**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, by searching for test cases (inputs, or pairs of input–output) that satisfy chosen testing criteria.

Most research in this area considers “white box” testing, or structural coverage, in which the aim is to ensure that executing a collection of test cases results in all parts of a program being tested. This can be interpreted in various ways, including “statement coverage” (when the program is tested with all of the test cases, somewhere along the line every statement in the programis executed at least once), ‘‘branch coverage’’ (both outcomes at every 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]. Evolutionary path testing, which uses an evolutionary algorithm (e.g. genetic algorithm “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 điều kiện so sánh phức tạp.

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

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 test cases (inputs to a program) that between them lead to the traversal of all logical paths through the program. In general, path testing process consists of two major steps: target paths generation, and test data generation.

*2.1.1. Target paths generation*

Target paths generation means identifying a set of logical execution path ways through the program, that we hope should all be 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 graph can be automatically generated by using appropriate programming language 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 execution through the program, which is determined by the decisions made at each decision point between the program’s entry point and its exit point.

*2.1.2. Test data generation*

Generating test data that fulfill path coverage is the main task in path testing. It is the process of creating test data, either heuristically or 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 and optimisation. GAs search methods are rooted in the mechanisms of evolution and natural genetics. GAs draw inspiration from the natural search and selection processes leading to the survival of the fittest individuals. GAs generate a sequence of populations by using a selection mechanism, and use crossover and mutation as search mechanisms.

The principle behind GAs is that they create and maintain a population of individuals represented by chromosomes (essentially a character string analogous to the 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 a measure of its fitness in the environment. Reproduction selects individuals with high fitness values in the population, and through crossover and mutation of such individuals, a new population is derived in which individuals may be even better fitted to their environment. The process of crossover involves two chromosomes swapping chunks of data (genetic information) and is analogous to the process of sexual reproduction. Mutation introduces slight changes into a small proportion of the population and is representative of an evolutionary step. The structure of a simple GA is 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**

**4. Proposed approach**

Chúng ta hãy xem 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)) // branch #1

{

if ((a != b) && (b != c) && (c != a)) // branch #2

trityp = 1; // Scalene

else

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

((c == a) && (a != b))) // branch #3

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]} // Path 4 (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 |
| Path 2 (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 đường dẫn mà có xác suất phủ được là thấp trong chương trình. Tìm các 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 |
| Path 2 (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 a 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 data được biểu diễn bằng một chromosome là một vector 3 chiều x = (x1, x2, x3).

Quay trở lại chương trình phân loại tam giác Tritype ở trên, các câu lệnh điều kiện mà cần điều chỉnh trong giải thuật di truyền là branch #2 và branch #3. Điều kiện so sánh mà cần điều chỉnh khi sinh ra các cá thể ở trong giải thuật di truyền là (a==b) và (a==b && b==c). Với các điều chỉnh này, thu được kết quả khi chạy 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 representthree 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 là:

|  |  |  |  |
| --- | --- | --- | --- |
| 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.