**Multi-Objective Particle Swarm Optimization-based**

**Test Data Generation for Path Coverage Testing**

**Abstract**

# 1. Introduction

Software has widely affected ourwork and lives, and brought

us great convenience. However, its failure may lead to significant

economic loss or threat to life safety. For example, the

explosion incident of Ariane-V rocket [1] and the BP deepwater

horizon disaster [2] are the typical instances of such

failures in the past years. As a consequence, software quality

has become an important concern for people in the current

information society.

Phần mềm đang trở nên ngày một phổ biến trong cuộc sống và công việc của chúng ta, và mang lại cho chúng ta những tiện nghi to lớn. Tuy nhiên, những failure của chúng có thể dẫn đến những thiệt hại về kinh tế và thậm chí đe dọa sự an toàn cho cuộc sống của chúng ta. Do đó, như là một lẽ tự nhiên,

Xem xét một program under test như sau:

int getDayNum(int year, int month)

{

int maxDay=0;

if( month>=1 && month<=12) //bch1: branch 1

{

if(month==2) //bch2: branch 2

{

if(year%400==0||(year%4==0&&year%100==0))

//bch1: branch 1

maxDay=29;

else //bch2: branch 2

maxDay=28;

}

else if(month==4||month==6||month==9||month==11)

//bch3: branch 3

maxDay=30;

else //bch4: branch 4

maxDay=31;

}

else //bch5: branch 5

maxDay=-1;

return maxDay;

}

Với program under test này, Mao [1] đã sử dụng giải thuật PSO với duy nhất 1 hàm fitness function bằng cách kết hợp branch functions for branch predicates với branch weight. Phương pháp này đã chứng minh được rằng giải thuật PSO có hiệu quả hơn giải thuật GA trong việc sinh test data, tuy nhiên vẫn có nhược điểm là việc tính toán các branch weight cho một program under test vẫn hoàn toàn là công việc manual.

Cách tiếp cận của chúng tôi là, với mỗi branch predicate, thì tạo ra một fitness function. Sau đó chúng tôi sẽ sử dụng multi-objective particle swarm optimization để tìm nghiệm thỏa mãn cho mỗi fitness function này.

# 2. Background

## 2.1. Fitness function

Table 1. The branch functions for several kinds of branch predicates

|  |  |  |
| --- | --- | --- |
| No | Predicate | Branch distance function *f*(bch*i*) |
| 1 | Boolean | If *true* then 0 else *k* |
| 2 | ¬a | Negation is propagated over *a* |
| 3 | a = b | If abs*(a* − *b)* = 0 then 0 else abs*(a* − *b)* + *k* |
| 4 | a ≠ b | If abs*(a* − *b)* ≠ 0 then 0 else *k* |
| 5 | a < b | If *a* − *b <* 0 then 0 else abs*(a* − *b)* + *k* |
| 6 | a ≤ b | If *a* − *b* ≤ 0 then 0 else abs*(a* − *b)* + *k* |
| 7 | a > b | If *b* − *a >* 0 then 0 else abs*(b* − *a)* + *k* |
| 8 | a ≥ b | If *b* − *a* ≥ 0 then 0 else abs*(b* − *a)* + *k* |
| 9 | a and b | *f (a)* + *f (b)* |
| 10 | a or b | min*( f (a), f (b))* |

Để thuận lợi cho việc trình bày thì có thể gộp các predicate từ 3 đến 8 ở Table 1 thành 1 function được viết như sau:

|  |
| --- |
| **Algorithm 1** Branch distance function (fBchDist) |
| **Input:** double a, operator type, double b |
| **Output:** Branch distance value |
| switch (operator type)  {  case “==”: |
| if abs(*a* − *b*) = 0 then retrun 0 else return (abs(*a* − *b*) + *k*) |
| case “!=”: |
| if abs*(a* − *b)* ≠ 0 then return 0 else return *k* |
| case “<”: |
| if *a* − *b <* 0 then return 0 else return (abs*(a* − *b)* + *k*) |
| case “≤”: |
| if *a* − *b* ≤ 0 then return 0 else return (abs*(a* − *b)* + *k*) |
| case “>”: |
| if *b* − *a >* 0 then return 0 else return (abs*(b* − *a)* + *k*) |
| case “≥”: |
| if *b* − *a* ≥ 0 then return 0 else return (abs*(b* − *a)* + *k*) |
| } |

# 3. Related work

Sinh test data là một trong các chủ đề chính của automated software testing, và thu hút được sự quan tâm của cộng đồng các nhà nghiên cứu trong một thập niên trở lại đây. Trong paper này, chúng tôi tập trung vào các phương pháp sinh test data dựa trên các thuật toán tìm kiếm meta-heuristic search.

Trong thập niên 1990, genetic algorithm là phương pháp được chọn lựa để sinh test data. Jones et al[2] và Pargas et al [3] đã nghiên cứu để sử dụng GA trong việc sinh ra test case để conver được các nhánh trong một program. Các thực nghiệm cũng đã cho thấy rằng với các small program thì GA có hiệu quả hơn hẳn so với các thuật toán tìm kiếm ngẫu nhiên. Trong những năm gần đây, McMinn [3] and Harman [4] đã thực hiện các nghiên cứu thực nghiệm dựa trên GA để sinh test data cho các large-scale programs, và cho thấy hiệu quả của nó hơn hẳn các thuật toán tìm kiếm meta-heuristic search khác.

Mặc dù GA đã đạt được các tiến bộ đáng kể trong việc sinh ra test data, tuy nhiên GA có một nhược điểm là tốc độ sinh test data để phủ được toàn bộ các branch của một program under test còn hạn chế. Do đó, phương pháp PSO, được đề xuất bởi Kennedy and Eberhart [5], có thể giải quyết được nhược điểm này.

# 4. Proprosed approach

## 4.1. Thiết lập fitness function cho mỗi branch

Mỗi branch trong sample getDayNum() bên trên được gán cho mỗi fitness function như sau:

|  |  |  |
| --- | --- | --- |
| No | Branch | Fitness function |
| 1 | bch1 | fBchDist(month, >=, 1) + fBchDist(month, <=, 12) (1) |
| 2 | bch2 | (1) + fBchDist(month, ==, 2) (2) |
| 3 | bch3 | (2) + min(fBchDist(year%400, ==, 0),  fBchDist(year%4, ==, 0) + fBchDist(year%100, ==, 0)) |
| 4 | bch4 | (2) + fBchDist(year%400, !=, 0) +  Min(fBchDist(year%4, !=, 0), fBchDist(year%100, !=, 0)) |
| 5 | bch5 | (1) + min(fBchDist(month, ==, 4), fBchDist(month, ==, 6),  fBchDist(month, ==, 9), fBchDist(month, ==, 11)) |
| 6 | bch6 | (1) + fBchDist(month, !=, 4) + fBchDist(month, !=, 6)  + fBchDist(month, !=, 9) + fBchDist(month, !=, 11)) |
| 7 | bch7 | min(fBchDist(month, <, 1), fBchDist(month, >, 12)) |

## 4.2. Multi-Objective Particle Swarm Optimization (MOPSO)

Sử dụng Multi-Objective trong Particle Swarm Optimization, với mỗi objective là một fitness function cho mỗi predicate branch của program under test.

# 5. Experimental result

So sánh với kết quả của Mao[1] theo các benchmark và tiêu chí mà Mao [1] đã sử dụng. Hai tiêu chí được mang ra so sánh là:

* Success rate (SR) is the probability of all branches which can be covered by the generated test data.
* Average time (AT) is the average execution time (ms) for realizing all-branch coverage.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program under test | Success rate (%) | | Average time (ms) | |
| Mao[10]’s PSO | MOPSO | Mao[10]’s PSO | MOPSO |
| triangleType | 99.80 | 100.0 | 0.19 |  |
| calDay | 100.0 | 100.0 | 0.35 |  |
| cal | 100.0 | 100.0 | 0.50 |  |
| remainder | 100.0 | 100.0 | 0.17 |  |
| computeTax | 99.80 | 100.0 | 0.25 |  |
| bessj | 100.0 | 100.0 | 0.66 |  |
| printCalendar | 99.10 | 100.0 | 1.41 |  |
| line | 99.20 | 100.0 | 2.69 |  |

# 6. Conclusion

**References**

1. C. Mao: Generating Test Data for Software Structural Testing Based on Particle Swarm Optimization. Arabian Journal for Science and Engineering, vol 39, issue 6, pp 4593–4607 (June 2014).

2. Jones, B.F.; Sthamer, H.H.; Eyres, D.E.: Automated structural testing using genetic algorithms. Softw. Eng. J. **11**(5), 299–306 (1996)

3. Pargas, R.P.; Harrold, M.J.; Peck, R.: Automated structural testing using genetic algorithms. Softw. Test. Verif. Reliab. **9**(4), 263–282 (1999)

4. McMinn, P.: Search-based software test data generation: a survey. Softw. Test. Verif. Reliab. 14, 105–156 (2004)

5. Harman, M.; McMinn, P.: A theoretical and empirical study of search-based testing: local, global, and hybrid search. IEEE Trans. Softw. Eng. 36(2), 226–247 (2010)

6. Kennedy, J.; Eberhart, R.C.: Particle swam optimization. In: Proceedings of IEEE International Conference on Neural Networks (ICNN’95), pp. 1942–1948 (1995)