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.

[10]Mutation operators define some operations like statement deletion, statement replacement, and so on.

[11]After manually studying the scientific functions used in 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 2: Basic statistics on MR inference

Number of MRs for each scientific program						
Library	**1-MRs**			**2-MRs**		
	Avg.	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**
Apache	87.96	353	0	46.63	401	0
JDK	85.04	348	0	47.72	395	0
GSL	80.85	331	0	52.78	239	0
MATLAB	47.24	168	0	13.52	78	0
Execution time of MR inference for each scientific program						
Library	**1-MRs (seconds)**			**2-MRs (seconds)**		
	Avg.	**Max.**	**Min.**	**Avg.**	**Max.**	**Min.**
Apache	49.87	261.03	15.19	33.33	96.20	16.25
JDK	42.37	404.08	14.92	45.35	421.88	17.02
GSL	97.70	315.05	9.87	352.88	1231.16	18.15
MATLAB	123.99	202.41	67.53	247.10	474.29	88.01

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.

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

Table 3: MRs inferred from three trigonometric functions

Function	Library	1-MRs	2-MRs
sin	Apache	$sin(x) - sin(x - 2\pi) = 0$	$\mathbf{sin^2(x) + sin^2(-x + 0.5\pi) - 1 = 0}$
		$\mathbf{sin(x) + sin(x - \pi) = 0}$	$\mathbf{sin^2(-0.5x - 0.75\pi) + 0.5sin(x) - 0.5 = 0}$
		$\mathbf{sin(x) + sin(-x) = 0}$	$\mathbf{sin^2(x) + sin^2(-x) + 2sin(x)sin(-x) = 0}$
		$sin(x) - sin(-x + \pi) = 0$...	$\mathbf{sin^2(x) + sin^2(x - \pi) + 2sin(x)sin(x - \pi) = 0}$...
	JDK	$sin(x) - sin(x - 2\pi) = 0$	$\mathbf{sin^2(x) + sin^2(x - 2.5\pi) - 1 = 0}$
		$\mathbf{sin(x) + sin(x - \pi) = 0}$	$\mathbf{sin^2(x) + 0.5sin(2x - 1.5\pi) - 0.5 = 0}$
		$\mathbf{sin(x) + sin(-x) = 0}$	$\mathbf{sin^2(x) + sin^2(-x) + 2sin(x)sin(-x) = 0}$
		$sin(x) + sin(-x - 4\pi) = 0$...	$\mathbf{sin^2(x) + sin^2(x - \pi) + 2sin(x)sin(x - \pi) = 0}$...
	GSL	$sin(x) - sin(-x - \pi) = 0$	$\mathbf{sin^2(x) - sin^2(-x - 2\pi) - sin(x) - sin(-x - 2\pi) = 0}$
		$\mathbf{sin(x) + sin(x + \pi) = 0}$	$\mathbf{sin(x) + 0.5sin^2(-x) + 1.5sin(x)sin(-x) + 0.5sin(x) + 0.5sin(-x) = 0}$
		$\mathbf{sin(x) + sin(-x) = 0}$	$\mathbf{sin^2(x) - sin(x)sin(-x + \pi) + sin(x) - sin(-x + \pi) = 0}$
		$sin(x) - sin(-x - 3\pi) = 0$...	$sin^2(x) + sin^2(-x) - 2sin(x)sin(-x) = 0$...
	MATLAB	$\mathbf{sin(x) + sin(-x) = 0}$	$\mathbf{sin^2(x) + sin^2(-x + 0.5\pi) - 1 = 0}$
		$\mathbf{sin(x) + sin(x - \pi) = 0}$	$\mathbf{sin^2(x) + sin^2(-x - \pi) - 2sin(x)sin(-x - \pi) = 0}$
		$sin(x) + sin(-x + 2\pi) = 0$	$\mathbf{sin^2(x) + sin(x)sin(x - \pi) + 2sin(x) + 2sin(x - \pi) = 0}$
		$sin(x) - sin(x - 2\pi) = 0$...	$\mathbf{sin^2(x) + 0.5sin(2x + 0.5\pi) - 0.5 = 0}$...
cos	Apache	$cos(x) + cos(-x - \pi) = 0$	$\mathbf{cos^2(x) + cos^2(-x - 0.5\pi) - 1 = 0}$
		$cos(x) - cos(x - 2\pi) = 0$	$\mathbf{cos^2(-0.5x - 1.5\pi) + 0.5cos(x) - 0.5 = 0}$
		$\mathbf{cos(x) + cos(x - \pi) = 0}$	$\mathbf{cos^2(x) - cos^2(x + \pi) = 0}$
		$\mathbf{cos(x) - cos(-x) = 0}$...	$\mathbf{cos^2(x) + cos^2(-x) - 2cos(x)cos(-x) = 0}$...
	JDK	$cos(x) + cos(-x - \pi) = 0$	$\mathbf{cos^2(x) + cos^2(-x + 0.5\pi) - 1 = 0}$
		$cos(x) - cos(x - 2\pi) = 0$	$\mathbf{cos^2(0.5x - 2\pi) - 0.5cos(x) - 0.5 = 0}$
		$\mathbf{cos(x) + cos(x - \pi) = 0}$	$\mathbf{cos^2(x) + cos(x)cos(-x + \pi) - cos(x) - cos(-x + \pi) = 0}$
		$\mathbf{cos(x) - cos(-x) = 0}$...	$\mathbf{cos^2(x) - cos(x)cos(-x) = 0}$...
	GSL	$cos(x) - cos(x + 2\pi) = 0$	$\mathbf{cos^2(x) + 0.5cos^2(x + \pi) + 1.5cos(x)cos(x + \pi) - 1.5cos(x) - 1.5cos(x + \pi) = 0}$
		$\mathbf{cos(x) + cos(x - \pi) = 0}$	$\mathbf{cos^2(x) - cos(x)cos(-x + 2\pi) - cos(x) + cos(-x + 2\pi) = 0}$
		$\mathbf{cos(x) - cos(-x) = 0}$	$\mathbf{cos^2(0.5x) - 0.5cos(x) - 0.5 = 0}$
		$cos(x) - cos(-x - 3\pi) = 0$...	$cos^2(x) + cos^2(x - 2\pi) - 2cos(x)cos(x - 2\pi) = 0$...
	MATLAB	$cos(x) - cos(-x + 2\pi) = 0$	$\mathbf{cos^2(-0.5x - 1.5\pi) + 0.5cos(x) - 0.5 = 0}$
		$\mathbf{cos(x) - cos(-x) = 0}$	$\mathbf{cos^2(x) + 3cos^2(x - \pi) + 4cos(x)cos(x - \pi) = 0}$
		$\mathbf{cos(x) + cos(x - \pi) = 0}$	$\mathbf{cos^2(x) + cos^2(x + 0.5\pi) - 1 = 0}$
		$cos(x) + cos(x + \pi) = 0$...	$\mathbf{cos^2(x) - cos^2(-x - \pi) + cos(x) + cos(-x - \pi) = 0}$...
tan	Apache	$\mathbf{tan(x) + tan(-x + \pi) = 0}$	$\mathbf{tan^2(0.5x + 1.25\pi) - 2tan(x)tan(0.5x + 1.25\pi) - 1 = 0}$
		$tan(x) - tan(x - 2\pi) = 0$	$\mathbf{tan^2(x) + tan^2(-x) + 2tan(x)tan(-x) = 0}$
		$\mathbf{tan(x) - tan(x - \pi) = 0}$	$\mathbf{tan^2(x) - tan^2(x + \pi) - tan(x) + tan(x + \pi) = 0}$
		$tan(x) - tan(x + 2\pi) = 0$...	$tan^2(x) + tan^2(x + 3\pi) - 2tan(x)tan(x + 3\pi) = 0$...
	JDK	$\mathbf{tan(x) + tan(-x + \pi) = 0}$	$\mathbf{tan^2(x) - 2tan(x)tan(2x - 1.5\pi) - 1 = 0}$
		$tan(x) + tan(-x + 2\pi) = 0$	$\mathbf{tan^2(x) - tan^2(-x) = 0}$
		$\mathbf{tan(x) - tan(x - \pi) = 0}$	$\mathbf{tan^2(x) - tan^2(x + \pi) + tan(x) - tan(x + \pi) = 0}$
		$tan(x) - tan(x + \pi) = 0$...	$tan^2(x) + tan^2(-x - 2\pi) + 2tan(x)tan(-x - 2\pi) = 0$...
	MATLAB	$\mathbf{tan(x) + tan(-x) = 0}$	$\mathbf{tan^2(x) + tan^2(-x + \pi) + 2tan(x)tan(-x + \pi) = 0}$
		$tan(x) - tan(x - 2\pi) = 0$	$\mathbf{tan^2(x) + 0.5tan^2(-x) + 1.5tan(x)tan(-x) - 0.5tan(x) - 0.5tan(-x) = 0}$
		$\mathbf{tan(x) + tan(-x + \pi) = 0}$	$tan^2(x) + tan^2(-x + 2\pi) + 2tan(x)tan(-x + 2\pi) = 0$
		$tan(x) + tan(-x - 2\pi) = 0$...	$\mathbf{tan^2(0.5x - 1.75\pi) - 2tan(x)tan(0.5x - 1.75\pi) - 1 = 0}$...

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

1-MRs	*abs_d*	*abs_f*	*abs_i*	*abs_l*	*acos*	*acosh*	*asin*	*asinh*	*atan*	*round_d*	*cbrt*	*ceil*	*cos*
Apache	141	160	64	73	0	19	0	21	62	118	23	123	207
JDK	132	145	96	72	0	-	0	-	60	120	30	97	213
MATLAB	148	-	-	-	19	8	44	10	50	78	-	12	129
1-MRs	*toDegrees*	*signum_f*	*expm1*	*floor*	*gE_d*	*gE_f*	*log*	*log10*	*log1p*	*nextUp_d*	*nextUp_f*	*rint*	*atanh*
Apache	0	302	73	115	106	106	40	133	50	331	350	126	0
JDK	5	306	70	120	115	117	60	135	54	348	334	110	-
MATLAB	-	-	-	84	-	-	28	95	44	-	-	-	52
1-MRs	*round_f*	*signum_d*	*exp*	*sin*	*sinh*	*sqrt*	*tan*	*tanh*	*cosh*	*toRadians*	*ulp_d*	*ulp_f*	
Apache	122	344	64	219	0	9	20	139	0	353	220	236	
JDK	122	317	68	227	0	13	13	134	0	336	217	229	
MATLAB	-	-	41	168	0	5	8	125	0	-	-	-	
2-MRs	*abs_d*	*abs_f*	*abs_i*	*abs_l*	*acos*	*acosh*	*asin*	*asinh*	*atan*	*round_d*	*cbrt*	*ceil*	*cos*
Apache	5	2	44	39	0	3	0	2	14	46	1	28	108
JDK	1	3	30	42	0	-	0	-	7	47	4	31	101
MATLAB	6	-	-	-	3	1	4	1	6	21	-	20	62
2-MRs	*toDegrees*	*signum_f*	*expm1*	*floor*	*gE_d*	*gE_f*	*log*	*log10*	*log1p*	*nextUp_d*	*nextUp_f*	*rint*	*atanh*
Apache	0	401	26	58	42	56	4	18	6	5	5	48	0
JDK	0	395	30	47	51	57	2	15	1	8	3	50	-
MATLAB	-	-	-	24	-	-	2	10	4	-	-	-	6
2-MRs	*round_f*	*signum_d*	*exp*	*sin*	*sinh*	*sqrt*	*tan*	*tanh*	*cosh*	*toRadians*	*ulp_d*	*ulp_f*	
Apache	38	391	31	131	0	0	8	80	0	303	214	246	
JDK	47	394	26	109	0	2	8	73	0	299	218	225	
MATLAB	-	-	9	78	0	7	6	58	0	-	-	-	

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 π 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 $cos^2(x) + cos^2(-x - 0.5\pi) - 1 = 0$, $cos^2(-0.5x - 1.5\pi) + 0.5cos(x) - 0.5 = 0$, $cos^2(x) - cos^2(x + \pi) = 0$, and $cos^2(x) + cos^2(-x) - 2cos(x)cos(-x) = 0$, 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 $tan^2(x) - 2tan(2x - 3\pi/2)tan(x) - 1 = 0$ 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.

Table 5: Fault-detection capability of MRs

	Seeded Faults	#By 1-MRs			#By 2-MRs		
		Total	FD	TD	Total	FD	TD
sin	17	9	0	9	9	0	9
cos	19	8	0	8	8	0	8
tan	18	8	0	8	8	0	8
log10	58	7	0	7	4	0	4
log1p	115	25	0	25	24	0	24
asinh	297	1	0	1	0	0	0
atan	94	15	0	15	31	0	31
abs_d	7	5	0	5	5	0	5
abs_f	7	5	0	5	5	0	5
abs_i	15	15	0	15	15	0	15
abs_l	15	15	0	15	15	0	15

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_i*, 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.

[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.

Table 6: Fault-detection capability of MRs without MR filtering

	Seeded Faults	#By 1-MRs			#By 2-MRs		
		Total	FD	TD	Total	FD	TD
sin	17	9	0	9	9	0	9
cos	19	9	0	9	19	11	8
tan	18	18	10	8	18	10	8
*log*10	58	58	51	7	58	54	4
*log*1*p*	115	29	0	29	113	85	28
asinh	297	297	292	5	4	0	4
atan	94	94	65	29	94	63	31
abs_d	7	7	2	5	5	0	5
abs_f	7	7	2	5	7	2	5
abs_i	15	15	0	15	15	0	15
abs_l	15	15	0	15	15	0	15

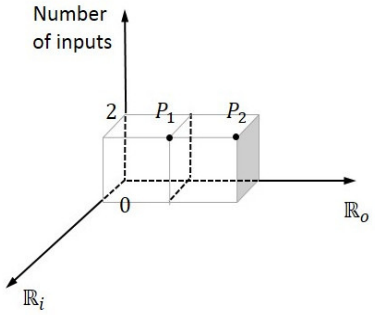

Figure 1: A 3-dimension solution space

7. DISCUSSION

7.1 Limitations

First, our approach requires the input of the program under test to be of numerical values. For example, our approach in the current stage may not be suitable for programs whose inputs are pointers or arrays. Therefore, we may improve the existing PSO algorithms by transforming these inputs into numerical values.

Second, our approach requires that the program under test should always produce the same output for the same input. That is to say, the behavior of the program under test should not depend on some external state. Thus, our approach may not be suitable for testing a program involving a database or a method in an object-oriented program relying on the state of the object.

7.2 Extensions

According to Chen et al. [14], an MR are supposed to hold among multiple executions. Based on this, we can change our Formula 3 to a more general form depicted in Formula 8.

$$\mathbb{R}_i(I_1, I_2, ...I_m) \Rightarrow \mathbb{R}_o(O_1, O_2, ...O_m) \quad (8)$$

Similar to the treatment presented in Section 3, we can also parameterize this generalized definition of MRs by confining $\mathbb{R}_i$ and $\mathbb{R}_o$ to be polynomial equations. In fact, we can further relax the requirement of polynomial equations to polynomial inequalities.

Therefore, the whole solution space of the extension can be depicted in Figure 1, where P_1 and P_2 represent the two cases our approach has solved. In this solution space, there are three major directions for further extending our approach.

First, the $\mathbb{R}_i$ 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 $\mathbb{R}_i$ may help infer "$log10(x^2) = 2log10(x)$" for the $log10$ function.

Second, the $\mathbb{R}_o$ 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 $\mathbb{R}_o$ may help infer "$sin(3x) = 3sin(x) - 4sin^3(x)$" for sin.

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

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 $log10(x)$, if x_1 is larger than x_2, $log10(x_1)$ is larger than $log10(x_2)$. 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

[1] E. Alba and F. Chicano. Management of software projects with gas. In *Proc. MIC*, pages 13–18, 2005.

[2] J. H. Andrews, L. C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proc. ICSE*, pages 402–411, 2005.

[3] J. H. Andrews, T. Menzies, and F. C. Li. Genetic algorithms for randomized unit testing. *IEEE Transactions on Software Engineering*, 37(1):80–94, 2011.

[4] A. Baars, M. Harman, Y. Hassoun, K. Lakhotia, P. McMinn, P. Tonella, and T. Vos. Symbolic search-based testing. In *Proc. ASE*, pages 53–62, 2011.

[5] E. T. Barr, T. Vo, V. Le, and Z. Su. Automatic detection of floating-point exceptions. In *Proc. POPL*, pages 549–560, 2013.

[6] S. Bensalem, M. Bozga, B. Boyer, and A. Legay. Incremental generation of linear invariants for component-based systems. In *Proc. ACSD*, pages 80–89, 2013.

[7] F. Benz, A. Hildebrandt, and S. Hack. A dynamic program analysis to find floating-point accuracy problems. In *Proc. PLDI*, pages 453–462, 2012.

[8] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *Proc. FOSE*, pages 85–103, 2007.

[9] J. Cabot, R. Clarisó, E. Guerra, and J. De Lara. An invariant-based method for the analysis of declarative model-to-model transformations. In *Proc. MODELS*, pages 37–52, 2008.

[10] G. Canfora and M. Di Penta. New frontiers of reverse engineering. In *Proc. FOSE*, pages 326–341, 2007.

[11] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: A new approach for generating next test cases. Technical Report HKUST-CS98-01, Hong Kong University of Science and Technology, 1998.

[12] T. Y. Chen, J. Feng, and T. H. Tse. Metamorphic testing of programs on partial differential equations: A case study. In *Proc. COMPSAC*, pages 327–333, 2002.

[13] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou. Case studies on the selection of useful relations in metamorphic testing. In *Proc. JIISIC*, pages 569–583, 2004.

[14] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Q. Zhou. Metamorphic testing and beyond. In *Proc. STEP*, pages 94–100, 2003.

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

[16] C. Csallner, N. Tillmann, and Y. Smaragdakis. Dysy: Dynamic symbolic execution for invariant inference. In *Proc. ICSE*, pages 281–290, 2008.

[17] E. Di Nitto, M. Di Penta, A. Gambi, G. Ripa, and M. L. Villani. Negotiation of service level agreements: An architecture and a search-based approach. In *Proc. ICSOC*, pages 295–306, 2007.

[18] M. Di Penta, M. Harman, and G. Antoniol. The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study. *Software: Practice and Experience*, 41(5):495–519, 2011.

[19] J. J. Dolado. A validation of the component-based method for software size estimation. *IEEE Transactions on Software Engineering*, 26(10):1006–1021, 2000.

[20] M. D. Ernst. *Dynamically discovering likely program invariants*. PhD thesis, University of Washington, 2000.

[21] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao. The Daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1):35–45, 2007.

[22] I. M. Gelfand and M. Saul. *Trigonometry*. Springer, 2001.

[23] P. Godefroid and J. Kinder. Proving memory safety of floating-point computations by combining static and dynamic program analysis. In *Proc. ISSTA*, pages 1–12, 2010.

[24] F. Gross, G. Fraser, and A. Zeller. Search-based system testing: high coverage, no false alarms. In *Proc. ISSTA*, pages 67–77, 2012.

[25] M. Harman. The current state and future of search based software engineering. In *Proc. FOSE*, pages 342–357, 2007.

[26] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[27] M. Harman, J. Krinke, J. Ren, and S. Yoo. Search based data sensitivity analysis applied to requirement engineering. In *Proc. GECCO*, pages 1681–1688, 2009.

[28] M. Harman, P. McMinn, M. Shahbaz, and S. Yoo. A comprehensive survey of trends in oracles for software testing. Technical Report CS-13-01, University of Sheffield, 2013.

[29] M. Harman and L. Tratt. Pareto optimal search based refactoring at the design level. In *Proc. GECCO*, pages 1106–1113, 2007.

[30] S. Hayashi, Y. Tsuda, and M. Saeki. Search-based refactoring detection from source code revisions. *IEICE Transactions on Information and Systems*, 93(4):754–762, 2010.

[31] S. Huang, M. B. Cohen, and A. M. Memon. Repairing GUI test suites using a genetic algorithm. In *Proc. ICST*, pages 245–254, 2010.

[32] G. Jiang, H. Chen, and K. Yoshihira. Discovering likely invariants of distributed transaction systems for autonomic system management. In *Proc. ICAC*, pages 199–208, 2006.

[33] U. Kanewala and J. M. Bieman. Using machine learning techniques to detect metamorphic relations for program without test oracles. In *Proc. ISSRE*, pages 1–10, 2013.

[34] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Encyclopedia of Machine Learning*, pages 1942–1948, 1995.

[35] I. Krka, Y. Brun, D. Popescu, J. Garcia, and N. Medvidovic. Using dynamic execution traces and program invariants to enhance behavioral model inference. In *Proc. ICSE*, pages 179–182, 2010.

[36] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295, 2006.

[37] W. Lin, M. Wu, Z. Yang, and Z. Zeng. Exact safety verification of hybrid systems using sums-of-squares representation. *Science China Information Sciences*, 57(5):1–13, 2014.

[38] H. Liu, X. Liu, and T. Y. Chen. A new method for constructing metamorphic relations. In *Proc. QSIC*, pages 59–68, 2012.

[39] M. T. Llano, A. Ireland, and A. Pease. Discovery of invariants through automated theory formation. *Formal Aspects of Computing*, 26(2):203–249, 2014.

[40] D. Lo and S. Maoz. Mining scenario-based specifications with value-based invariants. In *Proc. OOPSLA*, pages 755–756, 2009.

[41] R. E. Lopez-Herrejon and A. Egyed. Sbse4vm: Search based software engineering for variability management. In *Proc. CSMR*, pages 441–444, 2013.

[42] M. Z. Malik, A. Pervaiz, and S. Khurshid. Generating representation invariants of structurally complex data. In *Proc. TACAS*, pages 34–49, 2007.

[43] N. Mansour, R. Bahsoon, and G. Baradhi. Empirical comparison of regression test selection algorithms. *Journal of Systems and Software*, 57(1):79–90, 2001.

[44] J. Mayer and R. Guderlei. An empirical study on the selection of good metamorphic relations. In *Proc. COMPSAC*, pages 475–484, 2006.

[45] J. Mayer and R. Guderlei. An empirical study on the selection of good metamorphic relations. In *Proc. COMPSAC*, pages 475–484, 2006.

[46] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, and

G. Rothermel. A static approach to prioritizing junit test cases. *IEEE Transactions on Software Engineering*, 38(6):1258–1275, 2012.

[47] C. Murphy. *Metamorphic testing techniques to detect defects in applications without test oracles*. PhD thesis, Columbia University, 2010.

[48] C. Murphy, K. Shen, and G. Kaiser. Automatic system testing of programs without test oracles. In *Proc. ISSTA*, pages 189–200, 2009.

[49] T. Nguyen, D. Kapur, W. Weimer, and S. Forrest. Using dynamic analysis to discover polynomial and array invariants. In *Proc. ICSE*, pages 683–693, 2012.

[50] T. Nguyen, D. Kapur, W. Weimer, and S. Forrest. Using dynamic analysis to generate disjunctive invariants. In *Proc. ICSE*, pages 608–619, 2014.

[51] N. Pan, F. Zeng, and Y.-H. Huang. Test case reduction based on program invariant and genetic algorithm. In *Proc. WiCOM*, pages 1–5, 2010.

[52] C. S. Păsăreanu and W. Visser. Verification of java programs using symbolic execution and invariant generation. In *Proc. SPIN*, pages 164–181, 2004.

[53] Y. Pavlov and G. Fraser. Semi-automatic search-based test generation. In *Proc. ICST*, pages 777–784, 2012.

[54] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

[55] F. Qayum and R. Heckel. Search-based refactoring using unfolding of graph transformation systems. *Electronic Communications of the EASST*, 38, 2011.

[56] D. Romano, M. Di Penta, and G. Antoniol. An approach for search based testing of null pointer exceptions. In *Proc. ICST*, pages 160–169, 2011.

[57] C. Ryan. *Automatic re-engineering of software using genetic programming*, volume 2. Springer, 2000.

[58] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated test data generation on the analyses of feature models: a metamorphic testing approach. In *Proc. ICST*, pages 35–44, 2010.

[59] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. Automated metamorphic testing on the analyses of feature models. *Information and Software Technology*, 53(3):245–258, 2011.

[60] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Proc. CEC*, pages 1945–1950, 1999.

[61] Y. Shi, H. Liu, L. Gao, and G. Zhang. Cellular particle swarm optimization. *Information Sciences*, 181(20):4460–4493, 2011.

[62] B. H. Smith and L. Williams. On guiding the augmentation of an automated test suite via mutation analysis. *Empirical Software Engineering*, 14(3):341–369, 2009.

[63] E. Tang, E. Barr, X. Li, and Z. Su. Perturbing numerical calculations for statistical analysis of floating-point program (in)stability. In *Proc. ISSTA*, pages 131–142, 2010.

[64] P. Tonella, A. Susi, and F. Palma. Interactive requirements prioritization using a genetic algorithm. *Information and Software Technology*, 55(1):173–187, 2013.

[65] S. Wang, D. Lo, L. Jiang, and H. C. Lau. Search-based fault localization. In *Proc. ASE*, pages 556–559, 2011.

[66] E. J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25(4):465–470, 1982.

[67] A. Windisch, S. Wappler, and J. Wegener. Applying particle swarm optimization to software testing. In *Proc. GECCO*, pages 1121–1128, 2007.

[68] W.K.Chan, S.C.Cheung, and K. R.P.H.Leung. Towards a metamorphic testing methodology for service-oriented software applications. In *Proc. QSIC*, pages 470–476, 2005.

[69] W. E. Wong, J. R. Horgan, S. London, and H. Agrawal. A study of effective regression testing in practice. In *Proc. ISSRE*, pages 264–274, 1997.

[70] P. Wu. Iterative metamorphic testing. In *Proc. COMPSAC*, pages 19–24, 2005.

[71] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen. Application of metamorphic testing to supervised classifiers. In *Proc. QSIC*, pages 135–144, 2009.

[72] Z. Xu, M. B. Cohen, and G. Rothermel. Factors affecting the use of genetic algorithms in test suite augmentation. In *Proc. GECCO*, pages 1365–1372, 2010.

[73] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das. Perracotta: mining temporal API rules from imperfect traces. In *Proc. of ICSE*, pages 282–291, 2006.

[74] S. Yoo, M. Harman, and S. Ur. GPGPU test suite minimisation: search based software engineering performance improvement using graphics cards. *Empirical Software Engineering*, 18(3):550–593, 2013.

[75] Y. Yuan, Z. Fanping, Z. Guanmiao, D. Chaoqiang, and X. Neng. Test case generation based on program invariant and adaptive random algorithm. In *Proc. CSE*, pages 274–282, 2011.

[76] A. Zeller. Search-based program analysis. In *Proc. SSBSE*, pages 1–4, 2011.

[77] L. Zhang, D. Hao, L. Zhang, G. Rothermel, and H. Mei. Bridging the gap between the total and additional test-case prioritization strategies. In *Proc. ICSE*, pages 192–201. IEEE, 2013.

[78] L. Zhang, D. Marinov, L. Zhang, and S. Khurshid. Regression mutation testing. In *Proc. ISSTA*, pages 331–341, 2012.

[79] L. Zhang, G. Yang, N. Rungta, S. Person, and S. Khurshid. Feedback-driven dynamic invariant discovery. In *Proc. ISSTA*, pages 362–372, 2014.

[80] J. Zhao, C. Han, and B. Wei. Binary particle swarm optimization with multiple evolutionary strategies. *Science China Information Sciences*, 55(11):2485–2494, 2012.

[81] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei. MAPO: Mining and recommending API usage patterns. In *Proc. of ECOOP*, pages 318–343, 2009.

[82] H. Zhong, L. Zhang, T. Xie, and H. Mei. Inferring resource specifications from natural language API documentation. In *Proc. of ASE*, pages 307–318, 2009.

[83] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen. Metamorphic testing and its applications. In *Proc. ISFST*, pages 346–351, 2004.