Search-Based Inference of Polynomial Metamorphic Relations

Jie Zhang, Junjie Chen, Dan Hao, Yingfei Xiong, Bing Xie; Lu Zhang, Hong Mei
Key Laboratory of High Confidence Software Technologies (Peking University), MoE, China
Institute of Software, School of EECS, Peking University, China
{zhangjie12,chenjj14,haod,xiongyf04,xiebing,zhanglu,meih}@sei.pku.edu.cn

ABSTRACT
Metamorphic testing (MT) is an effective methodology for testing those so-called “non-testable” programs (e.g., scientific programs), where it is sometimes very difficult for testers to know whether the outputs are correct. In metamorphic testing, metamorphic relations (MRs) (which specify how particular changes to the input of the program under test would change the output) play an essential role. However, testers may typically have to obtain MRs manually.

In this paper, we propose a search-based approach to automatic inference of polynomial MRs for a program under test. In particular, we use a set of parameters to represent a particular class of MRs, which we refer to as polynomial MRs, and turn the problem of inferring MRs into a problem of searching for suitable values of the parameters. We then dynamically analyze multiple executions of the program, and use particle swarm optimization to solve the search problem. To improve the quality of inferred MRs, we further use MR filtering to remove some inferred MRs.

We also conducted three empirical studies to evaluate our approach using four scientific libraries (including 189 scientific functions). From our empirical results, our approach is able to infer many high-quality MRs in acceptable time (i.e., from 0.87 seconds to 1231.16 seconds), which are effective in detecting faults with no false detection.

Categories and Subject Descriptors
D.2.5 [Software Engineering]: Testing and debugging—Testing tools

General Terms
Algorithm, Experimentation, Verification

Keywords
Metamorphic testing; Invariant inference; Particle swarm optimization

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ASE’14, September 15-19, 2014, Sweden
Copyright 2014 ACM 978-1-4503-3013-8/14/09 ...$15.00.
http://dx.doi.org/10.1145/2642937.2642994

1. INTRODUCTION

In software testing, a test oracle is needed to determine whether the program under test exhibits an acceptable behavior. Typically, it is very costly for testers to obtain a suitable test oracle for the program under test [28]. For some programs (e.g., scientific programs), which Weyuker [66] refers to as “non-testable” programs, obtaining test oracles may be extremely difficult or even impossible. In such a circumstance, testers may be unable to decide whether the program outputs are correct for most given inputs.

To facilitate testing the so-called “non-testable” programs, Chen et al. [11] proposed metamorphic testing (MT), which detects program faults in by looking for violations of metamorphic relations (MRs). Typically, an MR specifies how a particular change to the input would change the output. For example, to test the sin program, in metamorphic testing, the tester can check whether \( \sin(x + \pi) = -\sin(x) \) without knowing the exact value of \( \sin(x) \).

To facilitate this labor-intensive work, Kanewala and Bie- man [33] proposed a machine-learning based technique to predict whether a program contains some forms of MRs. For example, for the sin program, some MRs like \( \sin(x + \pi) = -\sin(x) \) are easy to infer but some MRs like \( \sin^2(\pi/2 - x) + \sin^2(x) = 1 \) are not. As demonstrated by Chen et al. [14], more MRs may help achieve more adequate testing. Such manual acquisition of a number of MRs may become a bottleneck of metamorphic testing.

To facilitate this labor-intensive work, Kanewala and Bie- man [33] proposed a machine-learning based technique to predict whether a program contains some forms of MRs. For example, for the sin program, this technique predicts whether it contains a form of MR like \( \sin(x) - c_1 \sin(x + c_2) = 0 \) where \( c_1 \) and \( c_2 \) are constants, but does not generate the values of constants \( c_1 \) and \( c_2 \). Different from their technique, in this paper, we aim to automatically infer specific MRs instead.

As it is difficult to infer arbitrary relations automatically, we need to consider a particular class of relations. After manually studying 70 MRs reported in the literature [38, 12, 47, 68, 83, 14, 70, 58, 59, 71, 44], we found that 43 (61.4%) MRs are polynomial, i.e., the relations between inputs and the relations between outputs are both polynomial equations. Therefore, in this paper, we focus on inferring only such polynomial MRs.
In particular, we propose a novel search-based approach to inferring polynomial MRs\(^1\) by analyzing multiple executions of the program under test. Our approach aims to infer polynomial metamorphic relations in particular forms, which can be represented as a series of parameters. For those parameterized MRs, our approach uses particle swarm optimization [54] to search for suitable values of the parameters. That is, our approach tries to find a set of parameter values that characterize the analyzed executions in the form of an MR. Furthermore, to improve the quality of MRs, our approach applies statistics based filtering to the inferred MRs. Based on our study on the literature of MRs [38, 47, 68, 83, 12, 14, 70, 58, 59, 71, 44], 23 (53.5\%) polynomial MRs are simple polynomial MRs whose relations between inputs are linear equations and whose relations between outputs are linear or quadratic equations. Therefore, in the current stage our approach focuses on inferring these particular polynomial MRs. However, our approach is general and can be used to infer other polynomial MRs, which is illustrated in Section 7.2 as an extension to our approach.

Based on our approach, we implemented a tool named MRI (i.e., Metamorphic Relation Inferrer) and conducted three empirical studies on MRI to evaluate our approach using 189 scientific functions from Apache, JDK, GSL, and MATLAB. The first study investigates the feasibility of our approach. The second study investigates the quality of MRs inferred by our approach. The third study investigates whether MR filtering in our approach may improve the quality of inferred MRs. Our empirical results demonstrate that our approach is able to infer many high-quality MRs in acceptable time. Furthermore, MR filtering in our approach actually improves the quality of inferred MRs.

In summary, this paper makes the following contributions.

- A novel search-based\(^2\) approach to inferring polynomial MRs via dynamically analyzing multiple executions using particle swarm optimization.
- An evaluation on four scientific libraries demonstrating the feasibility of our approach and the quality of inferred MRs.

2. EXAMPLES

To further motivate our research, we use some example metamorphic relations for three trigonometric functions to demonstrate that even for some widely known functions there may be some unfamiliar metamorphic relations.

To most people, the sin function is one of the most popular trigonometric functions. Let us consider the metamorphic relation "\(\sin^2(\pi/2 - x) + \cos^2(x) = 1\)". At a glance, this metamorphic relation looks quite unfamiliar, but as we know that "cos\((x) + \sin^2(x) = 1\) and cos\((x) = \sin(\pi/2 - x)\)\), we may derive this metamorphic relation by ourselves. For the cos function, "\(\cos(2x) = 2\cos^2(x) - 1\)" is a metamorphic relation. Deriving this metamorphic relation would be much more difficult, because it involves complex mathematical deductions. Furthermore, it may be even more difficult to derive the metamorphic relation "\(\tan^2(x) - 2\tan(x)\tan(2x - 3\pi/2) - 1 = 0\)" for the tan function, as tan is much less used than sin and cos.

As mathematicians have thoroughly investigated all the preceding trigonometric functions, testers may find the preceding metamorphic relations by searching in a textbook or the Internet. However, when the program under test does not exactly match some thoroughly investigated function, it may be painful for testers to derive its metamorphic relations.

Previously, Kanewala and Bieman [33] proposed a machine learning based approach to predicting likely metamorphic relations by some features, which are extracted from the control flow graph of a program. However, this approach can only predict whether the program has a particular metamorphic relation or not. Let us take the sin function as an example. By analyzing the source code implementing the sin function, this machine-learning based approach may predict that this function has an MR whose input change and output change are both additive\(^3\). That is, for the sin function, this machine-learning based approach may infer metamorphic relations like sin\((x) + c_1\sin(x + c_2) = 0\), where \(c_1\) and \(c_2\) are constants and \(x\) is the input parameter of the function sin. However, this approach does not provide the values of constants \(c_1\) and \(c_2\) at all.

3. APPROACH

Before presenting our approach, we present some background information of particle swarm optimization (PSO) in Section 3.1. In our approach, we present definitions of MRs in their parameterized forms in Section 3.2, our PSO-based search algorithm for determining the parameters in Section 3.3, and our statistical based technique for filtering MRs in Section 3.4.

3.1 Background

Particle swarm optimization (PSO) [54], originally proposed by Kennedy and Eberhart [34] in 1995, is a swarm intelligence optimization algorithm simulating the birds foraging behavior. In PSO, each candidate solution is called a particle, and multiple particles coexist and optimize cooperatively to achieve the optimal solution. In particular, each particle has a velocity and a location, which keep changing during the search. There is a fitness function to evaluate how close the location of a particle is to an optimal location.

Consider searching in a \(D\)-dimensional space with \(N\) particles. We use \(v_{1}^{t} = \langle v_{11}^{t}, v_{12}^{t}, ..., v_{1D}^{t} \rangle\) and \(l_{1}^{t} = \langle l_{11}^{t}, l_{12}^{t}, ..., l_{1D}^{t} \rangle\) to denote the velocity and the location of the \(i\)-th (\(1 \leq i \leq N\)) particle at moment \(t\) \((t=1, 2, ...\)) , where \(v_{id}\) and \(l_{id}\) denote the values of \(V_{id}\) and \(L_{id}\) in the \(d\)-th \((1 \leq d \leq D\)) dimension, respectively. Then Formula 1 calculates the velocity of the \(i\)-th particle in the next moment (i.e., moment \(t+1\)).

\[v_{id}^{t+1} = \omega v_{id}^{t} + \xi_1 r_1(p_{id}^{t} - l_{id}^{t}) + \xi_2 r_2(p_{id}^{t} - l_{id}^{t})\] (1)

In Formula 1, \(\omega\), \(\xi_1\), and \(\xi_2\) are three weights in positive numbers (where \(\omega\) is referred to as the inertia weight and the other two weights referred to as the acceleration factors); \(r_1\) and \(r_2\) are two random numbers between 0 and 1; \(p_{id}^{t}\) is the \(d\)-th dimension of the personal optimum location that the

\(^1\)Strictly speaking, the MRs our approach inferred are likely MRs.

\(^2\)Search based software engineering [26], is an important branch of software engineering, which applies search techniques to solve various software-engineering problems [10, 8, 43]. To our knowledge, our approach is the first application of search based software engineering in MR inference.

\(^3\)If the input to a program is modified by adding or subtracting a constant, its output will remain or increase. Kanewala and Bieman [33] defined such a change to an input/output as additive.
that any particle has ever reached. Based on its velocity in moment \( t + 1 \), the location of the \( t \)-th particle can be calculated using Formula 2.

\[
l_{id}^{t+1} = l_{id}^t + v_{id}^{t+1}
\]  

In the beginning (i.e., moment 1), the \( N \) particles are assigned with locations and velocities randomly. The \( N \) particles keep updating their velocities and locations according to Formulae 1 and 2 until reaching moment \( T \) (which is a termination threshold). Then, the global optimum location that any particle has ever reached is returned as the final solution. Typically, each dimension is given a range, and if the location of any particle at any moment during the search is out of range, the location value in the corresponding dimension is set to the boundary values.

### 3.2 Parameterizing MRs

As a metamorphic relation (MR) usually specifies how a change to the input would result in a change to the output [33], we formalize an MR\(^4\) as Formula 3.

\[
R(i_1, i_2) \Rightarrow R(o_1, o_2)
\]  

In Formula 3, \( i_1 \) and \( i_2 \) are the original inputs and the changed inputs, \( O_1 \) and \( O_2 \) are the outputs corresponding to \( I_1 \) and \( I_2 \), \( R \) is a relation between input \( I_1 \) and input \( I_2 \), and \( R \) is a relation between output \( O_1 \) and output \( O_2 \). Without loss of generality, either \( I_1 \) or \( I_2 \) represents a vector of values, but either \( O_1 \) or \( O_2 \) represents only one value. The reason is that we can treat each output value individually when the program under test has more than one output value.

Although Formula 3 can exactly characterize an MR, it still cannot be directly used for MR inference, because both \( R \) and \( R \) can be in any form. As most MRs studied in the literature are polynomial, we further confine \( R \) and \( R \) in Formula 3 to polynomial equations. With this confinement, an MR can be characterized with the values of the parameters in the two polynomial equations.

Furthermore, as mentioned in the introduction, most polynomial metamorphic relations are simple polynomial metamorphic relations whose relations between inputs are linear equations and whose relations between outputs are linear equations or quadratic equations. That is, many MRs discussed in the literature are typically in much simpler forms than MRs involving two arbitrary polynomial equations. Therefore, in the current stage of our research, there are two cases under investigation: (1) \( R \) is a linear equation, and \( R \) is a linear equation, too. (2) \( R \) is a linear equation, while \( R \) is a quadratic equation. We will give a formula denoting MRs with each case.

Suppose that there are \( n \) input values for each input with \(<x_1, x_2, ..., x_n>\) and \(<y_1, y_2, ..., y_n>\) representing the two inputs (i.e., \( i_1 \) and \( i_2 \)). As we mentioned, \( i_1 \) is the original input and \( i_2 \) is the input derived from \( i_1 \). As \( R \) is a linear equation that represents the relation between \( i_1 \) and \( i_2 \), every input value of \( i_2 \) (i.e., \( y_j \)) can be denoted as a linear combination of all input values of \( i_1 \), i.e., \( y_i = \sum_{j=1}^{n} a_{ij} x_j + b_i \).

When \( R \) is a linear equation, we can represent \( R \) by \( c_1 O_1 + c_2 O_2 + d = 0 \), where \( O_1 \) and \( O_2 \) are the two outputs. Given a program under test (denoted as \( P \)), let us use \( P(x_1, x_2, ..., x_n) \) to denote output \( O_1 \), and \( P(\sum_{j=1}^{n} a_{ij} x_j + b_1, ..., \sum_{j=1}^{n} a_{ij} x_j + b_n) \) to denote output \( O_2 \). Then \( R \) can be further represented as:

\[
c_1 P(x_1, x_2, ..., x_n) + c_2 P(\sum_{j=1}^{n} a_{ij} x_j + b_1, ..., \sum_{j=1}^{n} a_{ij} x_j + b_n) + d = 0
\]  

(4)

Similarly, we can have a formula denoting MRs with \( R \) as a linear equation and \( R \) to be quadratic. As \( R \) is quadratic, we can represent \( R \) by \( c_1 O_1^2 + c_2 O_1 O_2 + c_2 O_2^2 + d O_1 + d O_2 + e = 0 \), where \( O_1 \) and \( O_2 \) are the two outputs. For simplicity, we use \( P(i_1) \) and \( P(a I_1 + \beta) \) to denote the two outputs \( O_1 \) and \( O_2 \), where \( a \) represents the matrix \( a[i, j] \) and \( \beta \) represents the vector \(<b_1, b_2, ..., b_n>\). Formula 5 denotes MRs with \( R \) to be quadratic.

\[
c_1 P^2(i_1) + c_2 P(i_1)P(a I_1 + \beta) + c_2 P^2(a I_1 + \beta) + d P(i_1) + d P(a I_1 + \beta) + e = 0
\]  

(5)

It should be noted that Formula 4 is actually a degenerate form of Formula 5, where the three coefficients of the three terms with degree 2 are all zero. However, in our search algorithm discussed in Section 3.3, we need to force the algorithm to produce non-degenerate MRs. For example, if we allow the algorithm to produce degenerate MRs, it may produce all zero for all the parameters. Therefore, we use both formulae in our search algorithm without allowing any degenerate MRs.

In the following, we demonstrate that all the MRs discussed previously can be represented with our two formulae. For \( \sin(x + \pi) = -\sin(x) \), when \( a_{11} = 1 \), \( b_1 = \pi \), \( c_1 = 1 \), \( c_2 = 2 \), and \( d = 0 \), Formula 4 actually represents this MR. Using Formula 5, when \( a_{11} = -1 \), \( b_1 = \pi/2 \), \( c_1 = c_2 = 1 \), \( c_2 = 0 \), and \( d_1 = d_2 = 0 \), and \( e = -1 \), it represents \( \sin^2(\pi/2 - x) + \sin^2(x) = 1 \); when \( a_{11} = 2 \), \( b_1 = 0 \), \( c_1 = 2 \), \( c_2 = c_2 = 0 \), \( d_1 = 0 \), \( d_2 = -1 \), and \( e = -1 \), it represents \( \cos(2x) = 2\cos^2(x) - 1 \). Similarly, we can also use Formula 5 to represent \( \tan^2(x) - 2\tan(x)\tan(2x - 3\pi/2) - 1 = 0 \).

In fact, given a program under test, the number of MRs that can be represented in Formula 4 or Formula 5 might still be infinite. Thus, it might still be infeasible to infer all the MRs satisfying the two formulae.

For ease of presentation, we refer to an MR inferred by using Formula 4 a type-one MR (denoted as 1-MR), whereas an MR inferred by using Formula 5 is called a type-two MR (denoted as 2-MR).

### 3.3 Searching for MR Parameters

Given a program under test (denoted as \( P \)) and \( M \) inputs (denoted as \( i_1, i_2, ..., i_M \)) of \( P \), the problem of inferring a polynomial MR for \( P \) can be turned into a search problem of finding some vector of parameter values in Formula 4 or Formula 5 such that for (almost) every input \( i_i \) (1 ≤ \( i \) ≤ \( M \))
the vector of parameter values and \( I_i \) satisfy Formula 4 or Formula 5.

To solve the preceding search problem, we adopt a PSO algorithm due to the following reasons. First, as PSO is very effective to search in continuous space, it may help find parameter values in real numbers. Second, as the location updating mechanism in PSO can keep particles from swaying among multiple optimal locations, it is suitable for the situation that there may be many MRs satisfying Formula 4 and/or Formula 5. However, when there are not many MRs, PSO is also effective due to the global optimal location and the velocity of the previous iteration can lead a particle to escape local optimal locations.

For simplicity of presentation, we focus on Formula 4 in the rest of this subsection. It is straightforward to extend to Formula 5. In our PSO, a candidate solution (i.e., a particle) is a set of parameter values (i.e., values of \( c_1, c_2, a_{ij}, b_i, \) and \( d \)). For every set of parameter values, the fitness function counts the number of inputs that satisfy Formula 4.

Formally, we define the fitness function for Formula 4 as follows. Given a vector (denoted as \( L \)) of values for \( c_1, c_2, a_{ij} \) (\( 1 \leq i, j \leq n \)), \( b_i \) (\( 1 \leq i \leq n \)), and \( d \), if \( L \) and input \( I_k \) (\( 1 \leq k \leq M \)) satisfy Formula 4, we define \( f(L, k) = 1 \); otherwise, we define \( f(L, k) = 0 \). Thus, the fitness of vector \( L \) can be defined as Formula 6, which actually counts the number of inputs that satisfy Formula 4 for \( L \).

\[
\text{fitness}(L) = \sum_{k=1}^{M} f(L, k)
\]  

As the location of every particle keeps changing, sometimes the values of \( c_1 \) and \( c_2 \) may be close to zero, which will make our MR pointless. To prevent our PSO algorithm from inferring MRs in degenerate forms, when both values of \( c_1 \) and \( c_2 \) in a particle are close to zero, we reset them to a new value. In particular, we use a threshold (denoted as \( \varphi \)). When both \( c_1 \) and \( c_2 \) are between \(-\varphi \) and \( \varphi \), we set them to \(-\varphi \) or \( \varphi \) depending on their being positive or negative.

As one execution of our PSO algorithm generates only one possible MR, we need to execute our PSO algorithm many times to obtain a number of MRs. Due to the random initialization of locations and velocities of the particles and the random factors (i.e., \( r_1 \) and \( r_2 \)) in Formula 1, different executions may produce different MRs. It is possible that an execution of our PSO algorithm may not always produce a good enough solution (whose fitness is lower than a threshold denoted as \( F \)). In such cases, we drop all not good enough solutions. Specific parameter-setting (e.g., \( F \)) of our PSO algorithm is presented in Section 4.

### 3.4 MR Filtering

Our PSO algorithm infers MRs based on the multiple executions of the program, and thus the quality of inferred MRs may be dependent on the test inputs of a program. Intuitively, our PSO algorithm tends to produce high-quality MRs if many test inputs are used. However, it is costly to conduct our PSO algorithm with a large number of test inputs. Therefore, our proposed approach uses statistics based filtering to remove low-quality MRs.

For each MRs inferred by our PSO algorithm, the statistics based filtering algorithm applies a large number of randomly generated test inputs to program \( P \), and records whether this MR is violated by each test input. If an MR is violated by an unignorable percentage (which is denoted as \( S \)) of test inputs, we deem such an MR as a low-quality MR and remove it from the set of inferred MRs. By repeating the preceding process several times (which is denoted as \( N_{of} \)), we deliver a set of high-quality MRs by removing some low-quality MRs.

### 4. IMPLEMENTATION

Based on the approach described in Section 3, we implemented a tool named MRI (Metamorphic Relation Inferrer).

For our PSO algorithm, we use the following settings as recommended in the literature of PSO [60, 36, 61]: we set the two acceleration factors (i.e., \( \xi_1 \) and \( \xi_2 \) in Formula 1) as 1.49445, the number of particles (i.e., \( N \)) as 20, and the termination threshold (i.e., the total number of moments \( T \)) as 350. The inertia weight (i.e., \( \omega \) in Formula 1) is a changing value for different moments. We set the value of \( \omega \) for moment \( t \) (denoted as \( \omega_t \)) according to Formula 7 (also recommended in PSO literature), where \( \omega_1 \) (which is the value for moment 1) is 0.9 and \( \omega_e \) (which is the value for moment \( T \)) is 0.4.

\[
\omega_t = \omega_1 - (\omega_1 - \omega_e)(t/T)^2
\]  

We set \( \varphi \) (which is the threshold to avoid degenerate MRs) as 0.5 and \( F \) (which is the threshold to select good enough solutions) as 95%*\( M \). Note that, in Formula 4, parameters \( c_1, c_2, \) and \( d \) can be inflated. If we multiply the three parameters with a common factor, we can have another MR that is semantically equivalent to the original MR. Thus, there can be a large number of MRs with the values of \( c_1, c_2, \) and \( d \) close to zero. Therefore, using a value significantly larger than zero for \( \varphi \) would also help save the efforts of inferring too many semantically equivalent MRs. Of course, 0.5 may not be the best value to achieve this goal. The setting of \( F \) as 95%*\( M \) is to due to statistical considerations, as such a setting implies that it is highly probable that an input can satisfy the inferred MR. Similarly, we also set \( S \) as 5%.

As PSO algorithm implies to set the boundary values for the parameters \( c_1, c_2, c_3, d_1, d_2, \alpha, \beta, \) and \( \epsilon \), we conducted a trial using the \( \sin \) function of Apache 2.2 to decide the proper boundary values of these parameters, with which MRI may infer high-quality MRs. In particular, we varied the boundary values for these parameters and applied MRI with any specified values of these parameters to the \( \sin \) function. After manually checking their inferred MRs for the \( \sin \) function, we determined the boundary values for these parameters based on the effectiveness of their inferred MRs. As a result, in our implementation, the boundary values for \( c_1, c_2, c_3, d_1, d_2, \) and \( \alpha \) are from -2 to 2, the boundary value for \( \beta \) is from -10 to 10, and the boundary value for \( e \) is from 0 to 10.

Moreover, in MRI, as the test inputs used in our PSO algorithm and MR filtering are randomly generated, we set the range of test inputs to be from 0 to 20. Besides, the number of test inputs used in our PSO algorithm is set to 100, the number of test inputs used in MR filtering is set to 1000, and \( N_{of} \) is set to 10.

### 5. EMPIRICAL SETUP

To evaluate our approach, we conducted three empirical studies on 189 scientific functions from four scientific libraries to answer the following research questions:
In the first study, we investigate the feasibility of our approach. In this study, we are interested in the following research question (RQ1): Is our approach able to infer MRs?

In the second study, we investigate the quality of inferred MRs. In this study, we are interested in the following research question (RQ2): How is the quality of MRs inferred by our approach?

In the third study, we investigate the impact of MR filtering in our approach. In this study, we are interested in the following research question (RQ3): Does MR filtering improve the quality of inferred MRs?

5.1 Subjects

Our approach infers MRs by analyzing multiple executions of a program rather than its source code. That is, our approach is a black-box technique, which requires no source code under test and may be applied to scientific functions in various languages. Therefore, in our empirical studies we used four scientific libraries which are written in Java and C/C++.

The four scientific libraries are Apache Commons Mathematics Library, the math class of Java Development Kit, the GNU Scientific Library, and the scientific library of MATLAB. Apache Commons Mathematics Library\(^5\) (abbreviated as Apache in this paper) is a library of self-contained mathematical and statistical components in Java. The math class of Java Development Kit\(^6\) (abbreviated as JDK in this paper) provides methods for numeric operations in the Java platform. The GNU Scientific Library\(^7\) (abbreviated as GSL in this paper) is a numeric library of mathematical routines in C/C++ language. MATLAB\(^8\) is a powerful computing environment developed by MathWorks, which also contains a very large number of scientific functions. Among the four libraries, Apache and GSL are open-source scientific libraries, whereas JDK and MATLAB are commercial scientific libraries. That is, the source code of the latter two libraries is not available.

As each scientific library contains a very large number of scientific functions, it is impossible for us to run our approach on all the scientific functions due to time limit on experimentation, so we further selected the packages that perform plain mathematical computations (e.g., not matrix operations or statistical calculations). In particular, we used the scientific programs in FastMath.java of Apache. For JDK and MATLAB, we used their same scientific programs as Apache. For GSL, we used the scientific programs in the “specfunc” directory. As a result, we got a dataset consisting of 56 scientific functions of Apache 2.2, 53 scientific functions of JDK 1.6, 55 scientific functions of GSL 1.8, and 25 scientific functions of MATLAB R2012b. Table 1 presents the basic information of these scientific functions, where the last three columns present the version information, the total number of scientific functions, and the total number of lines of code used in the studies. More information on these functions can be found in the project webpage: [http://infermrs.sourceforge.net/](http://infermrs.sourceforge.net/). As the source code of JDK 1.6 and that of MATLAB R2012b is not available, we do not list its number of lines of code in this table.

<table>
<thead>
<tr>
<th>Library</th>
<th>Version</th>
<th>#Function</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>2.2</td>
<td>56</td>
<td>1,626</td>
</tr>
<tr>
<td>JDK</td>
<td>1.6</td>
<td>53</td>
<td>-</td>
</tr>
<tr>
<td>GSL</td>
<td>1.8</td>
<td>55</td>
<td>1,309</td>
</tr>
<tr>
<td>MATLAB</td>
<td>R2012b</td>
<td>25</td>
<td>-</td>
</tr>
</tbody>
</table>

5.2 Process

In this subsection, we present the details of how we conducted the three empirical studies.

5.2.1 Study I

In the first study, for each function, we repeated our PSO algorithm 500 times. After MR filtering MRI inferred a set of 1-MRs and 2-MRs for each subject. We also recorded the total time spent on MR inference for each subject.

5.2.2 Study II

First, we investigated the correctness of MRs inferred in the first study. As \(\sin, \cos, \) and \(\tan\) are typical scientific functions, which are available through many mathematics books, we chose the three functions implemented in different libraries as representative subjects and manually checked the correctness of their inferred MRs as follows. For each of these inferred MRs, we looked through Wikipedia\(^9\) and a mathematics book [22] to check whether it is correct.

Some inferred MRs can be deduced by others. For example, \(\sin(-x) = \sin(x - \pi) = 0\) can be deduced by \(\sin(x) + \sin(-x) = 0\) and \(\sin(x) = \sin(x + \pi) = 0\). In software testing, if developers have already used the latter two MRs, they may not run again the \(\sin\) function with the former MR (i.e., \(\sin(-x) = \sin(x - \pi) = 0\)) because faults detected by the former MR may also be detected by the latter MRs. To acquire the set of MRs that cannot be deduced by each other, we define a set of representative MRs for any set of inferred MRs. In particular, for any given set of inferred MRs (i.e., 1-MRs or 2-MRs) of a subject, its set of representative MRs is the minimized subset of the given set, and each MR of the given set can be deduced by using one or more MRs in its representative set. For example, for a set of inferred MRs \(\{\sin(-x) = \sin(x - \pi) = 0, \sin(x) + \sin(-x) = 0, \sin(x) + \sin(x - \pi) = 0\}\), its set of representative MRs is \(\{\sin(x) + \sin(-x) = 0, \sin(x) = \sin(x - \pi) = 0\}\). In this study, for each trigonometric function, we manually summarized its set of representative 1-MRs and 2-MRs respectively. If multiple minimal subsets exist, we chose one that has the smallest size.

Besides, Apache 2.2, JDK 1.6, and MATLAB R2012b implement some common scientific functions. For each common scientific function (e.g., \(\exp\)), we compared the number of MRs inferred from different libraries.

Second, we investigated the fault-detection capability of MRs inferred in the first study through regression testing [69, 78]. In particular, to simulate the regression testing in software evolution, we applied the MRs inferred from the correct version of a project on detecting faults in its subsequent version. Among the functions used in the empirical studies, 11 functions (i.e., \(\sin, \cos, \tan, \log_{10}, \log_{10p}, \text{asinh}, \text{atan}, \) and four abs functions with various inputs) have changed from Apache 2.2 to Apache 3.2. The other functions used in the

\(^{9}\)http://en.wikipedia.org/wiki/Trigonometric_functions/#Sine.2C_cosine.2C_and_tangent
empirical studies have no change during software evolution. Therefore, in the second study we used these 11 functions of Apache 3.2 to investigate the fault-detection capability of MRs inferred from the first study. As the functions of Apache have been widely used in practical software development, they hardly contain any faults. Therefore, for these functions we constructed faults using program mutation following procedure similar to prior work [46, 77]. The difference between versions shows the developers’ modification on the previous version, and thus we generated faults only in such difference so as to simulate most developers’ faults in software evolution. In particular, for each of the 11 functions of Apache 3.2, we applied MuClipse [62] to generate a number of mutants whose mutation operators\(^{10}\) occur only on the different statements between Apache 2.2 and Apache 3.2. Each mutant, which is the result of applying a mutation operator on the source code, is viewed as a faulty program in our second study.

If an MR is violated by a faulty program, we deem the MR detects the fault. However, if the MR is also violated by the original, unseeded program of Apache 3.2, we deem this MR is of low quality and the detection is a false detection. For each of the 11 functions of Apache 3.2, we randomly generated 1000 test inputs. Then, we ran these test inputs on both the generated faulty versions of Apache 3.2 and the original version of Apache 3.2 for each MR inferred from Apache 2.2. As our approach infers MRs that are supported by at least 95% inputs, the inferred MRs should be used in a statistical way of metamorphic testing, which is referred to as statistical metamorphic testing in this paper. In statistical metamorphic testing, only when the violation of an inferred MR become statistically non-trivial, we deem the program to be likely to contain faults. In particular, we consider an MR was violated when at least 5% of the test inputs were violated considering anomaly detection.

5.2.3 Study III
To learn whether MR filtering improves the quality of inferred MRs, we compared the quality of MRs inferred with MR filtering and the quality of MRs inferred without MR filtering in regression testing following the same procedure of the second study. In this study, for each faulty program of Apache 3.2, we used statistical metamorphic testing to evaluate the quality of these MRs, recording the number of true detections and the number of false detections. Finally, we compared the numbers of true detections and false detections between MRs inferred with MR filtering and MRs inferred without MR filtering.

5.3 Threats to Validity
The threat to internal validity lies in the implementation of our approach. To reduce the threat from implementing errors, the authors of this paper reviewed the source code after implementing the proposed approach.

The main threats to external validity lie in the subjects and faults. First, similar to prior work [14] in metamorphic testing, we used four scientific libraries consisting of small\(^{11}\) scientific functions whose MRs may be manually checked. As our approach is a black-box technique that does not analyze the source code, whether the scientific functions are small or large does not affect the feasibility of our approach. However, the functionality of scientific functions has much impact on the feasibility of our approach because our approach infers MRs based on their executions. To reduce this threat, we used a large number of scientific programs from different libraries. Moreover, although our approach is evaluated based on scientific programs, the approach has no such restrictions and can be applied to any programs. To reduce this threat, we will evaluate our approach by other programs in the future. Second, the faults were generated by using a mutation tool because prior work [2] shows that such faults can be used in the empirical studies of software testing. As these faults may be not representative of real faults, we will conduct more empirical studies on more programs with real faults in the future. Furthermore, in the evaluation we used the implementation of our approach introduced in Section 4, whose value for each parameter is set based on the literature of PSO and our trial on the \(\sin\) function. However, as our approach does not have any restrictions on the values of the parameters, in future work we will conduct empirical studies to evaluate the effectiveness of our approach with other values of these parameters.

6. RESULTS

6.1 RQ1: MR Inference
Table 2 presents the basic statistics on MR inference, including the number of inferred MRs for each scientific function and the execution time of our approach on inferring MRs for each scientific function. The complete results can be found in http://infermrs.sourceforge.net/. From this table, for each scientific function, the number of 1-MRs is from 0 to 353, whereas the number of 2-MRs is from 0 to 401. That is, our approach infers an unignorable number of 1-MRs and 2-MRs for scientific functions. The execution time of our approach on inferring MRs for each scientific function is from 9.87 seconds to 1231.16 seconds, which is acceptable. Therefore, our approach is able to infer many MRs quickly.

### Table 2: Basic statistics on MR inference

<table>
<thead>
<tr>
<th>Library</th>
<th>1-MRs</th>
<th>2-MRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>57.96</td>
<td>0</td>
</tr>
<tr>
<td>JDK</td>
<td>85.04</td>
<td>0</td>
</tr>
<tr>
<td>GSL</td>
<td>80.85</td>
<td>0</td>
</tr>
<tr>
<td>MATLAB</td>
<td>47.24</td>
<td>0</td>
</tr>
</tbody>
</table>

### Table 3: Typical MRs of the three trigonometric functions \(\sin\), \(\cos\), and \(\tan\)

<table>
<thead>
<tr>
<th>Function</th>
<th>1-MR</th>
<th>2-MRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\sin)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\cos)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(\tan)</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

6.2 RQ2: Quality of Inferred MRs

6.2.1 Correctness
Table 3 presents typical MRs of the three trigonometric functions \(\sin\), \(\cos\), and \(\tan\). Our approach generates MRs

---

\(^{10}\) Mutation operators define some operations like statement deletion, statement replacement, and so on.

\(^{11}\) After manually studying the literature of software engineering [5, 63, 7, 23], we found that the scientific functions used in their evaluation are usually very small, which are usually smaller than 100 lines of code.
### Table 3: MRs inferred from three trigonometric functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Library</th>
<th>1-MRs</th>
<th>2-MRs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sin</strong></td>
<td>Apache</td>
<td>( \sin(x) - \sin(x - 2\pi) = 0 )</td>
<td>( \sin^2(x) + \sin^2(-x - 0.5\pi) - 1 = 0 )</td>
</tr>
<tr>
<td></td>
<td>JDK</td>
<td>( \sin(x) + \sin(x - \pi) = 0 )</td>
<td>( \sin^2(-0.5x - 0.75\pi) + 0.5\sin^2(x) - 0.5 = 0 )</td>
</tr>
<tr>
<td></td>
<td>GNU</td>
<td>( \sin(x) - \sin(-x) = 0 )</td>
<td>( \sin(x) + \sin^2(x - 2\pi) + 2\sin(x)\sin(-x) = 0 )</td>
</tr>
<tr>
<td></td>
<td>GSL</td>
<td>( \sin(x) - \sin(-x) = 0 )</td>
<td>( \sin(-0.5\sin^2(x) - 2\sin(x)\sin(-x) = 0 )</td>
</tr>
<tr>
<td></td>
<td>MATLAB</td>
<td>( \sin(x) + \sin(x + \pi) = 0 )</td>
<td>( \sin(x) - \sin(-x) - 2\sin(-x + x = 0 )</td>
</tr>
<tr>
<td><strong>cos</strong></td>
<td>Apache</td>
<td>( \cos(x) + \cos(-x) = 0 )</td>
<td>( \cos^2(x) + \cos(-x - 0.5\pi) - 1 = 0 )</td>
</tr>
<tr>
<td></td>
<td>JDK</td>
<td>( \cos(x) - \cos(-x) = 0 )</td>
<td>( \cos^2(-0.5x - 1.5\pi) + 0.5\cos(x - 0.5\pi) = 0 )</td>
</tr>
<tr>
<td></td>
<td>GNU</td>
<td>( \cos(x) + \cos(x) = 0 )</td>
<td>( \cos(x) + \cos(-x - 0.5\pi) - 1 = 0 )</td>
</tr>
<tr>
<td></td>
<td>GSL</td>
<td>( \cos(x) - \cos(-x) = 0 )</td>
<td>( \cos(x) + \cos(x)\cos(x + \pi) - \cos(x) = 0 )</td>
</tr>
<tr>
<td></td>
<td>MATLAB</td>
<td>( \cos(x) + \cos(x) = 0 )</td>
<td>( \cos(x) + \cos(-x - 0.5\pi) - 1 = 0 )</td>
</tr>
<tr>
<td><strong>tan</strong></td>
<td>Apache</td>
<td>( \tan(x) + \tan(-x + \pi) = 0 )</td>
<td>( \tan(-0.5x - 1.25\pi) + 2\tan(x)\tan(-0.5x + 1.25\pi) - 1 = 0 )</td>
</tr>
<tr>
<td></td>
<td>JDK</td>
<td>( \tan(x) - \tan(-x + 2\pi) = 0 )</td>
<td>( \tan(x) - \tan(-x) - 2\tan(x)\tan(-x) = 0 )</td>
</tr>
<tr>
<td></td>
<td>MATLAB</td>
<td>( \tan(x) + \tan(x + 2\pi) = 0 )</td>
<td>( \tan(x) - \tan(-x + x = 0 )</td>
</tr>
</tbody>
</table>

### Table 4: Comparison of some inferred MRs in the first study

<table>
<thead>
<tr>
<th>Function</th>
<th>1-MRs</th>
<th>2-MRs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>abs</strong></td>
<td>abs_d</td>
<td>abs_f</td>
</tr>
<tr>
<td>Apache</td>
<td>141</td>
<td>160</td>
</tr>
<tr>
<td>JDK</td>
<td>132</td>
<td>145</td>
</tr>
<tr>
<td>MATLAB</td>
<td>148</td>
<td>302</td>
</tr>
<tr>
<td><strong>acos</strong></td>
<td>log1p</td>
<td>log10</td>
</tr>
<tr>
<td>Apache</td>
<td>0</td>
<td>302</td>
</tr>
<tr>
<td>JDK</td>
<td>0</td>
<td>306</td>
</tr>
<tr>
<td>MATLAB</td>
<td>0</td>
<td>317</td>
</tr>
<tr>
<td><strong>asin</strong></td>
<td>atanh</td>
<td>atan</td>
</tr>
<tr>
<td>Apache</td>
<td>0</td>
<td>302</td>
</tr>
<tr>
<td>JDK</td>
<td>0</td>
<td>306</td>
</tr>
<tr>
<td>MATLAB</td>
<td>0</td>
<td>317</td>
</tr>
</tbody>
</table>

**Notes:**
- **abs** represents the absolute value function.
- **acos** represents the inverse cosine function.
- **asin** represents the inverse sine function.
- **atan** represents the inverse tangent function.
- **round** represents the rounding function.
- **cos** represents the cosine function.
- **sin** represents the sine function.
- **tan** represents the tangent function.
by assigning values to the parameters in the formulae with some precision. To ease understanding, we present the inferred MRs in this table by using the estimation of these values. For example, in the $\sin$ function, we use $\pi$ to denote $3.14^{12}$. GSL does not implement the $\tan$ function and thus we do not list its inferred MRs in this table. For each subject, we use the bold font to depict its complete sets of representative MRs. The left columns give 1-MRs inferred by our approach using Formula 4 whereas the right columns give 2-MRs inferred by our approach using Formula 5.

From this table, these typical MRs include most important MRs of the three trigonometric functions. For example, our approach infers series of 1-MRs and 2-MRs for the $\cos$ function of the four scientific libraries. These inferred 1-MRs, represented by $\cos(x) - \cos(-x) = 0$ and $\cos(x) + \cos(x - \pi) = 0$, show that the $\cos$ function is a symmetric and periodical function. The inferred 2-MRs, represented by $\cos^2(x) + \cos^2(-x - 0.5\pi) - 1 = 0$, $\cos^2(-0.5x - 1.5\pi) + 0.5\cos(x) - 0.5 = 0$, $\cos^2(x) - \cos^2(x + \pi) = 0$, and $\cos^2(x) + \cos^2(-x) - 2\cos(x)\cos(-x) = 0$, show the relation between $\cos(x)$, $\cos(2x)$ and $\cos(x - 0.5\pi)$ besides the symmetric and periodical characteristics of the $\cos$ function. Furthermore, our approach infers the complex MRs $\sin^2(\pi/2 - x) + \sin^2(x) - 1 = 0$ for the $\sin$ function and $\tan^2(x) - 2\tan(2x - 3\pi/2)\tan(x) - 1 = 0$ for the $\tan$ function from the libraries.

Table 4 presents the total number of MRs inferred from the common scientific functions. For each function, the numbers of inferred MRs from different libraries are close. This observation is as expected because these functions have the same functionality, suggesting the correctness of the inferred MRs. For the same scientific function, the average execution time of our approach for different libraries is close. That is, although different libraries may implement a scientific function in different ways$^{13}$ (i.e., resulting in different programs), their kernel source code may not differ much in efficiency and thus the execution time of our approach for the same scientific functions of different libraries is close.

From Table 4, the number of inferred MRs is larger than that of the representative MRs. For example, our approach generated 219 1-MRs for the $\sin$ function of Apache, but only 2 of them are representative. The other 217 MRs can be deduced by these 2 representative MRs. As Chen et al. [11] demonstrate that more MRs may help to achieve more adequate testing, the MRs that can be deduced by some representative MRs may not be redundant. Moreover, more MRs may reduce the cost in software testing. For example, in order to reveal the faults that can be detected only by $\sin(x) - \sin(-x + 3\pi) = 0$, it may be more costly to check the two representative MRs (i.e., $\sin(x) + \sin(-x) = 0$ and $\sin(x) + \sin(x - \pi) = 0$) rather than one MR. To check the former MR, developers may run the $\sin$ function twice, whereas checking the latter two MRs, developers may run the $\sin$ function four times.

### 6.2.2 Fault-Detection Capability

Table 5 gives the results of the second study, where the second column gives the total number of mutation faults in each scientific function of Apache 3.2. “FD” presents the number of false detections by the corresponding MRs, “TD” presents the number of true detections by the corresponding MRs, and “Total” is the sum of its previous two columns. From the fifth and eighth columns, the numbers of true detections for 1-MRs and 2-MRs are usually larger than 0. That is, the inferred MRs are able to detect faults. From the fourth and seventh columns, the numbers of false detections for 1-MRs and 1-MRs are 0. That is, the inferred MRs always make correct detection. On the other hand, for most scientific functions (including $\sin$, $\cos$, $\tan$, $abs_d$, $abs_f$, $abs_z$, and $abs_l$), the inferred MRs detect about half of the faults. The only exception is $asinh$, in which only one fault is detected out of 297 faults. By further investigating the injected faults, we found that this is probably because the rest 296 faults were not triggered by the test inputs: the mutated statements of all the 296 faults will be executed only when a strict condition (i.e., variable $a$ is smaller than 0.167) is satisfied. Overall, our inferred MRs were effective in detecting faults and produced no false detection.

Comparing the results of the two types of MRs (i.e., 1-MRs and 2-MRs), the number of faults detected by the former is close to that of the latter. After reviewing these inferred MRs and their detected faults, we found the reason to be that our approach always infers some important MRs that can detect a large number of faults no matter which formula (i.e., Formula 4 and Formula 5) it used.

### 6.3 RQ3: Necessity of MR Filtering

Table 6 presents the fault-detection capability of MRs inferred by our approach without MR filtering. For each scientific function, the number of false detections of MRs inferred without MR filtering is usually larger than 0. For example, the 1-MRs for the $\tan$ function have 10 false detections. As the functionality of these subjects does not change from Apache 2.2 to Apache 3.2, these inferred MRs should not be violated. That is, the false detections in this table result from low-quality MRs, which are inferred by our approach without MR filtering. As the number of false detections detected by the MRs inferred with MR filtering is 0 (shown by Table 5), MR filtering actually improves the quality of MRs by removing low-quality MRs.

On the other hand, the filtering did reduce the number of true detections, but the reduced number was small. Overall, the reduction on fault detection occurred only on 3 out of 11 functions, and in total only 16.9% true detections were filtered out. Considering the large number of false detections removed, we believe the filtering procedure is effective and necessary.

<table>
<thead>
<tr>
<th>Function</th>
<th>Seeded Faults</th>
<th>#By 1-MRs</th>
<th>#By 2-MRs</th>
<th>#By 1-MRs</th>
<th>#By 2-MRs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sin$</td>
<td>17</td>
<td>9</td>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>$\cos$</td>
<td>19</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>$\tan$</td>
<td>18</td>
<td>8</td>
<td>0</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>log10</td>
<td>58</td>
<td>7</td>
<td>0</td>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>log1p</td>
<td>115</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>0</td>
</tr>
<tr>
<td>asinh</td>
<td>297</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>atan</td>
<td>94</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>abs_d</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>abs_f</td>
<td>7</td>
<td>5</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>abs_z</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>abs_l</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
</tbody>
</table>

---

12 The complete list of inferred MRs for these functions can be found in the project webpage.

13 As the source code of JDK and MATLAB is not available, we cannot check the difference between the source code of the three libraries.
First, the $R_i$ relation on inputs can be extended to a polynomial with a higher degree (i.e., greater than 1). For example, using a polynomial with a degree of 2 for $R_i$ may help infer $"\log_{10}(x^2) = 2\log_{10}(x)"$ for the $\log_{10}$ function.

Second, the $R_o$ relation on outputs can also be extended to a polynomial with a higher degree (i.e., greater than 2). For example, using a polynomial with a degree of 3 for $R_o$ may help infer $"\sin(3x) = 3\sin(x) - 4\sin^3(x)"$ for $\sin$.

Third, the number of involved inputs can be extended to more than 2. For example, involving 3 inputs may help infer $"\sin(2x) = 2\sin(x)\sin(\pi/2 - x)"$ for the $\sin$ function.

As our approach in this paper cannot deal with these extensions, we will improve our PSO algorithm for further investigating these situations.

8. RELATED WORK

8.1 Metamorphic Testing

Chen et al. originally proposed the methodology of metamorphic testing [11] and formally established the methodology [15]. Besides application of metamorphic testing on various areas [12, 48], some researchers focus on the selection of good MRs, which is related to our work. Chen et al. [13] demonstrated that it is better to select MRs that make the multiple executions of the program as different as possible. Mayer and Guderlei [45] identified some MRs and classified them using mutation analysis. They also evaluated MRs according to their potential usefulness. Recently, Liu et al. [38] proposed to systematically construct MRs based on some already identified MRs. Differently, our approach aims to automatically infer MRs without any existing MRs.

Our work is mostly related to the technique proposed by Kanewala and Bieman [33], which automatically predicts the existence of some forms of MRs for a program using machine learning. Different from our approach, their technique does not produce specific MRs, but tells whether a program may have a particular form of MRs or not. Their technique and our approach may be viewed as complement to each other. In particular, we may use their technique to predict the existence of a form of MR and use our technique to produce the specific MR by giving the values of parameters. Furthermore, their technique is a white-box technique, which extracts features for prediction by analyzing the source code of the program under test, whereas our technique is a black-box technique. That is, with the source code of the program under test, our technique may be further improved to produce better MRs.

8.2 Program-Invariant Inference

As program invariants are important for fault detection and program repairing, researchers proposed to infer program invariants through analysis, especially dynamic analysis. For example, Ernst et al. [21] developed a tool named Daikon to discover program invariants for supporting program evolution. Jiang et al. [32] proposed a novel technique to automatically model and search relationships between the flow intensities that can be regarded as invariants. Csallner et al. [16] proposed to infer invariants using dynamic symbolic execution. Llano et al. [39] proposed to use theory formation to discover invariants. Recently, Nguyen et al. [50] inferred disjunctive invariants with a hybrid approach. Furthermore, as specifications tell the usage of API and may be used to detect faults, many researchers focus
on inferring specifications [81, 82, 73], which can be viewed as program invariants as well. Besides work on invariant generation [79, 37, 6, 49, 42], some researchers focused on using invariants to facilitate software testing (e.g., test-case generation [75] and test-suite reduction [51]), software verification [52], model inference and transformation [35, 9], specifications mining [40], and so on.

Our work is related to program-invariant inference because MRs can also be regarded as program invariants, both of them may be applied to reveal faults in software testing [20]. However, traditional program invariants are supposed to hold during each single execution, whereas MRs are supposed to hold across multiple executions. Similar to dynamic invariant inference, our approach is also based on the analysis of program executions. However, our approach is to search for the values of the parameters in an MR, whereas dynamic variant inference is to discover invariants that satisfying executions.

8.3 Particle Swarm Optimization

Particle swarm optimization (PSO) [54, 80] is a swarm intelligence optimization algorithm simulating the birds foraging behavior. Due to the efficiency of PSO in solving optimization problems, PSO has been applied to various areas, including multi-objective optimization, pattern recognition, signal processing, classification and data clustering.

Recently PSO is applied to some specific areas of software engineering, e.g., automated test-case generation [67]. In this paper, we use PSO for MR inference. To our knowledge, it is the first application of PSO in metamorphic testing.

8.4 Search-Based Software Engineering

Harman and Jones [26] coined the term Search Based Software Engineering (SBSE) and argued that software engineering is ideal for the application of metaheuristic search techniques, such as genetic algorithms. Typically, hill climbing, simulated annealing and genetic algorithms are the three main metaheuristic search techniques that have been widely used in software engineering [25].

Search-based optimization techniques have been widely applied to software testing, including test-suite generation [8, 4, 56, 24, 53] and optimization [72, 31, 43, 3, 74]. Besides software testing, search-based optimization techniques have also been applied to fault localization [65], program analysis [76], software refactoring [29, 30, 55], cost estimation [19], project scheduling [1, 18], decisions design optimization [10], automated negotiation [17], source code parallelization [57], requirement engineering [27, 64], variability management [41], and so on.

Although search-based software engineering is important and promising, very little research in search-based software engineering has used PSO as a metaheuristic search technique. Our work is the first application of search-based software engineering for program-invariant inference.

9. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel approach to automatically inferring polynomial metamorphic relations by analyzing multiple executions of the same program under test. To our knowledge, this is the first automatic approach to MR inference. In particular, we view the problem of MR inference as a searching problem and thus use a typical optimization algorithm PSO to solve the problem. Then we conducted three empirical studies and got the finding that our approach is able to infer many MRs with high quality in acceptable time, which are effective in detecting faults with no false detection.

In our future, we plan to investigate the following issues.

First, we will extend types of MRs in future work. Besides polynomial equations studied in this paper, some MRs may be represented by polynomial inequalities. For the scientific function \( \log(10(x)) \), if \( x_1 \) is larger than \( x_2 \), \( \log(10(x_1)) \) is larger than \( \log(10(x_2)) \). Relations between programs (e.g., \( \sin^2(x) + \cos^2(x) = 1 \)) may also help detect faults in the functions \( \sin \) and \( \cos \). In future work, we will extend the definition of MRs and investigate how to infer these MRs.

Second, we will improve our approach by investigating other PSO algorithms or optimization algorithms. Besides the PSO algorithm used in this paper, there exist many other optimization algorithms [1] like hill climbing, which have been used to solve similar search problems (e.g., test-suite reduction) in software testing. In future work, we will investigate some other PSO algorithms for the MR inference problem or optimization algorithms in MR inference.

10. ACKNOWLEDGMENTS

This work is supported by the National Basic Research Program of China under Grant No.2014CB347701, the High-Tech Research and Development Program of China under Grant No.2013AA01A605, and the National Natural Science Foundation of China under Grant Nos.61121063, 61332010, 61272157, 61228203, 61225007.

11. REFERENCES


