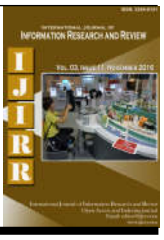

IJIRR

International Journal of Information Research and Review
Vol. 03, Issue, 11, pp. 3226-3229, November, 2016

Review Article

APPLICATIONS OF GENETIC ALGORITHM IN SOFTWARE TESTING

Aditi Jain, *Mohil Jain, Parv Badjatiya, Shivam Pandey and Shrey Khurana

SCOPE, VIT University, Vellore

ARTICLE INFO

Article History:

Received 18th August, 2016
Received in revised form
22nd September, 2016
Accepted 14th October, 2016
Published online November, 30th 2016

Keywords:

Frequently
Crossover
Algorithm
Implementation.

ABSTRACT

The applicability of evolutionary algorithms in soft- ware testing has been an area of importance for many re- searchers. In this paper, we have studied the implementation of one such evolutionary algorithm namely genetic algorithm. Genetic algorithm is an improvement technique established from the concepts of biological evolution such as natural selection, genetic recombination and survival of the fittest. It is frequently used to generate solutions to optimization and search problems. It uses three principles namely crossover, mutation and selection. Here, we are reviewing the applications of genetic algorithm in software project effort estimation and scheduling.

INTRODUCTION

Genetic algorithm is an approach to solve both uncon strained and constrained optimization situations based on natu ral selection, the phenomenon that drives biological expansion. It frequently alters a population of independent results. For every iteration, it determines parents randomly from the given population and produce off springs with its help for future generations. This helps in reaching an optimal solution. This algorithm can be used to solve diverse optimization problems that cannot be solved using other available techniques. Software testing helps in finding bugs in a software. It validates the software and verifies whether it meets the re quirements of the users. With the help of certain tools, software can be verified either with the help of machines or humans. On comparison between automated and manual methods, it has been found that automated testing is more efficient than manual testing. The approach of genetic algorithm helps in the testing pro cess of software. It is used for optimizing scheduling problems and for estimation of efforts. It also helps in optimizing other problems such as data flow testing, path testing, functional testing, mutation testing and model-based testing which are further discussed in detail.

Related Work

Estimation of effort in Software project management

It is an arduous task to accurately assess the efforts required by tasks for sustaining the entire software process cycle. Until now, it has been a topic for discussion among the researchers and various methods have been formulated to predict the same. Amid all the techniques proposed by researchers, Source Line of Codes (SLOC) and Function Point Analysis (FPA) are the most commonly used ones. A.J. Albrecht who worked at IBM came up with FPA in order to minimize the efforts to calculate the relative productivity of distinct programming languages. He identified five different types of elements in information systems and counted the occurrences of each. Those five function types include logical interface file (LIF) types, external interface file (EIF) types, external input (EI) types, external output (EO) types and external enquiry (EQ) types. FPA is used to find the testing effort required by the information systems and also to determine the software size in order to estimate the cost. Therefore, many organizations have started employing FPA. Albrecht and Gaffney also indicated that FPA can be used to determine SLOC which can further help in estimating the work effort. Subsequently, Salbrechter and Felfernig portrayed the application and development of FPA to estimate effort in the progression of knowledge-based composition systems. Niessink and Ahn *et al*. devised a model based on FPA that helps in predicting the same using agile software development methods.

Scheduling

Research on discovering new solutions to resourceconstrained project scheduling problems (RCPSPs) has advanced over several years.

***Corresponding author: Mohil Jain,**
SCOPE, VIT University, Vellore.

The methods used for the same are of two different types heuristic methods and exact methods. Exact methods consist of backtracking, branch and bound, dynamic programming, implicit enumeration and critical path method with its variations. Heuristic methods comprise of simulated annealing, tabu search and genetic algorithm. Heuristic methods are comparatively more popular in scheduling problems as compared to exact methods. These methods can further be classified into deterministic and stochastic approaches. Genetic algorithm falls under the category of stochastic search methods.

GENETIC ALGORITHM IN SOFTWARE TESTING

Applications of GA in White Box Testing

Data Flow Testing: A structural adapted automatic test data generation method that uses a GA guided approach was devised by M.R. Girgis (2005). The program that has to be verified is reformed Into a Control Flow Graph (CFG) where every node serves as a block in program and the edges of the flow graph represents control flow. The variables in the program are split into two different types c-uses and p-uses variables.

c-uses variables are used in computations or as implications in a given program. p-uses variables are affiliated to the edges of flow graph. To fulfill the all-uses norm, a def-clear path from each definition of variable to each use of that variable needs to be resolved. To locate the set of paths which satisfy the all-uses criteria, it is crucial to resolve def c-use (dcu) and def p-use (dpu) of a variable. Using the position of variable defs and its uses in the program under consideration, united with the Basic state reach algorithm, the sets are resolved where *reachi* is the set of variables def that reach *nodei* and set *availi* is the set of all available variable defs at that node which implies the union of global defs at node i and set of all defs that reach node i.

dcui = reachi ∩ c-usei (1)

dcu i,j = availi ∩ p-usei,j (2)

All dcu and dpu sets in the procedure calls fulfilling the all-uses model are resolved in addition with the killing nodes. Killing nodes are the nodes which contain other definition of a variable in its current path and these nodes must not be included. In this approach, GA obtains the arranged version of program under consideration, the list of def-use sets to be enclosed, the number of input variables, the domain and the accuracy of each input variables as input. Here, chromosome is represented by a binary vector. The inputs length is calculated by its domain and accuracy where domain is represented by $Di = (a_i, b_i)$ with each variable in the program taking values from range (a_i, b_i). To achieve accuracy for decimal places d_i, each domain should be cut into $(b_i \;\; a_i)$. 10^{di} size. The accuracy requirement is satisfied if mi denotes the length of a chromosome or string in a way that it fulfills the given equation $(b_i \;\; a_i).10^{di} \le 2\frac{m}{i}$ 1. The mapping from binary string i, into real numbers ranging between (*ai, bi*) is performed using the given formula $x = a_i + x_i'.(\frac{b_i-a_i}{2^{m_i}-1})$ (3). Here, x_i represents decimal value of string i.

$$x_i = a_i + int\left(x_i' \cdot \frac{b_i - a_i}{2^{m_i} - 1}\right) \qquad (3)$$

On putting $d_i = 0$, the given formula can be implemented to map binary string i into any integer ranging between (ai, bi). Each test case for a program which is represented using binary string of specified length is expressed by chromosomes.

Such chromosomes are then represented by decimal number using equations (3) or (4). The fitness value is calculated as

$$eval(v_i) = \frac{no.\ of\ def\text{ - }use\ paths\ covered\ by\ v_i}{total\ no.\ of\ def\text{ - }use\ paths} \qquad (4)$$

For a test case to be effective, its value should be greater than 0. Every test case is evaluated and executed to store the defuse paths covered by test cases as input. Test cases with good fitness values are selected and this selection is done using roulette wheel selection and random selection method. This test cases then become parents and it is repeated till optimal solution is achieved. If no test cases are effective, then individuals are selected randomly. From the above discussion, it is evident that Test Case Generation method proposed using application of GA is more efficient than random testing techniques. However, the sug gested method has not been assessed and compared with other methods such as uniform, stochastic and tournament selection methods

Path Testing

P.R.Srivastava *et al.* and Tai *et al.* (2009) presented an approach to optimize software testing efficiency by recognizing most crucial path clusters in a program. Here, SUT is changed to CFG and weights are designated to the edges of CFG by applying 80-20 rule. 80 percent of the weight of incoming credit is assigned to loops and branches whereas the remaining 20 percent is given to edges in the sequential path. The aggregate of weights along the edges consisting a path resolves the paths criticality. Higher summation denotes more critical path. Therefore, by identifying most critical paths and testing them first, the testing efficiency is elevated

P.R Srivastava proposed another test generation approach based on path coverage testing (Srivastava *et al*., 2008). In this approach, the test data is produced for Resource Request algorithm with the help of Ant Colony Optimization algorithm and GA. Resource request algorithm helps in the prevention of dead locks during resource allocation in the operating systems (Ribeiro *et al*., 2008). ACO is inspired from the behavior of real ants. Its based on the observation that ants look for the shortest possible route between food source and its destination. They produce chemical hormone named as pheromones which aids them to follow their path. The hormones content rises with the increase in number of ants following the trail. With the help of this algorithm, an optimized path ensuring safety sequence in resource request algorithm is accomplished which covers all edges of CFG. GA helps in the generation of suitable test data set which covers the requirements of each process wherein the fitness function acts as a backbone. This fitness function counts the number of times a particular data is entered and its continuation of the resource request algorithm.

Higher count denotes higher chances of avoiding a deadlock. Therefore, the test data with high count values is taken and GA principles are applied to it to yield better results. Concurrently, poor test data is eliminated every time.

Table 1. Last's fuzzy rule for crossover probability (Last *et al.*, 2012)

Parent	$Young_1$	Middle Age_1	Old_1
$Young_2$	Low	Medium	High
Middle Age_2	Medium	High	Medium
Old_2	Low	Medium	Low

Maha Alzabidi *et al.* (2009) schemed automatic structural test case design approach using evolutionary testing. This testing is done by taking path coverage. Here, CFG is adopted to represent a program where nodes serve as the basic blocks and edges between these nodes indicate the programs flow. Significant paths are drawn out from CFG and are then selected as target paths for testing. Several test cases are produced to trace the new paths which leads to the target path. Test result is assessed to resolve that the testing objective criteria is fulfilled because of the execution of the selected path. Here, the fitness function is called a Shifted-Modified Similarity or SMS which is an alteration to the hamming distance. Hamming distance is calculated for descending edges for current and target paths. The discovered similarities are then normalized and added in association with a weighing factor. The obtained value is used as an objective function to assess individuals in the population. The method stated above improves the fitness factor. Per formance of distinct GA principles are studied and applied on different test programs and results show that double crossover is more efficient than single crossover when applied to a test program. This approach is applied on small programs and has not been assessed on complicated programs involving arrays, loops and linked lists.

Applications of GA in Black Box Testing

Mutation Testing: Fuzzy based extension of genetic algorithm (FAexGA) approach was used by Mark Last (2005). The focus here is on finding the minimal set of test cases that are likely to show faults using mutated forms of the original program. In this approach, crossover probability differs as per the assigned age types. The probability of young and old individuals is set to low while the probability of middle age interval is set as high. The probability of very young off springs is also low which helps in exploring capability. Old off springs also has less crossover probability and its dying eventually helps in avoiding a local optimum or premature convergence. On contrary, the middle-age off springs are often used for crossover operations. The probability of crossover is determined using Fuzzy logic controller (FLC). The state variables of FLC consist of the age and lifetime of parent chromosomes. Here, the main focus is on the investigation and exploitation of individuals. The variables that determine the off springs age is included in the fuzzification interface of FLC. It allocates three value to the parents young, middle-age or old. These values help in re solving each roles membership in FLC rule. The fuzzification interface of FLC defines the truth values for each parents for all three variables as shown in the table I.

In table I, each cell elucidates a single fuzzy rule. For example, If Parent 1 is old and Parent 2 is young then the crossover probability is Low. Here, centre of gravity or COG is used as a defuzzification method which helps in calculating values for crossover probability on the basis of values of the linguistic labels as shown in Table 1. The test cases express the inputs of tested software and are interpreted as vectors of binary or continuous values. These cases are initialized on random basis in search of possible input values. Then, genetic operators are utilized and the test cases are resolved based on the faults, exposing capability using mutated versions of the original program. In this case, a Boolean expression comprising of 100 Boolean attributes and three logical operators (AND, OR and NOT) is considered in the case study. The expression was produced at random. Furthermore, to define an evaluation function for every case, a fallacious expression is generated. The chromosomes are 1-dimensional strings in binary of bit length 100. The value of the function F that is used for evaluation is calculated using the formula

$$F(T) = \begin{cases} 1, \ if \ Eval_{Correct} \neq Eval_{Erroneous}(T) \\ 0, \ respectively \end{cases} \tag{5}$$

where T is a 100 bit 1-D binary chromosome which represents a single test case . EvalCorrect (T) or EvalErroneous (T) are binary results of the applying chromosome T to the correct or fallacious expression. FAexGA has not been calculated on real program. Furthermore, practical and continuous evaluation functions need to be refined.

Regression Testing: In regression testing (You and Lu, 2012), all the repetitive test cases are removed and by applying Genetic Algorithm, the total running time of the test is reduced. To depict the connection between the requirements and test cases a satisfaction matrix S_{ij} is been applied. In this matrix, columns represent test cases and rows represent requirements. In this matrix, $S_{ij} = 0$ shows that *j*th test case t_j does not satisfy the *i*th requirement else $S_{ij} = 1$. The reduction problem for time aware regression testing can be defined as

$$Minimise \sum_{j=1}^{n} C_j x_j \tag{6}$$

After the removal of repetitive test cases, the above equation depicts fitness function and also shows the total running of time of remaining test cases. Here in the given equation, x_j is a test case of T_j and C_j shows the running time of test case T_j. Test case T_j exists in T_{min} if and only if $x_j = 1$whereas Test case T_j does not exist in $Tmin$ if $x_j = 0$. If $T_1 = t_1, t_3, t_5$, then Require $T_1 = r_1, r_2, r_3, r_4 = R =$ Require (T). $Tmin = T_1 = t_1, t_3, t_5$, by applying greedy algorithm. It comprises of seven test cases i.e. (You and Lu, 2012), amongst which $Tmin$ is the minimum regression testing. The reduced regression testing for $Tmin$ is $X = x_1, x_2, x_n$. For instance, $X = 1,0,1,0,1,0,0$ depicts $Tmin = t_1, t_3, t_5$. Chromosome

X is represented using a bit string $X = x_1, x_2, \ldots, x_n$. To change unfeasible solution into a feasible solution, a repair operator is implemented. The CHU's genetic algorithm is similar to crossover, mutation and repair operator. In CHU's genetic algorithm, mutation rate is equal to two bits per string whenever uniform crossover is implemented.

$$R_1 = \left(1 - \frac{\text{no. of reduced test suite}}{\text{no. of unreduced test suite}}\right) \times 100\% \quad (7)$$

$$R_2 = \left(1 - \frac{\text{total running time of reduced test suite}}{\text{total running time of unreduced test suite}}\right) \times 100\% \quad (8)$$

In the above equation, R_1 depicts the rate of reduction in the number of test cases and R_2 depicts the rate of reduction in the running time of test cases. This paper examines Genetic Algorithm based reduction with vector based reduction on all the test cases.

Conclusion

In this paper, uses of GA in various sorts of programming testing are talked about. The GA is utilized with fuzzy and also in the neural systems in various sorts of testing. It is found that by utilizing GA, the outcomes and the execution of testing can be accelerated. Our future work will include applying GA for relapse testing in online applications.

REFERENCES

Alzabidi, M., Kumar, A. and Shaligram, A. 2009. "Automatic software structural testing by using evolutionary algorithms for test data generations," *IJCNS International Journal of Computer Science and Network Security*, vol. 9, no. 4, pp. 390–395.

Girgis, M. R. 2005. "Automatic test data generation for data flow testing using a genetic algorithm." *J. UCS*, vol. 11, no. 6, pp. 898–915.

Last, M., Eyal, S. and Kandel, A. 2005. "Effective black-box testing with genetic algorithms," in *Haifa Verification Conference*. Springer, 2005, pp. 134–148.

Ribeiro, J. C. B., Rela, M. Z. and de Vega, F. F. 2008. "A strategy for evaluating feasible and unfeasible test cases for the evolutionary testing of object oriented software," in *Proceedings of the 3rd international workshop on Automation of software test*. ACM, pp. 85–92.

Srivastava, P. R. and Kim, T.H. 2009. "Application of genetic algorithm in software testing," *International Journal of oftware Engineering and its Applications*, vol. 3, no. 4, pp. 87–96.

Srivastava, P. R., Ramachandran, V., Kumar, M., Talukder, G., Tiwari, V. and Sharma, P. 2008. "Generation of test data using meta heuristic approach," in *TENCON 2008-2008 IEEE Region 10 Conference*. IEEE, pp.1–6.

You, L. and Lu, Y. 2012. "A genetic algorithm for the time-aware regression testing reduction problem," in *Natural Computation (ICNC), 2012 Eighth International Conference on*. IEEE, pp. 596–599.

****