Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 14, no. 6, June 2018, pp. 1330-1340
DOI: 10.23940/ijpe.18.06.p25.13301340

Input Domain Reduction of Search-based Structural Test Data Generation using Interval Arithmetic

Xuewei Lv[a,b], Song Huang[a,*], and Haijin Ji[a,b]

[a]*Command and Control Engineering College, Army Engineering University of PLA, Nanjing, 210074, China*
[b]*School of Computer Science and Technology, Huaiyin Normal University, Nanjing, 210074, China*

Abstract

The size of search space has an impact on the efficiency of test data generation by meta-heuristic algorithms. To enhance the efficiency of test data generation, a method that reduces search space utilizing interval arithmetic is proposed. Firstly, all input variables of the program are presented as interval variables. Then, the interval of each variable is gradually reduced by many constraint conditions in the target path. Finally, meta-heuristic algorithm with reduced search space is carried out to generate test data. Experimental results show the proposed method has advantages in the number of generations, running time, and success rate, which can significantly enhance the efficiency of test data by using meta-heuristic algorithms.

Keywords: search space reduction; interval arithmetic; meta-heuristic algorithm; test data generation

(Submitted on March 7, 2018; Revised on April 4, 2018; Accepted on May 8, 2018)

1. Introduction

With the rapid growth of software scale and complexity, software quality problems have become more and more prominent. Software testing has been a necessary part of software development to assure software quality. The report of NIST [1] shows that software testing can cut down one-third of software failure cost. The quality of test data has a great effect on the effectiveness and efficiency of software testing, and how to generate high-quality test data is one of the important issues in software testing. Recently, many researchers have applied many meta-heuristic algorithms (such as hill climbing (HC), genetic algorithm (GA), and ant colony optimization (ACO)) to automatically generate test data to meet specific coverage criterion (such as condition coverage and path coverage), which have achieved good effect. Because path testing has a stronger ability to detect faults, it has attracted the attention of many researchers. When meta-heuristic algorithms are applied to generate test data that cover the target path, the input domain of program under testing (PUT) is usually used as the search space of the meta-heuristic algorithms. The size of the input domain affects the efficiency of test data generation by using meta-heuristic algorithm. Hence, the input domain reduction can improve test data generation efficiency. By now, some researchers have proposed some different methods to reduce the input domain. However, the existing methods reduce input domain by deleting or fixing irrelevant variable, as far as by we know there are little studies that directly reduce input variable domain relevant with target path by static analysis.

Particle swarm optimization (PSO) algorithm is an emerging meta-heuristic algorithm. Since the PSO algorithm is easy to understand and implement, it is proverbially applied to generate test data. In this study, we take the PSO algorithm as an example to study search space reduction of meta-heuristic algorithms to make test data generation efficiency better. When PSO algorithm is utilized to generate test data covering the target path, we introduce interval arithmetic to reduce input domain of PUT. The input variables of the program are presented as interval variables, and then the input variable intervals are reduced by many constraint conditions in the target path. Finally, the reduced input domain is used as the search space of the PSO algorithm to improve the efficiency of test data generation.

* Corresponding author.
E-mail address: hs0317@163.com

This paper presented that how to apply interval arithmetic to reduce input domain. Section 2 provides concepts of PSO and interval arithmetic. Section 3 describes the related work. Test data generation based reduced input domain is presented in Section 4. Section 5 presents experimental evaluations of the proposed algorithm. Section 6 analyzes threats to validity. Section 7 concludes the paper and highlights directions for future research.

2. Backgrounds

2.1. Particle Swarm Optimization

The PSO algorithm proposed by Kennedy and Eberhart mimics the swarm behavior of insects, herds, birds and schools of fish [2]. The swarms search for food in a cooperative manner. Each member of the swarm changes its search position and velocity by learning own experience and other members' experience. In the PSO algorithm, each solution is regarded as a particle that moves in the search space to get a better position. Initially, each particle is placed in a random position, and each particle moves in the search space. Each particle's current position, velocity and individual best position are recorded. At the same time, the best position of all particles is recorded. And then particles update their positions and velocity by tracing their individual best position and the best position of all particles. Supposing a D-dimensional search, the update rules are described as follows:

$$V_i(t) = \omega V_i(t-1) + c_1 r_1 (pbest_i - X_i(t-1)) + c_2 r_2 (gbest - X_i(t-1)) \quad (1)$$

$$X_i(t) = X_i(t-1) + V_i(t) \quad (2)$$

Where V_i refers to the position of the i^{th} particle; X_i refers to the velocity of the i^{th} particle; t is the iteration number; $pbest_i$ is the personal best position that the i^{th} particle has obtained; $gbest$ is the best position that all particles have obtained; c_1 represents the i^{th} particle learning ability to $pbest_i$; c_2 represents the i^{th} individual particle learning ability to $gbest$. ω is the inertia weight, which presents the impact of previous velocity on current velocity.

2.2. Interval Arithmetic

Interval arithmetic proposed by Moore in the 1960s is mainly used to solve the calculation of reliable limits in the numerical analysis [3]. However, it is widely applied in the field of physics and economics. The definition and operations of the interval are as follows:

Definition 1 Interval [3]

For any given number $\underline{x}, \bar{x} \in R$, *if* $\underline{x} \le \bar{x}$, the bounded set $X = [\underline{x}, \bar{x}] = \{x \in R \mid \underline{x} \le x \le \bar{x}\}$ is called bounded closed interval. $\underline{x}$ is called the lower limit of the interval and $\bar{x}$ is called the upper limit of the interval.

Definition 2 Interval operation

For any given two intervals: $X = [\underline{x}, \bar{x}], Y = [\underline{y}, \bar{y}]$, basic interval operations are defined to be the following sets:

$X + Y = [\underline{x} + \underline{y}, \bar{x} + \bar{y}]$; $X - Y = [\underline{x} - \bar{y}, \bar{x} - \underline{y}]$

$X \times Y = [\min(\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y}), \max(\underline{x}\underline{y}, \underline{x}\bar{y}, \bar{x}\underline{y}, \bar{x}\bar{y})]$; $X / Y = [\underline{x}, \bar{x}] \times [\frac{1}{\bar{y}}, \frac{1}{\underline{y}}], 0 \notin [\underline{y}, \bar{y}]$;

$X \cap Y = [\max(\underline{x}, \underline{y}), \min(\bar{x}, \bar{y})]$, *if* X and Y have an intersection;

$X \cup Y = [\min(\underline{x}, \underline{y}), \max(\bar{x}, \bar{y})]$, *if* $X \cap Y \neq \varnothing$;

if $\bar{x} < \underline{y}$, *then* $X < Y$

Common numeric type variables of computer programs (Integer, Float, Character, etc.) can be presented as interval variables. The numerical test data are our study object.

3. Related Work

Since the numerical maximization techniques were applied to generate floating point test data by Miller and Spooner, more and more meta-heuristic algorithms and their improvements have already been used to generate test data.

Mei and Wang [4] combined bee colony characteristics and GA to generate test data to cover selected paths. In the proposed method, the best chromosome is called the "queen". According to a certain crossover probability, some drones are selected. These selected drones are used to cross with the queen. The method improves the global exploitation ability of the search. Several groups of experiments were carried to validate the effectiveness of the proposed method. The results proved this method is superior to similar methods.

Biswas *et al*. [5] used ACO to generate optimal paths and test data. The optimal paths covered the whole software have minimum redundancy. The testing order of optimal paths is determined based on their priorities. Then, ACO was used to generate test data sequence to use as the inputs for the paths generated previously.

Li and Zhang [6] applied the PSO algorithm to generate test data for all path coverage. This method adopted a new fitness function and recorded the frequencies for all paths. Experiment results showed that the proposed method had better efficiency than single-path test data generation.

In [7], negative selection algorithm (NSA) has been applied to generate test data. The proposed algorithm has been used to triangle classifier program. The experimental results presented that the proposed algorithm is more efficient and more effective than random method and genetic algorithm for test data generation. Sharma and Saha [8] applied the firefly algorithm to generate test cases of object-oriented programs. The experiment results presented that the test data generated using the firefly algorithm are not redundant, and the test data generated by the ACO algorithm are highly redundant. Therefore, firefly algorithm can generate more reduced test data compared with the ACO algorithm.

Except for improving existing meta-heuristic algorithms and proposing new meta-heuristic algorithms to automatically generate test data, many researchers have studied how to reduce the input domain of PUT to further enhance the efficiency of test data generation.

McMinn *et al.* [9] studied the relationship between the input domain size and the performance of generating test data using meta-heuristic algorithms. In their study, the irrelevant variables from testing targets were removed for reducing the input domain. The theory analysis was carried out, and a large of experiments proved that the domain reduction could enhance the performance of search-based test data generation.

Arcuri *et al.* [10] introduced a method for reducing the search space for Object-Oriented programs. This method dynamically removes the functions which cannot give any further help in the search. When there are lots of read-only functions under test, the method reduces the number of steps to achieve global optimality in case of assuring search quality. In the study, this method has double the speed of the local search.

Ribeiro *et al.* [11] proposed a novel input domain reduction method for Object-Oriented test data generation. The method is based on the concept of purity analysis, which determines and eliminates the irrelevant variables from test data generation. The experimental results proved the effectiveness of the method.

Zhang *et al.* [12] proposed a method based on the automatic reduction of input domain to evolutionarily generate test data for path coverage. The input sub-variables related to independent sub-paths of the target path are fixed. In the following evolution, these sub-variables do not carry out crossover and mutation operations. Therefore, the original population searches for test data in a reduced space so that the efficiency of test data generation can be improved. However, the method only has effectiveness on the target path that can be divided into independent sub-path.

These methods are based on irrelevant variable removal strategy, their reduction objects are variables irrelevant to the test target. By removing these variables, the number of variables becomes less and the dimension of input domain will decrease. Therefore, the input domain is reduced. In this paper, we proposed an innovative input domain reduction method by interval arithmetic. Different from existing methods, the proposed method reduces the domain of variable relevant to test objects and does not reduce the number of variables. The upper and lower limits of each relevant variable are reduced, resulting in input domain reduction.

4. Test Data Generation based on Reduced Input Domain

4.1. Fitness Function Construction

The fitness function construction is the key step in automatically generating test data by using meta-heuristic algorithms. By applying the fitness function, test data generation in software testing can be regarded as an optimization problem. The fitness function has a significant impact on the performance of test data generation by meta-heuristic algorithms. Until now, researchers have proposed several definitions of the fitness function, including branch distance and approval level. The combination of approach level and branch distance is widely used. Therefore, the combined fitness function is used in the study.

The approach level (*AL*) indicates how much the program execution deviates from the test target. The approach level is computed based on the number of branch predicates that have different predicate value in program executed path and the target path (called mismatch branch predicate). Because the branch predicates in a path form a predicate sequence, the backer the mismatch branch predicate in the sequence is, the closer the executed path and the target path are. Therefore, considering the position of mismatched branch predicate in the sequence, the definition is follows as:

$$AL(x,P)=\sum_{i=1}^{N}\omega\frac{L}{i} \tag{3}$$

Where x is the input; P is the target path; L refers to the length of the target path; N refers to the number of the branch predicates in the executed path of x and target path; i refers to the index number of the mismatched branch; ω is a weight value. Its value was set to L in the study.

The branch distance is calculated using the variable values at the mismatch branch predicate. For example, if the branch predicate is $x<5$ and the target is the true branch of this branch predicate. The branch distance of this branch predicate is defined by Formula 4

$$BD(x<5)=\begin{cases}0, & x<5\\ x-5, & others\end{cases} \tag{4}$$

The branch distance different branch predicates are calculated according to Table 1.

Table1. Fitness distance computation of different branch predicates [13]

Predicate	Branch distance computation
Boolean	*If* True *then* 0 *else* K
$a>b$	*If* $b-a<0$ *then* 0 *else* $(b-a)+K$
$a\geq b$	*If* $b-a\leq 0$ *then* 0 *else* $(b-a)+K$
$a<b$	*If* $a-b<0$ *then* 0 *else* $(a-b)+K$
$a\leq b$	*If* $a-b\leq 0$ *then* 0 *else* $(a-b)+K$
$a=b$	*If* $abs(a-b)=0$ *then* 0 *else* $abs(a-b)+K$
$a\wedge b$	$BD(a)+BD(b)$
$a\vee b$	$min(BD(a),BD(b))$

The branch distance of the input x and the target path P is calculated by summing the branch distance of all branched predicates both in the executed path of x and the target path P. Therefore, it is defined as follows:

$$BD(x,P)=\sum_{i=1}^{N}BD(predicate_i) \tag{5}$$

Where $BD(x,P)$ is the branch distance of the input x and the target path P, $BD(predicate_i)$ represents the branch distance of the i^{th} branch predicate, N refers to the number of all branch predicates included in the executed path of x and the target path P; i is the index value of branch predicates both in the executed path of x and the target path P.

We defined the general fitness function of input x to the target path P as follows:

$$F(x,P)=AL(x,P)+(1-1.001^{-BD(x,P)}) \tag{6}$$

In the second part of Formula 6, the branch distance $BD(x,P)$ is normalized.

By using this fitness function, the test data generation for path coverage is transformed into an optimization question. The smaller fitness function indicates the executed path of the input is closer to the target path. When the fitness function value is zero, the input must cover the target path.

4.2. Test data generation model based on reduced input domain

The test data generation model based on reduced input domain includes four parts: test environment construction, input domain reduction, PSO search and test execution. As shown in Figure 1.

(1) Test environment construction is the basis of the model. In this part, the source code is analyzed and instrumented, the fitness function is constructed, the target path is chosen and the parameters of the PSO algorithm are set.

(2) Input domain reduction is the innovation part of the model. This part performs domain reduction by interval arithmetic, interval combination and interval ordering. It reduces the search space of the PSO algorithm.

(3) PSO search is the important part of the model. The swarm is initialized and all particles' position and velocity are updated according to Formulas (1) and (2). The particle fitness value is calculated through executing the PUT.

(4) Test execution plays a bridge role in the model. During the execution, the particle positions as practical parameters are transferred into the instrumented program. By tracing the information of the branch, the fitness value is obtained.

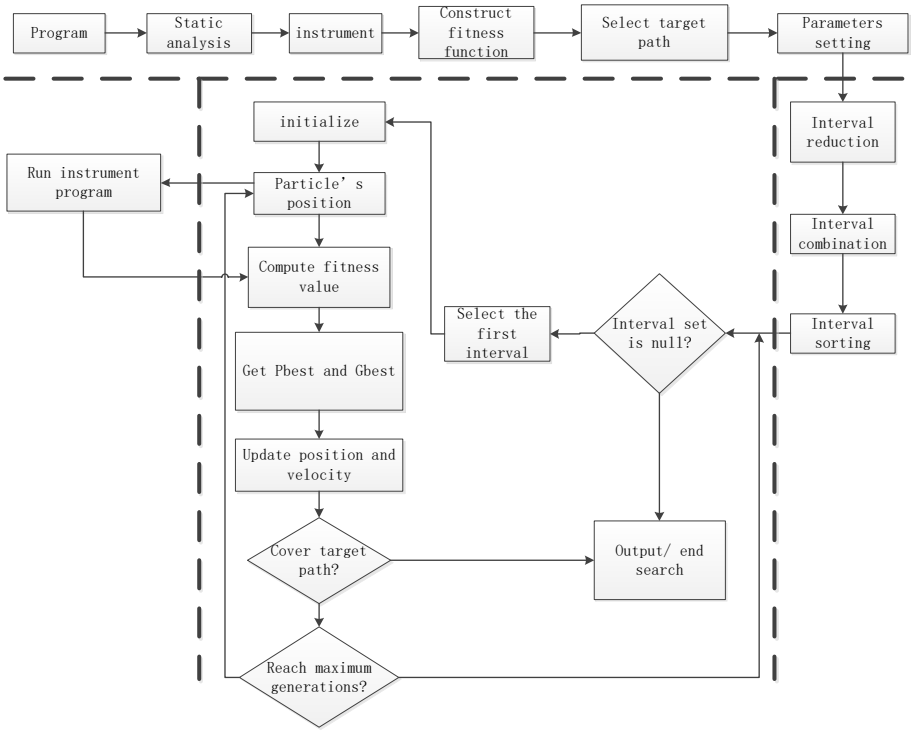

Figure 1. Test data generation model based on reduced input domain

4.2.1 Input domain reduction by interval arithmetic

To reduce input domain, all input variables are first presented as interval variables. When the interval variables are introduced into a conditional expression, the size of the variable intervals will be reduced to varying degrees due to the constraint relations of the condition expression. The interval reduction operation is defined as follows:

(1) For simple condition expression, we only consider that each side of the expression contains a single interval variable. Suppose that the interval variable $V=[\underline{v},\overline{v}]$, and another interval variable $C=[\underline{c},\overline{c}]$. When the result of the expression is true, the reduced interval of V can be seen in Table 2. l is the minimum variable precision. *MIN and MAX* are the minimum and maximum values in $\{\underline{v},\overline{v},\underline{c},\overline{c}\}$, respectively.

Table 2. Interval reduction operation of simple condition expression

Expression	Reduced V
$V < C$	$V \cap \{[MIN, \bar{c} - l\}$
$V \leq C$	$V \cap \{[MIN, \bar{c}\}$
$V > C$	$V \cap \{[\underline{c} + l, MAX]\}$
$V \geq C$	$V \cap \{[\underline{c}, MAX]\}$
$V = C$	$V \cap \{[\underline{c}, \bar{c}]\}$
$V \neq C$	V

(2) When the condition expression contains ‘and’, ‘or’, ‘not’ relations, the variable interval is calculated as follows: Supposed that V_X is the interval of the expression X, $V_{A||B} = V_A \cup V_B$; $V_{A\&\&B} = V_A \cap V_B$. The relation of ‘not’ can be translated into the relations of ‘and’ and ‘or’.

For path testing, a path consists of many branch nodes that include one or more conditional expression. These branch nodes are ordered. When the input variables are reduced by conditional expressions included in a branch node, the reduced variable intervals are used as the input variables of the next branch nodes. After being reduced by all branch nodes in the target path, the reduced intervals are used for the initial search space of the PSO algorithm to generate test data. The process is described in Algorithm 1.

Algorithm 1 The Input Domain Reduction of Input Variables

Input: the target path P, input variables set $vars$
Output: reduced intervals of input variables

```
initiate the intervals of all variables in vars;
for each branch node in P do
    if the node is assignment statement, and input variables are assigned then
        calculate the interval of the assignment expression according to the operations in Table 1, and the result is used as the interval of input variables ;
    end if
    if the node is condition statement, and input variables are in the condition expression then
        reduce the interval of each variable according to the operations in Table 1;
    end if
    if the node is case statement then
        the interval form of the constant value of case statement is assigned to the interval of variables of switch statement;
    end if
    if the node is default statement then
        the constant values of all case statements are subtracted from the variable intervals of switch statement;
    end if
    the reduced variable intervals are used as the input intervals of the same variables in the next node
end for
```

Figure 2 The algorithm of input domain reduction by interval arithmetic

4.2.2 Intervals combination

After being reduced, the input variable interval may be divided into many sub-intervals. When the spacing between two immediately adjacent intervals is very small, these two intervals can be combined to reduce the number of sub-intervals and computation consumption. For example, when two intervals are [1, 10] and [12,100] respectively, these two intervals could be combined into [1, 100]. Therefore, to decide when to combine two adjacent intervals, the definition of combination tendency is described as follows:

Definition 3 Assuming V_1, V_2 are the two adjacent intervals, combination tendency (*CT*) presents the possibility of combining two intervals. It is calculated according to Formula 7.

$$CT(V_1,V_2) = \frac{\overline{V_2} - \overline{V_1}}{\underline{V_2} - \underline{V_1}}, \quad (if\ V_2 > V_1) \tag{7}$$

Assuming β is a predefined threshold, when CT<β, the intervals V_1, V_2 are combined into an interval. In this study, β is equal to 0.01.

4.2.3 Intervals Sorting

When an input variable interval is divided into many sub-intervals, if there are many input variables, different sub-intervals of different input variable are combined to create a variable interval list. Each item of the list is a sub-domain of the input domain. To enhance search performance, we ordered these input sub-domains in accordance with their fitness values. The fitness value is calculated based on the middle value of the sub-interval of each input variable. The details of many variable sub-intervals ordering can be seen in Algorithm 2.

Algorithm 2 The Sorting of Multiple Intervals

Input: input variable intervals set $V[M][N]$, fitness function f
Output: the sorted input variable intervals set

1: $List$: combined list of different variable intervals(D) and the corresponding fitness function value($fitness$)
2: $List.D \leftarrow combine(V)$
3: **while** $List.D_i$ is not null **do**
4: **for** $j \rightarrow 1 : N$ **do**
5: $average[i][j] \leftarrow \frac{\overline{D_{ij}} + \underline{D_{ij}}}{2}$
6: **end for**
7: $List.fitness[i] \leftarrow f(average[i])$
8: **end while**
9: $SortedList \leftarrow quicksort(List, List.fitness)$
10: Return($SortedList.D$)

Figure 3. The algorithm of intervals sorting

The function *combine(V)* combines different sub-intervals of each variable into *List.D*. The fitness function(*List.fitness*) of the median of the sub-interval is calculated as a basis for sorting sub-intervals. *Quicksort* is utilized to sort the variable interval list *List.D* according to *List.fitness*

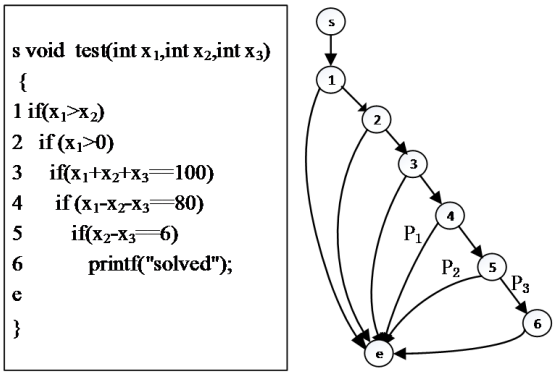

Figure 4. Test program and its control flow graph

4.3. Case Study

In order to better illustrate that interval arithmetic can effectively reduce the interval of the input variable, the test program shown in Figure 4 is taken as an example for description. Supposed the initial interval of all variables is [1, 100], and X_1, X_2 and X_3 denote the interval of the variables x_1, x_2 and x_3. *D* represents the input domain, the target path is P_3. The process of interval reduction operations can be seen in Figure 5. Because there are many variables, the variable alternate method is

used. When the interval of a variable is computed, other variables are regarded as constant. For example, the condition expression is $(x_2 - x_3 = 6)$, x_3 is regarded as constant, and the expression can be transformed into $(x_2 = 6 + x_3)$. According to the addition operation in Definition 2 and the formulas in Table 2, x_2 is computed. After that, x_2 is regarded as constant, and the interval of x_3 is computed.

5. Experiments and Analysis

To observe the effectiveness of the proposed approach, we carried out a large number of experiments. The experiments were performed in the environment of Windows 7 with Intel Core i5-4590 3.3GHz and 8GB RAM. The algorithms were implemented in C language by DEV C++ 5.11.

5.1. Subject Programs

To validate the proposed method, we chose eleven C programs as testing objects. Some basic information of each program are described in Table 3, including its name, the lines of code (LOC), the number of branches, the number of variables, the size of initial input domain, and description. These testing objects come from laboratory programs or industry programs. Also, their length and functions are different from each other. These programs have been applied by many researchers. After analyzing each program, we chose the path including the most condition branches as the target path.

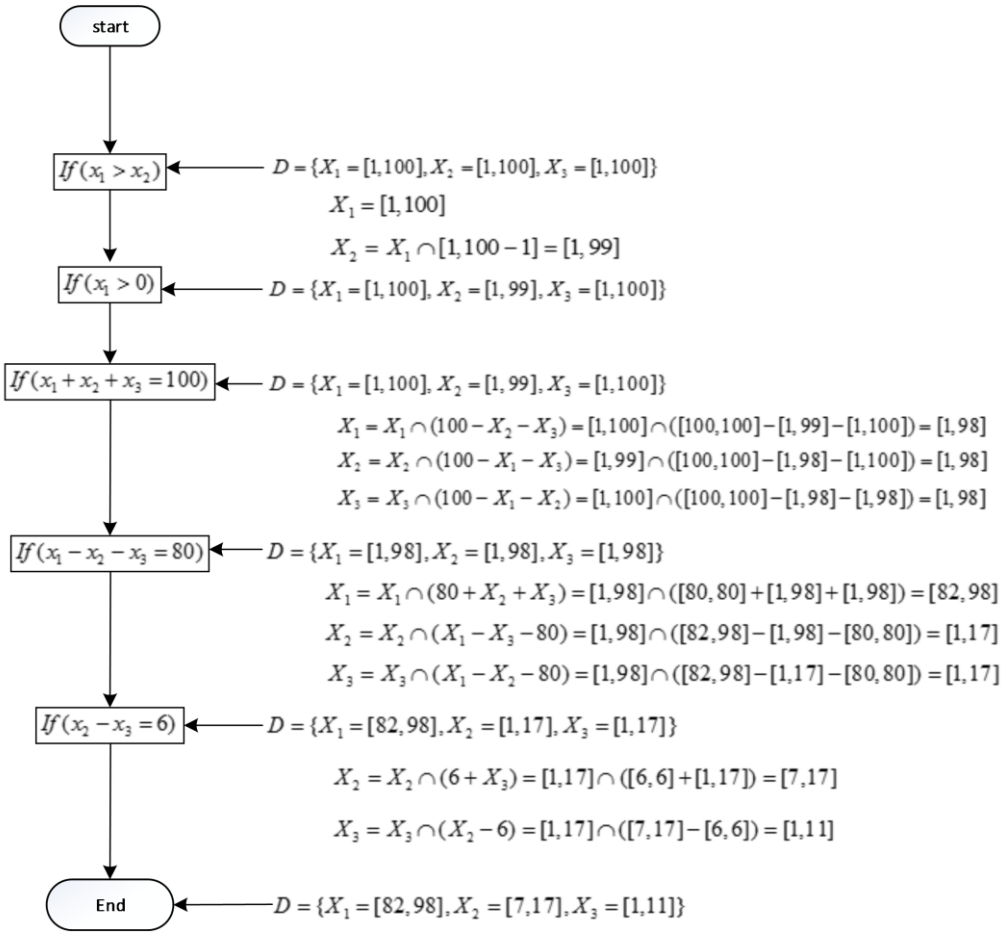

Figure 5. The process of interval reduction

5.2. PSO Parameters

When the experiments are conducted, the PSO parameter values must be set up beforehand. The number of particles at N=50 to improve the success rate of all PUTs as much as possible. c_1 and c_2 are set as 0.2 recommended in [14]. ω is set as 0.7. Maximum speed is set as 20. Maximum generation is 1000 in the experiments.

Table 3. The description of PUTs

Program	LOC	Number of branches	Number of variables	Initial domain	Description
Test	10	5	3	$[1,100]^3$	Described in Figure 4
Sumday	21	14	3	$[1,1000]^3$	Compute the given date is the number of days of the year
Days	48	28	6	$[1,100]^6$	Compute the number of day between two dates
Bonus	32	5	1	[1,1000]	Compute bonus according to given value
Equation	31	6	3	$[1,1000]^3$	Solve a Equation according to given parameters
asinl	71	5	1	[1,1000]	Solve the arcsine function
Statistics	21	8	5	$[1,1000]^5$	Compute statistics result according to given values
acos	45	4	1	[1,1000]	Solve the inverse cosine function
Star	17	6	3	$[1,1000]^3$	Calculate which value is marked with star
Deltat	163	9	1	[1,10000]	Compute the difference between Ephemeris and Universal Time
Nextdate	29	4	3	$[1,1000]^3$	Compute the next date of the given date

5.3. Experimental Design

When designing experiments, we paid particular attention to two issues that can be described as follows.

Proposition 1 Does the interval sorting has an impact on the domain reduction?

To validate the impact of sorting algorithm, two groups of experiments were conducted in the same reduced domain. The sorting algorithm is used in one group experiment, and not used in another group experiment.

Proposition 2 Can the input domain reduction can improve the performance of test data generation?

In order to verify the second proposition, we conducted two groups of experiments. In the first group of experiments, the PSO algorithm searches test data in the reduced domain. In the second group of experiment the PSO algorithm searches in the initial domain.

The test data search was repeated 1000 times and different seeds from random number were used each time. Three metrics were calculated according to the results. *AE* represents the average number of fitness evaluations accomplished to find test data. *TC* refers the total time consumption of 1000 runs to find the test data. *SR* refers to the number of successful coverage of the target path as a percentage of 1000 runs.

5.4. Experimental Results and Analysis

5.4.1 The Impact of Sorting Algorithm on the Efficiency of Input Domain Reduction

In this section, a group of experiments was conducted to evaluate the impact of the sorting algorithm. The results can be seen in Table 4. Sorted refers to the performance of PSO algorithm after interval reduction using the sorting algorithm. Unsorted refers to the performance of PSO algorithm after interval reduction not using the sorting algorithm.

Table 4 The impact of sorting algorithm on the efficiency of input domain reduction

Program	*AE*			*TC*(s)		
	Sorted	Unsorted	Rate	Sorted	Unsorted	Rate
Test	413	2911	14.19%	0.156	1.047	14.90%
Sumday	152	2700	5.63%	0.265	4.047	6.55%
Days	274	2821	9.71%	0.75	6.25	12.00%
Bonus	102	100	102%	0.031	0.031	100%
Equation	208	1860	11.18%	0.359	3.957	9.07%
asinl	102	100	102%	0.047	0.047	100%
Statistics	256	1860	13.76%	0.628	4.854	12.94%
acos	102	100	102%	0.032	0.032	100%
Star	163	1382	11.79%	0.238	2.253	10.56%
Deltat	102	100	102%	0.031	0.031	100%
Nextdate	212	2710	7.82%	0.328	3.719	8.82%

It can been seen from Table 4 that for some PUTs, the average fitness valuations and time consumption using sorting algorithm are less than that not using the sorting algorithm. For *test* program, the method using the sorting algorithm only needs 413 and 0.156s, but the method not using sorting algorithm needs 2911 and 1.047s, which is only 14.19% and 14.90% of the former. For *sumday* program, the method using sorting algorithm only needs 152 and 0.265s, but the method not using sorting algorithm needs 2700 and 4.047s, which is only 5.63% and 6.55% of the former. However, for some PUTs,

the sorting algorithm does not affect the performance of test data generation, even needs more fitness evaluation. For *bonus* program, the two methods both need 0.031s, but the method using sort algorithm needs more 2 fitness evaluations than the method not using the sorting algorithm. By analysis, we found that if PSO algorithm can find test data covering the target path in the first sub-interval, the sorting algorithm does not work, and because the sorting algorithm calls fitness function, it needs more fitness evaluations. However, if the test data covering the target path is not in the first sub-interval, PSO algorithm can first search the sub-interval that is most likely to contain test data by using sorting algorithm. To conclude, sorting algorithm has significant a impact on the performance of interval reduction.

5.4.2 The Impact of Input Domain Reduction on Performance of Test Data Generation

In this section, a group of experiments was conducted to evaluate the performance of input domain reduction. The results can be seen in Table 5. *Proposed method* refers to the performance of PSO algorithm using reduced space, while *Traditional method* refers to the performance of PSO algorithm using unreduced space.

Table 5. The impact of input domain reduction on the performance of test data generation

Program	AE		TC(s)		SR(%)	
	Proposed method	Traditional method	Proposed method	Traditional method	Proposed method	Traditional method
Test	413	37177	0.156	10.125	100	30.2
Sumday	152	3181	0.265	4.89	100	98.7
Days	274	32241	0.75	66.188	100	43.1
Bonus	102	1264	0.031	0.328	100	97.7
Equation	208	28546	0.359	6.882	100	61.1
asinl	102	48901	0.047	33.609	100	2.3
Statistics	256	29851	0.628	48.682	100	52.8
acos	102	48890	0.032	10.609	100	2.3
Star	163	2982	0.238	4.126	100	95.9
Deltat	102	4730	0.031	2.047	100	100
Nextdate	212	46707	0.328	60.75	100	7.6

Table 5 shows that 1) for all programs the proposed method can reduce the numbers of fitness evaluation and time consumption. The least fitness evaluation of the proposed method is 102(*asinl*), while that of the traditional method for the same program is 48901. The largest fitness evaluation of the proposed method is 413(*test*) while that of traditional method is 37177. The least time consuming of the proposed method is 0.031s(*bonus* and *deltat*), while that of the traditional method for the same programs are 0.328s and 2.047s. The largest time consuming of the proposed method is 0.75s(*days*) while that of traditional method is 66.188s. 2) the proposed method can greatly improve the success rate. For all programs the success rate of our method is 100%, while that of traditional method is from 2.3% to 100% for different programs. In conclusion, the results provide evidence that input domain reduction is an effective method that enhances the efficiency and effectiveness of test data generation using meta-heuristic algorithms.

6. Threats to Validity

The study focuses on how to reduce the input domain of search-based test data generation by interval arithmetic. One possible threat is the setting of the parameters of meta-heuristic algorithms. Researches showed the settings of parameters have a significant impact on the performance of meta-heuristic algorithms. Therefore, the settings of parameters influence the performance of test data generation for path coverage. However, how to set appropriate parameters is not the focus of the study. In the study, the values of parameters were set according to our experience. Another possible threat is the selection of PUTs. Although the PUTs in the study come from open sources or other literature, their lines of code are relatively small. Therefore, we need some relatively large PUTs to observe the effectiveness of the proposed method.

7. Conclusions

The approach applying meta-heuristic algorithms to generate test data has aroused widespread concern, but its efficiency needs to be further improved. The size of search space of the meta-heuristic algorithm affects the efficiency of test data generation. In this paper, the search space of meta-heuristic algorithm (PSO algorithm) is reduced by interval arithmetic, which improves the efficiency and success rate of test data generation. The experimental results demonstrate that input domain reduction has good effectiveness. However, since the strict interval arithmetic requires a large amount of storage space, how to decrease the search space by using the interval arithmetic without increasing a large amount of storage space is the research priority in the future.

Acknowledgments

This work is supported by the Natural Science Foundation of Jiangsu Province.

References

1. G. Tassey, "The economic impacts of inadequate infrastructure for software testing," National Institute of Standards and Technology, RTI Project, vol.7007, no.011, 2002.
2. R. C. Eberchart, J. Kennedy, "particle swarm optimization," in *Proceeding of IEEE International Conference on Neural Networks*, pp.1942–1948, Perth, Australia, 1995.
3. R. E. Moore, R.B. Kearfott, and M.J. Cloud, "Introduction to interval analysis," Siam, 2009.
4. J. Mei, S. Y. Wang, "An improved genetic algorithm for test cases generation oriented paths," *Chinese journal of electronics*, vol. 23, no. 3, pp. 494-498, 2014.
5. S. Biswas, M. S. Kaiser and S. A. Mamun, "Applying Ant Colony Optimization in software testing to generate prioritized optimal path and test data," in *Proceeding of Electrical Engineering and Information Communication Technology* (ICEEICT), pp. 1-6, IEEE, Khulna, Bangladesh, May 2015.
    A. Li, Y. Zhang, "Automatic generating all-path test data of a program based on PSO," in *Proceeding of Software Engineering*, pp. 189-193, IEEE, Xiamen, China, May 2009.
6. S. M. Mohi-Aldeen, R. Mohamad, and S. Deris, "Automated path testing using the negative selection algorithm, " *International Journal of Computational Vision and Robotics*, vol. 7, no. 1, pp. 160-171, 2017.
7. R. Sharma, A. Saha, "Optimization of object-oriented testing using firefly algorithm, " *Journal of Information and Optimization Sciences*, vol. 38, no. 6, pp. 873-893, 2017.
8. P. McMinn, M. Harman, K. Lakhotia, Y. Hassoun, and J. Wegener, "Input domain reduction through irrelevant variable removal and its effect on local, global, and hybrid search-based structural test data generation," *IEEE Transactions on Software Engineering*, vol. 38, no. 2, pp.453–477, 2012.
    A. Arcuri, X. A.Yao, " A memetic algorithm for test data generation of object oriented software," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation* (CEC), pp. 2048–2055, IEEE, Singapore, 2007.
9. J.C. B. Ribeiro, M.A. Zenha-Rela, and F.F. Vega, "Test case evaluation and input domain reduction strategies for the evolutionary testing of object-oriented software," *Information and Software Technology*, vol. 51, no. 11, pp. 1534–1548, 2009.
10. Y. Zhang, D. Gong, X. Yao, and Q. Lu, " Generating Test Data Covering Multiple Paths Using Genetic Algorithm Incorporating with Reducing Input Domain," in *Proceeding of Chinese Intelligent Systems Conference*, pp. 739-747, Springer, Singapore, October 2017.
11. N. Tracey, J. Clark, K. Mander, and J. McDermid , "An automated framework for structural test-data generation," in *Proceeding of Automated Software Engineering*, pp. 285-288, Hawaii, USA, 1998.
12. S. Jiang, J. Shi, Y. Zhang, and H. Han, "Automatic test data generation based on reduced adaptive particle swarm optimization algorithm, " *Neurocomputing*, vol. 158, pp. 109-116, 2015.