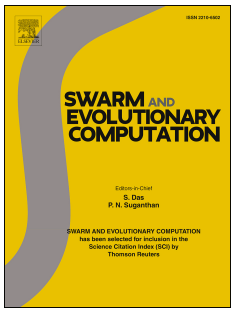
Accepted Manuscript



Structural test data generation using a memetic ant colony optimization based on evolution strategies

Hossein Sharifipour, Mojtaba Shakeri, Hassan Haghighi

PII: S2210-6502(17)30337-1

DOI: 10.1016/j.swevo.2017.12.009

Reference: SWEVO 340

To appear in: *Swarm and Evolutionary Computation BASE DATA*

Received Date: 7 May 2017

Revised Date: 30 November 2017

Accepted Date: 17 December 2017

Please cite this article as: H. Sharifipour, M. Shakeri, H. Haghighi, Structural test data generation using a memetic ant colony optimization based on evolution strategies, *Swarm and Evolutionary Computation BASE DATA* (2018), doi: 10.1016/j.swevo.2017.12.009.

Structural Test Data Generation Using a Memetic Ant Colony Optimization Based on Evolution Strategies

Abstract

Test data generation is one of the key activities that has a significant impact on the efficiency and effectiveness of software testing. Since manual test data generation is quite inefficient and even impractical, automated test data generation has been realized to produce an appropriate subset of input data to carry out effective software testing in reasonable times. This paper presents a memetic ant colony optimization (ACO) algorithm for structural test data generation. The proposed approach incorporates (1+1)-evolution strategies (ES) to improve the search functionality of ants in local moves and enhance search exploitation. Moreover, we have introduced a novel definition of the pheromone functionality in the way that it discourages ants from choosing mostly covered paths of the program to reinforce search exploration. Given that branch coverage is considered as the coverage criterion, two fitness functions are used accordingly for our proposed algorithm. The first fitness function is a Boolean function which is particularly defined to maximize branch coverage. It outputs one if a given solution is successful in traversing at least a yet uncovered branch; otherwise, it returns zero. The second fitness function is formulated according to the complexity of branches covered. The value of the second fitness function is not taken into account for solutions whose Boolean function value equals one. For these solutions, the decision-making process of ants is merely carried out based on the first fitness function. The experimental results indicate the superiority of our memetic ACO algorithm relative to existing test data generation techniques in terms of both branch coverage and convergence speed.

Keywords: automated test data generation, branch coverage, ant colony optimization, evolution strategies, pheromone trail, fitness functions

1. Introduction

Software testing is an important activity in the software development life cycle. One of the challenges in software testing is to generate a set of test data, called a test suite, such that it satisfies a certain test criterion [1, 2, 3]. Unfortunately, this process is normally tedious and costly. According to [4, 5], nearly up to 50% of software development costs have been related to testing. Automated test data generation on the other hand has the potential to significantly reduce software testing time and costs [6].

Since the amount of data representing the input space of a program may approach infinity, it is required that some methods be developed to generate an appropriate test suite that covers most parts of the program [7]. More precisely, assuming a program p, the input vector of p is represented as $X = (x_1, x_2, \ldots, x_n)$ where n is the number of inputs and x_i is the i^{th} input parameter. If the domain of input x_i is equal to D_i, the program input space then equals to $D = D_1 * D_2 * \ldots * D_n$. It is clear that the exhaustive testing of a program with such numerous inputs is not affordable, requiring that an efficient and effective test suite within the given domain be generated. The generated test suite should cover more parts of the program (as a criterion of effectiveness) while it has a low size in terms of the number of test data included in the suite (as a criterion of efficiency) to allow a low test execution time.

Test data generation methods can normally be divided into two distinct classes of functional and structural test data generation [8, 9, 10]. In the former, test data are chosen with reference only to the functional specification of the software under test (SUT) whereas test data for the latter are selected based on the structural elements of the SUT source code such as statements, branches, definition-usage pairs and paths. Compared with functional testing, structural test data generation is more cost-effective to detect failures in programs, thus has been widely applied and studied [6].

Various coverage criteria have been proposed to evaluate a generated test suite. Some important ones include statement coverage [11], branch coverage [12, 13], path coverage [14], coverage of a particular path in the program [7] and coverage of a particular node in a particular path in the program [7]. These coverage criteria can be calculated over the control flow graph (CFG) of the program. Among them, branch coverage is the most cost-effective. Compared to branch coverage, complete path coverage is labor-intensive and is not feasible for CFG graphs with cycles [11]. On the other hand, as opposed to its simplicity, statement coverage cannot be employed to test the false condition in the code. Accordingly, most studies adopt branch coverage as the coverage criterion [11].

With the growing size of software systems and the increase in their complexity, the use of traditional automated test data generation methods, such as symbolic execution [15] and random test data generation [10] is challenging and costly.