**Genetic algorithm for path testing using adjustment**

***Abstract***

**1. Introduction**

Testing is a critical but expensive part of the software development life cycle. There is considerable interest in waystoautomate testing, to reduce the cost and to gain more confidencein the result.A major task in software testing is test data generation.Search-based test data generation aims to automate this task, bysearching for test cases (inputs, or pairs of input–output) that satisfychosen testing criteria.

Most research in this area considers “white box” testing, or structural coverage, in which the aim is to ensure that executinga collection of test cases results in all parts of a program beingtested. This can be interpreted in various ways, including “statement coverage” (when the program is tested with all of the testcases, somewhere along the line every statement in the programis executed at least once), ‘‘branch coverage’’(both outcomes atevery logical branch in the program are executed at least once),and “path coverage”(every distinct path through the code is executedat least once). Path coverage is the strongest form of structural coverage [5]. This paper considers path coverage.

Many approaches have been used in path testing [cần dẫn nguồn]. Evolutionarypath testing, which uses an evolutionary algorithm (e.g. geneticalgorithm “GA”) as the search engine has been found effective[6,7]. In this research, GA is used as the search engine.

A challenge for any search-based approach is làm sao để sinh được những input test data mà có thể phủ được những test path có các path condition có xác suất phủ được là thấp.

Chúng tôi đề xuất một phương pháp sử dụng giải thuật di truyền để sinh ra input test data để phủ được các test path có xác suất phủ là thấp.

This paper is organized as follows: Section 2 presents some theoretical background to understanding this research. Section 3 describes related work, and Section 4 describes the proposed approach in detail. Section 5 presents the evaluation. Section 6 concludes the paper.

**2. Background**

*2.1. Path testing*

The objective of path testing is to search for a collection of testcases (inputs to a program) that between them lead to the traversalof all logical paths through the program.In general, path testing process consists of two major steps: target paths generation, and test data generation. Cả hai bước này đều được đề cập đến trong đề xuất của bài báo này.

*2.1.1. Target paths generation*

Target paths generation means identifying a set of logical executionpathways through the program, that we hope should allbe exercised during testing.The source code is needed to construct its logical control flow,which can be presented in a control flow graph (CFG). This graphcan be automatically generated by using appropriate programminglanguage grammar in which the program is written.

From the CFG, the different logical paths through the programneed to be enumerated. A logical path is a particular flow of executionthrough the program, which is determined by the decisionsmade at each decision point between the program’s entry pointand its exit point.

*2.1.2. Test data generation*

Generating test data that fulfill path coverage is the main task inpath testing. It is the process of creating test data, either heuristicallyor randomly. In a heuristic approach, the process is guided by some rules to search for required test data; the alternative is that random test data is generated.

*2.2. Generic algorithm*

The basic concepts of genetic algorithms (GAs) were developed by Holland [cần dẫn nguồn]. GAs are commonly applied to a variety of problems involving search andoptimisation. GAs search methods are rooted in the mechanisms of evolution andnatural genetics. GAs draw inspiration from the natural search and selection processesleading to the survival of the fittest individuals. GAs generate a sequence ofpopulations by using a selection mechanism, and use crossover and mutation as searchmechanisms.

The principle behind GAs is that they create and maintain a population ofindividuals represented by chromosomes (essentially a character string analogous tothe chromosomes appearing in DNA). These chromosomes are typically encoded solutions to a problem. The chromosomes then undergo a process of evolution according to rules of selection, mutation and reproduction.

Each individual in the environment (represented by a chromosome) receives ameasure of its fitness in the environment. Reproduction selects individuals with highfitness values in the population, and through crossover and mutation of suchindividuals, a new population is derived in which individuals may be even better fittedto their environment. The process of crossover involves two chromosomes swapping chunks of data (genetic information) and is analogous to the process of sexualreproduction. Mutation introduces slight changes into a small proportion of thepopulation and is representative of an evolutionary step. The structure of a simple GAis given below.

Simple Genetic Algorithm ()

{

initialize population;

evaluate population;

while termination criterion not reached

{

select solutions for next population;

perform crossover and mutation;

evaluate population;

}

}

The algorithm will iterate until the population has evolved to form a solution to the problem, or until a maximum number of iterations have taken place (suggesting that a solution is not going to be found given the resources available).

**3. Related work**

Basis path testing is one of the famous structural testing criteria. It is a methodology which searches the program domain for suitable test data, such that after executing the program with the test data, a predefined path is reached. Based on the cyclomatic complexity analysis [Thomas J. McCabe], each path is being tested for efficient functionality. Practically it is possible to apply path testing for a specific subset of paths in the control flow graph. This mechanism aims to compute the logical complexity of a procedural design and defines a set of execution paths. Test data are generated in such a way that they will execute every statement at least once.

**4. Proposed approach**

Xét function phân loại tam giác từ 3 tham số thể hiện chiều dài của 3 cạnh tam giác như sau:

int Tritype(int a, int b, int c)

{

inttrityp = 0;

if ((a + b > c) && (b + c > a) && (c + a > b))

{

if ((a != b) && (b != c) && (c != a))

trityp = 1; // Scalene

else

if (((a == b) && (b != c)) || ((b == c) && (c != a)) ||

((c == a) && (a != b)))

trityp = 2; // Isosceles

else

trityp = 3; // Equilateral

}

else

trityp = -1; // Not a triangle

return trityp;

}

CFG được sinh ra từ chương trình phân loại tam giác này là:



Tập target paths của được sinh ra từ CFG của function phân loại tam giác sẽ có 4 path như sau:

{[1,F]} // path 1 (Not Triangle)

{[1,T], [2,F], [3,F]} // path 2 (Equilateral)

{[1,T], [2,F], [3,T]} // path 3 (Isosceles)

{[1,T], [2,T]} // path4 (Scalence)

Giả sử rằng các input parameter a, b, c lấy giá trị nguyên dương trong khoảng a, b, c [1, 1000]. Mỗi tham số có thể lấy 1000 giá trị, cho nên không gian của input parameter sẽ là 10003. Do đó, có 1000 tam giác có các cạnh bằng nhau (1, 1, 1), (2, 2, 2)…(1000, 1000, 1000). Như vậy, xác suất để tạo ra một test data để có thể phủ được path 2 (Equilateral) là 1000/10003 = 0.000001.

Xác suất để có các phân loại tam giác khác cũng được thể hiện ở bảng sau:

|  |  |
| --- | --- |
| Classification | Probability |
| Path 4 (Scalence) | 0.4985015 |
| Path 3 (Isosceles) | 0.001998 |
| Path2 (Equilateral) | 0.000001 |
| Path 1 (Not Triangle) | 0.4994995 |

Nếu dùng giải thuật di truyền thông thường để sinh ra test data cho function phân loại tam giác này, thì chỉ sinh được test data cho hai path có xác suất cao để tạo được test data là path 4 (Scalence) và path 1 (Not Triangle).

Run No. 1 : Wait Please .........

Path 4: a = 14.744746849514955 b = 4.817243648636693 c = 12.301081756217084

Objective call: 1

Path 1: a = 7.768767568484256 b = 1.3747887684823534 c = 2.6602222750788584

Objective call: 3

Run No. 2 : Wait Please .........

Run No. 3 : Wait Please .........

Run No. 4 : Wait Please .........

Run No. 5 : Wait Please .........

Run No. 6 : Wait Please .........

Run No. 7 : Wait Please .........

Run No. 8 : Wait Please .........

Run No. 9 : Wait Please .........

Run No. 10 : Wait Please .........

Để có thể sinh ra được test data phủ được các path có xác suất được phủ thấp, chúng tôi đưa ra cách tiếp cận có hai bước chính như sau:

Step 1: Tìm các path mà có xác suất phủ được là thấp trong chương trình. Xác định path condition của các path khó được phủ này.

|  |  |  |
| --- | --- | --- |
| Classification | Probability | Path condition |
| Path 4 (Scalence) | 0.4985015 |  |
| Path 3 (Isosceles) | 0.001998 | a == b || b == c || c == a |
| Path2 (Equilateral) | 0.000001 | a == b == c |
| Path 1 (Not Triangle) | 0.4994995 |  |

Step 2: Thực hiện giải thuật di truyền để sinh test data.

***Representation***

Depend on type of input parametters, the proposed GA uses an integer/real vector as a chromosome to represent values of the program input variables x. The length of the vector depends on the required precision and the domain length for each input variable.

Với function phân loại tam giác ở trên, thì mỗi test case (là mỗi bộ ba a, b, c) được biểu diễn bằng một chromosome là một vector 3 chiều x = (x1, x2, x3).

***Initial population***

Bau đầu cần phải xác định được một số cố định popsize là số lượng chromosome trong một population. Sau đó khởi tạo giá trị ngẫu nhiên cho toàn bộ chromosome trong population ban đầu này.

***Evaluation function***

The algorithm evaluates each test case by executing the program with it as input, and recording the executed path in the program that are covered by this test case. The fitness value *eval(vi)* for each chromosome *vi* (*i* = 1,…, *popsize*) is calculated as follows:

if (executed path tập target paths)

{

eval(vi) = 0;

loại executed path ra khỏi tập target paths;

}

else

{

eval(vi) = *;*

}

Khoảng cách giữa hai path trong một function được định nghĩa như sau:

distance(path i, path j) =

Ví dụ, với các path của function phân loại tam giác ở trên, thì ta có khoảng cách giữa các path như sau:

distance(path 1, path 2) = 3

distance(path 2, path 3) = 0.3333

***Selection***

After computing the fitness of each test case in the current population, the algorithm selects test cases from the effective members of the current population that will be parents of the new population. If none of the members of the current population was effective, all the members of current population are considered the parents of the new population. In the selection process the GA uses random selection method which are described below.

The selection of parents is made randomly, so that every effective member of the current population has an equal chance of being selected for recombination.

Assume that l members of the current population were effective, where l ≤ *popsize*. The parents are selected as follows:

Isolate the effective members and number them from 1 to l;

for i=1 to popsize do

{

Generate an random integer number j from the range [0..l];

Select chromosome vj from the effective members;

}

***Recombination***

In the recombination phase, we use two operators, crossover and mutation, which are the key to the power of GAs. These operators create new individuals from the selected parents to form a new population.

*Crossover*: It operates at the individual level

. During crossover, two parents (chromosomes) exchange sub string information (genetic material) at a random position in the chromosome to produce two new strings (offspring). The objective here is to create better population over time by combining material from pairs of (fitter) members from the parent population. Crossover occurs according to a crossover probability. The probability of crossover *pc* gives us the expected number *pc · popsize* of chromosomes, which undergo the crossover operation. We proceed in the following way:

For each chromosome in the (new) population:

• Generate a random (float) number *r* from the range [0..1];

• If *r* < *pc* then select given chromosome for crossover.

Now we mate selected chromosomes randomly: For each pair of coupled chromosomes we generate a random integer number *pos* from the range [1..*m*-1] (*m* is the number of bit in a chromosome). The number *pos* indicates the position of the crossing point. Two chromosomes (*b1*…*bposbpos+1*…*bm*) and (*c1*…*cposcpos+1*…*cm*) are replaced by a pair of their offspring (*b1*…*bposcpos+1*…*cm*) and (*c1*…*cposbpos+1*…*bm*).

*Mutation*: It is performed on a bit-by-bit basis. Mutation always operates after the crossover operator, and flips each bit with the pre-determined probability. The probability of mutation *pm*, gives us the expected number of mutated bits *pm* · *m* · *popsize*. Every bit (in all chromosomes in the whole population) has an equal chance to undergo mutation, i.e., change from 0 to 1 or vice versa. So we proceed in the following way:

For each chromosome in the current (i.e. after crossover) population and for each bit within the chromosome:

• Generate a random (float) number *r* from the range [0..1];

• If *r* < *pm* then mutate the bit.

***Adjustment***

Sau khi thực hiện xử lý recombination, để có thể sinh được các test case phủ được path khó phủ, cần phải điều chỉnh các giá trị của mỗi chromosomes trong population.

Phương pháp điều chỉnh được tiến hành như sau:

• Duyệt danh sách path condition của các path khó được phủ

• Nếu có giá trị nào của chromosome gần thỏa mãn được path condition này thì điều chỉnh cho nó thỏa mãn.

Quay trở lại function phân loại tam giác Tritype, với các path condition đã được xác định ở trên, tiến hành điều chỉnh như sau:

for i=1 to popsize do

{

if (abs(chromosome[i].x1 - chromosome[i].x2) <= epsilon)

{

// Adjustment for Path 3 (Isosceles)

chromosome[i].x1 = chromosome[i].x2;

}

if ((abs(chromosome[i].x1 - chromosome[i].x2) <= epsilon) &&

(abs(chromosome[i].x1 - chromosome[i].x3) <= epsilon))

{

// Adjustment for Path 2 (Equilateral)

chromosome[i].x2 = chromosome[i].x1;

chromosome[i].x3 = chromosome[i].x1;

}

}

Với các điều chỉnh này, thu được kết quả khi chạy giải thuật di truyền để sinh test case cho function phân loại tam giác như sau:

Run No. 1 : Wait Please .........

Path 4: a = 14.744746849514955 b = 4.817243648636693 c = 12.301081756217084

Objective call: 1

Path 1: a = 7.768767568484256 b = 1.3747887684823534 c = 2.6602222750788584

Objective call: 3

Path 3: a = 10.153558006964193 b = 10.6389422858413 c = 10.6389422858413

Objective call: 462

Run No. 2 : Wait Please .........

Path 2: a = 0.18596209465759556 b = 0.18596209465759556 c = 0.18596209465759556

Objective call: 57595

**5. Experiments**

Bài báo này đã thực hiện thực nghiệm với các function như sau:

- tA2008 determines whether three given numbers that represent three lengths on a plane form a scalene, isosceles, equilateral, or not a triangle.

- tritypeBueno2002 accepts three numbers representing sides of atriangle, classifies its type, and computes its area.

- triangleMansour2004 classifies three numbers representing triangleside lengths into five type triangles: scalene, isosceles, right, iso-right, or equilateral.

Kết quả phân tích tĩnh của từng test function này thu được tập target paths, xác suất sinh data để có thể phủ cùng điều kiện cần điều chỉnh các path tương ứng như bảng sau.

|  |  |  |  |
| --- | --- | --- | --- |
| Function | PathID | Target paths | Path condition |
| tritypeBueno2002 | 1 | {[1,F], [2,F], [3,F], [4,F], [5,F]} | b ==c |
|  | 2 | {[1,F], [2,F], [3,F], [4,F], [5,T]} | a == b |
|  | 3 | {[1,F], [2,F], [3,F], [4,T]} | a == b == c |
|  | 4 | {[1,F], [2,F], [3,T], [6,F], [7,F]} |  |
|  | 5 | {[1,F], [2,F], [3,T], [6,F], [7,T]} |  |
|  | 6 | {[1,F], [2,F], [3,T], [6,T]} | a\*a == b\*b + c\*c |
|  | 7 | {[1,F], [2,T]} |  |
|  | 8 | {[1,T]} |  |
| triangleMansour2004 | 1 | {[1,F], [2,F], [3,F]} |  |
|  | 2 | {[1,F], [2,F], [3,T]} | a\*a == b\*b + c\*c |
|  | 3 | {[1,F], [2,T], [3,F]} | b = c |
|  | 4 | {[1,F], [2,T], [3,T]} | (a\*a == b\*b + c\*c)&&  (b == c) |
|  | 5 | {[1,T], [2,F], [3,F]} | a == b |
|  | 6 | {[1,T], [2,F], [3,T]} |  |
|  | 7 | {[1,T], [2,T], [3,F]} | a == b == c |
|  | 8 | {[1,T], [2,T], [3,T]} |  |
| tA2008\_Triangle | 1 | {[1,F]} |  |
|  | 2 | {[1,T], [2,F], [3,F]} | a == b |
|  | 3 | {[1,T], [2,F], [3,T]} | a == b == c |
|  | 4 | {[1,T], [2,T]} |  |
| QuadraticEquation2 | 1 | {[1,F], [2,F], [3,F]} |  |
|  | 2 | {[1,F], [2,F], [3,T]} | b\*b == 4\*a\*c |
|  | 3 | {[1,F], [2,T]} |  |
|  | 4 | {[1,T], [4,F]} | a == 0 |
|  | 5 | {[1,T], [4,T]} | (a == 0)&&(b == 0) |

Sau khi áp dụng phương pháp phân tích tĩnh kết hợp với giải thuật di truyền, so sánh với sử dụng giải thuật di truyền truyền thống, thu được kết quả như trong bảng sau:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Function | Target paths | Không điều chỉnh | | Có điều chỉnh | |
| Path ID | Object call | Path ID | Object call |
| tritypeBueno2002 | 8 | 8 | 1 | 8 | 1 |
|  |  | 7 | 4 | 7 | 4 |
|  |  | 5 | 10 | 5 | 10 |
|  |  | 4 | 16 | 4 | 16 |
|  |  |  |  | 2 | 924 |
|  |  |  |  | 1 | 3,900 |
|  |  |  |  | 6 | 31,434 |
|  |  |  |  | 3 | 129,295 |
| triangleMansour2004 | 8 | 1 | 1 | 1 | 1 |
|  |  |  |  | 3 | 1,370 |
|  |  |  |  | 5 | 2,358 |
|  |  |  |  | 2 | 8,882 |
|  |  |  |  | 7 | 18,374 |
|  |  |  |  | 4 | 25,080 |
| tA2008\_Triangle | 4 | 3 | 1 | 3 | 1 |
|  |  | 0 | 3 | 0 | 3 |
|  |  |  |  | 2 | 462 |
|  |  |  |  | 1 | 57,595 |
| QuadraticEquation2 | 5 | 3 | 1 | 3 | 1 |
|  |  | 1 | 5 | 1 | 5 |
|  |  |  |  | 5 | 2,086 |
|  |  |  |  | 4 | 11,610 |
|  |  |  |  | 2 | 14,468 |

**6. Conclusion**

**7. References**

[1] I. Hermadi, Path Testing Using Genetic Algorithm, Ph.D. Thesis, University of New South Wales, Canberra, Australia, August 2012 (submitted for examination).

[2] I. Hermadi, C. Lokan, R. Sarker, Dynamic stopping criteria for search-based test data generation for path testing, Information and Software Technology, volume 56 issue 4, April, 2014, pages 395-407.

[15] Thomas J. McCabe, A complexity measure, IEEE Transactions on Software Engineering, December 1976, volume 2, issue 4, pages 308-320.