as a simple social system. In PSO, each individual is called a particle and each particle denotes a potential solution, which forms the population set. The particles in searching space influence each other, exchanging information, to adjust the location and speed of itself and approach to the optimal solution. PSO initialize a group of particles, finding the optimal solution by iteration. During each iteration, the particle updates its speed and location by tracing individual extremism and global extremism

vi = {vi1, vi2, …, vid} denotes the speed of the i^{th} particle;
Xi = {xi1, xi2, …, xid},
i = 1, 2, …, N denotes a vector point of i^{th} particle in d-dimension solution space, t is the iteration number;
[Math Processing Error] denote the location after t^{th} iteration and d^{th} dimensional component of speed vector of i^{th} particle,
r1 and r2 are random numbers obeying the distribution of U(0, 1);
c1 and c2 are accelerate factors and usually c1 = c2 = 2.
We use Pi = {pi1, pi2, …,pid} to record the optimal point that is searched by the i^{th} particle, as pbest.
Thus there must exist an optimal point in the population, whose number is g.
Then Pg = {pg1, pg2, …,pgd} is the optimal value of the population search.

During the process of particle optimization, it is crucial to balance the local developing ability and the global detecting ability. For different problems the balance of these two abilities are not the same.

PSO algorithm has deficiency in premature convergence, and the local search accuracy is low. It also brings factors of program structure in testing case generation. Therefore the PSO-based testing case generation technology is discussed intensively in this article. To balance the ability of exploring and self-improvement of algorithms, and to achieve better convergence speed in global search, we provide adaptive particle inertia weight factor adjusting approach, integrated with fitness and particle aggregation degree. During evolution, a local searching strategy is performed on the optimal individual of each generation to further improve the efficiency of testing case generation.

The simulations indicate that Solving Particle Swarm Optimization with Local Search (SPSOLS) algorithm proposed in this article shows better testing case generation performance and it achieves coverage rate optimization of the tested cases. It also has certain advantages in testing case generation [14] compared to homogeneous algorithm under approximate environment. Executing the function under test using the input variables from a particular test case causes a particular control path through the function to be taken. In the case of the branch coverage metric and assuming the function under test contains branch statements, the function will typically need to be executed using several test cases in order to exercise (cover) each branch. For small functions containing few branch statements, the task of finding test cases, which use all branches, is relatively simple. For more complex functions with many branch statements and input variables it makes sense to automate the task, and one approach is to use evolutionary algorithms. Evolutionary algorithms use the principles of evolution to achieve optimization based on the result of a fitness function. The fitness of a first generation of random individuals is tested, and the characteristics, known as genes, of the fittest individuals are propagated to the next generation. This process is governed by rules regarding which percentage of individuals whether their genes have propagated to the next generation, how their genes are combined to form the next generation’s individuals and how the genes are randomly mutated. Figure 3 shows the flow chart for particle swarm optimization process.

Figure. 3: Flow chart for Particle Swarm Optimization

"Individual best": It is the individual best choice algorithm by assessing each individual position of the particle to its own best position *pbest*, only. The data about the other particles is not used in this ***pbest***.

"Global best": It is the universal best selection algorithm, which obtains the global information by making the movement of the particles encloses the position of the best particle from the complete swarm. Additionally, every particle uses its experience with earlier incidents in terms of its own best solution.

- **Swarm initialization**: For a population size u, arbitrarily generate a solution.
- **Define the fitness f unction**: According to the current solution, the fitness function chosen should be used for the constraints.
- *gb* **and** *pb* **Initialization:** In the start the fitness value estimated for every particle is placed as the Pbest value of each particle. Among the Pbest values, the optimal one is chooses as the *gb* value.
- **Velocity Computation:** The new velocity is calculated using the beneath equation

v[ ] = v[ ] + c1 * rand( ) * (pbest[ ] - present[ ]) + c2 * rand( ) * (gbest[ ] - present[ ])…. (a)

present[ ] = present[ ] + v[ ] …… (b)

v[ ] is the particle velocity, present[ ] is the current particle (solution).

pbest[] and gbest[] are defined as stated before.

rand () is a random number between (0,1). c1, c2 are learning factors. usually c1 = c2 = 2.

- Swarm Updating: Find out the fitness function once more and improve the pb and gb values. If the new value is better than the earlier one, replace the old by the current one. And as well select the optimal pb as the gb .
- Criterion to stop: Extend till the solution is really appropriate or maximum iteration is attained is

The pseudo code of the procedure is as follows

For each particle
  Initialize particle
END
Do
  For each particle
    Calculate fitness value
    If the fitness value is better than the best fitness value (pBest) in history
      set current value as the new pBest
  End
  Choose the particle with the best fitness value of all the particles as the gBest
  For each particle
    Calculate particle velocity according equation (a)
    Update particle position according equation (b)
  End
While maximum iterations or minimum error criteria is not attained

Particles' velocities on each dimension are clamped to a maximum velocity Vmax. If the sum of accelerations would cause the velocity on that dimension to exceed Vmax, which is a parameter specified by the user. Then the velocity on that dimension limited to Vmax.
Then we apply the cuckoo search algorithm

3.2.Cuckoo search algorithm

Cuckoo search algorithm is a metaheuristic algorithm which was inspired by the breeding behavior of the cuckoos and alleviates to implement. There are a number of nests in cuckoo search. Each egg points out a solution and an egg of cuckoo indicates a new solution. The new and better solution is replacing the most awful solution in the nest. The subsequent representation scheme is selected by Cuckoo Search algorithm: Each egg in a nest symbolizes a solution, and a Cuckoo egg symbolizes a novel solution. The plan is to employ the novel and probably better egg to substitute a not- so-good egg of Cuckoo in the nests. On the other hand this is the fundamental case i.e., one cuckoo per nest, but the extent of the approach can be raised by incorporating the property that each nest can have more than one egg which symbolizes a set of solutions. The process of clustering is given beneath,

- At a time only one egg is laid by cuckoo. Cuckoo dumps its egg in a arbitrarily selected nest.
- The number of accessible host nests is fixed, and nests with high quality of eggs will transmit over to the next generations.
- In case of a host bird found out the cuckoo egg, it can throw the egg away or discard the nest, and build a totally novel nest.

Step 1: Initialization Phase
The population (mi, where i=1, 2,…n) of host nest is started arbitrarily.

Step 2: Generating New Cuckoo Phase
Using levy flights a cuckoo is selected at random and it produces new solutions. After that the produced cuckoo is evaluated using the objective function for finding out the quality of the solutions.

Step 3: Fitness Evaluation Phase
Evaluate the fitness function based on the equation and next select the best one.

Step 4: Updating Phase
The superiority of the new solution is evaluated and a nest is selected among arbitrarily. If the excellence of new solution in the selected nest is better than the old solutions, it will be replaced by the new solution (Cuckoo). Otherwise, the previous solution is placed aside as the best solution.

Step 5: Reject Worst Nest Phase
The worst nests are thrown away in this part, based on their chance values and new ones are built. Presently, function the best solutions are ranked based on their fitness. After that the best solutions are identified and spotted as optimal solutions.

Step 6: Stopping Criterion Phase

Till the maximum iteration achieves this process is repeated. The optimized effect will be inspected for the measure of software quality. The précised process is clearly shown in flowchart in figure 4.

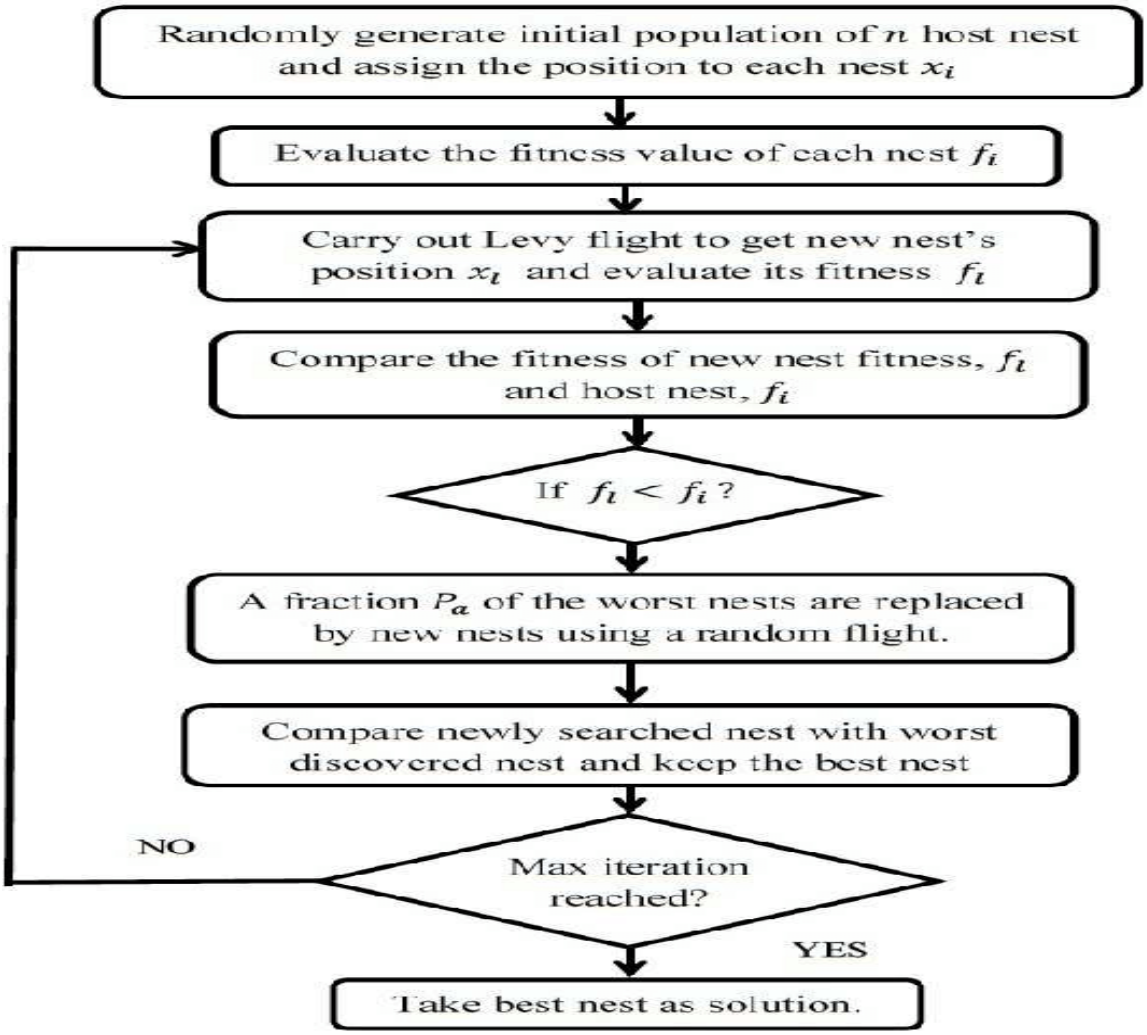

Figure. 4: Flow chart for Cuckoo search algorithm

IV. Implementation

The implementation of the algorithm is demonstrated in the following screens with the help of Booth's function and Three Hump Camel function.

The functions used in this work for optimization are as follows :

$$f(x) = -a\exp\left(-b\sqrt{\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d} cos(cx_i)\right) + a + \exp(1)$$

In the above equation, the values a, b and c are constants and are usually chosen as a=20, b=0.2 and c=2π.

On a 2-dimensional domain it is defined by:

$$f(x,y) = -20\,exp\left[-0.2\sqrt{0.5(x^2+y^2)}\right] - exp\left[0.5(cos2\Pi x + cos2\Pi y\right] + e + 20$$

4.1.Booth's Function

The Booth function is a unimodal, 2-dimensional convex mathematical function widely used for testing optimization algorithms.

Booth Function

Number of variables: n = 2.

Search domain: $-10 \leq x_i \leq 10$, i = 1, 2.

Number of local minima: several local minima.

Booth's Function

$$f(x,y) = (x + 2y - 7)^2 + (2x + y - 5)^2$$

4.2.Three Hump Camel Function

This function is used as a test function in order to evaluate the performance of optimization algorithms

$$f(x,y) = 2x_1^2 - 1.05\,x_1^4 + \frac{x_1^6}{6} + x_1x_2 + x_2^2$$

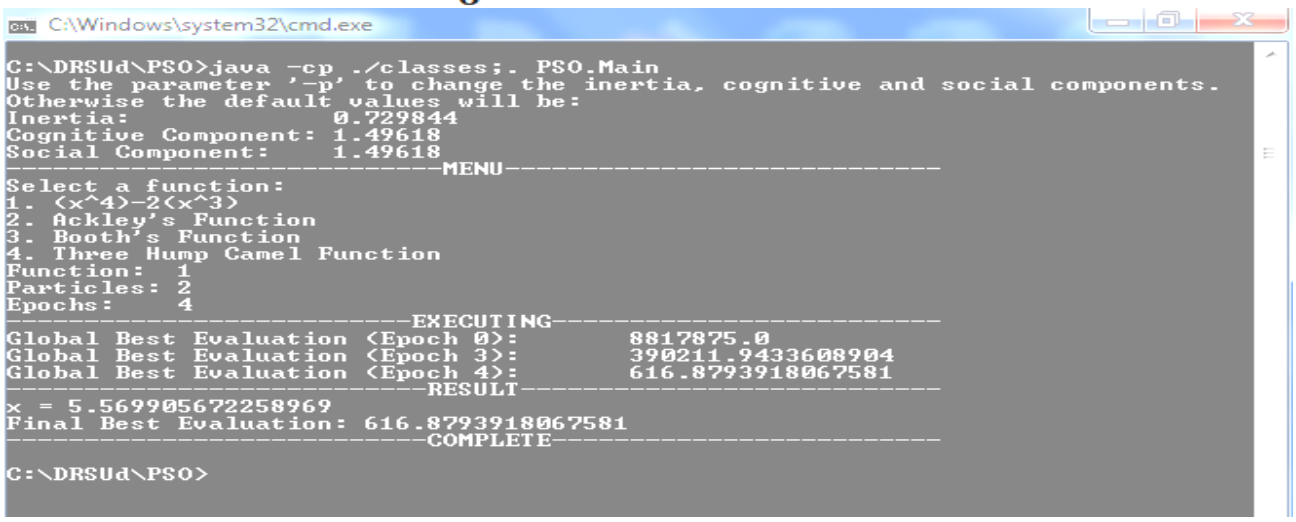

Fig 5. Find the Global Best Evaluation

The above figure displays a menu to calculate the global best.
Here the result of global best evaluation is obtained using the quadratic formula
X*4-2(x*3)

Using Particle Swarm optimization no of particles taken here is 2 and number of epochs is chosen as 4.
Final Best Evaluation calculated is 616.87939

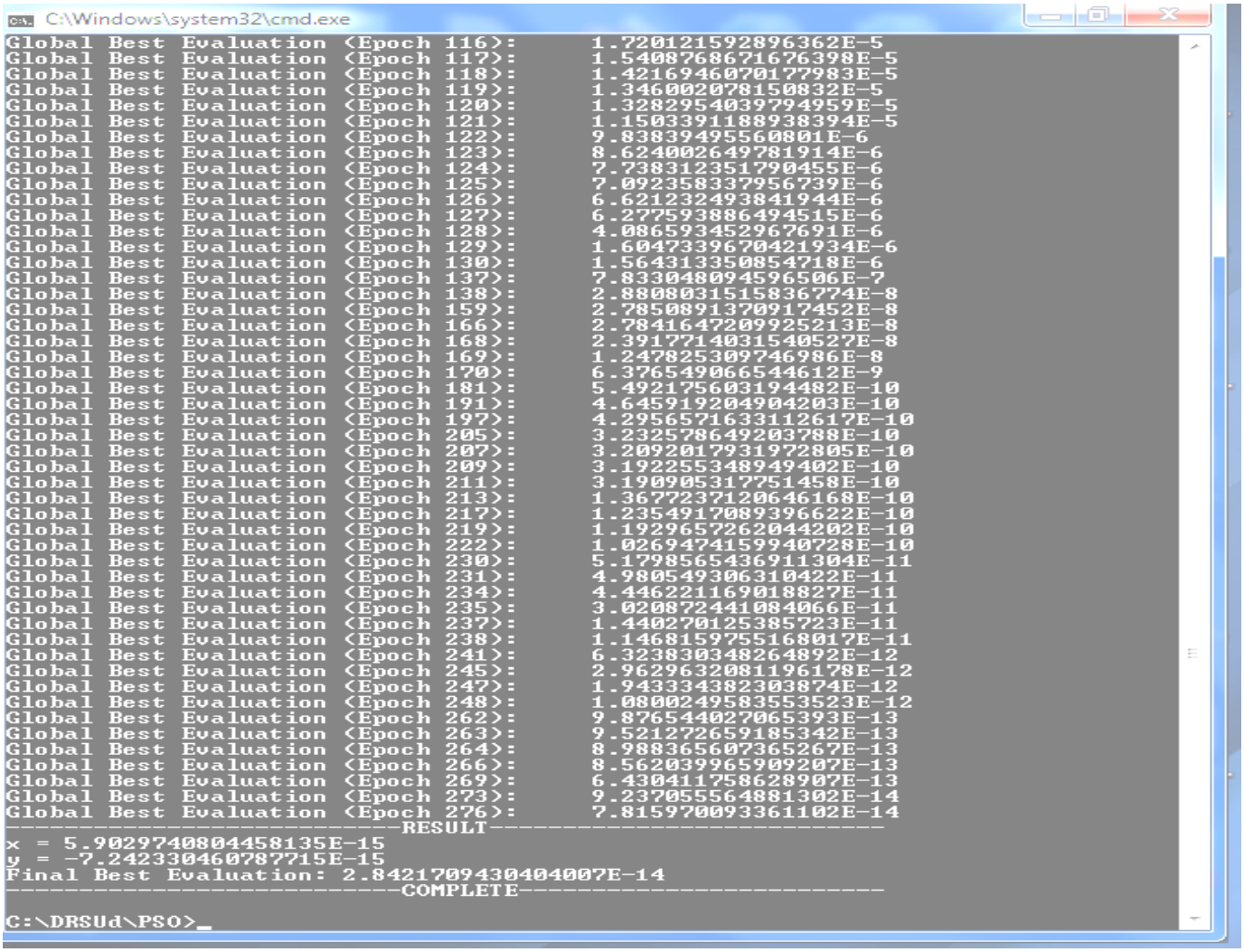

Fig 6. Find the Global Best

The above figure 6 displays a menu to calculate the global best evaluation and here it is computed as 2.8421709430404007E-14.

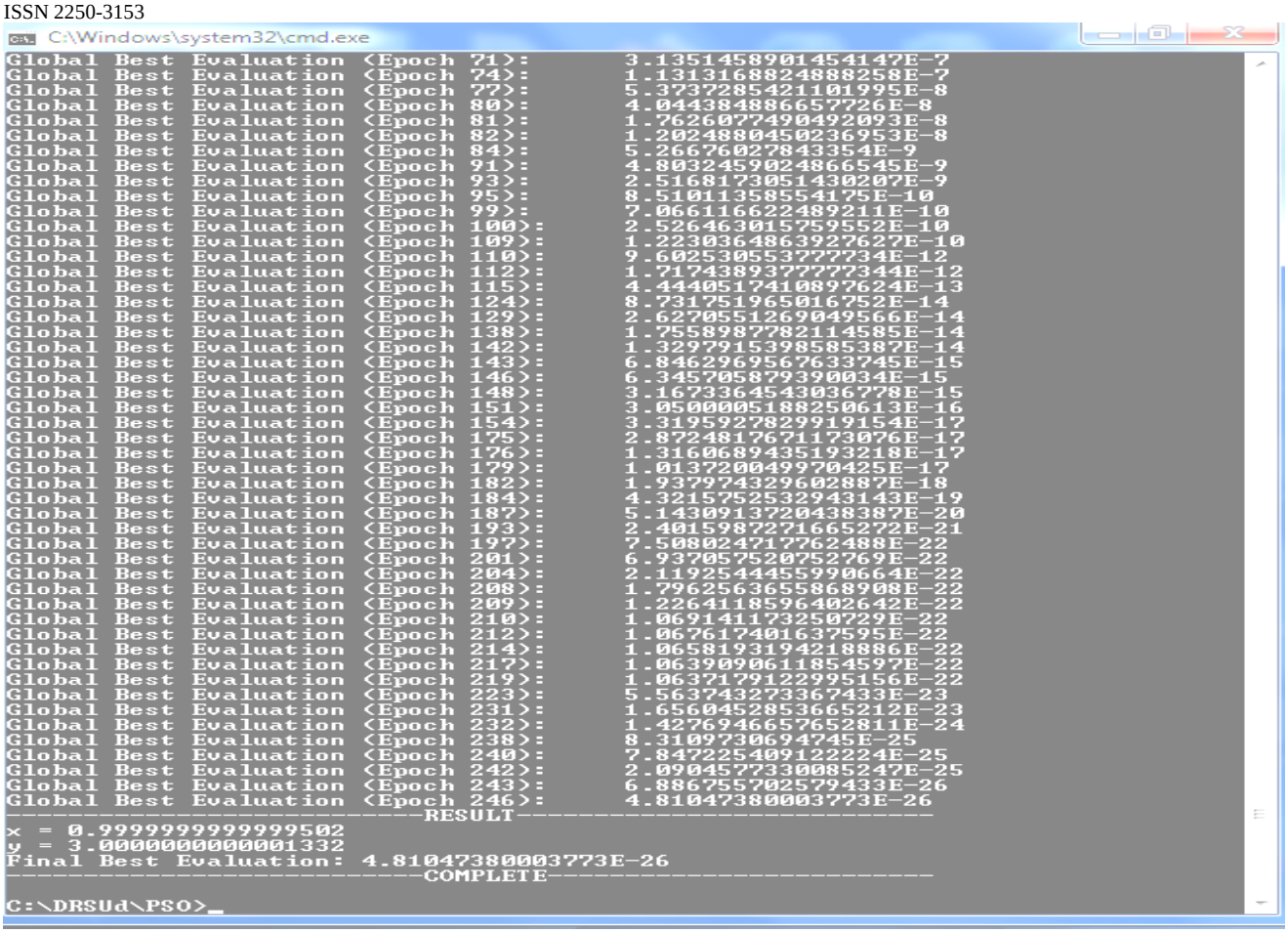

Fig 7. Find the Global Best

The above figure 7 demonstrates a menu to calculate the global best evaluation and here it is computed as 4.81047380003773E-26.

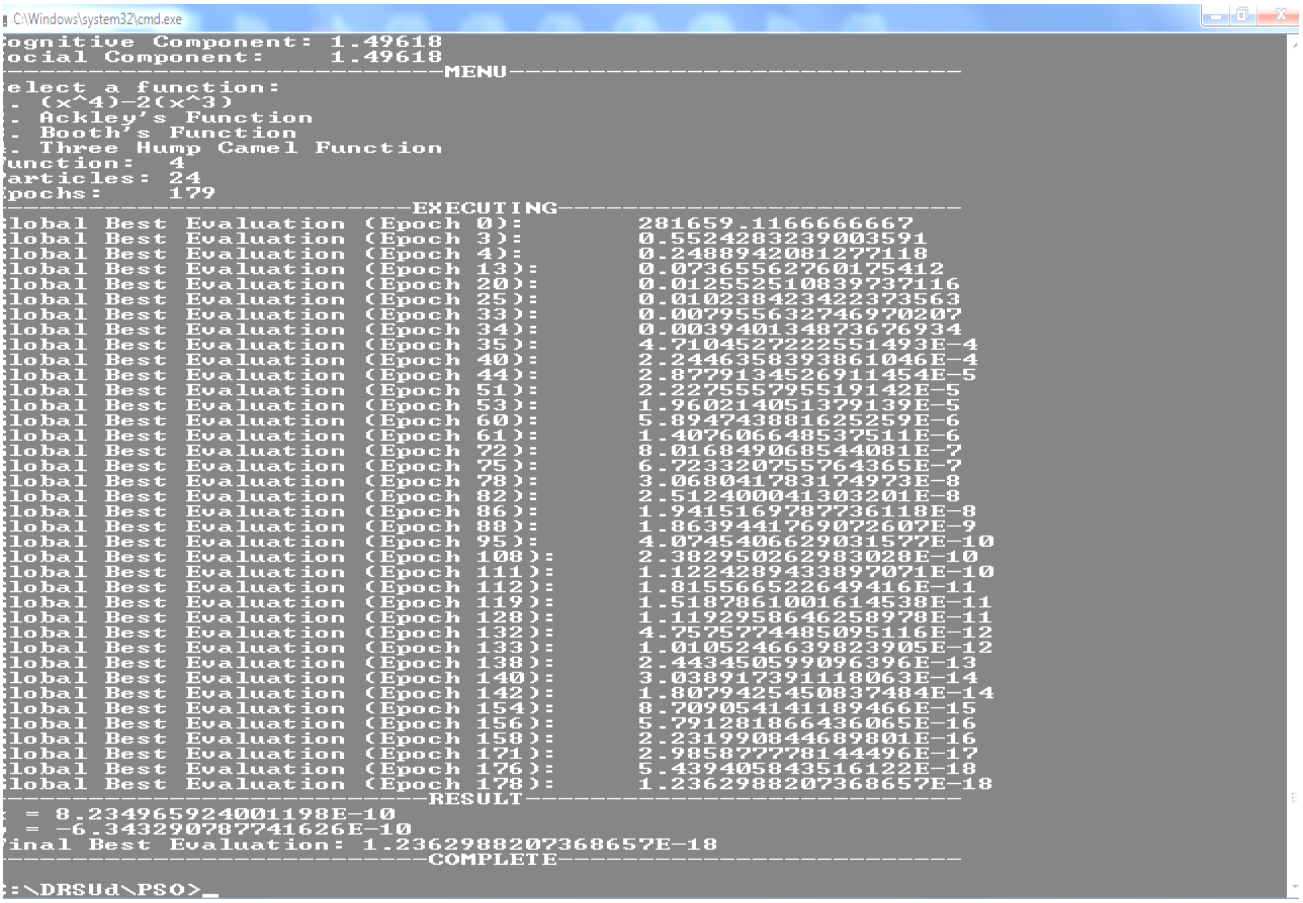

Fig 8. Find the Global Best

The above figure 8 demonstrates a menu to calculate the global best evaluation and here it is computed as 4.81047380003773E-26. The function used is Three Hump camel Function, particles used are 24 and epochs are 179.

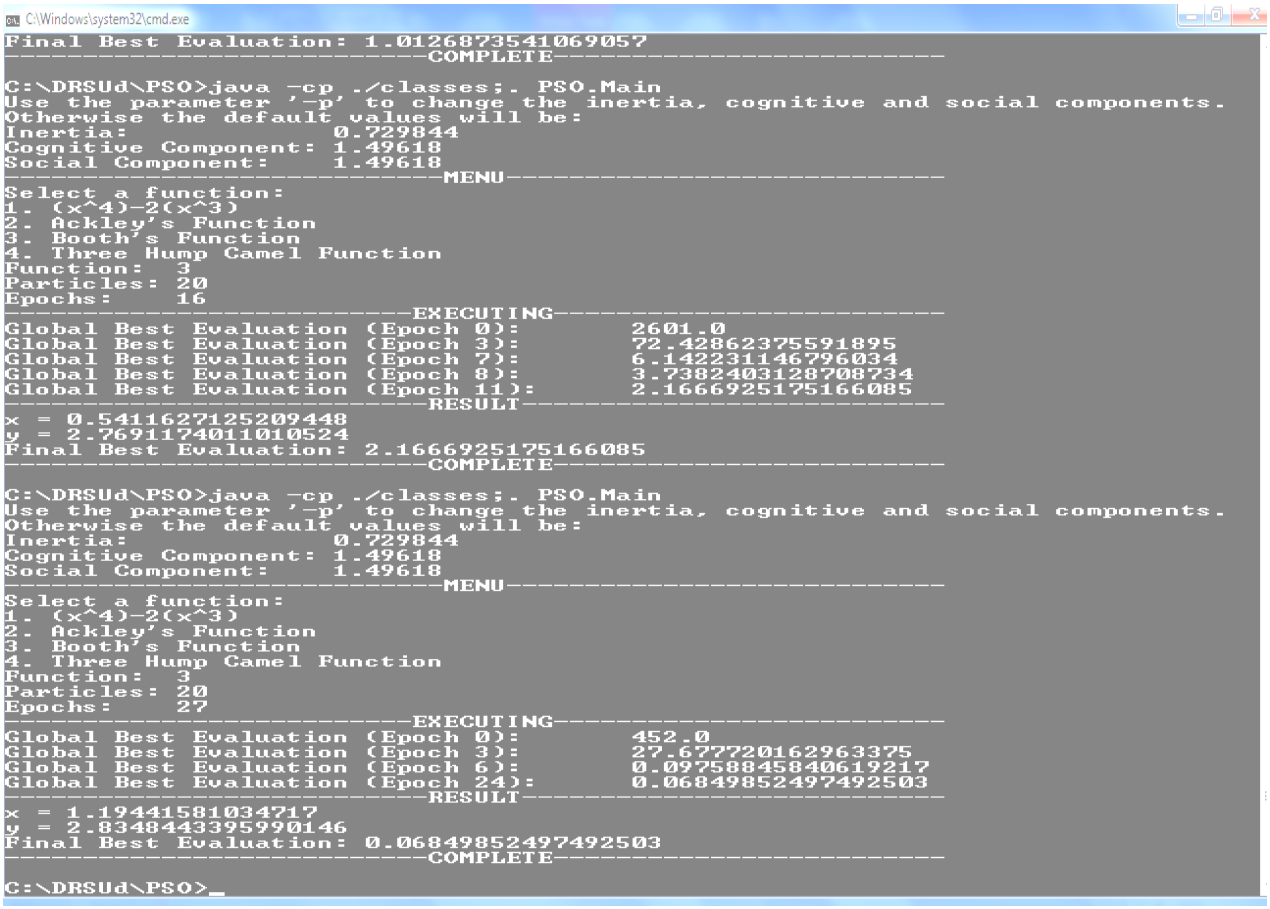

Fig 9. Find the Global Best

The above figure 9 displays a menu to calculate the global best evaluation with Booth's function, particles 20 and with 16 and 27 epochs.
Final best evaluation is 2.1666925175166085, particles are 20 and epochs are 16.
Final best evaluation is 0.06849852497492503, particles are 20 and epochs are 27.

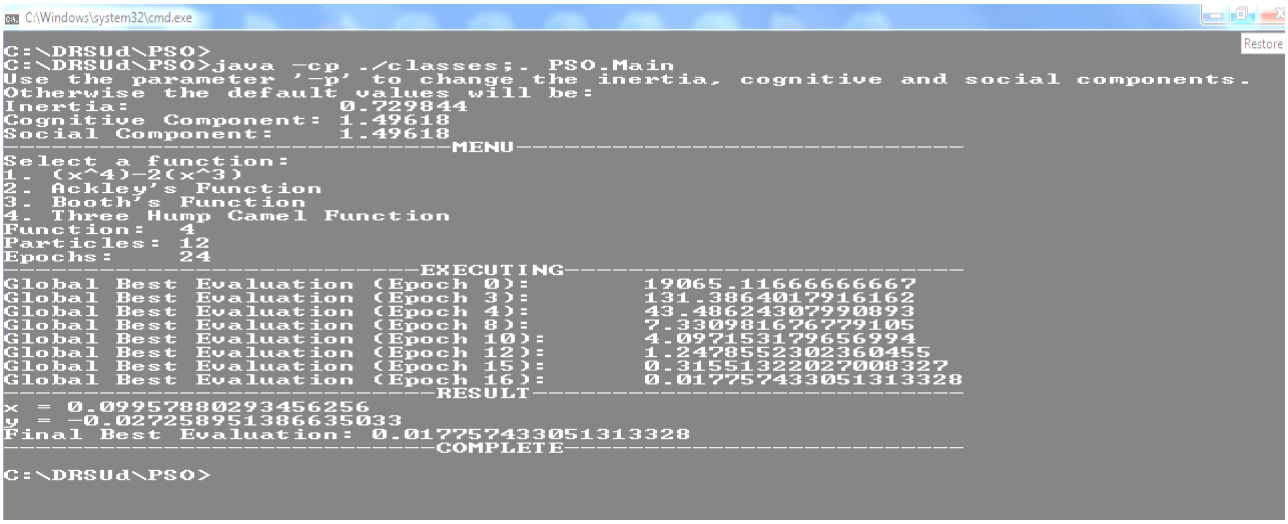

Fig 10. Find the Global Best

The above figure 10 displays a menu to calculate the global best evaluation with Hump Camel function, particles 12 and with 16 and 24 epochs.
Final best evaluation computed is 0.017757433051313328.

V. CONCLUSION

The main objective is to minimize the time, cost and effort required to test software where we need to implement certain evolutionary algorithms in software testing.

Genetic algorithm is one such evolutionary algorithm which can optimize the problem. The implementation study says that this novel genetic algorithm gives better results to increase quality of software by discovering errors. The overall aim was to develop a self-learning algorithm that has the ability to process and incorporate new information as the work environment changes (changing from one predicate to the next).

The GAs gives good results and their power lies in the good adaptation to the various and fast changing environments.

The primary objective of this research work was to propose a GA-based software test data generator and to demonstrate its feasibility. The main advantage of this novel Genetic Algorithm (GA) developed testing tool is the strength of GAs, in that the relationship between the predicate under investigation and the input variables may be complex and need not be known. Instrumentations capture the actual values of the predicates, so that the fitness can be based on these values.

In this research work a novel genetic algorithm (GA) is proposed, designed and implemented. The simulated results are shown in output screens. Global best evaluation is computed with the help of Booth's and Hump Camel functions.

All tests clearly show that the test data generated from Novel Genetic Algorithm are significantly better than randomly generated data.

REFERENCES

[1] Stacey, D. A.(2004), Software Testing Techniques Guide to the software, Engineering Body of Knowledge, Swebok – A project of the IEEE, Computer Society Professional Practices Committee.
[2] R.S. Pressman & Associates, Inc. (2005).
[3] Software Engineering: A Practitioner's Approach, 6/e;
[4] Software Testing Techniques, Myers, Glen ford J.(1979), IBM Systems Research Institute, Lecturer in Computer Science, Polytechnic Institute of New York, The Art of Software Testing, by John Wiley & Sons, Inc. Redmill, Felix (2005),
[5] Theory and Practice of Risk-based Testing, Vol. 15, No. 1. IEEE (1990), IEEE Standard Glossary of Software Engineering Terminology, Los Alamitos, CA: IEEE Computer Society Press.
[6] Software testing by Jiantao Pan available at http://www.ece.cmu.edu/~roopman/des-899/sw_testing/
[7] Software testing techniques available at http://pesona.mmu.edu.my/~wruslan/SE3/Readings/ GB1/pdf/ch14-GB
[8] White box testing from Wikipedia, the free encyclopedia.
[9] Software testing for Wikipedia available at http://en.wikipedia.org/wiki/grey_box_testing#grey_box_testing'
[10] Baker J. E.: 'Adaptive Selection Methods for Genetic Algorithms', Proceedings of the first international Conf. on Genetic Algorithm and their Applications, pp. 101-111, 1985.
[11] Bayer S. E. and Wang L.: 'A genetic Algorithm programming environment: Splicer', 3rd Intern. Conference on Tools for Artificial Intelligent [Cat.No.: 31ch30544] USA 1991; ISBN 0818623004, Part Nov, pp. 138-44, November 1991.
[12] Beizer, B.: 'Software Testing Techniques', Second Edition, New York: van Nostrand Rheinhold, ISBN 0442206720, 1990.
[13] Chayanika Sharma, Sangeeta Sabharwal, Ritu Sibal ,Software Testing techniques using Genetic Algorithm, IJCSI International Journal of Computer Science Issues, Volume 10, Issue 1,2013, 1694-0784.
[14] Moheb R. Girgis, Automatic Test Data Generation for Data Flow Testing, Using a Genetic Algorithm, Journal of Universal Computer Science, volume 11, Issue 6,2005, 898-915.