2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology

Security Assurance with Metamorphic Testing and Genetic Algorithm*

Guowei Dong, Shizhong Wu, Guisi Wang, Tao Guo+ and Yonggang Huang
China Information Technology Security Evaluation Center, Beijing 100085, China
{donggw, tiger, wanggs, guotao+, hyg}@itsec.gov.cn

***Abstract*—The correctness of mission-critical software is an important part of information security, but oracle problem and test data generation are constraints for some programs. Although metamorphic testing (MT) is practical for programs with oracle problem and evolutionary testing (ET) is a good application of genetic algorithm (GA) for automatic test data generation, fitness functions used in ET are not always effective at target search. This article provides a method for improving ET's efficiency by considering metamorphic relation (MR) when fitness function is constructed, and finally some conclusions are presented.**

Keywords-security; correctness; metamorphic testing; genetic algorithm

I. INTRODUCTION

Mission-critical software is important for people's life and property. The software used in aerospace, weapon equipment, process control, nuclear energy, transportation, medical and health are all belong to this kind. So testing for their correctness is significant for security assurance. But sometimes, there are oracle problems [1] in correctness testing, that is, it is difficult to construct expected results for software under test (SUT). To solve this problem, T.Y. Chen presented metamorphic testing (MT) [1], which tests software by checking relations among execution results.

Definition (metamorphic relation) Let program P is used to implement function f, and $x_1, x_2, \ldots, x_n$ (n>1) are different variables for f. If their satisfaction of relation r implies that $f(x_1),\ldots,f(x_n)$ satisfy relation r_f:

$$r(x_1, x_2, \ldots, x_n) \Rightarrow r_f(f(x_1), f(x_2), \ldots, f(x_n)) \quad (1)$$

then (r, r_f) is called a metamorphic relation (MR)[1] of P. It is obviously that if P is correct,

$$r(I_1, I_2, \ldots, I_n) \Rightarrow r_f(P(I_1), P(I_2), \ldots, P(I_n)) \quad (2)$$

will also be fulfilled. $I_1, I_2, \ldots, I_n$ are test cases according to x_1, x_2, …, x_n. The test cases given originally are original test cases (OTC), while the others generated upon MR and OTCs are follow-up test cases (FTC) [1].

MT has two prominent traits: (1) In order to check execution results, MR(s) should be constructed. (2) For the purpose of getting OTC, MT should work with other test case gene ration methods.

In the past ten years, great efforts have been done in producing effective OTCs, choosing useful MRs, constructing new test methods with MT and applying MT in new type software. In detail, Chen and Wu gave the evaluation and comparison of special case testing, MT with special and random test cases [2, 3]. Wu also presented iterative MT to generate OTC in a chain style[4], and we improved it[5]; Chen and Mayer introduced how to select useful MR[6, 7]; New proving and test methods were produced by integrating MT with global symbolic evaluation[8], fault-based testing[9]; Because of its validity, MT has been widely used in the testing of numerical programs[10], graph theory programs[11], computer graphics[11,12], compilers [11], ubiquitous computing[13, 14], SOA software[15, 16] and object-oriented programs[17]; We also provided some testing criteria for MT[18].

Genetic algorithm (GA) is a meta-heuristic search technique based on the Darwin's theory of biological evolution [19, 20]. The process of GA is: First, a population is randomly or heuristically initialized, in which each individual represents a candidate solution. And then a series of genetic operations (code, selection, crossover, mutation, survival), which mimic natural evolution are proceeded orderly and iteratively, according to the fitness values of the individuals, which is evaluated by a pre-defined fitness function. GA renews a population with successive generations of genetic operations until optimum is achieved, or another stopping condition is fulfilled [19, 20].

Evolutionary testing (ET) is the application of GA for test data generation, which has been used for systematizing and automating the conventional test methods, such as structural testing, functional testing, and the non-functional properties testing [21-26].

The search of test case in ET is based on fitness function, and the efficiency of test data generation is mostly decided by the complexity of fitness function. In this article, we improve previous ET by considering MR when fitness function is constructed. Some case studies are also provided to explain the strength of this new method.

II. IMPROVEMENT OF STRUCTURAL ET BASED ON METAMORPHIC METHOD

Structural ET utilizes GA for test data generation to satisfy certain coverage criteria. In the process of evolution, an individual is a single test input and the fitness function is determined by a particular structural construct. Depending on the construction of the fitness function, previous work on structural ET can be divided into three categories: the coverage-oriented approach, the distance-oriented approach, and the hybrid approach [27]. In distance-oriented approach, certain structure constructs as testing objects are partitioned into separate sub-goals, and the task is to generate test data that lead to an execution of all the sub-goals, such as branches or conditions [27]. In this approach a specialized

* This work was supported in part by the National Natural Science Foundation of China (90818021).
+ Corresponding Author.

978-0-7695-4191-4/10 $26.00 © 2010 IEEE
DOI 10.1109/WI-IAT.2010.101

fitness function is formulated for each sub-goal. For example, the fitness value $f(e)$ is | a - b | when expression e (a=b) is false, and zero otherwise [27].

Theoretical Principle

In distance-oriented approach, fitness function is constructed upon the logic product of some branch conditions. Sometimes it is difficult to hit test objective. Moreover, ET only considers test data generation, but test oracles are not mentioned. At the same time, MT is good at solving oracle problem and it is also an effective technique for test case generation [1, 11], so we decide to improve previous ET (PET) with metamorphic method.

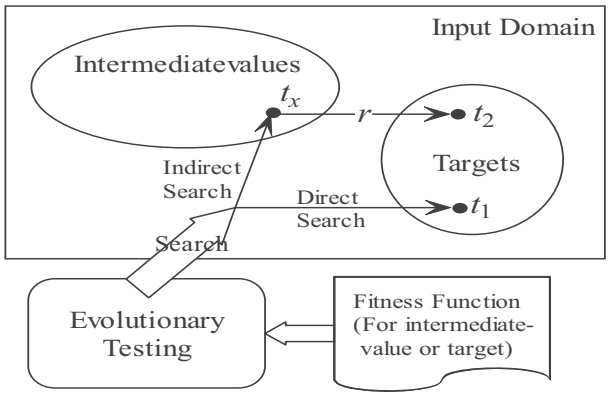

Fig.1. Generation of targets

If (r, r_f) is a MR for SUT P, and $TC_r(t)$ is the FTC of test case t upon relation r, then we can get target in two ways: (1) Attain t_1 which is a member of the objective set by direct search; (2) Get t_2 by searching out an inter-mediatevalue t_x and figuring out t_x's FTC, where $TC_r(t_x) = t_2$ is a target case. Fig.1 shows these two ways. In this situation, we should consider two parts when forming fitness function. The fitness function for a specified testing objective can be constructed as:

$$\text{Fitness}(t, \text{objective}) = \min(\text{distance}(t, \text{objective}), \text{distance}(TC_r(t), \text{objective})) \quad (3)$$

distance is the fitness function in distance-oriented approach, and it has been regulated into the range of [0, 1].

The improved ET (IET) is based on the theory above. The steps of IET are as follow: First, fitness function in the form of formula (3) is built upon SUT's structural construct (such as branch conditions) and relation r. And then, evolutionary process is started with the fitness function. This course is similar to PET. When a test case $test_0$ with the fitness value of zero is found, $TC_r(test_0)$ will be figured out soon. In this way, either $test_0$ or $TC_r(test_0)$ is an objective. Finally, SUT's function is validated by r_f and the executions of these two test cases. This entire process is illustrated in Fig.2.

In each experiment here, SUT's inputs are floating point numbers. So for a single individual, its integral part and decimal part are generated separately as two integers, which is good for the coding of individuals. Here, each initial population is generated randomly and contains 8 individuals. Every individual's two parts (integral and decimal) are coded with gray code respectively [19]. Individuals do single-point crossover with a probability of 0.9, while each of them makes bit mutation with a probability of 0.1 [20].

III. CASE STUDIES

ET is good at searching for particular numbers limited by intricate constricts or special array with numerous members automatically. We provide two programs, TriSquare and Determinant here to show IET's efficiency. Their inputs belong to the two types of data above, and their expected outputs are difficult to get.

A. TriSquare

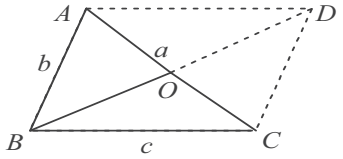

Fig.3. mr_4 and mr_5

In reference [5], code of TriSquare is described in detail. This program first decides whether 3 positive real numbers, a, b and c, could form a triangle, and then calculates the triangle's square. We construct two MRs, mr_4 and mr_5, according to parallelogram's characteristics. In Fig.3, it is obvious that $|AC|^2 + |BD|^2 = |AB|^2 + |BC|^2 + |CD|^2 + |DA|^2$, so $|OB|$ is easy to get. Triangle ABC is twice as large as OAB, and also twice as large as OBC. The relations contained in

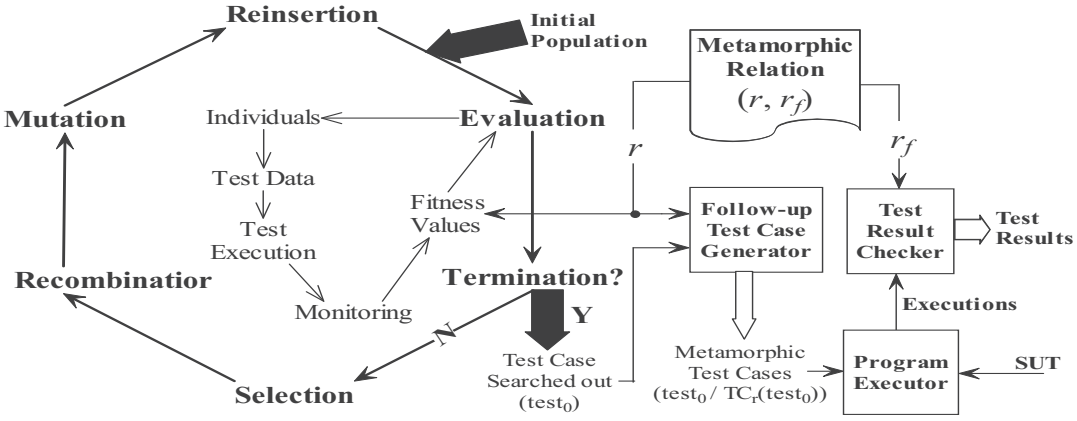

Fig.2. Process of IET

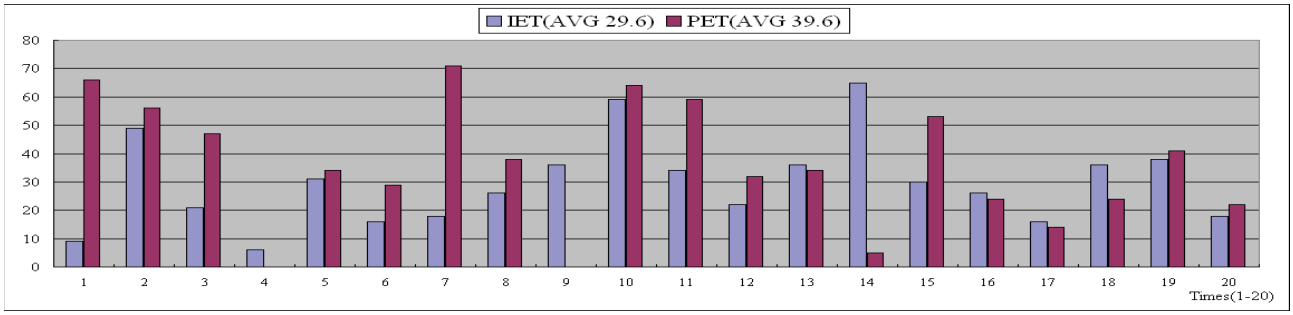

Fig.4. Iteration numbers needed for searching equilateral triangles with IET and PET

two MRs are:

$(a' = b, b' = a/2, c' = \sqrt{2b^2 + 2c^2 - a^2}/2) \Rightarrow$
TriSquare(a, b, c) = 2×TriSquare(a', b', c')

$(a' = \sqrt{2b^2 + 2c^2 - a^2}/2, b' = c, c' = a/2) \Rightarrow$
TriSquare(a, b, c) = 2×TriSquare(a', b', c')

Equilateral triangle (a=b=c) is a particular kind of test case for TriSquare, and it executes $Path_5$ [5]. It is difficult to generate this type of test case with random method, but much easier with ET. Here, we take mr_4 and mr_5 for fitness function construction. The evolutionary process will terminate when any one of $ABC(a, b, c)$, $OAB(b, a/2, \sqrt{2b^2 + 2c^2 - a^2}/2)$ and $OBC(\sqrt{2b^2 + 2c^2 - a^2}/2, c, a/2)$ is equilateral. Following formula is used as the measure for distance between test case and objective:

$$\min\ (\ (|a^2 - b^2|+|a^2 - c^2|+|b^2 - c^2|),\ (|b^2-a^2/4\ |+|b^2-(2b^2+2c^2-a^2)/4\ |+|a^2/4 - (2b^2+2c^2-a^2)/4\ |),\ (|(2b^2+2c^2-a^2)/4 -c^2\ |+|(2b^2+2c^2-a^2)/4 -a^2/4|+|c^2- a^2/4\ |)\ )$$

It can be simplified to:

$$\min\ (\ (|a^2-b^2| + |a^2-c^2| + |b^2-c^2|),\ ((|\ 4b^2-a^2\ |\ +|\ a^2+2b^2-2c^2\ |)/4 + |\ a^2-b^2-c^2\ |/2),\ ((|\ 2b^2-2c^2-a^2\ |\ +|\ 4c^2-a^2\ |)/4 + |\ b^2+c^2-a^2\ |/2)\ ) \quad (4)$$

Formula (4) is only about a^2, b^2 and c^2, so in evolutionary process, each individual represents a combination of three square values, that is, (a^2, b^2, c^2). The range for each square value is assigned to [1, 100].

Mutant3 (Replacing "/2" in sentence 18, 23, 28 with "*2") and Mutant4 (Replacing sentence 30 with "return sqrt(3) *a*a/2") are imported into TriSquare for evaluation [5].

We searched for objectives with IET and PET (fitness function is built upon (a=b=c)) 20 times respectively, and the largest iteration number for each search was 100. Fig.4 gives the iteration numbers needed for hitting targets. Although IET is not always better than PET, its average value shows prominent superiority (29.6<39.6), which indicates that IET could accelerate evolutionary convergence in most situations. IET's hit percent is 100%, higher than PET's (90%). Because of the complexity of calculating formula (4), the execution time for IET is longer. But when a computer with 1.66GHz CPU and 1GB memory is used, the execution times for 20 evolutions with IET and PET are 0.117 second and 0.098 second respectively. In one word, IET is more effective in test data generation.

Tab.1 lists three test cases generated by IET. "FTC4" and "FTC5" are the FTCs based upon mr_4 and mr_5. Test cases with "OBJ" behind are the ones that satisfy (a=b=c). From this table, we can see that Case2 and its FTCs could not detect Mutant3. But with PET, all test cases searched out belong to this kind. After detail analysis, we find that the sentence containing Mutant3 can be covered when executing Case1's or Case3's FTCs, while cannot when executing Case2's. That is, if MR is considered in fitness function building, the test area will might be enlarged. As a result, IET performs better in fault detection.

B. Determinant

Determinant is another Java implementation for determinant computation, whose code is shown in Fig.5. **AlgComp**(**int**[] A, **int** n, **int** i) is a method for calculating complement minor of the element at matrix A's 1st row and (i+1)th column. We construct a MR mr_6 for this program on the basis of determinant's trait:

$$|A| = \begin{cases} |A'| & A' \text{ is generated by performing an even number of row-swaps on A} \\ -|A'| & A' \text{ is generated by performing an odd number of row-swaps on A} \end{cases}$$

where A is a square matrix, and |A| is the output of

Tab.1. Several test cases for TriSquare

Values searched out (a^2, b^2, c^2)	Test cases (OTC/FTC (MR_1 / MR_2))		Outputs / Satisfy corresponding MR?			
			SUT with Mutant3		SUT with Mutant4	
(24, 6, 18) Case1	OTC	(4.8990, 2.4495, 4.2426)	5.19615242		5.19615242	
	FTC4	(2.4495, 2.4495, 2.4495) OBJ	2.59807621	Y	5.19615242	**F**
	FTC5	(2.4495, 4.2426, 2.4495)	10.39230485	**F**	2.59807621	Y
(55, 55, 55) Case2	OTC	(7.4162, 7.4162, 7.4162) OBJ	23.81569860		47.63139721	
	FTC4	(7.4162, 3.7081, 6.4226)	11.90784930	Y	11.90784930	**F**
	FTC5	(6.4226, 7.4162, 3.7081)	11.90784930	Y	11.90784930	**F**
(40, 30, 10) Case3	OTC	(6.3246, 5.4772, 3.1623)	8.66025404		8.66025404	
	FTC4	(5.4772, 3.1623, 3.1623)	17.32050808	**F**	4.33012702	Y
	FTC5	(3.1623, 3.1623, 3.1623) OBJ	4.33012702	Y	8.66025404	**F**

```
int Determinant (int[] A, int n) {
  // tags for 4 types of triangular determinants
  boolean f1=f2=f3=f4=true;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      if(f1 && (j>i) && A[i*n+j]!=0) f1=false;
      if(f2 && (i>j) && A[i*n+j]!=0) f2=false;
      if(f3&&(i+j<n-1)&&A[i*n+j]!=0)f3=false;
      if(f4&&(i+j>n-1)&&A[i*n+j]!=0)f4=false;
    }
  }
  //zeros at one side of principal diagonal
  if (f1 || f2){
    int result = 1;
    for (int i = 0; i < n; i++) result*=A[i*n+i];
    return result;
  }
  //zeros at one side of minor diagonal
  else if (f3 || f4) {
    int result = (int) pow(-1.0, n*(n-1)/2);
    for (int i=0;i<n;i++) result*=A[i*n+(n-1-i)];
    return result;
  }
  else
    return DeterComp(A, n);
}

//Computation by determinantal expansion
//Elements of each row are stored in A in turn
//n is the rank
int DeterComp (int[] A, int n) {
  int mid = 0;      // For middle results
  int result;       // For the final result
  int[] temp;
  //Computing with the elements in temp
  for (int i=0;i<n*n;i++)  temp[i] = A[i];
  //If rank is 1, return result directly
  if (n == 1)  result = A[0];
  //Else, calculating in recursive style
  else {
    //Expanding with the first row
    for (int i = 0; i < n; i++) {
      mid += (int) pow(-1.0, 2 + i) * A[i] *
      DeterComp(AlgComp(temp, n, i), n-1);
    }
    result = mid;
  }
  return result;
}
```

Fig.5. Code of Determinant

Determinant.

In practical applications, triangular determinants as

$$|M| = \begin{vmatrix} 0 & \cdots & 0 & a_{1n} \\ \vdots & \iddots & \cdots & a_{2n} \\ 0 & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{vmatrix} \text{ and } |N| = \begin{vmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & \ddots & \vdots & 0 \\ \vdots & \cdots & \iddots & \vdots \\ b_{n1} & 0 & \cdots & 0 \end{vmatrix}$$

are often required. In these determinants, the elements at top-left or bottom-right are all zero, while none of the ones on minor diagonal is. We only take M-type matrix for discussion here. When this kind of test cases is used, sentences in the broken rectangle of Fig.5 will be executed.

As we know, some elements of M-type matrix must be zero, while others must not be. This condition is strict. But when thinking about mr_6, we only need a matrix whose first non-zero elements of all rows locate in diverse positions, and the test target (M-type matrix) will emerge after several row swaps. Terminal condition for searching with IET is:

$$(\bigvee_{i=0}^{n-1}(A[i\times n]\neq 0)) \bigwedge$$
$$(\bigvee_{i=0}^{n-1}((A[i\times n]=0)\bigwedge(A[i\times n+1]\neq 0))) \bigwedge \cdots \bigwedge$$
$$(\vee_{i=0}^{n-1}(\wedge_{j=0}^{n-2}(A[i\times n+j]=0)\wedge(A[i\times n+j+1]\neq 0)))$$

following formula is used as the measure for distance between test case and objective:

$$\begin{aligned} &\min(\textstyle\bigcup_{i_0=0}^{n-1}\{1\text{-sign}(|A[i_0\times n]|)\}) + \\ &\min(\textstyle\bigcup_{i_1=0}^{n-1}\{|A[i_1\times n]|+1\text{-sign}(|A[i_1\times n+1]|)\}) + \\ &\min(\textstyle\bigcup_{i_2=0}^{n-1}\{\sum_{j=0}^{1}|A[i_2\times n+j]|+1\text{-sign}(|A[i_2\times n+2]|)\}) + \ldots + \\ &\min(\textstyle\bigcup_{i_{n-1}=0}^{n-1}\{\sum_{j=0}^{n-2}|A[i_{n-1}\times n+j]|+1\text{-sign}(|A[i_{n-1}\times n+(n-1)]|)\}) \end{aligned} \quad (5)$$

Function sign(x)'s return number (1, -1 or 0) is depended upon the property of x (positive, negative or zero). In the evolutionary process, each element of test case (array A) is limited to the range of [0, 30].

We also import a mutant, replacing the gray code in Fig.5 with "mid += A[i]*Determinant (AlgComp(temp, n, i), n-1)", into Determinant for evaluation.

Test cases were prescribed to be 3 or 4-order matrices in this experiment and the largest iteration numbers for these two kinds of matrices were 100 and 200 respectively. We also executed IET and PET 20 times for each type of target (3 or 4-order). The symbols AVG^{i}_{ETkd} and Per^{i}_{ETkd} are defined to indicate the average iteration number and hit percent for 20 searches of **i**-order (3 or 4) objective matrix with the

Tab.2. Five group of test cases for Determinant

n	A (Arrays searched out)	FTC A' (Target)	Row Swap	| A |	| A'|	Satisfy mr_6?
4	$\begin{bmatrix}14&6&23&2\\0&0&6&15\\0&5&26&12\\0&0&0&18\end{bmatrix}$	$\begin{bmatrix}0&0&0&18\\0&0&6&15\\0&5&26&12\\14&6&23&2\end{bmatrix}$	**1 (1 ↔ 4)**	**7560**	**7560**	**F**
	$\begin{bmatrix}19&7&26&10\\0&0&0&28\\0&23&27&4\\0&0&15&18\end{bmatrix}$	$\begin{bmatrix}0&0&0&28\\0&0&15&18\\0&23&27&4\\19&7&26&10\end{bmatrix}$	**2** $\begin{pmatrix}1\leftrightarrow 2\\2\leftrightarrow 4\end{pmatrix}$	**183540**	**183540**	**Y**
	$\begin{bmatrix}0&20&11&21\\3&20&22&5\\0&0&19&26\\0&0&0&7\end{bmatrix}$	$\begin{bmatrix}0&0&0&7\\0&0&19&26\\0&20&11&21\\3&20&22&5\end{bmatrix}$	**3** $\begin{pmatrix}1\leftrightarrow 4\\2\leftrightarrow 3\\3\leftrightarrow 4\end{pmatrix}$	**7980**	**7980**	**F**
3	$\begin{bmatrix}0&2&13\\0&0&12\\29&3&9\end{bmatrix}$	$\begin{bmatrix}0&0&12\\0&2&13\\29&3&9\end{bmatrix}$	**1 (1 ↔ 2)**	**696**	**696**	**F**
	$\begin{bmatrix}0&29&14\\3&4&28\\0&0&30\end{bmatrix}$	$\begin{bmatrix}0&0&30\\0&29&28\\3&4&14\end{bmatrix}$	**2** $\begin{pmatrix}1\leftrightarrow 3\\2\leftrightarrow 3\end{pmatrix}$	**2610**	**2610**	**Y**

ETkd (IET or PET) method. The results are listed as follow:
AVG^3_{IET} =18.50, AVG^3_{PET} =32.68, AVG^4_{IET} =126.84, AVG^4_{PET} =153.06;
Per^3_{IET} = 100%, Per^3_{PET} = 95%, Per^4_{IET} = 95%, Per^4_{PET} = 85%.
It is obvious that IET's performance is better. With the computer described in 3.2.1, the execution time of IET's 20 searches for 4-order targets is 1.297 seconds, while that of PET's is 1.115 seconds. The difference is tiny.

Tab.2 shows five test cases that were generated by IET. The fourth column demonstrates the process of row swap for each test case. Data in this table indicate that the mutant can be easily detected. But the test cases generated by PET could not find it. Once again, IET's is proved to be better in fault detection.

IV. CONCLUSION

Following conclusions can be drawn from this article:

- IET is more effective in test case generation than PET. IET's average iteration numbers are much smaller than PET's (29.6<39.6, 18.50<32.68, 126.84< 153.06), and its hit percents are higher than PET's (100%>90%, 100%>95%, 95%>85%).
- IET is better than PET in fault detection. Because MR is considered for the construction of fitness function, the search region is enlarged, and then the test area, such as the number of sentences that are executed in testing process, might also be expanded. This is the main reason why Case1's and Case3's FTCs could reveal Mutant3, while Case2's couldn't.

REFERENCES

[1] Chen, T.Y., Cheung, S.C., and Yiu, S.M., "Metamorphic testing: a new approach for generating next test cases", Technical Report HKUST-CS98-01, 1998, Hong Kong.

[2] Chen, T.Y., Kuo, F.C., Liu, Y., and Tang, A., "Metamorphic testing and testing with special values", in Proceeding of the 5th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'04), Beijing, China, 2004.

[3] Wu, P., Shi, X.C., Tang, J.J., Lin, H.M., and Chen, T.Y., "Metamorphic Testing and Special Case Testing: A Case Study", Journal of Software, 2005, 16(7), pp: 1210-1220.

[4] Wu, P., "Iterative Metamorphic Testing", in Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05), Edinburgh, UK, 2005.

[5] Dong, G.W., Nie, C.H., Xu, B.W., and Wang., L.L., "An Effective Iterative Metamorphic Testing Algorithm Based on Program Path Analysis", in Proceedings of the 7th Annual International Conference on Quality Software (QSIC'07), Oregon, USA, 2007.

[6] Chen, T.Y., Huang, D.H., and Tse, T.H., "Case Studies on the Selection of Useful Relations in Metamorphic Testing", in Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC'04), Madrid, Spain, 2004.

[7] Mayer, J., and Guderlei, R., "An Empirical Study on the Selection of Good Metamorphic Relations", in Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06), Chicago, USA, 2006.

[8] Chen, T.Y., Tse, T.H., and Zhou, Z.Q., "Semi-proving: an integrated method based on global symbolic evaluation and metamorphic testing", ACM SIGSOFT Software Engineering Notes, 2002, 27(4), pp:191-195.

[9] Chen, T.Y., Tse, T.H., and Zhou, Z.Q., "Fault-based testing without the need of oracles", Information and Software Technology, 2003, 45(1), pp:1-9.

[10] Chen, T.Y., Feng, J., and Tse. T.H., "Metamorphic testing of programs on partial differential equations: a case study", in Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02), Oxford, England, 2002.

[11] Zhou, Z.Q., Huang, D.H., Tse, T.H., Yang, Z.Y., Huang, H.T., and Chen, T.Y., "Metamorphic Testing and Its Applications", in Proceedings of the 8th International Symposium on Future Software Technology (ISFST'04), Xi'an, China, 2004..

[12] Chan, W.K., Ho, J.C.F., and Tse., T.H. "Piping Classification to Metamorphic Testing: An Empirical Study towards Better Effectiveness for the Identification of Failures in Mesh Simplification Programs", in Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007), Beijing, China, 2007.

[13] Tse, T.H., Yau, S.S., Chan, W.K., Heng, L., and Chen, T.Y., "Testing context-sensitive middleware-based software applications", in Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04), Hong Kong, 2004.

[14] Chan, W.K., Chen, T.Y., Heng, L., Tse, T.H., and Yau, S.S., "A metamorphic approach to integration testing of context-sensitive middleware-based applications", in Proceedings of the 5th Annual International Conference on Quality Software (QSIC'05), Melbourne, Australia, 2005.

[15]Chan, W.K., Cheung, S.C., and Leung, K.P.H., "Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications", in Proceedings of the 1st International Conference on Services Engineering (SEIW'05), Melbourne, Australia, 2005.

[16] Chan, W.K., Cheung, S.C., and Leung. K.P.H., "A metamorphic testing approach for on-line testing of service-oriented software applications", in A Special Issue on Services Engineering of International Journal of Web Services Research, 2007, pp:60-80.

[17] Chen, H.Y., Tse, T.H., Chan, F.T., and Chen, T.Y., "In black and white: An integrated approach to class-level testing of object-oriented programs", ACM Transactions on Software Engineering and Methodology, 1998, 7(3), pp: 250-295.

[18] Dong, G.W., Nie, C.H., and Xu, B.W., "Effective Metamorphic Testing Based on Program Path Analysis", Accepted by Chinese Journal of Computers.

[19] Srinivas, M., and Patnaik, L., "Genetic algorithms: A survey", IEEE Computer, 1994, 27(6), pp:17-26.

[20] Back, T., "Evolutionary Algorithms in Theory and Practice", Oxford University Press, 1996.

[21] Sthamer, H., "The Automatic Generation of Software Test Data Using Genetic Algorithms", PhD Thesis, University of Glamorgan, Pontyprid, Great Britain, April 1996.

[22] Wegener, J., Sthamer, H., Jones, B., and Eyres, D., "Testing Real-time Systems using Genetic Algorithms", Software Quality Journal, Chapman Hall, 1997, 6(2), pp:127-135.

[23] Tracey, N., Clark, J., McDermid, J., and Mander, K. "Integrating Safety Analysis with Automatic Test-Data Generation for Software Safety Verification", in Proceedings of the 17th International System Safety Conference, Orlando, USA, 1999.

[24] Wappler, S., and Wegener, J., "Evolutionary Unit Testing of Object-oriented Software Using Strongly-Typed Genetic Programming", Conference on Genetic and Evolutionary Computation, Seattle, Washington, USA, 2006.

[25] Xie, X.Y., Xu, B.W., Nie, C.H., and Shi, L., "Configu-ration Strategies for Evolutionary Testing", in Proceedings of the 29th Annual International Computer Software and Applications Conference, Edinburgh, Scotland, 2005.

[26] Xie, X.Y., Xu, B.W., Shi, L., Nie, C.H., and He, Y.X., "A Dynamic Optimization Strategy for Evolutionary Testing" in Proceedings of the 12th ASIA-PACIFIC Software Engineering Conference, Taipei, Taiwan, 2005.

[27] Wegener, J., Baresel, A., and Sthamer, H. "Evolutionary Test Environment for Automatic Structural Testing." Information and Software Technology Special Issue on Software Eng. Using Metaheuristic Innovative Algorithms, 2001, 43(14): pp. 841-854.