关于蜕变测试和特殊用例测试的实例研究*

吴 鹏[1,2+], 施小纯[1,2], 唐江峻[1,2], 林惠民[1], 陈宗岳[3]

[1](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

[2](中国科学院 研究生院,北京 100049)

[3](School of Information Technology, Swinburne University of Technology, Victoria 3122, Australia)

Metamorphic Testing and Special Case Testing: A Case Study

WU Peng[1,2+], SHI Xiao-Chun[1,2], TANG Jiang-Jun[1,2], LIN Hui-Min[1], CHEN T.Y.[3]

[1](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100049, China)

[3](School of Information Technology, Swinburne University of Technology, Victoria 3122, Australia)

+ Corresponding author: Phn: +86-10-62562796, Fax: +86-10-62563894, E-mail: wp@ios.ac.cn, http://lcs.ios.ac.cn/~wp

Received 2004-05-24; Accepted 2004-10-09

Wu P, Shi XC, Tang JJ, Lin HM, Chen TY. Metamorphic testing and special case testing: A case study. *Journal of Software*, 2005,16(7):1210−1220. DOI: 10.1360/jos161210

Abstract: This paper presents an integrated metamorphic testing environment MTest and reports an experimental analysis of the effectiveness of metamorphic testing, which is carried out using MTest with a real program of sparse matrix multiplication. Quantitative evaluation and comparison of special case testing, metamorphic testing with special and random test cases are illustrated with two measurements: mutation score and fault detection ratio. The case study shows that metamorphic testing and special case testing are complementary to each other, and with respect to source test case for metamorphic testing, random test cases are preferred to special test cases.

Key words: software testing; metamorphic testing; special case testing; random testing; mutation analysis

摘 要: 提出了基于蜕变测试方法的集成测试环境 MTest,进而为检验蜕变测试方法的能力和效率,以稀疏矩阵乘法程序为例设计了一组实验.该实验基于变异分析技术,分别以 mutation score 和错误发现率为度量指标,定量地分析和对比了特殊用例测试,以特殊测试用例和随机测试用例为源测试用例的蜕变测试这 3 种方法的测试能力和效率.该实验可在MTest测试环境下自动完成.实验结果表明,蜕变测试与特殊用例测试之间是互补的,而且就蜕变测试

* Supported by the National Natural Science Foundation of China under Grant No.60223005 (国家自然科学基金); the "Knowledge Innovation Initiative" of the Chinese Academy of Sciences (中国科学院知识创新工程)

WU Peng was born in 1977. He is a Ph.D. candidate at the Institute of Software, the Chinese Academy of Sciences. His current research interests include analysis and testing of concurrent systems, software testing and formal methods. **SHI Xiao-Chun** was born in 1978. He is a Ph.D. candidate at the Uppsala University, Sweden. His current research interests include model checking and formal methods. **TANG Jiang-Jun** was born in 1976. He received his Master degree from the Institute of Software, the Chinese Academy of Sciences. His current research interests include web application and software testing. **LIN Hui-Min** was born in 1947. He is a research professor and a doctoral supervisor at the Institute of Software, the Chinese Academy of Sciences. His research interests include concurrency, tools and algorithms for concurrent systems, formal methods etc. **CHEN T.Y.** is a professor at the Swinburne University of Technology. His research interests include software testing, debugging, software maintenance and software design.

的源测试用例而言,随机测试用例在测试能力和效率上优于特殊测试用例.

关键词: 软件测试; 蜕变测试; 特殊用例测试; 随机测试; 变异分析

中图法分类号: TP311 文献标识码: A

1 Introduction

Testing is a practical technique to validate the correctness of software. A successful test is the one that can detect faults in the software under test (SUT). A test that SUT has passed does not contribute to fault detection. Recently, *metamorphic testing* was proposed to exploit information from such tests by examining multiple executions of SUT against some metamorphic relations[1]. A remarkable advantage of this method over other testing methods is that it does not require oracle, which is one of intractable limitations on software testing[2].

Suppose SUT computes an *n*-nary function $f: D_1 \times ... \times D_n \to D_{out}$, which is also referred to as the specification that SUT must conform to. Let $D = D_1 \times ... \times D_n$ denote its input domain. A metamorphic relation of the function *f* is defined as a pair *MR*=(*R*,R_f) such that $R \subseteq \bigcup_{k\geq 2} D^k$, $R_f \subseteq \bigcup_{k\geq 2} (D \times D_{out})^k$, and for any $\bar{x}_1, \ldots, \bar{x}_k \in D$,

$$(\bar{x}_1, \ldots, \bar{x}_k) \in R \quad \text{implies} \quad ((\bar{x}_1, f(\bar{x}_1)), \ldots, (\bar{x}_k, f(\bar{x}_k))) \in R_f \tag{1}$$

A metamorphic relation *MR* is a necessary condition for the correctness of SUT. Let *P* be an implementation of the function *f*, and $P(\bar{x}_1), \ldots, P(\bar{x}_k)$ the outputs resulted from *k* inputs $\bar{x}_1, \ldots, \bar{x}_k \in D$ respectively. If *P* is correct, that is, conforming to its specification *f*, the following proposition should hold.

$$(\bar{x}_1, \ldots \bar{x}_k) \in R \quad \text{implies} \quad ((\bar{x}_1, P(\bar{x}_1)), \ldots, (\bar{x}_k, P(\bar{x}_k))) \in R_f \tag{2}$$

Hence, if $k-1$ tests have been performed without detecting any fault, the k^{th} test can be derived in accordance with *R*. By examining these k ($k \geq 2$) executions against *MR*, metamorphic testing may detect the invalidity of SUT[3,4], and identify its immunity from pre-specified faults in the absence of test oracles[2].

In this way, better performance in fault detection may be achieved by integrating metamorphic testing with other testing methods, in particular test case generation techniques. However, further research should be carried out to validate the effectiveness of this method. We investigate the following problems in this paper:

1. How well can metamorphic testing be integrated with two common test case generation techniques, namely special case and random methods?
2. What is the relationship between metamorphic testing and special case testing in terms of effectiveness?

Special case testing has been widely used in practice. A special test case is the one for which the oracle is already known or can be obtained almost requiring no computation effort. By and large, there are two ways to derive special test cases. One is directly from the special input values of the software with their expected outputs. The other is from the atomic properties of the software, by which test oracles can be determined for single executions of SUT with certain kinds of inputs. The difference lies in that not all inputs of an atomic property are special values.

Note that although metamorphic testing can be categorized as a testing method based on properties of SUT, it is definitely different from atomic-property-based testing in that it requires at least two sets of inputs (correlated by *R*), which means SUT should be executed more than once for each test.

In this paper, we present an experimental analysis to compare metamorphic testing and special case testing. JASPA[5], a mathematical program for sparse matrix multiplication, is taken as an example to evaluate and compare the effectiveness of three testing methods, namely special case testing, metamorphic testing with special test cases and metamorphic testing with random test cases[6], using mutation analysis technique[7,8]. Two measurements are

used to quantify their test effectiveness in different granularities: mutation score and fault detection ratio. We have developed an integrated environment MTest to perform all test campaigns automatically.

Reference [1] presents a brief survey on metamorphic testing with potential research directions. Reference [2] proposes to enhance fault based testing to address the oracle problem with metamorphic testing. References [3,4] are closely related to our paper in the sense that Ref.[3] illustrates the applicability of metamorphic testing on solving partial differential equations, but with no comparison on the effectiveness of metamorphic testing against other testing methods; Reference [4] demonstrates the use of metamorphic testing to detect faults in programs, which could not be detected by special test values. Our paper gives a more practical and general consideration of special case testing and presents a more systematic evaluation and comparison with statistical data.

The paper is organized as follows. Section 2 illustrates the architecture and workflow of MTest. The reference implementation of sparse matrix multiplication is introduced in Section 3 together with its mutants, atomic properties and metamorphic relations used in the subsequent testing experiments. Section 4 describes two measurements employed in mutation analysis for those testing methods. Experiment results are presented in Section 5 with detailed analysis. The paper is concluded with Section 6 where some future work is also outlined.

2 Metamorphic Testing

In this section we first briefly introduce the workflow of metamorphic testing, then present an integrated metamorphic testing environment MTest that can automate test generation and execution with domain-specific metamorphic relations and special cases (optional).

Based on requirements (1) and (2), metamorphic testing is organized as follows:

1. Test SUT with source test cases $X_1,X_2,\ldots,X_{k-1}$ ($k \geq 2$) and record its results, denoted as $Y_1,Y_2,\ldots,Y_{k-1}$ respectively. If any source test case causes an unexpected exception resulting in the abnormal termination of SUT, e.g. bus error, segmentation fault etc., the test can be terminated with failure reported.
2. For each metamorphic relation $MR=(R, R_f)$,
   a) Derive a new test case X_k from source test cases above, in accordance with relation R;
   b) Test SUT with X_k;
   c) If SUT terminates at X_k abnormally by certain unexpected exception, the test will be terminated with failure reported.
   d) If SUT terminates normally with output Y_k, validate it according to R_f. If $R_f((X_1,Y_1), \ldots,(X_k,Y_k))$ does not hold, a failure will be reported;
3. If no failure is reported, SUT is deemed to pass all test cases.

Various test generation techniques can be adopted to derive source test cases. In this paper, two techniques will be applied: one is to derive test cases from special values or atomic properties of the software. Such test cases are referred to as special test cases, of which their input-output relationships are already known or easy to obtain. The other is to select test cases randomly from the entire input domain. Such test cases are referred to as random test cases. The common ground of special-value-based and atomic-property-based testing is that input-output relationships for executions of SUT can be obtained almost with no computation effort. Therefore, although metamorphic testing can be generally categorized as a testing method based on properties of the software, it is definitely different from special case testing, especially atomic-property-based testing, in that for each metamorphic relation $MR=(R,R_f)$, it is required that SUT be executed more than once with distinct test inputs correlated by R.

To illustrate the compatibility of metamorphic testing with other testing methods, in particular test case generation techniques, we design and implement an integrated environment MTest to automatically perform metamorphic testing with special and random inputs as source test cases. The architecture of the test environment is

shown in Fig.1. Given a SUT, MTest customizes a metamorphic tester with two configuration files: one supplies special cases and corresponding oracles of SUT, while the other supplies the interface definition and metamorphic relations of SUT.

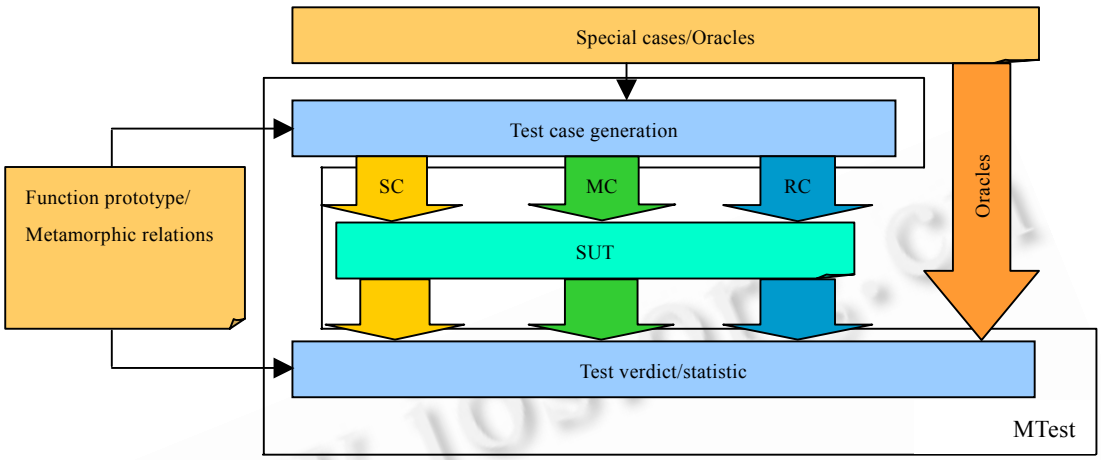

Fig.1 Metamorphic test architecture

The metamorphic tester consists of two parts: Test Case Generation and Test Verdict/Statistic. The former is to derive test cases and call SUT with them thereafter, while the latter is to collect outputs from SUT and then determine whether SUT passes test cases according to pre-defined test oracles (in special case testing) or metamorphic relations (in metamorphic testing). A test case may be a special test case (denoted as SC); or a random test case (denoted as RC); or resulted from transforming one or more special/random test cases in accordance with certain metamorphic relation (denoted as MC).

The tester can carry out as many rounds of test campaigns as configured by the user. In each round, SUT is tested firstly with source test cases (SC or RC). Their variations (MC) in accordance with each metamorphic relation are then applied to SUT again for more tests. Special test cases, if supplied, are always used as source test cases in the first round, while random test cases are used in all other rounds. Finally the tester reports verdicts of SUT and counts the number of test cases that SUT fails in each round of test.

3 Sparse Matrix Multiplication

A representative matrix operation, sparse matrix multiplication, is considered in our work. We have chosen this case for the following reasons:

1. It has been widely used in scientific and engineering applications;
2. It requires efforts to validate the computation results because oracles cannot be obtained easily, especially for large-order matrices;
3. It features a number of atomic properties that can be used in defining special test cases, besides its special values.

In this section, a reference implementation will be described, along with its mutants, atomic properties and metamorphic relations. In the sequel, the usual matrix notations and conventions will be followed.

3.1 Implementations

A *C* program, which is a part of JASPA, is chosen as a reference implementation for sparse matrix multiplication. JASPA is a benchmark for evaluating the I/O and compute performance of sparse matrix

multiplication kernels in large-scale scientific and engineering applications. The main segment of its *C* source code, *SpMatMul*, is shown in Fig.2 with minor layout modifications.

In JASPA, a sparse matrix *M* is stored in a compressed row format, i.e. *M*=(*v*, *iv*, *jv*), where *v* is a vector for all non-zero float-point numbers, *iv* and *jv* for integers. Vector *v* stores all non-zero elements of the matrix as they are traversed in a row-wise fashion; vector *iv* stores the locations of those elements in *v*, which appear as the first non-zero element in a row of the matrix; vector *jv* stores the column indices of elements in *v*.

```
          void SpMatMul (int n, int m,
               const double *a, const int *ia, const int *ja,
               const double *b, const int *ib, const int *jb,
               double *c, int *ic, int *jc)
          {
            int nz = 0, *mask, i, j, k, icol, icol_add;
            const double *aij = a;
            const int *neighbour = ja;
1.
2.          mask = (int *)malloc(m * sizeof(int));
3.          for (i = 0; i < m; i++) mask[i] = -1;
4.          ic[0] = 0;
5.          for (i = 0;i < n;i++) {
6.            for (j = ia[i];j < ia[i + 1];j++) {
7.              for (k = ib[*neighbour];k < ib[(*neighbour) + 1];k++) {
8.           icol_add = jb[k];
9.           icol = mask[icol_add];
10.          if (icol == -1) {
11.             jc[nz] = icol_add;
12.             c[nz] = (*aij) * b[k];
13.             mask[icol_add] = nz;
14.             nz ++;
15.          }
16.          else
17.             c[icol] += (*aij) * b[k];
18.           }
19.           aij ++;
20.           neighbour++;
21.         }
22.         for (j = ic[i]; j < nz; j++) mask[jc[j]] = -1;
23.         ic[i + 1] = nz;
24.       }
25.       free(mask);
26.     }
```

Fig.2 *SpMatMul* - C source code for spare matrix multiplication from JASPA

The function *SpMatMul* computes the product of two sparse matrices, *A*=(*a*,*ia*,*ja*) and *B*=(*b*,*ib*,*jb*) with the result stored in matrix *C*=(*c*,*ic*, *jc*). Parameters *n* and *m* denote the row and column dimension of the product matrix, respectively. Note that *SpMatMul* assumes that matrices *A* and *B* are multipliable and the memory usages for *c*, *ic* and *jc* have been allocated before the function is called.

Mutation analysis is a powerful technique to assess the quality of a test set or determining its sufficiency. In our research, we apply this technique to assess the effectiveness of a testing method by evaluating the quality of test sets it can derive. Five mutants are designed for the function *SpMatMul* based on two types of mutation operators: SDL (Statement Deletion) and AOR (Arithmetic Operator Replacement), which are defined in a mutation-testing environment Mothra[9,10].

- **Mutant 1** is resulted by deleting line **19** so that only the first non-zero element of matrix *A* is retrieved for computation.
- **Mutant 2** is resulted by replacing line **12** with `c[nz] = (*aij);`
- **Mutant 3** is resulted by replacing line **12** with `c[nz] = b[k];`
- **Mutant 4** is resulted by replacing line **17** with `c[icol] += (*aij);`

- **Mutant 5** is resulted by replacing line **17** with `c[icol] += b[k];`

3.2 Atomic properties and metamorphic relations

For sparse matrix multiplication, the following atomic properties, shown in Table 1, can be applied to derive special test cases (denoted as SC in the sequel) for which oracles are also well known (for SC_1 to SC_4) or can be obtained easily (for SC_5 to SC_8). In Table 1, elements without explicit assignments will be dynamically assigned a random number for test case generation.

Table 1 Atomic properties for sparse matrix multiplication

SC_i	$A=[a_{ij}]_{n\times r}$	$B=[b_{ij}]_{r\times m}$	$C=AB=[c_{ij}]_{n\times m}$
SC_1	$A=0$		0
SC_2	$B=0$		0
SC_3	$A=I$		B
SC_4	$B=I$		A
SC_5	$n=1, a_{ij}=\begin{cases}1 & i=j=1\\ 0 & otherwise\end{cases}$		$c_{ij}=b_{nj}$
SC_6	$m=1, b_{ij}=\begin{cases}1 & i=j=1\\ 0 & otherwise\end{cases}$		$c_{ij}=a_{im}$
SC_7	$r=1, a_{ij}=1$		$c_{ij}=b_{rj}$
SC_8	$r=1, b_{ij}=1$		$c_{ij}=a_{ir}$

Nine metamorphic relations, shown in Table 2, will be applied in the subsequent case study. MR_2 and MR_3 are symmetric in that they apply the same transmutation method to different inputs, i.e. the first input A is transmuted in MR_2, while the second one, B, is in MR_3. Similarly, MR_4 and MR_5, MR_6 and MR_7, as well as MR_8 and MR_9 are also symmetric. In the sequel, it will be seen that symmetric metamorphic relations may exhibit quite different test effects.

Table 2 Metamorphic relations

MR_i	R	R_f	Notes
MR_1	$A'=B^T, B'=A^T$	$A'B'=(AB)^T$	
MR_2	$A'=PA, B'=B$	$A'B'=P(AB)$	P is resulted by exchanging two rows of the identity matrix I.
MR_3	$A'=A, B'=BP$	$A'B'=(AB)P$	
MR_4	$A'=QA, B'=B$	$A'B'=Q(AB)$	Q is resulted by multiplying a principal diagonal element of I with a scalar c.
MR_5	$A'=A, B'=BQ$	$A'B'=(AB)Q$	
MR_6	$A'=cA, B'=B$	$A'B'=c(AB)$	c is a scalar.
MR_7	$A'=A, B'=cB$		
MR_8	$A'=A+I, B'=B$	$A'B'=AB+IB$	
MR_9	$A'=A, B'=B+I$	$A'B'=AI+AB$	

4 Measurements

The effectiveness of a testing method can be measured quantitatively at two levels in different granularities: one is by counting the number of mutants that could be detected; and the other by calculating how many of its test cases are able to detect a particular mutant. Correspondingly two measurements are employed in our work: mutation score and fault detection ratio.

In mutation analysis, the quality of a test set is measured by mutation score, which is the percentage of mutants detected, that is

$$MS(T)=\frac{M_k}{M_t-M_q} \tag{3}$$

where T is the test set; M_k the number of mutants detected by T, M_t the total number of mutants, and M_q the number of equivalent mutants that cannot be detected by any set of test data.

Mutation score is also appropriate for assessing the effectiveness of a testing method. Regarding test sets resulted from different testing methods, higher mutation score implies higher quality of the corresponding test set. Therefore, mutation score can serve as a macroscopical measurement in the sense that it does not matter which mutant is detected or not. Definition (3) can be reused for this task except that T stands for a testing method in the sense that multiple test sets derived by the testing method are evaluated and the results are averaged as the measurement for the testing method.

On the other hand, a microscopical measurement is desirable to analyze to what extent a testing method can detect a fault in SUT, in the sense that each mutant is considered separately. A simple way is to calculate the fault detection ratio FD, the percentage of test cases that could detect certain mutant m, that is

$$FD(m,T)=\frac{N_f}{N_t-N_e} \tag{4}$$

where N_f is the number of times SUT fails, N_t the number of tests, and N_e the number of infeasible tests.

For special case testing, N_t is just the number of special cases N_c and N_e=0. For metamorphic testing, N_t is the product of the number of source test cases N_c and the number of metamorphic relations N_m, since each source test case should be executed once for each metamorphic relation; while N_e is the total number of times that source test cases could not satisfy the requirement of a metamorphic relation.

In the case study, mutation score and fault detection ratio are taken to evaluate and compare three testing methods: special case testing, denoted as T_s, metamorphic testing with special test cases as source test cases, denoted as T_{ms}, and metamorphic testing with random test cases as source test cases, denoted as T_{mr}. Note that M_t = 5 and M_q=0 because none of the five mutants are functionally equivalent to *SpMatMul*.

Furthermore, both measurements are also appropriate for assessing the test effect of a single metamorphic relation. In the sequel, T_{ms}^r and T_{mr}^r denote metamorphic testing using metamorphic relation MR_i with special test cases and random test cases, respectively.

5 Experimental Results

This section will illustrate our experimental results on metamorphic testing for sparse matrix multiplication programs, which has been performed automatically with MTest.

5.1 Metamorphic testing with special test cases

The special test set consists of 8 test cases, derived from atomic properties mentioned in Section 3.

Table 3 reports test verdicts of all mutants by special test case testing, in Column 2, and by metamorphic testing with each metamorphic relation MR_i, in Column 3 to 11, where "✓" represents that SUT passes the corresponding special test case by the corresponding test method, "×" represents that SUT fails and "-" represents that the corresponding special test case is not applicable for the corresponding metamorphic relation.

Table 3 Test verdicts report by special case testing and metamorphic testing with special test cases

(a) Mutant 1

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
SC_1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_2	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
SC_3	✓	×	✓	✓	×	✓	✓	✓	✓	✓
SC_4	×	×	×	✓	×	✓	✓	✓	×	×
SC_5	✓	×	-	✓	✓	✓	✓	✓	✓	✓
SC_6	×	×	×	-	×	✓	✓	✓	×	×
SC_7	✓	×	✓	✓	×	✓	✓	✓	×	✓
SC_8	×	×	×	✓	×	✓	✓	✓	×	×
N_f	3	6	3	0	5	0	0	0	4	4

(b) Mutant 2

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
SC_1	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
SC_2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_3	×	×	✓	✓	✓	×	✓	×	×	×
SC_4	✓	×	✓	✓	✓	×	✓	×	✓	×
SC_5	×	×	-	✓	✓	×	✓	×	×	×
SC_6	✓	×	✓	-	✓	×	✓	×	✓	×
SC_7	×	×	✓	✓	✓	×	✓	×	×	×
SC_8	✓	×	✓	✓	✓	×	✓	×	✓	×
N_f	3	6	0	0	0	6	0	6	4	6

(c) Mutant 3

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
SC_1	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
SC_2	✓	✓	✓	✓	✓	✓	✓	✓	✓	×
SC_3	✓	×	✓	✓	×	✓	×	✓	×	✓
SC_4	×	×	✓	✓	×	✓	×	✓	×	×
SC_5	✓	×	-	✓	×	✓	×	✓	×	✓
SC_6	×	×	✓	-	×	✓	×	✓	×	×
SC_7	✓	×	✓	✓	×	✓	×	✓	×	✓
SC_8	×	×	✓	✓	×	✓	×	✓	×	×
N_f	3	6	0	0	6	0	6	0	6	4

(d) Mutant 4

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
SC_1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_5	✓	✓	-	✓	✓	✓	✓	✓	✓	✓
SC_6	✓	✓	✓	-	✓	✓	✓	✓	✓	✓
SC_7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
N_f	0	0	0	0	0	0	0	0	0	0

(e) Mutant 5

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
SC_1	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_2	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_3	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_4	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_5	✓	✓	-	✓	✓	✓	✓	✓	✓	✓
SC_6	✓	✓	✓	-	✓	✓	✓	✓	✓	✓
SC_7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
SC_8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
N_f	0	0	0	0	0	0	0	0	0	0

Table 4 reports the mutation score and fault detection ratio of special case testing in Column 2, and metamorphic testing with each metamorphic relation MR_i, in Columns 3 to 11.

Table 4 Mutation score and fault detection ratio of special case testing and metamorphic testing with special test cases ($N_c = 8$, $N_m = 9$)

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
Mutant 1	37.5%	75%	37.5%	0	62.5%	0	0	0	50%	50%
Mutant 2	37.5%	75%	0	0	0	75%	0	75%	50%	75%
Mutant 3	37.5%	75%	0	0	75%	0	75%	0	75%	50%
Mutant 4	0	0	0	0	0	0	0	0	0	0
Mutant 5	0	0	0	0	0	0	0	0	0	0
M_k	3	3	1	0	2	1	1	1	3	3
$MS(T)$	0.6	0.6	0.2	0	0.4	0.2	0.2	0.2	0.6	0.6

Based on the above data, the following conclusions can be drawn:

1. Metamorphic testing with a single metamorphic relation, as well as with special test cases, may not outperform special case testing. For sparse matrix multiplication, MR_1, MR_8 and MR_9 seem better than other metamorphic relations, among which MR_3 is the worst one. Therefore, metamorphic relation selection is as crucial to metamorphic testing as test case selection is. A simple way is to use all metamorphic relations together in a parallel way. Fault detection capability for such strategy is illustrated in Table 5, from which it can be seen that the fault detection capability of metamorphic testing with specail test cases is comparable to that of specail case testing.

Table 5 Fault detection ratio of metamorphic testing vs. special case testing

FD	Mutant 1 (%)	Mutant 2 (%)	Mutant 3 (%)	Mutant 4 (%)	Mutant 5 (%)	Average (%)
T_s	37.5	37.5	37.5	0	0	22.5
T_{ms}	31.43	40	40	0	0	22.3

2. A comparison of column 1 and other columns in Table 3 shows that applying metamorphic relations with special test cases could reveal faults that are not detected by special test cases alone. Hence, metamorphic testing with special test cases well supplements the fault detection capabilities of special test cases.

3. The symmetry among metamorphic relations does not imply equivalent test effectiveness, as evidenced by the observation that MR_2 and MR_3, MR_4 and MR_5 are symmetric, but $MS(T_{ms}^2) > MS(T_{ms}^3)$, $MS(T_{ms}^4) > MS(T_{ms}^5)$.

5.2 Metamorphic testing with random test cases

As random testing is actually a statistical method to assess the reliability of SUT, metamorphic testing can also be carried out in a statistical way. Obviously it is desirable to perform metamorphic testing with random test cases many times to make evaluation and comparison more reliable. However, more tests require more effort and cost. Therefore, we would also like to investigate the relationship between the test effectiveness of such method and the number of random test cases, as well as the minimum order of a random matrix (denoted as $D_{\min}$ in the sequel), to find how such task can be performed in a more cost-effective way. In each round, the number of random test cases or the minimum order of a random matrix is increased by a certain constant amount.

Table 6 reports the mutation score and (average) fault detection ratio of metamorphic testing with 8 random test cases, where $D_{\min}$ is 2. This order is also the minimum requirement for using all metamorphic relations. Figure 3 illustrates fault detection ratios, $FD(T_{mr})$, for Mutant 1 with an increasing number of test cases, as well as an increasing minimum order.

Table 6 Mutation score and fault detection ratio of metamorphic testing with random test cases (N_c = 8, N_c = 9)

	T_s	T_{ms}^1	T_{ms}^2	T_{ms}^3	T_{ms}^4	T_{ms}^5	T_{ms}^6	T_{ms}^7	T_{ms}^8	T_{ms}^9
Mutant 1	98.44%	87.89%	0	92.19%	0	0	0	98.83%	97.66%	52.78%
Mutant 2	100%	0	0	0	92.18%	0	100%	99.61%	95.70%	54.17%
Mutant 3	99.61%	0	0	91.41%	0	99.61%	0	95.31%	100%	54.00%
Mutant 4	89.45%	0	0	0	64.84%	0	89.45%	85.55%	79.30%	45.40%
Mutant 5	95.31%	0	0	72.66%	0	95.31%	0	83.60%	89.45%	48.48%
M_k	5	1	0	3	2	2	2	5	5	5
$MS(T)$	1	0.2	0	0.6	0.4	0.4	0.4	1	1	1

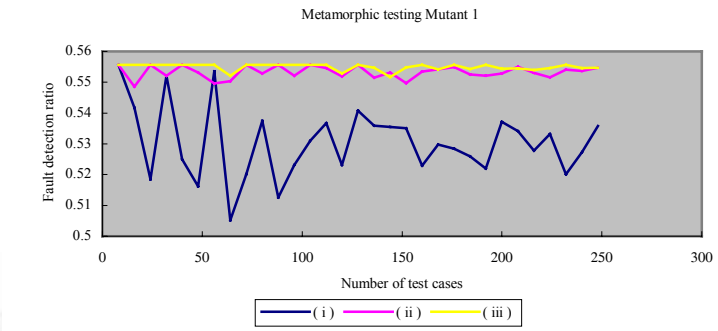

Fig.3 $FD(T_{mr})$ with increasing number of random test cases and minimum order of a random matrix: (i) $D_{\min}$=2, (ii) $D_{\min}$=6, (iii) $D_{\min}$=18

The following conlcusions can be drawn:

1. With MR_3, none of the five mutants could be detected as faulty by metamorphic testing, but special case testing does detect three of them, namely Mutants 1, 2 and 3. On the other hand, special case testing is unable to detect Mutants 4 and 5 using all available special test cases, but all metamorphic relations except MR_2 and MR_3 can detect their faults with random test cases. Hence, the case study clearly shows that metamorphic testing and special case testing are complementary to each other.

2. Metamorphic testing with random test cases always outperforms that with special test cases, as evidenced by the observation that $MS(T_{mr})>MS(T_{ms})$ and $FD(T_{mr})>FD(T_{ms})$ hold for any mutant. Therefore, random test cases are the preferred source test cases for metamorphic testing.

3. Metamorphic relation selection is also identified as crucial to metamorphic testing with random test cases. Furthermore, those metamorphic relations, which exhibit better test effectiveness than others during metamorphic testing with special test cases, also exhibit much better test effectiveness during that with random ones. Regarding sparse matrix multiplication, MR_1, MR_8 and MR_9 detect Mutant 4 and Mutant 5 with higher fault detection ratios than other metamorphic relations do. By inspecting Table 4 and Table 6, we observe that higher fault detection ratio a metamorphic relation exhibits with special test cases, higher possibility it may have to detect other faults in SUT with random test cases.

4. When the number of random test cases increases, the fault detection ratio of metamorphic testing may vibrate in a small amplitude, which is reduced as $D_{\min}$ increases. Nevertheless, more test cases may not result in better performance. In the case study, using more than 80 random test cases and the order of a matrix larger than 18 would not help to increase the fault detection ratio. An instructive way to determine the cutoffs is to perform metamorphic testing incrementally in two aspects, i.e. by increasing the number of random test cases and the minimum order of a random matrix gradually until the curve of fault detection ratio tends stable or going down.

6 Conclusions

This paper investigates the effectiveness of metamorphic testing with a real program of sparse matrix multiplication, which is a part of the spare matrix multiplication benchmark JASPA. The technique of mutation analysis has been used to evaluate and compare three testing methods, namely special case testing, metamorphic testing with special test cases and random test cases. Mutation score and fault detection ratio are applied to assess the effectiveness of a testing method from a macro and a micro point of view, respectively.

The case study has clearly shown that metamorphic testing and special case testing are really complementary. As compared with special test cases, random test cases are the preferred source test cases for metamorphic testing. Furthermore, it is interesting to see that those metamorphic relations exhibit better performance with special test cases, and exhibit much better performance with random test cases.

In the case study, all metamorphic relations are taken alone to derive follow-up test cases. As future work, we would like to extend metamorphic testing by using their sequential combinations, which may result in a general framework for metamorphic testing. On the other hand, we would also like to investigate the way to make metamorphic testing more cost-effective and to integrate it with other testing methods.

Acknowledgement The authors acknowledge and thank Prof. Yidong Shen for the stimulating discussion on statistic data analysis. The authors gratefully acknowledge the anonymous referees for their valuable comments in the course of reviewing process.

References:

[1] Chen TY, Kuo F-C, Tse TH, Zhou ZQ. Metamorphic testing and beyond. In: O'Brien L, Gold N, Kontogiannis K, eds. Proc. of Int'l Workshop on Software Technology and Engineering Practice (STEP2003). Los Alamitos: IEEE Computer Society Press, 2004. 94−100.

[2] Chen TY, Tse TH, Zhou ZQ. Fault-Based testing in the absence of an Oracle. In: Tse TH, ed. Proc. of the 25th Annual Int'l Computer Software and Application Conf.(COMPSAC'01). Piscataway: IEEE Computer Society Press, 2001. 172−180.

[3] Chen TY, Feng JQ, Tse TH. Metamorphic testing of programs on partial differential equations: A case study. In: Yang HJ, ed. Proc. of the 26th Annual Int'l Computer Software and Application Conf. (COMPSAC'02). Piscataway: IEEE Computer Society Press, 2002. 327−333.

[4] Chen TY, Kuo F-C, Liu Y, Tang A. Metamorphic testing and testing with special values. In: Hu GZ, Huang T, Ni XZ, Zhou AY, eds. Proc. of the 5th ACIS Int'l Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2004). Mt. Pleasant, Michigan: Int'l Association for Computer and Information Science, 2004. 128−134.

[5] Hu YF, Allan RJ, Maguire KCF. Comparing the performance of JAVA with Fortran and C for numerical computing. Daresbury Laboratory, CCLRC. http://www.dl.ac.uk/TCSC/UKHEC/JASPA/bench.pdf

[6] Hamlet R. Random testing. In: Marciniak J, ed. Encyclopedia of Software Engineering. New York: Wiley, 1994. 970−978.

[7] DeMillo RA, Lipton RJ, Sayward FG. Hints on test data selection: Help for the practicing programmer. Computer, 1978,11(4): 34−41.

[8] Budd TA. Mutation analysis: Ideas, examples, problems and prospects. In: Chandrasekaran B, Radicchi S, eds. Proc. of the Summer School on Computer Program Testing. Amsterdam: North-Holland, 1981. 129−148.

[9] DeMillo RA, Guindi DS, McCracken WM, Offutt AJ, King KN. An extended overview of the mothra software testing environment. In: Proc. of the 2nd Workshop on Software Testing, Verification, and Analysis. New York: IEEE Computer Society Press, 1988. 142−151.

[10] Untch RH, Offutt AJ, Harrold MJ. Mutation analysis using mutant schemata. Software Engineering Notes, 1993,18(3):139−148.