Search-Based Hierarchical Regression Testing Dataset Augmentation Method Postprint

Authors: Wang Shuyan, Gao Lu, Sun Jiaze

Date: 2018-07-23T00:00:00+00:00

Abstract

To address the issue that existing test datasets in regression testing often fail to meet the testing requirements of new software versions, this paper proposes a search-based hierarchical regression test dataset augmentation method, which mainly consists of a coverage target method set acquisition module and a test data generation module. First, abstract analysis is performed on the new version of the program to extract the method call graph, and method coverage information is established using method call traces and existing test data to obtain the target method set, with priority selection of the target method set performed by calculating Bayesian conditional probabilities. An orthogonal population is designed using Hadamard matrix, while combining with the existing test dataset for population initialization, and a memetic algorithm is adopted to generate test data for methods in the target set. Compared with random method, genetic algorithm, and particle swarm optimization-based test data generation methods on four benchmark programs, the test data generation efficiency is improved by an average of 95.2%, 78.2%, and 50.5%, respectively, and the fault detection capability of test data is improved by an average of 47.9%, 33.6%, and 18.2%, respectively. Experimental results demonstrate that the proposed method is more suitable for regression test data augmentation.

Full Text

Preamble

Title: Search-Based Hierarchical Regression Test Suite Augmentation Method

Authors: Wang Shuyan, Gao Lu, Sun Jiaze
Affiliation: School of Computer Science & Technology, Xi' an University of Posts & Telecommunications, Xi' an 710121, China

Abstract: In regression testing, the original test dataset often fails to meet the testing requirements of the new software version. To address this problem,

this paper proposes a search-based hierarchical regression test suite augmentation method, which mainly includes a target method set acquisition module and a test data generation module. First, the new version program is abstractly analyzed to extract the method call graph. Method coverage information is established using method call traces and existing test data to obtain the target method set, and Bayesian conditional probability is calculated to prioritize the target method set. The Hadamard matrix is used to design an orthogonal population, which is combined with the existing test dataset for population initialization, and a memetic algorithm is employed to generate test data for the methods in the target set. Compared with random methods, genetic algorithms, and particle swarm optimization-based test data generation methods on four benchmark programs, the proposed method improves test data generation efficiency by an average of 95.2%, 78.2%, and 50.5% respectively, and improves error detection capability by an average of 47.9%, 33.6%, and 18.2% respectively. Experimental results demonstrate that the proposed method is more suitable for regression test suite augmentation.

Keywords: function call graph; regression test suite augmentation; orthogonal population; memetic algorithm

0 Introduction

Software testing is a crucial means to improve software quality and reliability while also enhancing the credibility of the tested software. Regression testing refers to the process where testers re-test the new version of software to ensure that no new errors or other potential defects have been introduced. During the software lifecycle, frequent modifications often make the original test dataset unable to meet the testing requirements of the new version, making it difficult to guarantee coverage of newly added or changed components and to reveal existing or potential errors. To supplement the completeness of the original test dataset, augmentation becomes necessary. Statistics show that regression testing costs typically account for approximately 80% of the entire testing budget, making it highly meaningful to improve regression testing efficiency.

Initially, Menager et al. proposed the concept of test data augmentation, which was originally intended as a counterpart to test dataset reduction methods, without elaborating on how to perform regression test suite augmentation. In recent years, test data augmentation methods have been divided into two main categories: behavior-oriented and coverage-oriented regression test data augmentation methods. The former aims to describe execution changes in newly added and modified parts through generated test datasets, while the latter expects generated test datasets to cover newly added and modified parts of the software, as well as methods that may contain errors after bug fixes.

Santelices et al. utilized symbolic execution technology to analyze software control flow and data flow graphs for test data augmentation, proposing that regres-

sion test augmentation techniques should satisfy state requirements and chain requirements. Subsequently, they introduced the PIE model into regression test dataset augmentation methods, further employing symbolic execution and constraint solving to obtain test datasets. Xu et al. proposed a directed test dataset augmentation technique that primarily obtains branch coverage targets —i.e., sets of branches not covered by any test data—through re-execution analysis of test data, combined with symbolic execution technology to generate new test data. Xu et al. further introduced genetic algorithms into test data augmentation technology and analyzed the impact of genetic algorithm parameter settings on augmentation results. Zhang et al. proposed a code-based regression testing method for generating test data for unit testing, mainly by analyzing the tree structure of programs to identify changed paths, using branch paths as the primary targets for test data generation. Xu et al. utilized original test data to run the modified program, using changed statements as target statements, with the initial population selected from existing test datasets based on the number of target statements, and finally using genetic algorithms to generate test data covering target statements. Gong Dunwei et al. analyzed the similarity between paths traversed by original test data and target paths to select partial test datasets as part of the initial population, using genetic algorithms to evolve and generate test data. Wu Chuan et al. focused on utilizing existing test datasets, proposing a mathematical model for data generation based on path correlation and using genetic algorithms for model solving. Wang Shuyan et al. proposed a regression test augmentation method based on an adaptive particle swarm algorithm, using path similarity to identify new paths and select the initial population, then establishing a mathematical model for test dataset augmentation, and finally solving it using an adaptive particle swarm algorithm.

In the above regression test data generation methods, test data coverage rate is used as the final goal of test data generation, while the defect detection capability of the test dataset is ignored. The purpose of test data is to discover as many program defects as possible with minimal test data. This paper considers both test data coverage and error detection capability. Building upon the work in reference [14] and targeting two scenarios requiring regression testing—(1) regression testing when new methods are added to a software system, and (2) regression testing after program defect repair—this paper proposes a search-based hierarchical regression test data augmentation method. The "hierarchical" aspect refers to sequentially extracting coverage targets at the method level and statement level: first extracting the target method set (including new or changed methods) at the method level, then performing static analysis on the target method set to extract coverage paths, aiming to improve the efficiency of identifying and extracting coverage targets. Bayesian theory is used to prioritize the coverage target method set, and the memetic algorithm (MA) is employed to generate test data for the target path set, aiming to improve test data coverage and error detection capability, thereby reducing regression testing costs. The framework of the proposed method is shown in Figure 1 [Figure 1: see original paper].

1 Obtaining Coverage Target Method Set

1.1 Weighted Method Call Graph

A method call graph is represented as a directed graph, denoted as , with directed edges denoted as . Each edge is assigned a weight . This paper uses the average execution probability of edges as weights. Assuming the method call graph has edges, with each edge appearing times in execution paths, the weight of edge is calculated by the formula:

For the same fixed point, the sum of weights of outgoing edges equals 1.

Consider the method call graph shown in Figure 2 [Figure 2: see original paper]. With test cases , their method coverage paths are . During execution of , method can only call method , so the weight of edge is 1. For method , it can call , , or . The edges , , and appear 2 times, 1 time, and 1 time respectively. Therefore, the weight of edge is $2/(2+1+1) = 0.5$, the weight of edge is $1/(2+1+1) = 0.25$, and the weight of edge is 1 - 0.5 - 0.25 = 0.25. The weighted method call graph during execution of is shown in Figure 3 [Figure 3: see original paper]. By calculating the weight of each node' s outgoing edges in the method call graph for the test dataset, the final weight of each outgoing edge is obtained as the calling probability of that edge during execution of different test data.

1.2 Method Coverage Information

Assuming a software system consists of components , where is the test execution set, where represents the set of test cases with erroneous execution results, and represents the set of test cases with correct execution results, then program coverage information is a 2-dimensional matrix.

The method coverage information needed in this paper is defined as follows: the i-th row represents method , and the j-th column represents test data . If test data covers method , then ; otherwise, . The vector represents the execution results of test data, where element represents the execution result of the i-th test data. If successful, then ; otherwise, .

Given that most source program methods do not contain errors, where is the number of methods and is the number of methods in the target method set , we use maximum likelihood estimation to calculate and based on expressions about . Substituting the obtained into calculation formula (1), we arrange in increasing order and select methods with higher conditional probability values to prioritize test data generation.

2 Test Data Generation

This paper uses method coverage information as input to obtain the target method set , which includes the set of new methods and the set of methods that may contain errors after modification, where . The test data generation framework is divided into a preprocessing module (obtaining method set and obtaining target paths) and a test data generation module (memetic algorithm), as shown in Figure 4 [Figure 4: see original paper]. The preprocessing module is the prerequisite for the test data generation module, responsible for obtaining the ordered coverage target method set in the early stage, performing static analysis on the target method set to select target paths and extract relevant parameters for algorithm parameter initialization, and constructing the fitness function using branch functions. Test data generation is the core module, which guides the population toward the target solution through fitness values, population evaluation, and cooperative competition operators, ultimately generating test datasets that cover the target paths in the target method set.

Assuming the method set is mutually independent, this paper uses Bayes' theorem to calculate the conditional probability of method set as:

where is the execution probability of the k-th node' s outgoing edges, and is a normalization constant defined by all that does not need direct calculation. Since each execution is independent, is defined as:

where is defined as:

represents the probability that method does not contain errors. In the above definition, represents the probability that method does not contain errors. Substituting the obtained into formula (1) and arranging in increasing order, we select methods with higher conditional probability values to prioritize test data generation.

2.1 Initialization Population

Utilizing existing test datasets in regression test data generation can significantly reduce the number of test data generations. The diversity of the initial population ensures that heuristic search algorithms have high evolutionary efficiency during iterative evolution, which is beneficial for evolutionary algorithms to converge to the global optimum more quickly. However, during algorithm evolution, the reduction of population diversity in later stages often leads to premature convergence. To ensure population diversity and improve regression test data generation efficiency, this paper uses a combination of orthogonal population and partial original test datasets for population initialization.

Orthogonal design, proposed by Genichi Taguchi in 1949, has been widely applied to optimization problems. Initializing the population using orthogonal design can distribute individuals more evenly in the solution space, thereby ensuring population diversity. Following Zhang et al.' s application of orthogonal design in genetic algorithms to maintain population diversity and improve

solution optimality and algorithm convergence speed, this paper adopts the orthogonal array designed by Montgomery et al. for population initialization in the memetic algorithm. The orthogonal array is represented as , where is the number of rows in the orthogonal array, representing the number of tests or test cases; is the number of columns, representing the number of tested objects; and is the number of codes in the orthogonal array, representing the number of levels for tested objects. When the number of levels is 2, the orthogonal array only takes values 0 or 1. Current algorithms for constructing orthogonal arrays include row exchange algorithms, coordinate exchange algorithms, or anthropomorphic algorithms. This paper designs orthogonal arrays through Hadamard matrices because this approach can quickly and efficiently generate orthogonal arrays with two levels. A matrix is represented as:

where all matrix elements are 1 or -1, is the transpose matrix, and is the identity matrix. Then is called an order Hadamard matrix. A 4th-order Hadamard matrix is:

If is a Hadamard matrix of order , then a Hadamard matrix of order is:

First, construct a order Hadamard matrix, where and is the initial population size, is the number of tested objects. Then sort the rows of in ascending order and delete the first column of where all elements are 0, forming matrix . Transform into orthogonal array . Finally, use the orthogonal array to generate the orthogonal population, and then perform population initialization using the orthogonal population combined with partial existing test datasets.

2.2 Fitness Function Design

An individual' s viability is mainly reflected in its fitness, which determines its survival and competition capability. Therefore, designing a reasonable fitness function is crucial. This paper uses the branch distance method to design the fitness function. Branch distance refers to the degree to which a predicate condition is satisfied as true (or false). The goal is to cover target paths, i.e., each branch node in the target path. By inserting branch functions at each branch, we represent the distance between current test data and the true branch. A branch distance less than or equal to 0 indicates that the true branch is covered, and the branch distance is set to 0. Assuming the target path contains branches, the fitness function is transformed into normalized branch distance as:

The necessary and sufficient condition for is that exactly covers all branches of the target path. The smaller the value, the closer is to the target. Therefore, the test data generation problem for covering target paths can be transformed into a minimization problem of the fitness function.

2.3 Memetic Algorithm

The memetic algorithm is a combination of population-based global search and individual-based local heuristic search. Its advantage lies in the global-local com-

bined search strategy. Solutions to problems are represented as "chromosomes," and through simulating cultural evolution processes of competition, cooperation operations combined with local search, individual fitness is improved. Through continuous iteration, the optimal or near-optimal solution is gradually sought.

This section details the implementation steps of the proposed method using the test dataset of the original program to generate test data covering the method set of the new version program as an example.

Input: Test dataset of old program and new version program .
Output: Test dataset covering the method set of .

a) Extract the program method call graph, run the new version program combined with the original test dataset, establish method coverage information, and obtain the coverage target method set.

b) Sequentially select methods from the target method set for static analysis to obtain the coverage target path set and related parameters.

c) Select target paths one by one, perform branch function instrumentation, set initial parameters, design the orthogonal population, and initialize the population.

d) Evaluate the population.

e) Determine whether the termination condition is met, i.e., whether the generated test data covers the target path of or reaches the maximum evolution generation. If the termination condition is met, proceed to step i); otherwise, proceed to step f).

f) Select parent individuals , , generate offspring individuals , through crossover and mutation operators, calculate the fitness values of parents and offspring and their survival ability.

g) If , then ; otherwise, accept offspring individuals according to , where is obtained through maximum likelihood estimation, and is the temperature in the local strategy.

h) Generate a new generation population through mutation operators and return to step d).

i) The algorithm ends and outputs the test dataset.

Based on the method coverage information of old and new versions, the coverage target method set is obtained as . The coverage method set is . Taking and the source code as input to the test data generation module, static analysis is performed on first to obtain the number of tested objects and the theoretical path set . Paths and are selected as target paths sequentially, and branch

functions are inserted. The method in Section 2.1 is used to design the initial population, perform population initialization, and evaluate population fitness. If the output condition is satisfied, the algorithm ends; otherwise, iteration continues until test data covering the target path is generated or the maximum iteration count is reached.

---

3 Case Study

The proposed method is illustrated through a triangle classification program. Test data is selected from the original program' s test dataset. The method is explained using this example. First, the method call graphs of the old and new version programs are extracted using the Doxygen tool. Then, following the approach in Section 1.1, the average weight of each node' s outgoing edges in the new version program' s method call graph is calculated to obtain the weighted method call graph. The method call graphs of the old version and the weighted method call graph of the new version are shown in Figure 5 [Figure 5: see original paper].

The methods in the program include: main function, triangle type judgment, triangle definition validation, side length acquisition, difference calculation between two sides, output method, and triangle type determination. In the new version program, is the correct program, and is the program containing errors after defect repair. By analyzing the method coverage information of the old and new versions, as shown in Table 1 , the coverage target method set is obtained as . The coverage method set is . Taking and the source code as input to the test data generation module, static analysis is performed on first to obtain the number of tested objects and the theoretical path set . Paths and are selected as target paths sequentially, and branch functions are inserted. The method in Section 2.1 is used to design the initial population, perform population initialization, and evaluate population fitness. If the output condition is satisfied, the algorithm ends; otherwise, iteration continues until test data covering the target path is generated or the maximum iteration count is reached.

---

4 Experiments

4.1 Experimental Setup

To verify the effectiveness of the proposed method, four benchmark programs were selected: the triangle classification program Triangle, Schedule and Tcas from the Siemens industrial program suite, and the NextDay program. Their basic information is shown in Table 2 .

Table 2. Information of Tested Programs

Program	Lines of Code	Number of Methods	Input Range
Triangle	[value]	[value]	[0,100]
Schedule	[value]	[value]	[0,2047]
Tcas	[value]	[value]	[0,10000]
NextDay	[value]	[value]	[0,10000]

All experimental programs were written in Java. The computer configuration was Windows Intel(R) Core(TM) i5-2450M CPU @ 2.50GHz, 64-bit operating system. Programs ran in Eclipse MARS.2 4.5.2 with jdk1.7.0_80. The strategies and parameter settings related to the proposed method are shown in Table 3 . Each program was run independently 20 times. The algorithm terminated when test data covering the target path was generated or the maximum iteration count was reached.

Table 3. Parameter Settings of the Proposed Method

Algorithm Strategy (Parameters)	Value
Maximum Iterations	[value]
Encoding	Binary
Selection Criterion	Metropolis criterion
Selection Method	Roulette wheel selection
Crossover Operator	Single-point random crossover

To verify the effectiveness of the proposed method, comparisons were made with genetic algorithm (GA), particle swarm optimization (PSO), and random methods. The parameters for benchmark algorithms are shown in Table 4 .

Table 4. Parameter Settings of Benchmark Algorithms

Algorithm	Parameters
GA	Related to input range
PSO	Related to input range
Random	Related to input range

4.2 Experimental Results

To verify the effectiveness of the proposed method, genetic algorithm (GA), particle swarm optimization (PSO), and random methods were selected for comparison with the proposed method. Each method was run under the same conditions, and the iteration count and time consumption for generating test data were recorded. The experimental results are shown in Table 5 and Table 6 .

Taking the Traffic Collision Avoidance System (Tcas) from Table 5 as an example, compared with genetic algorithm requiring an average of 189.3 iterations and 6.513 seconds, particle swarm algorithm requiring an average of 49.1 iterations and 3.235 seconds, and random method failing to complete the test requirements within the specified maximum iteration count, the proposed method only requires 26.9 iterations and 0.972 seconds to generate test data covering the target paths in the target method set. Considering the generation efficiency of various methods across different benchmark programs, compared with particle swarm algorithm, genetic algorithm, and random method, the proposed method improves test data generation efficiency by approximately 50.5%, 78.2%, and 95.2% respectively. The standard deviation obtained from multiple runs shows that the proposed method also has good stability advantages.

Table 5. Iteration Count for Test Data Generation by Different Methods

Program	Random	GA	PSO	Proposed Method
Triangle	[value]	[value]	[value]	[value]
Schedule	[value]	[value]	[value]	[value]
Tcas	[value]	189.3	49.1	26.9
NextDay	[value]	[value]	[value]	[value]

Table 6. Time Consumption for Test Data Generation by Different Methods (seconds)

Program	Random	GA	PSO	Proposed Method
Triangle	[value]	[value]	[value]	[value]
Schedule	[value]	[value]	[value]	[value]
Tcas	[value]	6.513	3.235	0.972
NextDay	[value]	[value]	[value]	[value]

The capability of different regression test data generation methods was evaluated from both test data coverage and defect detection perspectives. Using Tcas as an example, the path coverage rates of test data generated by different methods are shown in Figure 6 [Figure 6: see original paper]. The results demonstrate that the proposed method has obvious advantages in path coverage compared with the other three methods.

Under the same conditions, the MuJava tool was used to inject mutants into the experimental source code. The mutation operators used are described in Table 7 .

Table 7. Description of Mutation Operators

Mutation Operator	Description
AOI	Arithmetic Operator Insertion
SDL	Statement Deletion
ODL	Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
LOI	Logical Operator Insertion
COI	Conditional Operator Insertion
SDI	Statement Duplication Insertion

The test data generated by four different methods were used to run the mutated programs. The mutation operators used and the number of test data are shown in Table 8 .

Table 8. Description of Experimental Programs

Program	Original Test Data	Current Test Data	Running Time (s)
Triangle	[value]	[value]	[value]
Schedule	[value]	[value]	[value]
Tcas	[value]	[value]	[value]
NextDay	[value]	[value]	[value]

When test data can distinguish the original program from a mutant, the mutant is considered killed. The remaining mutants not killed by test data are called surviving mutants, some of which can be further killed by improving the test dataset. However, a certain proportion of mutants are syntactically different from the original program but semantically equivalent, and thus cannot be killed by any test data. These mutants are called equivalent mutants. This paper uses mutation testing to evaluate the error detection capability of test datasets.

First, the original test dataset was used to execute the mutated programs. The experimental results are shown in Table 9 . Then, the augmented test dataset was used to execute the mutated programs, with results shown in Table 10 . The comparison clearly shows that the augmented test dataset can increase the number of killed mutants, thereby reducing the number of potential equivalent mutants.

Table 9. Experimental Results (Original Test Dataset)

Program	Mutants Injected	Mutants Killed	Mutants Survived
Triangle	[value]	[value]	[value]
Schedule	[value]	[value]	[value]
Tcas	[value]	[value]	[value]

Program	Mutants Injected	Mutants Killed	Mutants Survived
NextDay	[value]	[value]	[value]

Table 10. Experimental Results (Augmented Test Dataset)

Program	Mutants Injected	Method	Mutants Killed	Potential Equivalent Mutants
Triangle	[value]	[value]	[value]	[value]
Schedule	[value]	[value]	[value]	[value]
Tcas	[value]	[value]	[value]	[value]
NextDay	[value]	[value]	[value]	[value]

The mutation score for each program was calculated as: Mutation Score = (Number of Mutants Killed / Number of Mutants Injected) × 100%. The average mutation scores are shown in Table 11 . Compared with random method, genetic algorithm, and particle swarm algorithm, the proposed method improves test data error detection capability by approximately 47.9%, 33.6%, and 18.2% respectively.

Table 11. Mutation Scores

Method	Triangle	Schedule	Tcas	NextDay
Random	[value]	[value]	[value]	[value]
GA	[value]	[value]	[value]	[value]
PSO	[value]	[value]	[value]	[value]
Proposed	[value]	[value]	[value]	[value]

5 Conclusion

Augmenting the completeness of the original test dataset is a key issue that needs to be addressed in regression test data generation. Effectively analyzing and extracting changed or newly added components is the prerequisite for regression test data generation. Ensuring high coverage and error detection capability of the augmented test dataset is crucial for improving regression testing efficiency. This paper proposes a search-based hierarchical regression test data augmentation method. The method was tested on multiple benchmark programs and Siemens industrial programs, and compared with random methods and regression test data generation methods using genetic algorithms and particle swarm algorithms. Experimental results demonstrate that the test data generated by the proposed method significantly improves both coverage and error detection capability.

During regression test data generation, the method coverage information at the method granularity is used to analyze changed and newly added methods. Bayesian theory is then applied to determine the probability that a method body may contain errors. Test datasets covering the target path set are generated for methods with high error probability and newly added methods, which improves test dataset coverage and error detection capability to a certain extent. Based on this research, future work will focus on two main aspects: first, expanding the experimental scale; second, applying the proposed method to large, complex real-world software systems, and utilizing software evolution information to efficiently prepare and analyze coverage targets for test data generation.

References

[1] Qiu XiaoKang, Li Xuandong. A path-oriented tool supporting for testing [J]. Acta Electronica Sinica, 2004, 32 (S1): 235-237.

[2] Xu Renzuo. Software reliability engineering [M]. Beijing: Tsinghua University Press, 2007.

[3] Zhang Zhiyi, Chen Zhenyu, Xu Baowen, et al. Research progress on test case evolution [J]. Journal of Software, 2013, 24 (4): 663-674.

[4] Beizer B, Software testing techniques [M]. NY: Van Nostrand Reinhold, 1990.

[5] Harrold M J. Reduce, reuse, recycle, recover: Techniques for improved regression testing [C]// Proc of IEEE International Conference on Software Maintenance. Picataway, NJ: IEEE Press, 2009.

[6] Harder M, Mellen J, Ernst M D. Improving test suites via operational abstraction [C]// Proc of International Conference on Software Engineering. Picataway, NJ: IEEE Press, 2003: 60-71.

[7] Santelices R, Chittimalli P K, Apiwattanapong T, et al. Test-suite augmentation for evolving software [C]// Proc of IEEE//ACM International Conference on Automated Software Engineering. Picataway, NJ: IEEE Press, 2008: 218-227.

[8] Xu Z, Rothermel G. Directed test suite augmentation [C]// Proc of Asia-Pacific Software Engineering Conference. Picataway, NJ: IEEE Press, 2011: 1110-1113.

[9] Xu Z, Cohen M B, Rothermel G. Factors affecting the use of genetic algorithms in test suite augmentation [C]// Proc of Conference on Genetic and Evolutionary Computation. New York: ACM Press, 2010: 1365-1372.

[10] Zhang Zhihao, Huang Jun, Zhang Bo, et al. Regression Test Generation Approach Based on Tree-Structured Analysis [C]// Proc of International Conference on Computational Science and ITS Applications. Picataway, NJ: IEEE Press, 2010: 244-249.

[11] Xu Z, Kim Y, Kim M, et al. Directed test suite augmentation: techniques and tradeoffs [C]// Proc of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2010: 257-266.

[12] Gong Dunwei, Ren Lina. Evolutionary generation of regression test data [J]. Chinese Journal of Computers, 2014, 37 (3): 489-499.

[13] Wu Chuan, Gong Dunwei. Evolutionary generation of test data for regression testing based on path correlation [J]. Chinese Journal of Computers, 2015, 38 (11): 2247-2261.

[14] Wang Shuyan, Wen Chunyan, Sun Jiaze. Test data augmentation method based on adaptive particle swarm optimization algorithm [J]. Journal of Computer Applications, 2016, 36 (9): 2492-2496.

[15] Wen Chunyan. Research on Test Data Augmentation for Regression Testing [D]. Xi' an: Xi' an University of Posts & Telecommunications, 2017.

[16] Maaranen H, Miettinen K, Penttinen A. On initial populations of a genetic algorithm for continuous optimization problems [J]. Journal of Global Optimization, 2007, 37 (3): 405.

[17] Chen Ligu, Cai Zhihua. Application of improving orthogonal-based genetic algorithm in function optimization [J]. Computer Engineering & Design, 2008, 29 (6): 3413-3418.

[18] Zhang Qingfu, Leung Yiuwing. An orthogonal genetic algorithm for multimedia multicast routing [J]. IEEE Trans on Evolutionary Computation, 1999, 3 (1): 53-62.

[19] Montgomery D C. Design and analysis of experiments [M]. NY: Wiley, 2015.

[20] Craigen R. Hadamard Matrices and Designs [M]// Combinatorial Designs. NY: Springer, 2004: 73-100.

[21] Wang Wgnjiang. The Construction of Hadamard Matrix and Its Application, Tianjin: Tianjin University of Technology, 2007.

[22] Stinson D R. Combinatorial designs: constructions and analysis [M]. Berlin: Springer-Verlag, 2003.

[23] Seberry J. Orthogonal designs: hadamard matrices, quadratic forms and algebras [M]. Berlin: Springer-Verlag, 2017.

[24] Crnković D, Egan R, Švob A. Orbit matrices of Hadamard matrices and related codes [J]. Discrete Mathematics, 2018, 341 (5): 1199-1209.

[25] Fraser G, Arcuri A, Mcminn P. Test suite generation with memetic algorithms [C]// Proc of International Conference on Genetic and Evolutionary Computation. 2013: 1437-1444.

[26] Jiang Shujuan, Wang Lingsai, Xue Meng, et al. Test case generation based on combination of schema using particle swarm optimization [J]. Journal of

Software, 2016, 27 (4): 785-801.

[27] Liu Mandan. The development of the memetic algorithm [J]. Control Theory and Applications, 2007 (11): 14-18.

[28] Fraser G, Arcuri A, Mcminn P. A memetic algorithm for whole test suite generation [J]. Journal of Systems & Software, 2015, 103 (2): 311-327.

[29] Mundade A A, Pattewar T M. Test suite generation using Memetic algorithm on adaptive local search [C]// Proc of International Conference on Communications and Signal Processing. Picataway, NJ: IEEE Press, 2015: 0630-0633.

[30] Nejad F M, Akbari R, Dejam M M. Using memetic algorithms for test case prioritization in model based software testing [C]// Swarm Intelligence and Evolutionary Computation. Picataway, NJ: IEEE Press, 2016.

[31] Islam M M, Singh H K, Ray T, et al. An enhanced memetic algorithm for single-objective bilevel optimization problems [J]. Evolutionary Computation, 2017, 25 (4): 607-642.

[32] Islam M M, Singh H K, Ray T, et al. An enhanced memetic algorithm for single-objective bilevel optimization problems [J]. Evolutionary Computation, 2017, 25 (4): 607-642.

Note: Figure translations are in progress. See original paper for figures.