**PHƯƠNG PHÁP SINH TEST DATA ĐỂ THỰC HIỆN TEST PATH SỬ DỤNG GIẢI THUẬT DI TRUYỀN**

***Abstract***

**1. Introduction**

Testing is a critical but expensive part of the software development life cycle [1–3]. There is considerable interest in ways to automate testing, to reduce the cost and to gain more confidence　in the result [4]. A major task in software testing is test data generation [3]. Search-based test data generation aims to automate this task, by searching for test cases (inputs, or pairs of input–output) that satisfy chosen testing criteria.

Most research in this area considers “white box” testing, or structural coverage, in which the aim is to ensure that executing a collection of test cases results in all parts of a program being tested. This can be interpreted in various ways, including “statement coverage” (when the program is tested with all of the test cases, somewhere along the line every statement in the program is executed at least once), ‘‘branch coverage’’ (both outcomes at every logical branch in the program are executed at least once), and “path coverage” (every distinct path through the code is executed at least once). Path coverage is the strongest form of structural coverage [5]. This paper considers path coverage.

Many approaches have been used in path testing [2,6]. Evolutionary path testing, which uses an evolutionary algorithm (e.g. genetic algorithm “GA”) as the search engine has been found effective [6,7]. In this research, GA is used as the search engine.

A challenge for any search-based approach is làm sao để sinh được những input test data mà có thể phủ được những test path có các điều kiện so sánh phức tạp.

Chúng tôi đề xuất một phương pháp để có thể sinh ra được input test data để phủ được các test path có xác suất đi qua là thấp trong giải thuật di truyền.

This paper is organized as follows: Section 2 presents some theoretical background to understanding this research. Section 3 describes related work, and Section 4 describes the proposed approach in detail. Section 5 presents the evaluation. Section 6 concludes the paper.

**2. Background**

*2.1. Path testing*

The objective of path testing is to search for a collection of test cases (inputs to a program) that between them lead to the traversal of all logical paths through the program. In general, path testing process consists of two major steps: target paths generation, and test data generation.

*2.1.1. Target paths generation*

Target paths generation means identifying a set of logical execution pathways through the program, that we hope should all be exercised during testing. The source code is needed to construct its logical control flow, which can be presented in a control flow graph (CFG). This graph can be automatically generated by using appropriate programming language grammar in which the program is written.

From the CFG, the different logical paths through the program need to be enumerated. A logical path is a particular flow of execution through the program, which is determined by the decisions made at each decision point between the program’s entry point and its exit point.

*2.1.2. Test data generation*

Generating test data that fulfill path coverage is the main task in path testing. It is the process of creating test data, either heuristically or randomly. In a heuristic approach, the process is guided by some rules to search for required test data; the alternative is that random test data is generated.

*2.2. Evolutionary path testing*

Path testing that uses any methods from the evolutionary algorithms　family is called evolutionary path testing. In this work, genetic algorithm (GA) is used as the test data generator.　A chromosome represents one set of test data (a collection　of input values that represents a single test case). Thus the population is a collection of test cases. Each test case causes one targetpath to be executed; most of the time a target path can be covered by many test cases. The aim is to evolve a set of test cases that causes all target paths to be executed.

Generic steps in GA are (1) Initialization, (2) Evaluation, and (3) Do the following until any stopping criteria is met: (3.a) Selection, (3.b) Perturbation, and (3.c) Go back to Step (2). Initialization generates the first population, randomly or with some knowledge. Step (2) evaluates all members of the population using a given fitness function. In (3.a) some members of the population are selected for perturbation using genetic operators. Section (3.b) applies those operators: crossover is responsible for mixing the genetic traits, and mutation for introducing new genetic traits.

The generator keeps a list of target paths that have not yet been covered. At the beginning of the evolution, every target path is in that list. In each generation, each test case in the population is evaluated (its fitness is calculated) against each uncovered target path. When a test case is found to cover a target path, it is remembered and that target path is removed from the list. As the search progresses, the list of paths for which test data is sought changes dynamically. Searching can stop if the list becomes empty, or when some other stopping criterion is reached. If the list of target paths contains infeasible paths, the list of uncovered paths will never be empty, and another stopping criterion is essential.

**3. Related work**

**4. Proposed approach**

**5. Evalution**

**6. Conclusion**

**7. References**

[1] I. Hermadi, Path Testing Using Genetic Algorithm, Ph.D. Thesis, University of New South Wales, Canberra, Australia, August 2012 (submitted for examination).

[2] I. Hermadi, C. Lokan, R. Sarker, Dynamic stopping criteria for search-based test data generation for path testing, Information and Software Technology, Volume 56 Issue 4, April, 2014, Pages 395-407.