Test Case Prioritization Based on Genetic Algorithm and Test-Points Coverage

Weixiang Zhang, Bo Wei, and Huisen Du

Beijing Institute of Tracking and Telecommunications Technology, Beijing, China
wxchung@msn.com

Abstract. By optimizing the execution order of test cases, test case prioritization techniques can effectively improve the efficiency of software testing. Test case prioritization is becoming a hot topic in software testing research. Combining genetic algorithm with test-points coverage, this paper obtains some meaningful research results in test case prioritization, especially for the functional testing. Firstly, presents two new test case prioritization evaluations APTC and its improvement APRC_C. As focused on test-points coverage, these evaluations are more suitable for black-box testing. Then, proposes a test case prioritization method based on genetic algorithm, whose representation, selection, crossover and mutation are designed for black-box testing. Finally, verifies the proposed method by experiments data. The experimental results show that the proposed method can achieve desired effect.

Keywords: Software Testing, Test Case Prioritization, Genetic Algorithm, Evaluation Function, Black-box Testing, Software Engineering.

1 Introduction

Software testing is one of the most important means to ensure software quality. Statistics show that software testing accounts for more than 50% of the total cost of software development in general [1]. With the increasing software complexity, software testing is becoming more and more difficult and expensive. How to select a few of test cases from huge available collection to test software effectively has become an outstanding problem.

Test case prioritization technology (TCP) is one of the most important research directions of software testing. According to the importance degree based on some specific criteria, TCP sorts the test cases and then executes them sequentially. By optimizing the execution order of test cases, TCP can effectively improve the efficiency of software testing.

A general description of TCP is as follow: for a given test case set T, the total permutation PT of T and a sort function $f: PT \to R$, to find $T' \in PT$, let $f(T') \geq f(T'')$, for $\forall T'' \in PT(T'' \neq T')$. A typical sort goal is to maximize the rate of early defect detection. Other sort goals can be considered such as code coverage, defect detection rate of high-risk, defect detection rate of modified code related and system reliability and so on.

X.-h. Sun et al. (Eds.): ICA3PP 2014, Part I, LNCS 8630, pp. 644–654, 2014.

The current researches focus on code-based test case prioritization techniques. Usually based on the coverage power to program (e.g. statement, branch or function) of every test case, set weights and apply the greedy method to guide test ordering [2]. Some researchers try using typical machine learning methods to improve the effect of the implementation of TCP technology, such as meta-heuristic search [3], Bayesian networks [4] and cluster analysis [5].

By simulating the process of biological evolution, genetic algorithm (GA) searches the optimal solution for the optimization problem. GA maintains a population of potential solutions; it randomly samples in the entire search space, and evaluates each sample in accordance with its fitness function. In the genetic algorithm, some operators such as selection, crossover and mutation are used, which constantly iterates (each iteration is equivalent to one cycle of biological evolution) to search for a global optimal solution, until the termination condition is met.

In the test data generation, the search space of GA is the input domain of the software and the optimal solution is some test data to meet for the specified testing purposes. GA is more often used in the structural testing, taking running path as optimization goal [6-9]. Researches on using GA in the functional testing are not very extensive, including a method based on Z language specification [10] and the method based on pre / post-conditions [11]. Because of the high cost of formalization, these methods are difficult for large systems.

Aiming at the functional testing, this paper proposed a new test case prioritization evaluation, named Average Percentage of Test-Points Coverage (APTC) and its improvement APRC_C. Then, using APTC or APTC_C as fitness function, we proposed a test case prioritization method based on genetic algorithm. At last, we verified our method by simulation experiments.

2 Evaluation of Test Case Prioritization

2.1 Some Existing Evaluation Function

The purpose of software testing is as much and as quickly as possible detection of defects in the software. Compared with software testing in random sequence, one of the biggest benefits of TCP is able to check out the errors faster. Rothermel pointed out that TCP can be evaluated by the relationship between the number of test cases executed and errors detected.

Rothermel proposed an evaluation named APFD (Average Percentage of Fault Detection). Elbaum [12] gave the formula of APFD.

Suppose test case set T contains n test cases and m defects can be detected. For given test cases sequence, TF_i represents the precedence of the first test case to detect the i^{th} defect in the sequence, so there is $APFD=1-\frac{TF_1+TF_2+\cdots+TF_m}{nm}+\frac{1}{2n}$.

APFD ranges between 0 to 100%, the higher the value, the faster the defect detection.

Since people cannot predict defect detection information of test cases, Li Zheng et al [13] then proposed APBC (average percentage of block coverage), APDC (average percentage of decision coverage) and APSC (average percentage of statement coverage) that could evaluate the coverage power to program (block, decision, statement respectively) of the test cases sequence. Formulas of these evaluations are similar to APFD, e.g. $APBC = 1 - \frac{TB_1 + TB_2 + \cdots + TB_m}{nm} + \frac{1}{2n}$, where TB_i represents the precedence of the first test case to cover the i^{th} block.

As the coverage information can be obtained through coverage analysis tools without test cases execution, so it is feasible to use APBC, APDC and APSC before software testing is over. However, these evaluations are clearly more suitable for structural testing or white-box testing.

Therefore, we propose a new test case prioritization evaluation based on test-points coverage.

2.2 Average Percentage of Test-Points Coverage (APTC)

In black-box testing, testers design test cases based on software specifications. Firstly, transform the software requirements into the software test requirements using of requirements analysis; secondly, decompose the test requirements into a lots of test points; then, design test cases aimed at test points respectively and then form a collection of test cases. Each test case may correspond to one or more test points.

For a test cases collection $\Phi = \{T_1, T_2, \cdots, T_m\}$, define APTC(average percentage of test-point coverage) as follows:

$$APTC = 1 - \frac{TT_1 + TT_2 + \cdots + TT_m}{nm} + \frac{1}{2n} \tag{1}$$

Wherein, n denotes test cases count, m denotes test-points count, TT_i represents the precedence of the first test case to cover the i^{th} test-points. APTC ranges between 0 to 100%, the higher the value, the faster the test-points coverage.

Taking into account the different importance level of each test-point and the different cost of each test case, adjust the evaluation object to the importance value of test-points covered by unit test case cost. Its formulation is expressed as follows:

$$APTC\_C = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TT_i}^{n} t_j - \frac{1}{2}t_{TT_i}\right)\right)}{\sum_{j=1}^{n} t_j \times \sum_{i=1}^{m} f_i} \tag{2}$$

Wherein, f_i represents the value of the importance of the i^{th} test-point, t_j represents the cost of the j^{th} test case, and the other variables is consistent with formula (1). When the importance of all test-points is the same and the costs of all test cases is equal, equation (2) reduces to equation (1). That is, APTC_C degenerates into APTC.

Consider the examples shown in Table 1, in which there are 5 test-points and 10 test cases. For test case sequences A-B-C-D-E and E-D-C-B-A, the APTC values is 50% and 64% respectively, so the effect of the latter is better than the former. Figure 1 shows the relationship between the rate of executed test case and the rate of covered test-points in different test case sequences. APTC is equal to the ratio of the area of the shaded portion under the line to whole area.

Table 1. An example of the relationship between test cases and test-points

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*
A	X				X					
B	X				X	X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X

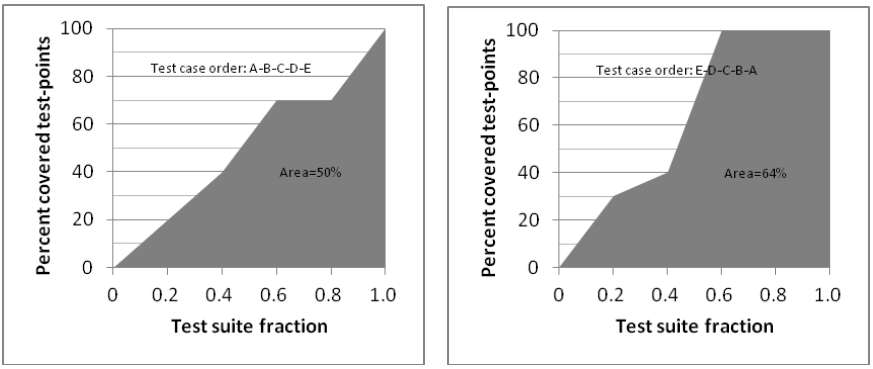

Fig. 1. APTC of different test case sequences

Supposed that the cost of test case B is two times of the other test cases, and the importance value of test-points covered by test case B (that is, test-point 1, 5, 6, 7) is two times of the other test-points. According to the formula, it is easy to calculate that APTC_C of A-B-C-D-E and E-D-C-B-A are 56% and 68%.

After replacing the horizontal and vertical coordinates by "test suite fraction with cost" and "percent covered test-points with importance", we can draw the diagram of APTC_C in a similar way to APTC, as shown in Figure 2.

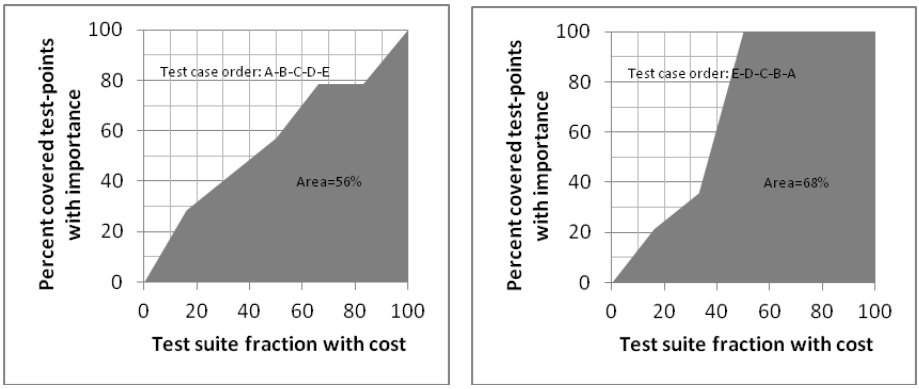

Fig. 2. APTC_C of different test case sequences

In the case of large differences in test-points' importance or/and test cases' costs, APTC_C is more appropriate than APTC.

3 Test Case Prioritization Based on Genetic Algorithm

3.1 Process of Genetic Algorithm

In genetic algorithm (GA), each effective solution to the problem is called a "chromosome", with respect to each individual of population. A chromosome is a coded string using a specific encoding method, and each unit of the coded string is called a "gene". By comparing the fitness values, GA distinguishes the pros and cons of chromosomes. The chromosome with larger fitness value is more outstanding.

In GA, fitness function is used to calculate the fitness value of corresponding chromosome; selection is used to choose some individual in accordance with certain rules, and form the parent population; crossover is used to interchange part of genes of two individuals to generate their offspring chromosomes; mutation is used to change a few genes of selected chromosome to get a new one.

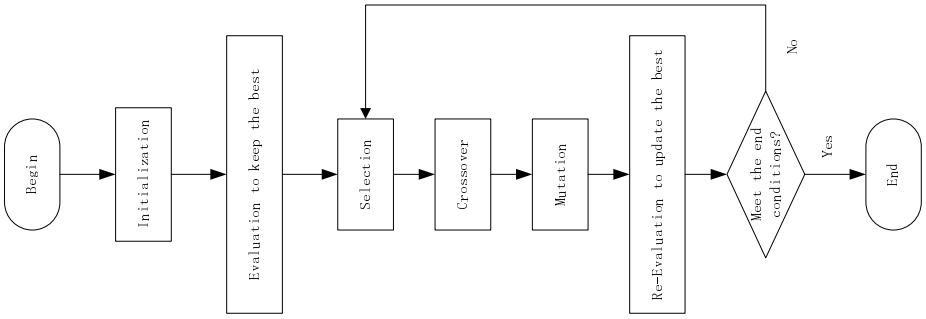

Fig. 3. Flowchart of GA

The main steps of GA include:

STEP1. To Initialize a population with N chromosomes, get the genes of every chromosome in random manner and keep them inside the range of the problem definition. Denote the count of generation $Generation$ and let $Generation = 0$.

STEP2. To evaluate each chromosome using fitness function, calculate the fitness value of every chromosome, save the best one whose fitness is largest and name it $Best$.

STEP3. To do selection using of the manner such as Roulette wheel, generate the population with N selected chromosomes.

STEP4. To do crossover in accordance with the probability p_c. Each couple of selected chromosomes interchange some genes to generate their two offspring and replace themselves; other chromosomes retain in the population.

STEP5. To do mutation in accordance with the probability p_m. Some new chromosomes are generated separately through altering a few genes of corresponding selected chromosome; other non-selected chromosomes retain in the population.

STEP6. Re-evaluate each chromosome using the fitness function. If the largest fitness value in the new population is better than $Best$'s, replace $Best$.

STEP7. Let $Generation++$. If $Generation$ exceeds the specified maximum generation or $Best$ achieves the specified error requirement, end the algorithm; otherwise, goto STEP3.

3.2 Design of Representation

In TCP, each chromosome is a test case sequence. What we concern is the order of test cases instead of their specific composition. So, we assign a unique natural number for every test case, and then each individual can be encoded as an ordered sequence of the numbers. The scale of initial population affects to the search capability and operational efficiency of GA, so it usually range from 20 to 150.

For an example, the representation of an individual $T_1 - T_2 - T_3 \cdots - T_m$ of test cases suite $\Phi = \{T_1, T_2, \cdots, T_m\}$ is shown as in Figure 4.

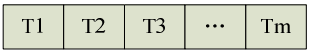

Fig. 4. Design of representation

3.3 Design of Fitness Function

Fitness function is used to calculate the fitness value of each chromosome and to guide the search direction of GA. So, it is the key part of genetic algorithm implementation. Generally, the fitness value is between 0 and 1. The individuals with larger value are more excellent, and have greater probability to evolve to the next generation.

In general, for a test case, the more the count of test points to cover, the larger the probability to find defects. Therefore, taking consider of improving the coverage of test cases to test-points as evolutionary goal, APTC is good to be used as the fitness function. Furthermore, if the goal is designed to improve the importance degree of covered test-points in unit test cost, APTC_C is more suitable for fitness evaluation.

3.4 Design of Selection

Use roulette wheel selection. For example, for a population with k individuals, denotes $fitness_i$ for fitness of i^{th} individual, the roulette wheel selection include 5 steps: first, calculate the fitness percent $fitness_i / \sum fitness_i$ which show the capability of each individual to yield offspring; second, sort the individuals by descending order of their fitness percent; third, for each individual, sum up all the fitness percent of individuals that are in ahead of it; then, select the first individual whose summed fitness is greater than the random number $r_s \in [0,1]$; lastly, loop above steps until enough individuals is born. As can be seen, the individuals with greater fitness will have larger probability to be selected to produce the next generation, which is consistent with the principles of evolution.

3.5 Design of Crossover

Suppose Q_1 and Q_2 are the selected individuals to do crossover, D_1 and D_2 are the individuals after crossover. Set crossover probability p_c and get a random number $r_c \in [0,1]$, obtain D_1 and D_2 when r_c is less than p_c in a manner as follows. For a random crossover point $k\,(1 \le k \le m)$, D_1 consists of two parts: the first part is the first k test cases of Q_1; the second part is from Q_2 excluding the first k test cases of Q_1. Similarly, D_2 consists of two parts: the first part is the first $(m-k)$ test cases of Q_2, the second part is from Q_1 excluding the first $(m-k)$ test cases of Q_2. Figure 5 shows an example. The value of p_c is generally between 0.4 and 0.99.

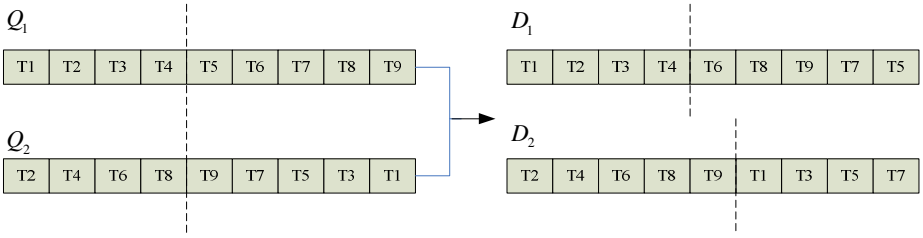

Fig. 5. Design of crossover

3.6 Design of Mutation

Mutation change some genes of the selected chromosome to generate a new individual. Set mutation probability p_m and get a random number $r_m \in [0,1]$, obtain the new individual when r_m is less than p_m in a manner as follows: according to the importance of test-points covered by test cases, select two test cases of the parent individual using of the manner as roulette wheel selection; then, exchange the positions of the two selected test cases to form a new individual. Figure 6 shows an example. The value of p_m is generally between 0.001 and 0.1.

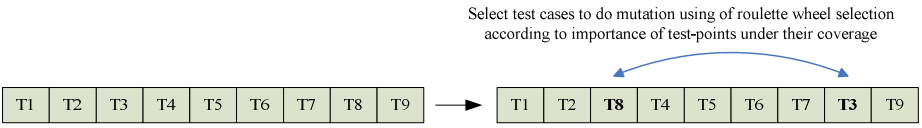

Fig. 6. Design of mutation

4 Simulation Experiment

We use triangle classification program to illustrate the effect of the proposed method. Triangle classification program analyzes the values and their relationship of the three input variables (x, y, z) to determine the type of the triangle composed by them. There are total 7 test-points of triangle classification program, as shown in Table 2. We design total 6 test cases by black-box testing method. The relationship between test cases and test-points is shown in Table 3.

Table 2. Test-points of triangle classification program

No.	Name	Description	Importance
1	IsNumError	Judge if (x, y, z) is out of the valid range or not, i.e., check whether each variable is less than 0 or greater than the maximum value.	1
2	IsTriangleError	Judge if (x, y, z) is able to form triangle or not, i.e., check whether the sum of any two sides is large than the other side.	1
3	IsTriangle	Judge if (x, y, z) is suitable for triangle sides or not, i.e., check whether each variable is within the valid range and the sum of any two sides is large than the other side.	1
4	IsScalTriangle	Judge if the triangle is inequilateral or not in the condition that (x, y, z) can form triangle, i.e., check whether $x \neq y \neq z$.	2
5	IsRightTriangle	Judge if the triangle belongs to right triangle or not in the condition that (x, y, z) can form triangle, i.e., check whether x^2+y^2=z^2.	2
6	IsIsosTriangle	Judge if the triangle belongs to isosceles triangle or not in the condition that (x, y, z) can form triangle, i.e., check whether x=y、y=z or x=z.	2
7	IsEquiTriangle	Judge if the triangle belongs to equilateral triangle or not in the condition that (x, y, z) can form isosceles triangle, i.e., check whether x=y =z.	3

Table 3. Relationship between test cases and test-points of triangle classification program

No.	Test case name	Test case cost	1	2	3	4	5	6	7
A	NumErrorTest	1	X						
B	TriangleErrorTest	1		X					
C	ScalTriangleTest	2			X	X			
D	RightTriangleTest	2			X		X		
E	IsosTriangleTest	2			X			X	
F	EquiTriangleTest	3						X	X

Set the values of GA parameters as shown in Table 4. The best three results is obtained by the proposed method using of APTC and APTC_C separately, as shown in Table 5 and Table 6.

Table 4. Values of GA parameters in triangle classification program

No.	Name	Value
p_c	Probability of crossover	0.85
p_m	Probability of mutation	0.05
$G_{\max}$	Maximum count of iterations	50
N	Scale of population	150

Table 5. Partial results of APTC althgram

No.	Sequence of test cases	APTC
1	C-F-D-E-A-B/D-C-E-F-A-B/F-E-D-C-A-B	0.4881
2	A-C-D-F-E-B/C-A-D-E-F-B/F-D-C-A-E-B	0.4643
3	A-C-D-B-F-E/C-A-D-B-F-E/D-B-F-A-C-E	0.4405

Table 6. Partial results of APTC_C althgram

No.	Sequence of test cases	APTC_C
1	D-C-E-A-B-F/D-C-E-B-A-F	0.8030
2	C-D-E-A-B-F/C-D-E-B-A-F	0.7955
3	D-C-A-E-B-F/D-C-B-E-A-F	0.7879

5 Conclusion

The use of genetic algorithms in test case prioritization, can effectively reduce the blindness in test cases executed order and so improve the efficiency of software testing.

This paper proposed a new test case prioritization evaluation APRC and its improvement APRC_C for functional testing. As focused on test-points coverage, these evaluations are more suitable for black-box testing.

In addition, this paper presented an automated test case prioritization method using of genetic algorithms which adopted APTC or APTC_C as fitness function. The designs of representation, selection, crossover and mutation in GA are aimed at black-box testing. We gave the specific steps of the method and validated it by experimental data.

The experimental results show that the proposed method can achieve expected results. It provides an effective technical approach to the test case prioritization problem. In the future, we will do further research in test-points automatically conversation and applications of GA in the automated generation of black-box test cases.

References

1. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press, Cambridge (2008)
2. Chen, X., Chen, J.H., Ju, X.L., Gu, Q.: Survey of test case prioritization techniques for regression testing. Journal of Software 24(8), 1695–1712 (2013) (in Chinese)
3. Li, Z., Harman, M., Hierons, R.M.: Search algorithms for regression test case prioritization. IEEE Trans. on Software Engineering 33(4), 225–237 (2007)
4. Mirarab, S., Tahvildari, L.: A prioritization approach for software test cases based on bayesian networks. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 276–290. Springer, Heidelberg (2007)
5. Yoo, S., Harman, M., Tonella, P., Susi, A.: Clustering test cases to achieve effective and scalable prioritization incorporating expert knowledge. In: Proc. of the Int'l Symp. on Software Testing and Analysis, pp. 201–212. ACM Press (2009)
6. Jones, J.A., Harrold, M.J.: Test-suite reduction and prioritization for modified condition/decision coverage. IEEE Trans. Software Eng. 29(3), 195–209 (2003)
7. Quan, J., Lu, L.: Research test case suite minimization based on genetic algorithm. Computer Engineering and Applications 45(19), 58–61 (2009)
8. Lin, J.C., Yeh, P.L.: Using genetic algorithms for test case generation in path testing. In: Proc. of the Asian Test Symposium, pp. 241–246 (2000)
9. Jones, B.F., Sthamer, H.H., Eyres, D.E.: Automatic structural testing using genetic algorithms. Software Engineering Journal 11(5), 299–306 (1996)
10. Baresel, A., Sthamer, H., Schmidt, M.: Fitness function design to improve evolutionary structural testing. In: Genetic and Evolutionary Computation Conference, pp. 1329–1336. IEEE Press, New York (2002)
11. Wegener, J., Buhler, O.: Evaluation of different fitness functions for the evolutionary testing of an automatic parking system. In: The Genetic and Evolutionary Computation Conference, pp. 1400–1412. Seattle, Washington (2002)

12. Elbaum, S., Malishevsky, A., Rothermel, G.: Prioritizing test cases for regression testing. In: Proc. of the Int'l Symp. on Software Testing and Analysis, pp. 102–112. ACM Press (2000)
13. Elbaum, S., Malishevsky, A., Rothermel, G.: Incorporating varying test costs and fault severities into test case prioritization. In: Proc. of the Int'l Conf. on Software Engineering, pp. 329–338. IEEE Press, New York (2001)