Optimum Selection of PSO Parameters for Automated Test Data Generation of Software Programs

Nirupamakumari[1] Venukamadan[2]
[1,2]Shivalik Institute of Engineering & Technology, Aliyaspur, Ambala

Abstract— The proposed research work implemented and fine-tuned meta-heuristic based search algorithms namely PSO algorithm for automatic test case generation using path testing criterion. The three parameters namely size of population (NIND), inertia weight (W), social learning rate (C1), and cognitive learning rate (C2) have been chosen for setting PSO algorithm for test data generation. For input generation, symbolic execution method has been used in which first, target path is selected from Control Flow Graph (CFG) of Software under Test (SUT) and then inputs are generated using search algorithms which can evaluate composite predicate corresponding to the target path true. We have experimented on two real world programs showing the applicability of these techniques in genuine testing environment. The algorithm is implemented using MATLAB programming environment. The performance of the algorithms is measured using average test cases generated per path (ATCPP) and average percentage coverage (APC) metrics.

I. INTRODUCTION

Test cases can be generated in several ways: automated, semi-automated or manual. While automated test cases generation saves a lot of resources and these don't encompass human biases and are less intelligent. On the other side manual test cases, although consist human intelligence, take much efforts and time. It is relatively easy to generate test cases manually which are able to detect many faults but at the same time it is very slow, boring and costly process. It is highly desirable to achieve almost same quality test cases through an automated process so that the probability of fault identification becomes maximum. This process of automation requires a good amount of intelligence in order to handle several test adequacy criteria and must solve problems involving state propagation and constraint satisfaction either implicitly or explicitly [8]. While automated test cases generation saves a lot of resources and doesn't encompass human biases, these have not been as intelligent as desired. In order to improve the quality of this automation and to add more intelligence to this process, many researchers have started exploring the ways to devise new artificial intelligence based techniques, which can generate apt test cases automatically [3, 4].

Considerable effort has been put in by various researchers to apply soft computing in the area of software engineering [6]. One of the most intensively applied areas of soft computing in software engineering is test data generation, which is categorized as NP-hard or NP-complete [5] problem due to requirement of enormous efforts in finding the data out of large search space satisfying the complex and non-linear search objectives. Some of the Meta-heuristic search techniques which we have employed for our experimentation are described below with their respective algorithm workflow.

In [16] four Meta-heuristic algorithms namely Genetic Algorithm (GA), Artificial Bee Algorithm (ABC), Particle Swarm Optimization (PSO) and Big Bang Big Crunch (BBBC) algorithms are evaluated to identify a suitable testing technique which has general applicability and can be scaled to larger programs in software testing by following the method of empirical investigation. Besides this authors have also worked for identifying the key factors responsible for software testing efforts by making a correlation between the characteristics of programs and their respective efforts made in test data generation. But this work is silent about the optimum parameters selection of these algorithms for better performance in software test data generation.

In this thesis, we have worked in two areas, one on test case generation methodology and another on PSO based search algorithm implementation for this purpose. Although there are so many different Meta-heuristic algorithms used for test data generation but till now no significant research has been taken for assessing optimum selection of various PSO parameters for test data generation, hence we plan to propose to generate test cases using PSO algorithm and find optimum value of various types of PSO parameters such as population size, inertia weight, social learning rate and cognitive learning rate for these operators for test data generation problem. Based on discussion following objective may be suggested for research work; to implement PSO algorithm for software test data generation and to compare performance of various parameters of PSO algorithm and thus select optimum values of parameters for test data generation.

The paper is divided in five sections. Section 1 introduces the motivation of automatic testing. Section 2 is devoted to the literature survey containing works related to automatic test data generation during last decade. Chapter 3 presents test problem as a search problem and PSO algorithm for implementing test data generation using symbolic execution method with fitness function of search algorithm. Chapter 4 describes experimental setup, selection of test objects and experimental results. Chapter 5 concludes paper with research directions and future scope in automatic test data generation.

II. RELATED WORKS

Different types of search algorithms employed in testing research can be categorized into three classes.

First class belongs to random testing based search algorithms. In this type of testing, random values are generated from domains of inputs and program is executed using these values. If these inputs are able to satisfy the testing criterion then they form a test case. In recent times Mayer and Schneckenburger [7] empirically investigated different flavors of adaptive random testing and concluded that distance based random testing and restricted random testing are the best methods for this class of testing techniques. Although, in general, random testing is a simple, cost-effective and unbiased technique, but researchers do

not find it suitable for large and complex programs, especially where input domain size is large.

Another class of search methods used in testing is algorithmic such as numerical maximization techniques, alternate variable method which was employed by Korel for its dynamic test data generator TESTGEN [14] and domain reduction procedure used by Demillo et al for its fault based test data generator named as GODZILA [4]. Last decades experienced a number of research attempts in automation of test cases generation [8], specifically heuristic approaches were used to fulfill the requirements of this activity. Korel's technique is based upon the dynamic execution of software under test (SUT). On the other hand, Demillo et al used symbolic execution for testing purpose.

In the past decades several soft computing based methods such as genetic algorithm (GA) [11], simulated annealing (SA) [9], tabu search [15], ant colony optimization (ACO) [2] were used to fulfill this requirement. Out of these techniques, GA has been the most successful and frequently used technique by researchers. Xanthakis [13] first time applied GA for automatic test case generation. He constructed an objective function for GA program using the concept of branch distance.

III. METHODOLOGY

Search techniques are applied for generation of test data by transforming testing objective into search problem. Two components are essential for a problem which is to be modeled as search target. First a mechanism should be derived through which the problem is encoded in search algorithm and second component is assessment of the suitability of solutions produced by search technique to guide the individuals for exploring search space.

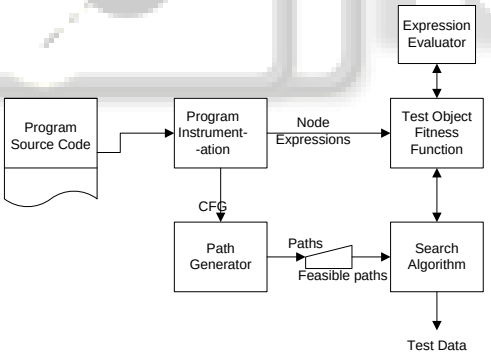

Fig. 1: Path based automatic test data generator

A. Particle Swarm Optimization (PSO)

PSO is a population-based a biologically inspired algorithm which applies to concept of social interaction to problem solving where each individual is referred to as particle and represents a candidate solution [12]. Each particle in PSO flies through the search space with an adaptable velocity that is dynamically modified according to its own flying experience and also flying experience of other particles using the following equations.

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

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

where

- $v_{id}^{t+1}$is a velocity vector at t+1 time for i particle in d^{th} dimension
- x_{id}^{t+1}position vector at t+1 time for ith particle in d dimension
- $r_1, r_2$are random number generators.
- $C_1$and$C_2$are are learning rates governing the cognition and social components.
- p_{id}^lrepresents the particle with best p-fitness.
- W is the inertia factor that dynamically adjust the velocities of particles gradually focusing the PSO into a local search.

B. PSO Algorithm Workflow

Following steps illustrate the overall optimization scheme of PSO.

Initialize the particle population by randomly assigning locations (X-vector for each particle) and velocities (V-vector with random or zero velocities- in our case it is initialized with zero vector)

Evaluate the fitness of the individual particle and record the best fitness P_{best} for each particle till now and update P-vector related to each P_{best}.

Also find out the individuals' highest fitness G_{best} and record corresponding position P_g.

Modify velocities based on P_{best} and G_{best} position using eq3.

Update the particles position using eq4.

Terminate if the condition is met

Go to Step 2

In equation (1) above, new velocity at t+1 is generated with the help of global fitness which all the particles have achieved till iteration t. In this equation, (given in bold) is position given by the global best fitness in dimension d. Usually, global best fitness concept is expected to give a global search exploration possibilities in the search space.

Ahmed *et al* [1] have proposed several techniques such as normalization, weighting and rewarding schemes for making fitness function effective and useful. Watkins *et al* [10] compared many fitness function construction techniques and concluded through their experiments that branch-distance based function is best performer in the static structural testing category.

Violated individual 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)
A and B are operands and ζ is a smallest constant of operands' universal domains. In case integer it is 1 and in case real values it can be 0.1 or 0.01 depending on the accuracy we need in solution.	

Table. 1: Fitness function of a branch predicate

To calculate the fitness of an individual in population, first, values from the individual are assigned to the symbols related to each input of the program. In CFG of a program, a node is either branch node if it can deviate the control flow of program from target path otherwise it is a non-branch node. Non branch nodes' statements are evaluated whenever they are encountered in the course of path traversal so that any future predicate dependency on internal variables can be addressed properly. The CP of each branch node corresponding to the target path is evaluated using the current values of inputs and any internal variable. If it is not evaluated true then DP(s) are extracted from CP one by one and fitness for each DP is determined by means of the branch distance concept as explained above. After this integrated fitness for whole of CP is determined by adding fitness 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 fitness of two DPs is considered for the evaluation of whole CP fitness. Negation is resolved by propagating inward over variables and changing *'and'* into *'or'* and vice versa. If integrated fitness is zero or negative then CP is called evaluated or satisfied by the individual and search process for particular path is terminated otherwise search is allowed to proceed further.

IV. EXPERIMENTAL SETUP & RESULTS

All experiments are performed in the MATLAB framework. MATLAB programming is used for the PSO algorithm implementation. In these experimentations, main aim is to fine tune the parameters of the PSO algorithm to prove the usefulness and utility of algorithm for test case generation concept and compare the performance of PSO with various parameters. 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 10^7 and one small with a size of order of 10^3 for each path and experiment is conducted 100 times for averaging results. In each attempt, the PSO is 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.

A. Test Measures

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.

B. Test Objects

We have taken two real world programs for test data generation activity:Triangle classifier (TC)&Line-rectangle classifier (LRC). TC is one of the most used programs for experimentation of test data generation in structural testing environment. It accepts three inputs as sides of a triangle and then decides whether these sides form a triangle and if yes then of what type. LRC program identifies whether a line cuts a rectangle or lies completely outside or lies completely inside of the rectangle. In this program total eight inputs are entered; four for co-ordinates of rectangle and other four inputs to define the line. Some of the nodes in CFG of this program have very high level of nesting. This is main reason of using this program so that the difficulty of testing nested structures can be found Test cases for TC and LRC programs are generated from inputs by taking small domain of size 10^3 for each path.

C. Experimental Results

We have experimented on two most standard benchmark programs regularly used in testing research [Diaz2007, Lin2000]. These are triangle classifier and rectangle classifier programs. Table 5.4 lists the results of test cases generation PSOmethod applied on two programs. Eighty one mutant programs differ in selection of various parameters such as size of population (NO_PART), inertia weight (W), social learning rate (C1) and cognitive learning rate (C2). For TC program, we have considered two domains large and small. The large domain is of the order of -10^7 to $+10^7$ while the small domain is of the order of -10^3 to $+10^3$.

Table2 shows the experiments results. First and second column of each category programs register invalid average test cases per path (ATCPP) and average percentage coverage (APC) of all paths for both programs in 10 attempts of test case generation for each path.

	PSO parameters				TC(SD)		TC(LD)		LRC(SD)	
S. No.	NIND	W	C1	C2	ATCPP	APC	ATCPP	APC	ATCPP	APC
1	20	0.9	2.8	1.8	857	100%	5336	81%	2078	97%
2	20	0.9	2.8	1.5	781	100%	3826	96%	1246	99%
3	20	0.9	2.8	1.2	656	100%	3089	97%	1167	100%
4	20	0.9	2.4	1.8	633	100%	2906	99%	1283	99%
5	20	0.9	2.4	1.5	528	100%	1647	100%	1307	99%
6	20	0.9	2.4	1.2	592	100%	1588	100%	1274	99%
7	20	0.9	2.1	1.8	617	100%	1466	100%	1167	100%
8	20	0.9	2.1	1.5	571	100%	1180	100%	1130	100%
9	20	0.9	2.1	1.2	547	100%	1113	100%	1070	99%

10	20	0.7	2.8	1.8	878	100%	6434	84%	1780	99%
11	20	0.7	2.8	1.5	593	100%	2885	100%	1238	99%
12	20	0.7	2.8	1.2	449	100%	1706	100%	916	99%
13	20	0.7	2.4	1.8	539	100%	1366	100%	916	100%
14	20	0.7	2.4	1.5	429	100%	1299	100%	1256	99%
15	20	0.7	2.4	1.2	488	100%	1083	100%	1308	99%
16	20	0.7	2.1	1.8	397	100%	1126	100%	1045	100%
17	20	0.7	2.1	1.5	364	100%	906	100%	1115	100%
18	20	0.7	2.1	1.2	334	100%	912	100%	1131	100%
19	20	0.5	2.8	1.8	486	100%	3820	99%	1609	99%
20	20	0.5	2.8	1.5	439	100%	1479	100%	1865	99%
21	20	0.5	2.8	1.2	330	100%	1380	100%	1265	99%
22	20	0.5	2.4	1.8	371	100%	1212	100%	1476	100%
23	20	0.5	2.4	1.5	340	100%	920	100%	1288	100%
24	20	0.5	2.4	1.2	291	100%	744	100%	1538	99%
25	20	0.5	2.1	1.8	304	100%	785	100%	1252	99%
26	20	0.5	2.1	1.5	261	100%	717	100%	1749	99%
27	20	0.5	2.1	1.2	246	100%	694	100%	1563	99%
28	30	0.9	2.8	1.8	1101	100%	6070	90%	2009	98%
29	30	0.9	2.8	1.5	1030	100%	5260	99%	1400	99%
30	30	0.9	2.8	1.2	817	100%	3115	100%	1088	100%
31	30	0.9	2.4	1.8	859	100%	2680	100%	1297	100%
32	30	0.9	2.4	1.5	773	100%	2113	100%	1382	99%
33	30	0.9	2.4	1.2	850	100%	1851	100%	1155	100%
34	30	0.9	2.1	1.8	836	100%	2070	100%	1057	100%
35	30	0.9	2.1	1.5	769	100%	1813	100%	960	100%
36	30	0.9	2.1	1.2	678	100%	1678	100%	1084	100%
37	30	0.7	2.8	1.8	896	100%	7006	96%	1584	99%
38	30	0.7	2.8	1.5	830	100%	2627	100%	1280	99%
39	30	0.7	2.8	1.2	681	100%	2685	100%	1238	100%
40	30	0.7	2.4	1.8	697	100%	2125	100%	1439	98%
41	30	0.7	2.4	1.5	644	100%	1728	100%	933	100%
42	30	0.7	2.4	1.2	581	100%	1495	100%	1110	100%
43	30	0.7	2.1	1.8	603	100%	1635	100%	1154	100%
44	30	0.7	2.1	1.5	613	100%	1346	100%	1254	100%
45	30	0.7	2.1	1.2	521	100%	1278	100%	1100	100%
46	30	0.5	2.8	1.8	635	100%	3875	100%	1872	99%
47	30	0.5	2.8	1.5	520	100%	2019	100%	1679	99%
48	30	0.5	2.8	1.2	489	100%	1354	100%	1467	100%
49	30	0.5	2.4	1.8	504	100%	1651	100%	1178	100%
50	30	0.5	2.4	1.5	409	100%	1196	100%	1135	100%
51	30	0.5	2.4	1.2	410	100%	1078	100%	1120	100%
52	30	0.5	2.1	1.8	434	100%	1176	100%	1329	100%
53	30	0.5	2.1	1.5	398	100%	1031	100%	1343	100%
54	30	0.5	2.1	1.2	327	100%	979	100%	1292	100%
55	40	0.9	2.8	1.8	1267	100%	9283	90%	1887	100%
56	40	0.9	2.8	1.5	1161	100%	4740	100%	1685	99%
57	40	0.9	2.8	1.2	1088	100%	3320	100%	1384	99%
58	40	0.9	2.4	1.8	1163	100%	5067	100%	1400	100%
59	40	0.9	2.4	1.5	912	100%	2425	100%	1428	100%
60	40	0.9	2.4	1.2	1066	100%	2835	100%	1125	100%
61	40	0.9	2.1	1.8	1120	100%	2420	100%	1295	100%
62	40	0.9	2.1	1.5	1007	100%	2407	100%	1275	100%
63	40	0.9	2.1	1.2	959	100%	2226	100%	954	100%
64	40	0.7	2.8	1.8	1307	100%	6648	97%	1867	100%
65	40	0.7	2.8	1.5	1014	100%	3835	100%	1667	99%
66	40	0.7	2.8	1.2	940	100%	3068	100%	1369	99%
67	40	0.7	2.4	1.8	884	100%	2543	100%	1385	100%
68	40	0.7	2.4	1.5	754	100%	1832	100%	1413	100%
69	40	0.7	2.4	1.2	708	100%	1832	100%	1113	100%

70	40	0.7	2.1	1.8	763	100%	1987	100%	1281	100%
71	40	0.7	2.1	1.5	679	100%	1587	100%	1261	100%
72	40	0.7	2.1	1.2	619	100%	1832	100%	944	100%
73	40	0.5	2.8	1.8	774	100%	4308	100%	1836	100%
74	40	0.5	2.8	1.5	610	100%	2050	100%	1639	99%
75	40	0.5	2.8	1.2	562	100%	1600	100%	1346	99%
76	40	0.5	2.4	1.8	674	100%	2094	100%	1362	100%
77	40	0.5	2.4	1.5	530	100%	1510	100%	1389	100%
78	40	0.5	2.4	1.2	552	100%	1417	100%	1094	100%
79	40	0.5	2.1	1.8	549	100%	1416	100%	1260	100%
80	40	0.5	2.1	1.5	459	100%	1311	100%	1240	100%
81	40	0.5	2.1	1.2	448	100%	1311	100%	928	100%

Table. 2: Average Test cases generation per path (ATCPP) and Average percentage coverage with PSO

Following results can be interfered from the table.

- PSO's performance depends on input domain size. If domain is small the PSO is able to find test case each and every time. But in large domain it sometime fails to generate test case and it is also taking more effort. The PSO is taking much time for those path constraints which involve equality constraints. But before making general statement, a further study needs to be done to find out the cause and effect relationship between nature of predicates and efforts made toward generation of test data.
- From Sr. No. line 27, 54, 81 it can be clearly deduced that for different value of Number of Individual NIND, combination of inertia weight W, social learning rate C1 and cognitive learning rateC2 with value 0.5, 2.1 and 1.2 respectively give the excellent results.
- If we compare performance of values of C1 and C2 as 2.1 and 1.2 for different values of inertia weight W, then these values give better result as compared to other value.
- Number of individual or population size doesn't affect performance much but it should be from 20 to 40 to maintain a tradeoff between effort and computation time.

V. CONCLUSION

From experiments it has been also observed that PSO performance greatly affected by mutation rate for test case generation problem. Ideal value forinertia weight, social learning rate and cognitive rate is found to be 0.5, 2.1 and 1.2 respectively. Number of individual or population size doesn't affect performance much but it should be between 20 and 40.

It has also been found that the performance of the PSO method is not good for path constraints which are having large number of equality predicates. Hence, it has been observed to be ideally suited for test case generation problem where less equality predicates has to be solved. During experimentation we also observed that Meta-heuristic techniques exhibits domain dependency in test case generation, as these have to generate huge number of ATCPP In larger domain. PSO is also taking more effort to generate test cases for larger domain especially when path constraint involves equality predicates.

REFERENCES

[1] Ahmed MA, Hermadi I. GA-based multiple paths test data generator. Computers and Operations Research, 2007
[2] Ayari, K., Bouktif, S., and Antoniol, G. Automatic mutation test input data generation via ant colony. Proceedings of the 9th annual conference on Genetic and evolutionary computation GECCO '07, New York, NY, USA, 2007, ACM, pp. 1074-1081.
[3] Boyapati C, Khurshid S, Marinov D. Automated testing based on Java predicates. MIT ISSTA 2002
[4] Demillo R. A., and Offutt A. J., Constraint-based automatic test data generation, IEEE transaction on Software engineering. Vol.17, No.9, September, 1991 pp. 900-910
[5] Mansour N, Salame M. Data generation for path testing. Software Quality Journal 2004; 12:121–136.
[6] Mantere T and Alander JT, Evolutionary Software Engineering, A Review, Applied Soft Computing, 5(3) 2005, pp. 315-331
[7] Mayer J, Schneckenburger C, An Empirical Analysis and Comparison of Random Testing Techniques, ISESE'06, September 21–22, 2006, Rio de Janeiro, Brazil pp. 105-114.
[8] McMinn P. Search-based Software Test Data Generation: A Survey. Software Testing, Verification and Reliability June 2004; 14(2):105-156.
[9] Tracey N. A Search-Based Automated Test-Data Generation Framework for Safety Critical Software. PhD thesis, University of York, 2000.
[10] Watkins A, Hufnagel E. M. Evolutionary test data generation: a comparison of fitness functions. Software Practice & Experience 2006; 36:95–116
[11] Wegener J, Baresel A, Sthamer H., "Evolutionary test environment for automatic structural testing". Information and Software Technology2001;43:841–854,
[12] Windisch A, Wappler S and Wegener J, Applying Particle Swarm Optimization to Software Testing, Proceedings of the 2007 conference on Genetic and evolutionary computation GECCO'07, July 7–11, 2007, London, England, United Kingdom.
[13] Xanthakis S, Ellis C, Skourlas C, Gall AL, Katsikas S and Karapoulios K. Application of

genetic algorithms to software testing. In The fifth international conference on software engineering 1992; 625–36.
[14] Korel B. Dynamic method for software test data generation. Software Testing, Verification and Reliability, 1992; 2(4):203-213.
[15] Díaz E, Javier T, Raquel B, José JD. A tabu search algorithm for structural software testing. Computers and Operations Research, 2007
[16] Surender Singh and Parvin Kumar "Empirical Evaluation of Metaheuristic Approaches for Symbolic Execution based Automated Test Generation" International Journal of Information Technology and Knowledge Management (ISSN: 0973-4414) July-December 2012, Volume 5, No. 2, pp. 489-493

