Operational Profile-based Test Suite Generation using a Genetic Algorithm

Tomohiko Takagi, Shinichiro Hashimoto, and Zengo Furukawa
Dept. of Reliability-based Information Systems Engineering,
Faculty of Engineering, Kagawa University
Takamatsu, Kagawa, Japan

***Abstract*—This paper shows that a test suite (a set of testcases) that enables software test engineers to rapidly and comprehensively execute software features of high use probability is generated from an operational profile by using a genetic algorithm. Test suites are represented as chromosomes, and genetic operations such as crossover, mutation and selection are applied to them. Usage distribution coverage (UDC) is used for evaluating the fitness of chromosomes. We have developed a tool for automating this method, and showed a simple example.**

Keywords - software testing; testcase; coverage; operational profile; state machine; genetic algorithm

I. Introduction

An operational profile is a model that represents the usage characteristics of software in its operational environments as a probabilistic state machine, and it is typically used for generating testcases (state transition sequences from an initial state to a final state) in software testing. It is well-known that the operational profile allows the infinite generation of testcases in order to evaluate software reliability based on the law of large number [1]. On the other hand, software test engineers having a limited period of time require a new testcase generation method to perform agile testing (i.e., to rapidly and comprehensively test software features of high use probability).

A recent study [2] proposed usage distribution coverage (UDC) that indicates what percentage of software use has been executed by a test suite (a set of testcases) and each individual testcase. Therefore, the new testcase generation method described above can be viewed from the optimization problem of maximizing UDC within the size of a test suite (the number of state transitions) that engineers can perform. Since solving this problem deterministically is difficult, one of effective techniques is to introduce meta-heuristics. This paper shows a new method of generating a test suite optimized by a genetic algorithm from an operational profile.

II. Method Overview

A genetic algorithm is the simulation imitated from biological evolution, and it can be used for finding approximate solutions of a problem that is NP-hard or is hard to be formulated. There are some studies about the software testing technique that effectively applies it to testcase generation [3]. The procedure of a genetic algorithm usually consists of the following steps:

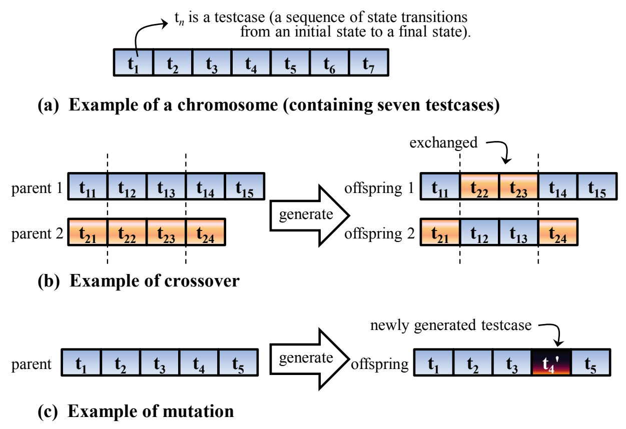
t_n is a testcase (a sequence of state transitions from an initial state to a final state).
t_1 t_2 t_3 t_4 t_5 t_6 t_7
(a) Example of a chromosome (containing seven testcases)
exchanged
parent 1 t_{11} t_{12} t_{13} t_{14} t_{15}
parent 2 t_{21} t_{22} t_{23} t_{24}
generate
offspring 1 t_{11} t_{22} t_{23} t_{14} t_{15}
offspring 2 t_{21} t_{12} t_{13} t_{24}
(b) Example of crossover
newly generated testcase
parent t_1 t_2 t_3 t_4 t_5
generate
offspring t_1 t_2 t_3 t_4' t_5
(c) Example of mutation

Figure 1. Overview of genetic representation and genetic operations in this method.

Step 1. An initial set of chromosomes is generated, and the fitness of each chromosome is evaluated.

Step 2. New chromosomes are generated by crossover and mutation, and the fitness of each new chromosome is evaluated.

Step 3. Chromosomes of the next generation are selected based on the fitness.

Step 4. If termination conditions (e.g., the number of generations) are not satisfied, this procedure returns to step 2.

Step 5. A chromosome of the highest fitness is outputted as a solution.

Our method is based on the above procedure. The genetic representation and genetic operations (crossover, mutation and selection) are the points of a genetic algorithm. Those for our method are set as follows:

A. Genetic representation

Genetic representation is the way of mapping from candidate solutions onto chromosomes. In our method, a test suite is represented as a chromosome. See Fig.1 (a). A testcase (a gene) is randomly generated in proportion to probability distributions of an operational profile. This straightforward mapping obviates the necessity for an encoding routine that translates candidate solutions into chromosomes and a decoding routine that translates chromosomes into candidate solutions.

B. *Crossover*

Crossover is the operation in which a pair of parent chromosomes exchanges their genes and generates offspring chromosomes. In our method, test suites (parent chromosomes) are randomly selected with a crossover probability P_c ($0.0 < P_c < 1.0$), and then they are paired up. After that, each pair selects two cut positions randomly, and testcases between the cut positions are exchanged for the purpose of generating new test suites (offspring chromosomes) as shown in Fig.1 (b). This is generally called two-point crossover.

C. *Mutation*

Mutation is the operation in which part of a parent chromosome is changed for generating a offspring chromosome. In our method, testcases (genes of parent chromosomes) are randomly selected with a mutation probability P_m ($0.0 < P_m < 1.0$). After that, replacing the selected testcases with new testcases generated randomly produces new test suites (offspring chromosomes). See Fig.1 (c).

D. *Selection*

Selection is the operation in which chromosomes of a next generation are selected based on fitness. UDC ($0.0 \leq UDC \leq 1.0$) is used as fitness in our method. UDC of a test suite (i.e., fitness of a chromosome) is a total sum of UDC of testcases that the test suite consists of, and UDC of a testcase is a total product of transition probabilities that appear in the testcase; for example, UDC of a test suite {a1b2d3g2f, a1b2e2f} in Fig.2 is about 0.12 ($1.00 \times 0.67 \times 0.61 \times 0.88 \times 0.28 + 1.00 \times 0.67 \times 0.11 \times 0.28$). However, if the size of a test suite exceeds a fixed upper limit L, its fitness is calculated as $UDC - UDC \times M$ by introducing a penalty M ($0.0 \leq M \leq 1.0$). For a next generation, S chromosomes are randomly selected in proportion to fitness; where S is population size of each generation. This is generally called roulette wheel selection, in which a chromosome of high fitness has many chances of survival.

III. Simple Example

We applied this method to a simple operational profile shown in Fig.2. Setting parameters to L=100, S=20, P_c=0.5, P_m=0.1, M=0.3 resulted in Fig.3. It wouldn't be easy to grow UDC well even though new testcases are added to a test suite without the upper limit of the size L. However, this method improved UDC of a test suite from 0.45 to 0.53 within L through 1000 generations.

We have developed a tool HOUMA2 (High-Order Usage Model Analyzer 2) for automating this method. When HOUMA2 receives data about an operational profile and parameter settings, it outputs the best test suite and a detailed report about its process and results. The execution time of this example was about 11.9 seconds on a notebook computer with 1.2GHz CPU and 2.0GB RAM.

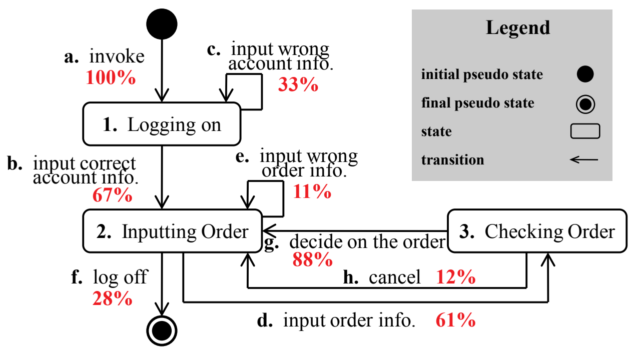

Figure 2. Operational profile of simple internet shopping system.

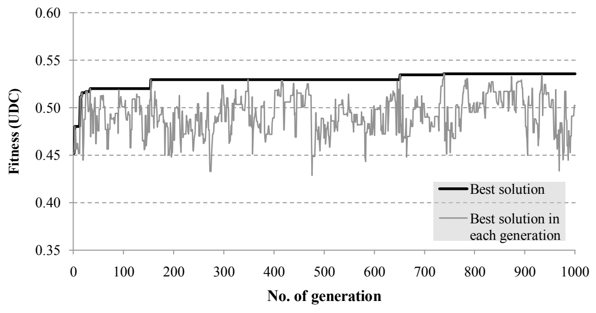

Figure 3. UDC growth of the best test suite.

IV. Conclusion and Future Work

This paper showed that an effective test suite can be automatically generated from an operational profile by using a genetic algorithm. The characteristic of this method is to use UDC as fitness. Since this method would enable software test engineers to test software features of high use probability rapidly, it can be useful for finding serious software faults and taking an overview of software reliability level in a hard-pressed test process.

The genetic representation and operations are important factors that determine the outcome of this method. Future work includes developing them to improve its solution search capability, and evaluating the effectiveness through trial applications to actual software development.

References

[1] G.H. Walton, J.H. Poore, and C.J. Trammell, "Statistical Testing of Software Based on a Usage Model", Software Practice and Experience, Vol.25, No.1, pp.97-108, 1995.

[2] T. Takagi, K. Nishimachi, M. Muragishi, T. Mitsuhashi, and Z. Furukawa, "Usage Distribution Coverage: What Percentage of Expected Use Has Been Executed in Software Testing?", Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, Studies in Computational Intelligence, Springer, pp.57-68, 2009.

[3] C. Doungsa-ard, K. Dahal, A. Hossain, and T. Suwannasart, "Test Data Generation from UML State Machine Diagrams using GAs", Proc. International Conference on Software Engineering Advances, pp.47, 2007.