2012 Third International Conference on Emerging Intelligent Data and Web Technologies

Automatic Test Data Generation for Software Path Testing using Evolutionary Algorithms

Genţiana Ioana Laţiu, Octavian Augustin Creţ, Lucia Văcariu
Computer Science Department
Technical University of Cluj-Napoca
Cluj-Napoca, Romania
e-mail: gentiana.latiu@cs.utcluj.ro, octavian.cret@cs.utcluj.ro, lucia.vacariu@cs.utcluj.ro

Abstract — Software testing is a very expensive and time consuming process. Test methods which generate test data based on the program's internal structure are intensively used. This paper presents a comparison between three important Evolutionary Algorithms used for automatic test data generation, a technique that forces the execution of a desired path of the program called target path. Two new approaches, based on Particle Swarm Optimization and Simulated Annealing algorithms, used in conjunction with the approximation level and branch distance metrics, are compared with Genetic Algorithms for generating test data. The results obtained based on the proposed approaches suggest that evolutionary testing strategies are very well suited to generate test data which cover a target path inside a software program.

Keywords — software testing, evolutionary algorithms, path testing.

I. INTRODUCTION

Software testing is the most significant analytic quality assurance measure for software products [1]. It is an expensive process which increases the total development cost of a software product. The automation process of test-data generation is an important step in reducing the cost of software development and maintenance [2].

A promising achievement in Software Testing is considered to be the use of Evolutionary Algorithms for structural testing. The structural testing method uses the internal structure of the software application in order to derive test cases. Application of Evolutionary Algorithms in Software Testing is often called in the literature Evolutionary Testing. The first work in applying Evolutionary Algorithms to generate data for structural testing was presented by McMinn in [3]. The aim of applying evolutionary testing to software testing is the generation of a quantity of test data, leading to the best possible coverage of the respective structural test criterion [1]. The main benefit of evolutionary testing is that it can be applied for software products which have a large input domain.

For the Evolutionary testing to succeed, it needs to be given some guidance (heuristics), represented in the form of a cost function that links a program input to a measure of how "good" it is, where "good" means the appropriateness of the current individual to the problem solution [4]. Evolutionary testing has been used by Korel for generating test data, which should traverse a selected path inside the program [5]. Korel's method uses data flow graph and a function minimization approach to generate test data which is based on the execution of the software product under test [6]. In the work of Watkins, Roper, Weichselbaum and Pargas et al. [1] the fitness of an individual is determined based on the coverage percentage, that is measured for the associated individual data, i.e. test data sets that cover more program branches than others are assigned higher fitness values.

The research presented in this paper builds on automatic generation of test data for covering a selected path. In Section II a theoretical background related to software testing and evolutionary algorithms is presented. Section III discusses the application of three Evolutionary Algorithms (Genetic Algorithms, Simulated Annealing Algorithm and Particle Swarm Optimization Algorithm) for path testing. Section IV presents experimental results obtained by applying Evolutionary Algorithms for target path testing on a number of commonly known programs that are used as benchmarks for comparison purposes. Section V concludes the paper and presents directions for future research.

II. THEORETICAL BACKGROUND

A. Software Testing

The definition of software testing was given by Miller: "The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances" [7]. Software testing procedure also represents a dynamic analysis of a software program, requiring execution of the program in order to produce results which are then compared with the expected outputs [8]. Software testing may be categorized in three major classes: black-box testing, white-box testing and grey-box testing [9].

Black-box testing is referred in the literature as functional testing, because it focuses on the program's outputs generated in the response to certain input data and execution conditions. With black-box testing technique, the software tester does not have access to the source code of the software product. Based on the product's requirements, the tester knows what to expect the black box to yield. For this testing

978-0-7695-4734-3/12 $26.00 © 2012 IEEE
DOI 10.1109/EIDWT.2012.25

procedure, testers just utilize functional requirements of the software product for designing the test cases.

White-box testing methodology is referred in the literature as structural testing, because it takes into account the internal structure of the system. During white-box testing, the tester is aware of the internal structure of the software product and designs test cases by executing different methods with specific parameters.

Grey-box testing is a software testing technique that uses a combination of black-box testing and white-box testing techniques. Grey-box testing is just partially based on the internal structure of the software product. During grey-box testing, the tester tests the software product from the outside, but he/she is better informed because he/she is partially aware of the internal structure of the software product.

B. Evolutionary Algorithms

Evolutionary Algorithms are stochastic iterative procedures for generating tentative solutions for a given problem. The algorithms manipulate a collection of individuals, each of which comprises one or more chromosomes [10].

C. Genetic Algorithms

Genetic Algorithms (GAs) are a subclass of the Evolutionary Algorithms, because they are inspired from biological evolution processes. GAs were first introduced in 1970s by Holland [11] and are currently used for solving optimization and search problems. A basic GA is composed of the steps shown in Fig. 1.

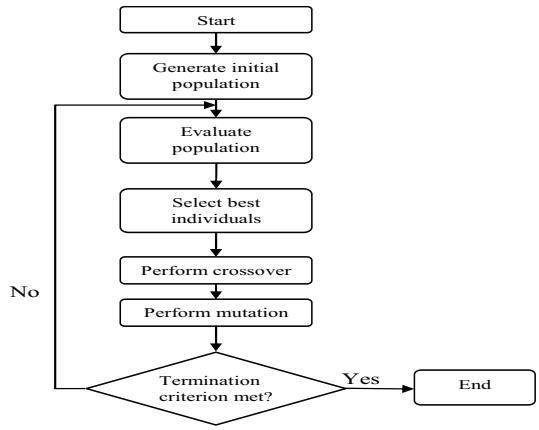

Figure 1. Genetic Algorithms

During the GA process, the population is represented by chromosomes. The main idea for GAs is to guide a population of individuals from an initial collection of values to a point in the solutions space where the fitness function is optimized, i.e. has the biggest value.

The main operators used by GAs are: *selection*, *crossover* and *mutation*. Through the *selection* process, the best individuals are chosen to form the next population of individuals. There are many methods of selection: roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others [12]. *Crossover* and *mutation* are two operators that strongly influence the performance of the GA. *Crossover* selects genes from parent chromosomes and creates new offsprings. *Mutation* changes the new offspring by randomly selecting one gene and changing it from 1 to 0 or 0 to 1. The crossover and mutation procedures should be performed with some probability, which represents *crossover* and *mutation rates*.

D. Simulated Annealing

Simulated annealing (SA) is a probabilistic search method proposed by Kirkpatrick, Gelett and Vecchi [13] in 1983 and by Cerny in 1985 [14] for finding the global minimum of a cost function that may possess several local minima values. Goldman and Mays in their publication [15] presents a water distribution system developed using Simulated Annealing approach. They stated that Simulated Annealing algorithm has the flexibility to consider different objective functions and constraints.

The essence of the SA process is an analogy with the way molten metals cool and anneal. For slowly cooled process, the system is able to find the minimum energy state [16]. The main steps for the SA method are illustrated in Fig. 2.

SA constitutes a method that is suitable for solving large scale optimization problems [16] (e.g. scheduling problems).

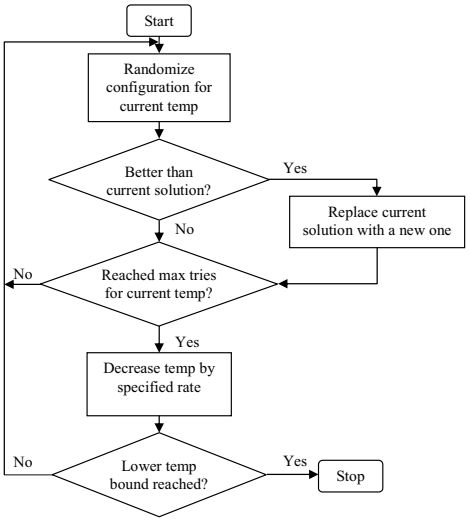

Figure 2. Simulated Annealing

E. Particle Swarm Optimization

Particle Swarm Optimization (PSO) is an evolutionary computation technique developed by Kennedy and Eberhart in 1995 [17]. The PSO technique is similar to the genetic algorithm method, because it begins with a group of randomly generated individuals (called the initial population) and also utilizes a fitness value to evaluate each particle from the population. For each potential solution in PSO, a randomized velocity is assigned. During the PSO process, each particle tracks its coordinates (location and velocity),

which are associated with the best solution it has achieved so far.

The particle swarm optimization concept consists of, at each step, changing the velocity (accelerating) each particle towards its best fitness (*pbest*) achieved so far and the overall best value obtained so far by any particle in the population (*gbest*). Acceleration is weighted by a random term *w* (*weight inertia*). Separate random numbers are generated for acceleration towards *pbest* and *gbest* locations [18].

The main steps of the particle swarm optimization method are illustrated in Fig. 3. The PSO algorithm is used in many areas including neural networks, telecommunications, control, data mining, power systems, signal processing, etc.

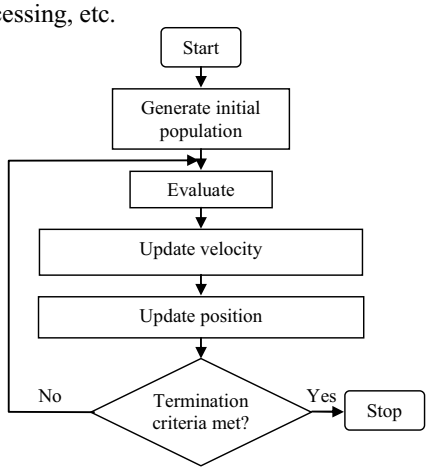

Figure 3. Particle Swarm Optimization

F. Software Path Testing

Path testing is a structural testing method that involves, using the source code of a program, to attempt finding every possible executable path in the program [19]. The path testing problem is considered to be an NP-complete problem [20].

For guiding search algorithms to generate test data which cover the target path, special heuristics are used to define the fitness function. The two main heuristics used for evaluating test data are *the approximation level* and *the branch distance*.

The first heuristic, *approximation level*, calculates the distance in branching nodes between the actual test data and the optimal test data, which would cover the target path (Fig. 4). This metric specifies how far away the individual is from fulfilling the target path branching condition [21]. Fig. 4 illustrates the target path which contains three decision nodes: A, B and C. If the individual diverges from the target path at the level of node A, the *approximation level* used for calculating the fitness function will be 2, because there are two critical branching nodes between the path taken by the individual during execution and the target path. If the individual diverges away at level of node B or level of node C, then the *approximation level* value will be 1, respectively 0.

The second heuristic, *branch distance*, is performed in order to distinguish between different individuals who execute the same program path [21]. Branch distance is calculated for an individual by using branching conditions in the branching node in which the target node is missed. The branching conditions are evaluated based on a table – Table I shows an example of such a distance function [5].

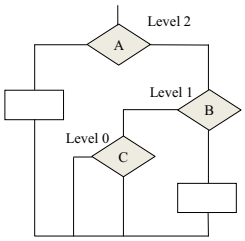

Figure 4. Aproximation level

Because the branch distance is less important than the approximation level, it is usually normalized in the range [0..1]. This guarantees that a better approximation level is always preferred regardless of the branch distance measure [22].

TABLE I. KOREL'S DISTANCE FUNCTION

No	Branch Predicate	Branch Function
1	A = B	ABS(A-B)
2	A ≠ B	K
3	A < B	(A-B) + K
4	A ≤ B	(A-B)
5	A > B	(B-A) + K
6	A ≥ B	(B-A)
7	X OR Y	Min(Distance(X), Distance(Y))
8	X AND Y	Distance(X) + Distance(Y)

In our experiments, a normalized value of the branch condition is used (1):

$$branch_{dist} = \frac{branch_{dist}}{branch_{dist} + 1} \quad (1)$$

III. AUTOMATIC TEST DATA GENERATION

Our approach is based on using three evolutionary algorithms (GA, SA and PSO) for solving the path testing problem. The original feature of this approach is the usage of SA and PSO algorithms in conjunction with the approximation level and branch distance metrics – the association of GAs with these metrics was already reported in the literature [1]. The metric used in this method calculates the difference between the selected path to be traversed, and the actual path traversed by the input values.

The three evolutionary algorithms described in the Section II try to find test data which cover particularly the chosen branch – it is important to compare these algorithms in order to determine which one is most appropriate for solving the path testing problem.

For generating test data which traverses the target path for a given problem there are five steps to follow [23]:

1. Construction of a control flow graph – it helps testers to select the target path.

2. Selection of the target path – the most important paths should be considered target paths.

3. Fitness function construction – the fitness function should be used for evaluating the distance between the actual path, which is traversed by actual test data, and the target path.

4. Program instrumentation – it consists of inserting probes at the beginning of every block of the source code to monitor the program's execution.

5. Test data generation and execution of instrumented program – initial test data are generated and then the evolutionary process chooses the best ones in order to achieve the target path.

For all the used algorithms the previous five steps were performed. The current test data evaluation was realized using the sum between the approximation level and the normalized branch distance – this approach applied for Simulated Annealing and Particle Swarm Optimization algorithms constitutes an original contribution.

For each search algorithm there are some important parameters which must be set up before launching the execution of the algorithm.

For the Genetic Algorithm the most important parameters are: *population size*, *number of generations*, *crossover method*, *crossover rate*, *mutation rate*, and *selection procedure*. The *population size* represents the total number of individuals in each population. Each individual from the population is represented by an array of bits, which encode the input parameter values for the program under test. For example in case of the triangle classification function each individual is encoded as an array of 24 bits, each byte representing the value of a triangle edge. The *number of generations* represents the total number of iterations for which the algorithm is executed.

For the Simulated Annealing algorithm there are three important parameters, which should be set up before starting the algorithm implementation: the *initial temperature*, the *final temperature*, which should have a very low value, and the *alpha* parameter which has the value equal with 0.999 and is used to decrease at each step the current temperature value. For Simulated Annealing, the random configuration for the initial temperature is composed by an array of bits, which represents the values of the input parameters of the function under test.

For the Particle Swarm Optimization algorithm the *number of particles* and the *total number of iterations* should be chosen. Each particle location is characterized by an array of bits which represents the test values for the input parameters. Velocity and position of the current particle are calculated based on the formulas (2), (3):

$$v_i(t) = wv_i(t) + c_1 \cdot rand() \cdot (p_{best} - x_i \cdot (t-1)) + c_2 \cdot rand() \cdot (g_{best} - x_i \cdot (t-1)) \quad (2)$$

$$x_i(t) = x_i(t-1) + v_i(t) \quad (3)$$

where we have the following parameters: *inertia weight* (w), *cognitive weight* (c_1) and *social weight* (c_2). The *inertia weight* represents the weighting contribution of the previous velocity for calculating the particle's actual velocity. It is used to calculate the new velocity of the particle according to its previous velocity. The *cognitive weight* and *social weight* are learning factors. The combination of cognitive weight and social weight parameters determines the convergence property of the algorithm.

IV. EXPERIMENTAL RESULTS

The proposed methodology was applied on a subset of some of the most commonly used benchmark programs:

- triangle classification program;
- quadratic equation solver;
- determination of ascendant order for a group of three numbers;
- a program for finding three numbers which have the sum equal with 250;
- a program that finds three numbers which have the product 0;
- the minimum function;
- the maximum function;
- the middle value function;
- the Fibonacci function;
- a program for finding three numbers which have the arithmetic mean value equal to 150.

Next, the step by step procedure for generating test data is illustrated just for the most significant example, which is the triangle classification function. This benchmark is widely used in the software testing literature (see [23]) and that is why the most important aspects of our approach will first be illustrated on this benchmark; for the other nine benchmarks, the procedure is similar and only the final results will be presented.

The triangles classification program receives as inputs three integers which represent the triangle's edges and checks if they can form a triangle. Fig. 5 presents the pseudo code for this program, while Fig. 6 shows its control flow graph, designed using Visio tool .

```
    TriangleClasification (int x, int y, int z)
begin
    if ((x + y > z) and (y + z > x) and (z + x > y))
     if ((x != y) and (y != z) and (z != x))
            "Triangle is scalene"
     else
    if ((x == y) and (y != z) or (y == z) and (z != x) or (z ==
x) and (x != y))
            "Triangle is isosceles"
     else
          "Triangle is equilateral"
     else
          "Not a triangle"
    end
```

Figure 5. Triangle classification program

According to the probability theory, the path for the equilateral triangle is the most difficult one to cover and therefore this was chosen to be the target path. The source code instrumented for the triangle classification program is shown in Fig. 7.

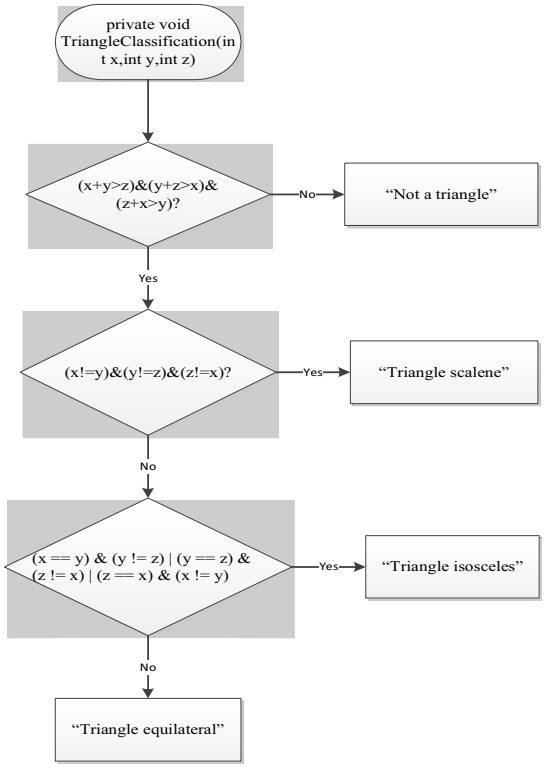

Figure 6. Triangle classification flow chart (generated by Visio)

```
TriangleClassification (int x, int y, int z)
begin
  approximation_level: = 2;
  fitness:=0;
  k: =1;
if ((x + y > z) and (y + z > x) and (z + x > y))
      approximation_level: = 1;
  if ((x != y) and (y != z) and (z != x))
      fitness + =  (3*k /3*k + 1) + approximation_level;
  else
       approximation_level = 0;
  if ((x == y) and (y != z) | (y == z) and (z != x) | (z == x) and
                  (x != y))
        fitness  +=(  min(min(|x-y|+2*k,|y-z|+2*k),|z-x|+2*k))  /
                  (min(min(|x-y|+2*k,|y-z|+2*k),|z-x|+2*k) + 1)) +
                   approximation_level;
  else
        fitness += ((x + y + z + 3 * k) / ((x + y + z + 3 * k) + 1))
                   + approximation_level;
end
```

Figure 7. Triangle classification instrumented pseudocode

The test data generation was performed using all the three evolutionary algorithms discussed above. The experimental settings for these algorithms are presented in Table II:

TABLE II. EVOLUTIONARY ALGORITHMS SETTINGS

Search Algorithm		Triangle classification	Quadratic equation	Ascendant order array values, Sum, Prod, Min, Max, Middle, Fibonacci, Average
GA	Pop. size	40	40	10
	Crossover rate (one point crossover)	0.75	0.75	0.75
	Mutation rate	0.1	0.1	0.1
	Generations	100	1000	100
SA	Initial temp	100	400	100
	Epsilon	0.001	0.001	0.001
	Alpha	0.999	0.999	0.999
PSO	No of particles	40	40	10
	w	0.796	0.796	0.796
	c_1	1.4962	1.4962	1.4962
	c_2	1.4962	1.4962	1.4962
	Iterations	100	1000	100

Based on some recommended parameter values presented in [25] and [26] for GA, we used some values adapted to the search space of each function under test. For the population size parameter, we have chosen a medium value, because if it is too small the algorithm may prematurely converge and if it is too large then the computation time will increase. Typically the number of individuals in the population (population size) should be between 100 and 1000. For the crossover rate a value between 0.5 and 1.0 was chosen [27]. For the mutation rate the typical values should be very small (0.1%), because if the values are higher the algorithm will degrade into a random search [28].

For SA, the initial temperature, cooling parameter and final temperature values were chosen to be close to the values published in [29].

The values for the parameters used in PSO were chosen based on [30]. The number of iterations is the same for each algorithm and was chosen based on some previous experiments of ours. There isn't a typical value which should be used for setting the number of iterations. This value is tightly coupled with the problem to be solved. The number of particles parameter was chosen to be 10 except for the triangle classification problem and the quadratic equation solver, where the number of particles was set to 40. The number of iterations parameter was chosen to be 100, except for the quadratic equation solver, where it was set to 1000. These values are different because the experiments have shown that 10 particles are not able to solve the problem in only 100 iterations.

The evolutionary algorithms parameters values should be chosen based on each developer/tester's previous experience, because there aren't universally accepted "best values" which should be used for solving each search problem.

All the experiments were performed on a computer having the following configuration: Intel I3 processor, 2.2 GHz, Windows 7 Operating System. Fig. 8 illustrates a comparison between all the three evolutionary algorithms. It presents the

iteration number at which each evolutionary algorithm finds the solution for the target benchmark. Fig. 9 presents the time spent by each evolutionary algorithm for solving each benchmark.

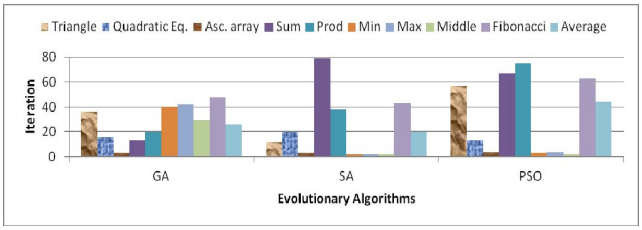

Figure 8. Iteration at which Evolutionary Algorithms solved each function (10 executions were performed for each algorithm)

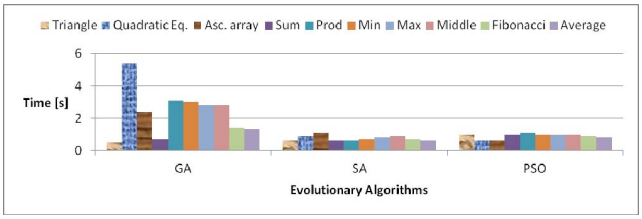

Figure 9. Time spent by each Evolutionary Algorithm to solve each function (average time for 10 separate experiments)

Fig. 10 ÷ Fig. 19 present a comparison between the three evolutionary algorithms in generating test data which cover the target path for each benchmark. Each figure corresponds to a benchmark and for each benchmark (except for the quadratic equation function) each algorithm was executed for 100 generations. For the quadratic equation (Fig. 11) the SA algorithm was not able to solve the benchmark in 100 generations; therefore the runs were performed for 1000 generations. It was decided to increase the number of generations in order to be able to observe the convergence of SA in this case.

Each algorithm was executed 10 times for each benchmark and the best execution was posted on the graphic. For guiding the search process of all the three evolutionary algorithms the same original metric presented in Section III, composed by summing up the normalized value of the branch distance and the approximation level, was used.

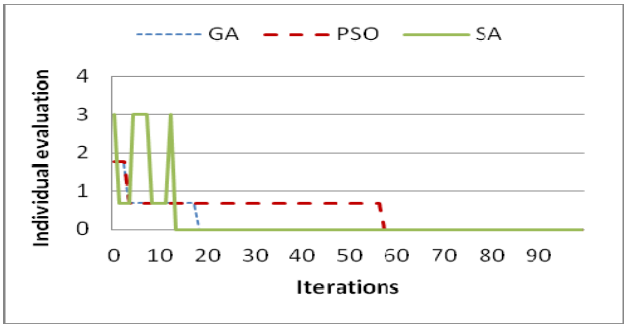

Figure 10. Convergence comparison for triangle problem based on 100 runs

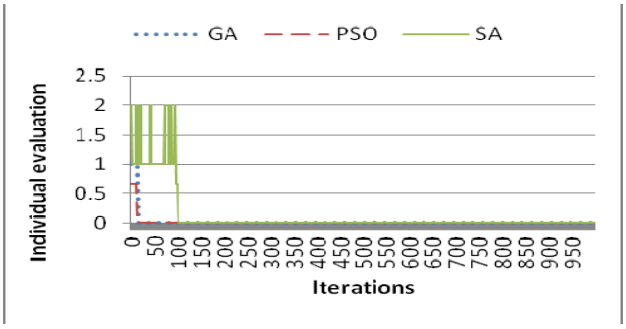

Figure 11. Convergence comparison for quadratic equation based on 1000 runs

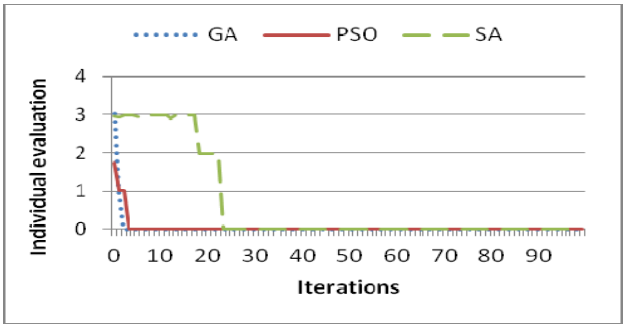

Figure 12. Convergence comparison for ascendant array values problem based on 100 runs

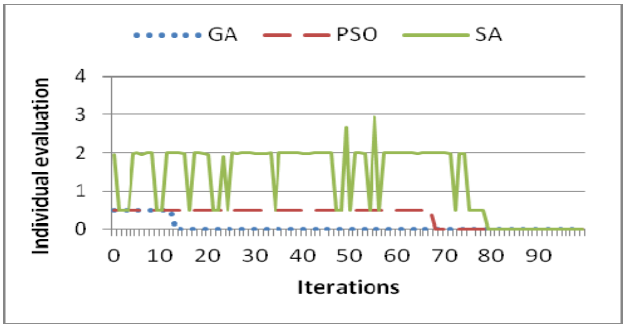

Figure 13. Convergence comparison for Sum function problem based on 100 runs

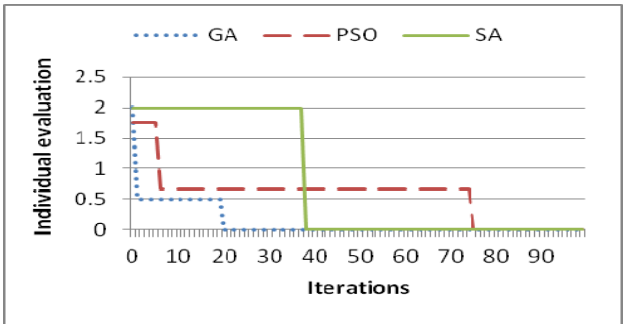

Figure 14. Convergence comparison for Prod function problem based on 100 runs

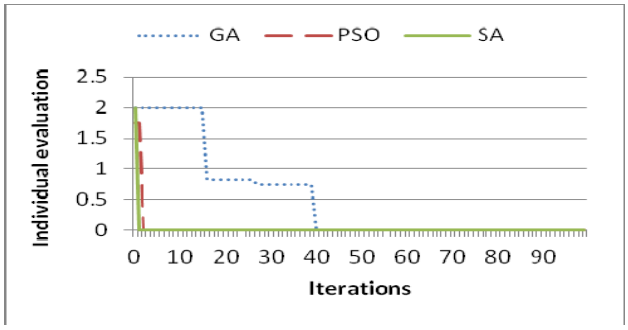

Figure 15. Convergence comparison for Min function problem based on 100 runs

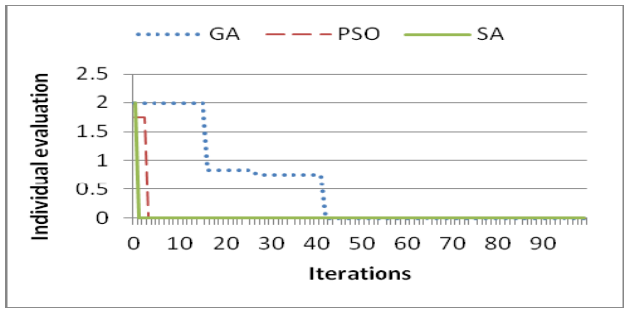

Figure 16. Convergence comparison for Max function problem based on 100 runs

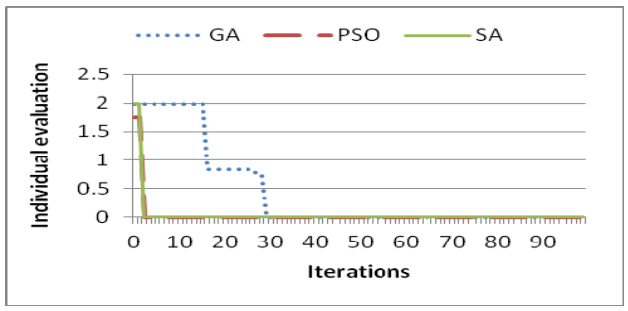

Figure 17. Convergence comparison for Middle value function problem based on 100 runs

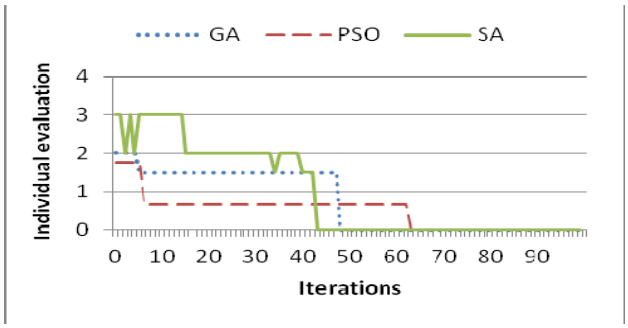

Figure 18. Convergence comparison for Fibonacci function problem based on 100 runs

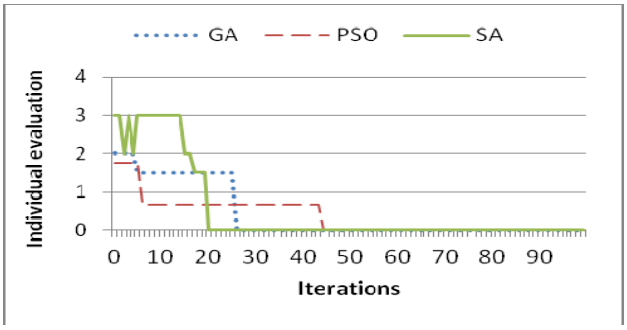

Figure 19. Convergence comparison for Arihmetic Mean function problem based on 100 runs

The results show that evolutionary algorithms are useful in finding test data for a target path using the branch distance and approximation level metrics. Table III shows a comparison between each algorithm's convergences. It presents the number of iterations needed by each evolutionary algorithm to generate appropriate test data for covering the target path.

For all the benchmarks, except Sum and Prod, the Simulated Annealing algorithm is able to generate test data which covers the target path quicker than the other two algorithms. For the Sum and Prod benchmarks (which have the same structure of their data flow graph), the fastest algorithm which generates test data is the Genetic Algorithm. So for the software problems which have the same very basic tree complexity as Sum and Prod benchmarks the Genetic Algorithm should be used for generating test data.

TABLE III. EVOLUTIONARY ALGORITHMS CONVERGENCE (MEASURED IN NUMBER OF ITERATIONS)

	Tc	Qe	Asc	S	P	Min	Max	Mid	Fib	Avg
GA	36	16	3	13	20	40	42	29	48	26
SA	12	15	3	20	38	2	2	2	43	20
PSO	57	13	4	67	75	4	4	2	63	44

V. CONCLUSIONS AND FUTURE WORK

In this paper, three evolutionary algorithms: Genetic Algorithm (GA), Simulated Annealing (SA) and Particle Swarm Optimization (PSO) were used for generating test data for software path testing. A significant number of different benchmarks conduct the study to a clear conclusion: evolutionary algorithms are very appropriate for generating test data for covering a target path.

Experimental results show that the best evolutionary algorithm for path testing is the Simulated Annealing one (SA) with a starting temperature of 100.0º, because the quality of the test data produced by this algorithm is higher than the quality of the other data produced by the two others algorithms. The quality of the test data produced by SA is higher than the data produced by the other two algorithms, because it manages to generate test data which cover the target path quicker (SA converges faster than GA and PSO).

It is interesting to notice that for very simple functions like Sum and Prod, the GA yields the results faster, but in this case also SA comes on the second position. As the complexity of the software program under test increases, SA reveals to be the best solution.

Evolutionary algorithms are also useful for reducing the time required for path testing. In our research, these algorithms were adapted for structural testing in order to reduce execution time and generate suitable test data for covering the target path.

As a working methodology: in order to generate test data which cover the target path in software programs, the testers should first generate the programs' flow graphs and then, based on the flow graphs structure, they can decide which evolutionary algorithm(s) should be used for generating test data. Our recommendation is to use Simulated Annealing (SA) with a starting temperature of 100.0º, because it is the fastest to converge.

Future work will involve using evolutionary algorithms for path testing in larger projects and compare them with other evolutionary techniques to assure their efficiency in structural testing. A testing framework based on evolutionary algorithms could be designed and implemented, in order to make the test data generation process completely automated.

REFERENCES

[1] J. Wegener, A. Baresel, H. Sthamer, "Evolutionary test environment for automatic structural testing", Information and Software Technology vol. 43, pp. 841-854, December 2001, doi: 10.1016/S0950-5849(01)00190-2

[2] R.P.Pargas, M.J.Harrold, R.R.Peck, "Test-Data Generation Using Genetic Algorithms", Journal of Software Testing, Verification and Reliability, vol. 9, pp. 263-282, December 1999, doi: 10.1002/(SICI)1099-1689(199912)9

[3] P. McMinn, Search-based, "Software Test Data Generation: A Survey", Journal of Software Testing Verification and Reliability, vol. 14, pp. 105-156,June 2004, doi:10.1002/stvr.294

[4] N. Tracey, J. Clark, K. Mander, J. McDermid, "An Automated Framework for Structural Test-Data Generation",13th IEEE International Conference on Automated Software Engineering,pp. 285,October 1998, doi: 10.1109/ASE.1998.732680

[5] B. Korel. "Automated software test data generation", IEEE Transactions on software engineering 16, no. 8,August 1990, doi: 10.1109/32.57624

[6] H-H. Sthamer, "The Automatic Generation of Software Test Data Using Genetic Algorithms", Phd. Thesis, 1995.

[7] E.F. Miller, "Introduction to Software Testing Technology, Tutorial: Software Testing & Validation Techniques", Second Edition, IEEE Catalog No. 180-0,pp. 4-16., 1980.

[8] L. Luo, "Software Testing Techniques, Technology Maturation and Research Strategies", Institute for Software Research International, Carnegie Mellon University, Pittsburgh, USA, 2009, class report.

[9] A.Sofokleous, A. Andreou, "Automatic, evolutionary test data generation for dynamic software testing", Journal of Systems and Software, vol. 81, pp. 1883-1898, 2008, doi: 10.1016/j.jss.2007.12.809

[10] E. Alba, C. Cotta, "Evolutionary Algorithms", Handbook of Bioinspired Algorithms and Applications,pp. 3-19, Chapman & Hall/CRC Computer & Information Science Series, September 2005.

[11] M. Srinivas, "Genetic Algorithms: A Survey", Journal Computer, vol. 27, pp. 17-26 June 1994, doi: 10.1109/2.294849

[12] D. Whitley, "A genetic algorithm tutorial, Statistics and Computing", Statistics and Computing, vol. 4, pp. 65-85, 1994, doi: 10.1007/BF00175354

[13] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi,, "Optimization by Simulated Annealing", Statistical Science, vol. 220, pp. 671-680, May 1983.

[14] V. Cerny, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm", Journal of Optimization Theory and Applications, vol. 45, pp. 41-51, 1985, doi: 10.1007/BF00940812

[15] F. Goldman, L. Mays, "Water distribution system operation: Application of Simulated Annealing", Water Resources Systems Management Tools, December 2004.

[16] F. Kolahan, M. Hossein Abolbashari, S. Mohitzadeh, "Simulated Annealing Application for Structural Optimization", Engineering and Technology, pp. 326-329, 2007, ISSN 1307-7473

[17] J. Kennedy, R. Eberhart, "Particle swarm optimization", Proceedings of IEEE International Conference on Neutral Networks, pp. 1942-1948, 1995, doi: 10.1371/journal.pone.0021036

[18] R. Eberhart, Y.Shi, "Particle Swarm Optimization: Developments, Applications and Resources", Proceedings Congress Evolutionary Computation, pp. 81-86, 2001, doi: 10.1109/CEC.2001.934374

[19] L. Gregory, "Advanced Topics in Computer Science: Testing",Path Testing, Dissertation Paper, December 2006.

[20] P.Mishra, B.S.P.Mishra, Eccentric "Test Data generation for Path Testing Using Genetic Algorithm", International Proceedings of Computer Science and Information Technology, vol. 2, pp. 536, July 2009.

[21] J. Wegener, K. Buhr, H. Pohlheim, "Automatic Test data Generation for Structural Testing of Embedded Software Systems by Evolutionary Testing", Research and Technology, GECCO '02 Proceedings of the Genetic and Evolutionary Computation Conference, 2002, doi: 10.3724/SP.J.1001.2009.00580

[22] A. Arcuri, "It Does Matter How You Normalise the Branch Distance in Search Based Software Testing", Software Testing, Verification and Validation, pp. 205-214, April 2010, doi: 10.1109/ICST.2010.17

[23] P. Nirpal, V. Kale, "Comparison of Software Test Data for Automatic Path Coverage Using Genetic Algorithm", International Journal Of Computer Science & Engineering Technology (IJCSET), vol. 1,pp. 12-16., February, 2011.

[24] Visio software http://office.microsoft.com/en-us/visio/

[25] DeJong,K.A and Spears,W.M "An Analysis of the Interacting Roles of Population Size and Crossover in Genetic Algorithms", Proc. First Workshop Parallel Problem Solving from Nature, Springer-Verlag,pp. 38-47, Berlin, 1990.

[26] J.Grefenstette,"Optimization of Control Parameters for Genetic Algorithms",IEEE Trans. Systems,Man, and Cybernetics, vol. 16, pp. 122-128,January 1986.

[27] M.Srinivas, L.M. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms", IEEE Transactions On Systems Man and Cybernetics, vol. 24, pp. 656-66, April, 1994, doi: 10.1109/21.286385

[28] L.Yalan, C.Nie, J.Kauffman, G.Kapfhammer, H. Leung, "Empirically Identifying the Best Genetic Algorithm for Covering Array Generation", ESEC Conference, September 2011.

[29] J. Lam, "An Efficient Simulated Annealing Schedule" ,Report 8818, Department of Computer Science, September 1988.

[30] I.C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection", Information Processing Letters, vol. 85,pp. 317-325, 2003, doi: 10.1016/S0020-0190(02)00447-7