Generating Test Case for Object-Oriented Software Using Genetic Algorithm and Mutation Testing Method

Yamina Mohamed Ben Ali, University of Badji Mokhtar, Algeria
Fatma Benmaiza, University of Badji Mokhtar, Algeria

ABSTRACT

This paper presents an automatic creation of software test cases based on the use of a genetic algorithm and a mutation testing technique. The aim of this work is then the optimization of a score function in order to give the best set of optimal test case needed for testing an oriented-object program. Therefore, the proposed search-based approach generates in a first time a set of mutants according to an input program for testing the output of methods belonging in the tested class. On the second time, the output of the mutants and the input program are compared to evaluate the performance of all chromosomes in the genetic population. Finally, at the end of the chromosomes evolution the best test case in retrieved as the optimal one.

Keywords: Automatic Testing, Genetic Algorithm, Java Class, Mutation Testing, Oriented Object Program, Test Case

1. INTRODUCTION

Despite that software testing is time consuming, it remains an effective way to estimate the confidence one can have in software. The confidence measure is based on evaluating the consistency between implementation, specification and test cases. This consistency is evaluated by testing the software (Sacha, 2006; Mathur, 2008). The quality of test cases is then the main factor to evaluate this confidence. Since manual test case design is a time-consuming, tedious, difficult, and error prone activity, many testing methods are developed to increase the effectiveness and efficiency of the test and thus to reduce the overall development costs for software-based systems (Wegener, 2005). An extensive automation of testing can be achieved by transforming testing objectives into search problems which are then solved using evolutionary computation. The application of evolutionary computation to test automation is called Evolutionary Testing (Xanthakis et al., 1992; Hierons et al., 1999; Wegener & Grochtmann, 1998; Wegener, 2005). Evolutionary Testing which aims at generating relevant test cases for a given software unit by
means of an evolutionary algorithm, has been shown to be successful for different test objectives, such as structural testing (Jones et al., 1996; McMinn, 2004; Wappler & Lammerman, 2005; Wappler & Wegener, 2006).

In the context of our studies, we propose to use the technique of mutation testing which involves creating a set of faulty versions of the program, called mutants, and then realizing a set of test cases on each of these erroneous programs. The goal is the validation and improvement of the test cases set. This paper is organized as follows: Section 2 presents an introduction to genetic algorithms. Section 3 exposes the used approach combining genetic algorithms and mutation testing. Section 4 shows some experimental tests, and finally we conclude our work by some remarks.

2. GENETIC ALGORITHMS

Genetic algorithms (GA) (Holland, 1975; Goldberg, 1989) are probabilistic search algorithms inspired from biology. They represent an optimization tool where their aim is to find a problem solution to a given problem. Thus, genetic algorithms try to explore the problem search space in order to find progressively potential solutions through time. Based on inheritance, natural selection, mutation, and sexual reproduction, they try to give after many generations the optimal solution in a finite time.

The evolution process of a genetic algorithm works on an encoding of the search space, represented by a set of individuals (chromosomes) that form the genetic population. Each individual, called solution, is randomly created and evaluated according to a measure of the solution quality. Once initially created the individuals undergo changes according to recombination operators (crossover, mutation). A new population is then generated improving the fitness of its individuals. The genetic algorithm steps can be summarized as follows:

Step1. Initialize (population)
Step2. Evaluate (population)
Step3. Repeat
    3.1 Selection (population)
    3.2 Crossover (population)
    3.3 Mutate (population)
    3.4 Evaluate (population)
Step4. Until (stopping condition is satisfied)

3. PROPOSED METHOD

The problem of improving automatic test cases is a nonlinear problem, which can easily be reduced to an optimization function problem. It means finding the best set of test case that maximizes the mutation score for a given program. To achieve this aim, a genetic algorithm is used for the exploration of large sets of solutions based on the results of the mutation technique as an evaluation tool. For this purpose a strong analogy is done between the phenomena of natural selection and the generation of test cases from a population of initial data. This analogy is expressed as follows: a test case is considered as a predator and the created mutant as the prey. The role of the selection is then to detect the best predators (best test cases) those able to kill most mutants and improving the accuracy of the testing process through generations.

3.1. Evaluation of Test Cases Based on Mutation Testing

Mutation testing is performed as part of unit testing. It is a fault-based testing technique proposed by DeMillo et al. (1978) to validate and improve a set of test cases. In Mutation Testing (Frankl et al., 1997; Offutt & Pan, 1997; Kim et al., 2001; Alexander et al., 2002; Ma et al., 2002; Bradbury et al., 2006) from a program \( p \), a set of faulty programs \( p' \), called mutants, is generated by injecting faults into the original program \( p \). The motivation for Mutation Testing is that injected faults should represent mistakes that programmers often make. Traditionally, a mutant is generated by a single small change to the original program. Figure 1 illustrates the architecture of the generation of mutants via the user interface. A set of mutation...
Related Content

Modified Sub-Population Based Heat Transfer Search Algorithm for Structural Optimization
[www.igi-global.com/article/modified-sub-population-based-heat-transfer-search-algorithm-for-structural-optimization/182234?camid=4v1a](www.igi-global.com/article/modified-sub-population-based-heat-transfer-search-algorithm-for-structural-optimization/182234?camid=4v1a)

Identification of Helicopter Dynamics based on Flight Data using Nature Inspired Techniques
Sugeno Fuzzy-Inference-System-Based Land Cover Classification of Remotely Sensed Images
www.igi-global.com/chapter/sugeno-fuzzy-inference-system-based-land-cover-classification-of-remotely-sensed-images/190167?camid=4v1a

Reinforcement Learning for Improving Gene Identification Accuracy by Combination of Gene-Finding Programs
www.igi-global.com/article/reinforcement-learning-improving-gene-identification/64643?camid=4v1a