Generating Test Data for Software Structural Testing using Particle Swarm Optimization

Abstract

Search-based test data generation is a very popular domain in the field of automatic test data generation. However, existing search-based test data generators suffer from some problems. By combining static program analysis and search-based testing, our proposed approach overcomes one of these problems. Considering the automatic ability and the path coverage as the test adequacy criterion, this paper proposes using Particle Swarm Optimization, an alternative search technique, for automating the generation of test data for evolutionary structural testing. Experimental results demonstrate that our test data generator can generate suitable test data has higher path coverage than the previous one.

*Keywords:* Automatic test data generation, search-based software testing, Particle Swarm Optimization.

1. Introduction[[1]](#footnote-1)\*

Software is a mandatory part of today's life, and has become more and more important in the current information society. However, its failure may lead to significant economic loss or threat to life safety. As a consequence, software quality has become a top concern today. Among the methods of software quality assurance, software testing has been proved as one of the effective approaches to ensure and improve software quality over the past three decades. However, as most of the software testing is being done manually, the workforce and cost required are accordingly high [1]. In general, about 50 percent of workforce and cost in the software development process is spent on software testing [2]. Considering those reasons, automated software testing has been evaluated as an efficient and necessary method in order to reduce those effort and costs.

Automated structural test data generation is being the research topic attracting much interest in automated software testing, for it not only enhances the efficiency but also can reduce considerably costs of software testing. In our paper, we will focus on path coverage test data generation, considering that almost all structural test data generation problems can be transformed to the path coverage test data generation one. Moreover, Kernighan and Plauger [3] also pointed out that path coverage test data generation can find out more than 65 percent of bugs of the given program under test (PUT).

Although path coverage test data generation is the major unsolved problem [20], various approaches have been proposed by researchers. These approaches can be classified into two types: constraint-based test data generation (CBTDG) or search-based test data generation (SBTDG).

Symbolic execution (SE) is the state-of-the-art of CBTDG approaches [21]. Even though there have been significant achievements, SE still faces difficulties in handling infinite loops, array, procedure calls and pointer references in each PUT [22].

There are also random testing, local search [10], and evolutionary methods [23, 24, 25] in SBTDG approaches. As the value of input variables is assigned when program executes, problems encountered in CBTDG approaches can be avoided in SBTDG.

Being an automated searching method in a predefine space, genetic algorithm (GA) was applied to test data generation since 1992 [26]. Micheal et al [22], Levin and Yehudai [25], Joachim et al [27] indicated that GA outperforms other SBTDG methods e.g. local search or random testing. However even though they can generate test data with appropriate fault-prone ability [4, 5], they fail to produce them quickly due to their slowly evolutionary speed. Recently, as a swarm intelligence technique, Particle Swarm Optimization (PSO) [6, 7, 8] has become a hot research topic in the area of intelligent computing. Its significant feature is the simplicity and fast convergence speed.

Even so, there are still certain limitations in current research on PSO usage in test data generation. For example, consider one PUT which was used in Mao’s paper [9] as below:

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))

//bch3: branch 3

maxDay=29;

else //bch4: branch 4

maxDay=28;

}

else if(month=4||month=6||

month=9||month=11)

//bch5: branch 5

maxDay=30;

else //bch6: branch 6

maxDay=31;

}

else //bch7: branch 7

maxDay=-1;

return maxDay;

}

Regarding this PUT, Mao [9] used PSO to generate test data through building the one and only fitness function which was the combination of Korel formula [10] and the branch weights. This proposal has two weaknesses: the branch weight function being entirely performed manually and some PUTs not being able to generate test data to cover all test paths. To overcome these weaknesses, we still use PSO to generate test data for the given PUT. However, unlike Mao, our approach is to assign one fitness function for each test path. Then we will use simultaneous multithreading of PSO to find simultaneously the solution corresponding to this fitness function, which is also the one being able to generate test data for this test path.

The rest of this paper is organized as follows: Section 2 gives some theoretical backgrounds including fitness function and particle swarm optimization algorithm. Section 3 summarizes some related works, and Section 4 presents the proposed approach in detail. Section 5 shows the experimental results and discussions. Section 6 concludes the paper.

2. Background

This section describes the theoretical background being used in our proposed approach.

2.1. Fitness function

When using PSO, a test path coverage test data generation is transformed into an optimization problem. To cover a test path during execution, we must find appropriate values for the input variables which satisfy related branch predicates. The usual way is to use Korel’s branch distance function [10]. As a result, generating test data for a desired branch is transformed into searching input values which optimizes the return value of its Korel function. Table 1 gives some common formulas which are used in branch distance functions. To generate test data for a desired path P, we define a fitness function *F*(*P*) as the total values of all related branch distance functions. For these reasons, generating path coverage test data can be converted into searching input values which can minimize the return value of function *F*(*P*).

Similar to Mao [9], we also set up the punishment factor k = 0.1. Basing on this formula, we will develop a function calculating values at branch predication, which is to be explained in the next part.

**2.2. Particle Swarm Optimization**

Particle Swarm Optimization (PSO) was first introduced in 1995 by Kennedy and Eberhart [11], and is now widely applied in optimization problems. Comparing to other optimal search algorithms such as GA or SA, PSO has the strength of faster convergent speed and easier coding. PSO is initialized with a group of random particles (solutions) and then it searches for optima by updating generations. In every iteration, each particle is updated by the following two "*best*" values. The first one is the best solution (fitness) it has achieved so far (the fitness value is also stored). This value is called *pbest*. Another "*best*" value that is tracked by the particle swarm optimizer is the best value, obtained so far by any particle in the population. This best value is a global best and called *gbest*.

After finding the two best values, the particle updates its velocity and positions with following equation (1) and (2).

(1)

(2)

*v*[] is the particle velocity, *persent*[] is the current particle (solution). *pbest*[] and *gbest*[] are defined as stated before. *rand*() is a random number between (0,1). *c*1, *c*2 are learning factors, usually *c*1 = *c*2 = 2.

The PSO algorithm is described by pseudo code as below:

Particles' velocities on each dimension are clamped to a maximum velocity *Vmax*, which is an input parameter specified by the user.

# **3. Related work**

From the 1990s, genetic algorithm (GA) has been adopted to generate test data. Jones et. al. [13] presented a GA-based branch coverage test data generator. Their fitness function made use of weighted Hamming distance to branch predicate values. They used unrolled control flow graph of a test program such that it is acyclic. Six small programs were used to test the approach. In recent years, Harman and McMinn [14] performed empirical study on GA-based test data generation for large-scale programs, and validated its effectiveness over other meta-heuristic search algorithms.

Although GA is a classical search algorithm, their convergence speed is not very significant. PSO algorithm, which simulates to birds flocking around food sources, was invented by Kennedy and Eberhart [11] in 1995, and was originally just an algorithm used for optimization problems. However with the advantages of the convergence speed and easier construction than other optimization algorithms, it was promptly adopted as a meta-heuristic search algorithm in the automatic test data generation problem.

Automatic test data generation literature using PSO started with Windisch et al. [6] in 2007. They improved the PSO into comprehensive learning particle swarm optimization (CL-PSO) to generate structural test data, but some experiments proved that the convergence speed of CL-PSO was perhaps worse than the basic PSO.

Jia et al. [8] created an automatic test data generating tool named particle swarm optimization data generation tool (PSODGT). The PSODGT is characterized by two features. First, the PSODGT adopts the condition-decision coverage as the criterion of software testing, aiming to build an efficient test data set that covers all conditions. Second, the PSODGT uses a particle swarm optimization (PSO) approach to generate test data set. In addition, a new position initialization technique is developed for PSO. Instead of initializing the test data randomly, the proposed technique uses the previously-found test data which can reach the target condition as the initial positions so that the search speed of PSODGT can be further accelerated. The PSODGT is tested on four practical programs.

Khushboo et al. [15] described the application of the discrete quantum particle swarm optimization (QPSO) to the problem of automated test data generation. A discrete quantum particle swarm optimization algorithm based on the concept of quantum computing. They had studied the role of the critical QPSO parameters on test data generation performance and based on the observation an adaptive version (AQPSO) had been designed. Its performance compared with QPSO. They used the branch coverage as their test adequacy criteria.

Tiwari et al. [16] had applied a variant of PSO in the creation of new test data for modified code in regression testing. The experiment results demonstrated that this method could cover more code in less number of iterations than the original PSO algorithm.

Zhu et al. [17] put forward an improved algorithm (APSO) and applied to automatic test data generation, in which inertia weight was adjusted according to the particle fitness. The results showed that APSO had better performance than basic PSO.

Dahiya et al. [18] proposed PSO based hybrid testing technique and solved many of the structural testing problems such as dynamic variables, input dependent array index, abstract function calls, infeasible paths and loop handling.

Singla et al. [19] presented a technique that based on a combination of genetic algorithm and particle swarm algorithm. It is used to generate automatic test data for data flow coverage with using dominance concept between two nodes, which is compared to both GA and PSO for generation of automatic test cases to demonstrate its superiority.

Mao [9] and Zhang et al. [7] had the same approach, in which they did not execute any PSO improvement but only built a fitness function by combining the branch distance functions for branch predicates and the branch weights of a PUT, then applied PSO to find the solution for this fitness function. The experiment result with 1 benchmark having 8 programs under test proved that PSO algorithm was more effective than GA in generating test data. However, there remained a weakness that the calculation of branch weight for a PUT was still entirely manual work, which reduced the automatic nature of the proposal. In this paper, our proposal can overcome this limitation while being able to assure the efficiency of a PSO-based automatic test data generation method.

# **4. Proposed approach**

Our proposed approach can be divided into two separate parts: performing statistical analysis and applying simultaneous multithreading of PSO to generate test data. It is presented as below chart:

## 4.1. Perform statistical analysis to find out all test paths

At first, we perform the statistical analysis to find all test paths of the given PUT. It can be done through the below 2 small steps:

1) *Control flow graph generation*: Test data generated from source code directly is more complicated and difficult than from control flow graph (CFG). CFG is a directed graph visualizing logic structures of program simplify [12] and is defined as follow:

**Definition 1 (CFG).** *Given a program, a corresponding CFG is defined as a pair G* =(*V*, *E*), *where V* ={*v*0*, v1*,…*vn*} *is a set of vertices representing statements, E* ={(*vi, vj*)|*vi, vj* ∈ *V*} ⊂ *V* × *V is a set of edges. Each edge* (*vi*, *vj*) *implies the statement corresponding to vj is executed after vi.*

This paper uses the CFG generation algorithm from a given program which was presented in [28].

Apply this GenerateCFG algorithm to the above mentioned PUT getDayNum, we will get a CFG which has 5 test paths (presented by decision nodes) as below chart:

2) *Test paths generation*: In order to generate test data, a set of feasible test paths is discovered by traversing the given CFG. Path and test path are defined as follows:

**Definition 2 (Path).** *Given a CFG G =* (*V, E*)*, a path is a sequence of vertices* {*v0, v*1*,..., vk* |(*vi, vi*+1)∈ *E,* 0< *k* < *n*}*, where n is the number of vertices.*

**Definition 3 (Test path).** *Given a CFG G =* (*V, E*)*, a test path is a path* {*v*0*, v*1*,..., vk* |(*vi, vi*+1)∈ *E*}*, where v*0 *and vi+*1 *are corresponding to the start vertex and end vertex of the CFG.*

This research also uses CFG traverse algorithm [28] to obtain feasible test paths from a CFG as below:

1. \* Corresponding author. E-mail.: [duongdt@vnu.edu.vn](mailto:duongdt@vnu.edu.vn) [↑](#footnote-ref-1)