ICIC Express Letters
Volume **13**, Number **7**, July **2019**

 ISSN 1881-803X
pp. 585–591

OPTIMIZATION OF TEST CASE GENERATION FROM UML ACTIVITY DIAGRAM AND SEQUENCE DIAGRAM BY USING GENETIC ALGORITHM

Meiliana, Lusiana Citra Dewi and Alvin Chandra

Computer Science Department
School of Computer Science
Bina Nusantara University
Jl. K. H. Syahdan No. 9, Kemanggisan, Palmerah, Jakarta 11480, Indonesia
{ meiliana; lcdewi; achandra }@binus.edu

Received December 2018; accepted February 2019

Abstract. *Software testing is the most exhausting phase of software development life cycle that needs a lot of resource. Moreover, the current rapid development of mobile technology multiplies the efforts to test software in varying mobile devices. This challenge drives many researchers to implement automated testing that would increase efficiency of resource usage. The first stage of automated testing is to generate and optimize test cases automatically. Test case could be generated from requirement artifacts such as UML diagram to provide test case on early phase of software development process. There are two approaches to optimize the generated test case, by using mathematical or computational approach. This research aims to implement test case generation from combinational UML diagram and optimize the test case by using genetic algorithm. Simple example of login case is provided to simulate the proposed method. The experimental result of this paper proved that genetic algorithm implementation is able to optimize generated test paths by showing the fitness value.*
Keywords: Test case optimization, Genetic algorithm, Test case generation, Software testing

1. **Introduction.** Software testing is the most exhausting phase of software development life cycle that needs a lot of resource, such as time, effort and cost. Simple to medium system will have hundreds of test cases, while complex system might have thousands of test cases to be executed. Once software is changed (due to bug fixing, changes requirement or other causes) regression testing needs to be conducted that require all test cases to be re-executed one more time. Moreover, the current rapid development of mobile technology multiplies the efforts to test software in varying mobile devices (in terms of different possible operating system, screen size, and other compatibilities). This challenge drives many researchers to implement automated testing that would increase efficiency of resource usage, especially in regression testing.

Automated testing research could be divided into three parts, test case generation, test case optimization and development of tool or framework to automatically execute the testing. Test case generation could be derived from requirement artifacts to source code produced in software development process. However, deriving test case from requirement specification is more preferable to provide test case at early phase of development. Among all requirement artifacts, the Unified Modeling Language (UML) diagrams are the most common tools to design software. Thus, UML diagrams will be used as model to generate test cases in this research.

DOI: 10.24507/icicel.13.07.585

Test case optimization aims to optimize path coverage with less effort or time by choosing best test path. There are two approaches to doing test case optimization: mathematical and computational methods. Mathematical approach such as orthogonal array yields the best possible covering arrays in certain cases. However, most mathematical methods are limited to be applied in specific or restrictive sets or factors. Computational approach comprises greedy algorithm and heuristic optimization techniques of evolutionary algorithm. Greedy algorithms have been effective to find shortest path, but their accuracy is often trapped in local optima. On the other hand, implementation of evolutionary algorithm (such as genetic algorithm) for optimization problem shows promising result.

This paper aims to extend previous research [1] in generating test case by using greedy algorithm: depth first search algorithm. Test cases are generated from combination of two UML diagrams, sequence and activity diagram and will be optimized by using heuristic approach of genetic algorithm. The rest of this paper will be organized as follows. The second section will discuss about related works and basic concepts. Subsequent section describes the proposed method to generate test case from UML combination diagrams and explains detailed optimization process by using genetic algorithm. Proposed method will be simulated with simple example of login system in the fourth section. Conclusions and future works are given in the last section.

2. Related Works and Basic Concepts.

Test cases are defined as a set of condition or variables which determine the level of correctness and quality of the product. Simple way to present test case is by providing test path to be followed when conducting a testing. The studied literature shows there are various methods described by numerous researchers for generating test cases and comparing test case from different UML diagrams. We have classified the literature according to different aspects of testing from UML design using different UML diagrams.

Asad et al. [2] presented automated test case generation using UML class and sequence diagram. UML class and sequence diagram are converted into XML format by using Visual Paradigm and read by C# code. Another research of Khurana and Chillar [3] used state chart diagram that is converted into state chart graph and sequence diagram being converted into sequence graph. State chart graph and sequence graph are being combined into a graph called System Testing Graph (SYTG), and derived test case from this combination diagram. Many other researchers [4-8] use only single diagram of UML to generate test cases. This research uses SYTG diagram as combination from two UML diagrams: activity diagram and sequence diagram. Activity diagrams provide sequential flow of logic system, while sequence diagrams provide detail process involved object interaction that could provide all information needed to generated system testing graph.

Greedy search algorithms are used by several researchers to generate and validate test cases by prioritization. For example, Shanthi [9] implemented TABU search algorithm as an approach to generating test path automatically. Activity diagram is generated from software design, and then all possible information extracting using write parser in java. Based on the extracted information, an Activity Dependency Table (ADT) is generated. Test case is generated with the help of ADT by applying TABU search algorithm. Other implementation of greedy algorithm is depth first search algorithm by Tripathy and Mitra [10] and the modification of DFS by Meiliana et al. [1]. Application of greedy paradigm to generate test case is found to be effective on some cases, but lack of accuracy on some other cases. Therefore, heuristic approaches of evolutionary algorithm are highly considered in this research.

Evolutionary algorithm is a part of evolutionary computation in artificial intelligent, generic population-based that is focused on optimization problem. Two main types of evolutionary algorithm are genetic algorithm and swarm algorithm. Both algorithms temp to model biological evolution that contains explorer particle on a population to get the

fitness value. Fitness function evaluation is one of the solutions that overcome optimization problem in this paradigm. Genetic algorithm seeks solution of a problem in the form of strings of number by applying operators such as recombination and mutation. Swarm algorithm is inspired by colony of animal that cooperates to find source of food. There are several types of swarm algorithm based on their colony: Ant Colony Optimization (ACO), Bee Colony Algorithm (BCA), Particle Swarm Optimization (PSO) and cuckoo search.

Sahoo et al. [11] proposed hybrid colony algorithm from BCA and PSO for generating and optimizing the test case from activity and sequence diagram. Wu et al. [12] and Patidar [13] proposed improvement of PSO, Discrete PSO (DPSO), for covering array generation. Covering array generation is implemented for testing purpose to get best combinatorial test case. Hybrid algorithm of GA and PSO to generate test cases is conducted by Singla et al. [14] and Singh et al. [15]. Other researches try to implement basic evolutionary algorithm to generate, to prioritize or to optimize test case, such as PSO [16,17], bat-inspires algorithm [18], genetic algorithm [3] and ACO [19-21]. All results discussed on this research show promising approach to generate and optimize test cases. Hence, genetic algorithm is chosen as preliminary work to optimize our previous work in generating test case by using depth first search algorithm.

3. **Proposed Approach.** Proposed method in this paper used two UML diagrams: activity diagram and sequence diagram. Both diagrams are converted into graph diagram before combined into system testing graph. Integrating the two graphs is more appropriate to generate test case for system testing. From system testing graph, all possible test paths that show control flow sequence are generated by implementing DFS algorithm. Our previous research found some redundant nodes that are represented on several test paths; therefore, modification needs to be applied on this algorithm to getting better test cases. Although the experiment showed expected result, further study showed that the algorithm might not work well on other cases. Adopted from this limitation, enhancement by implementation of genetic algorithm is proposed to optimize the test cases. Figure 1 depicts method proposed in this paper.

Activity and sequence diagrams are converted into graph diagram by getting detail information of both diagrams from XML format. Activity diagram graph is generated by getting the information from activity diagram to form the node contain ID, activity

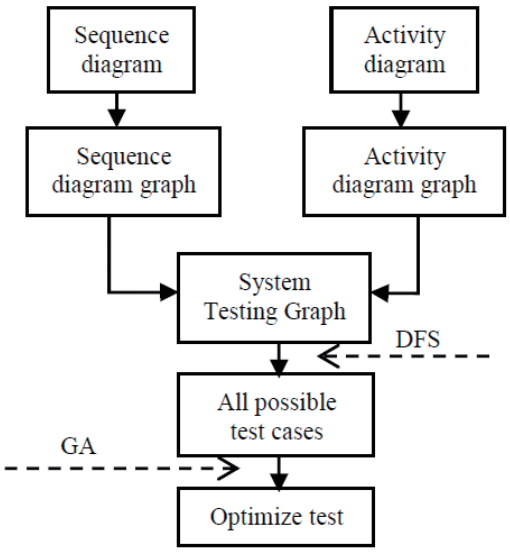

FIGURE 1. Optimization test

name, activity target, activity status. ID is a unique number used to identify each activity, activity name is the name of each activity, activity target is the next activity that we would use to make adjacency list, and the activity status is the status after passing the decision component. Sequence diagram node contains information of ID, message name, message target, message number, operand status. ID is a unique number used to identify each message, message name is the name of each message, message target is the next message that we would use to make adjacency list, message number is the sequence number of each message that we would use to reorder every message, and the operand status is the status of what alternative the message in. After we form the activity diagram graph and sequence diagram graph, the next step is to form the System Testing Graph (SYTG) by combining the activity diagram graph and sequence diagram graph using following algorithm.

SYTG algorihtm
input: Graph activity and sequence
output: List of combined graphs (LN)

```
LN ← activity_graph
foreach key, value in LN:
  if LN[key].adjacency_list length > 1 and LN[key].status = "To Sequence"
     curr_adj ← value.adjacency_list
     first_SDG ← SDG first key
     N[key] ← first_SDG
     exit
  foreach key, value in SDG:
     LN[key] ← value
  foreach key, value in LN:
     if value.adjacency_list length = 0
      if value.status is not null and value.status = true
         add curr_adj[1] to LN[key].adjacency_list
      elseif value.status is not null and value.status = false
         add curr_adj[0] to LN[key].adjacency_list
return LN
```

Depth first search algorithm is then applied to SYTG to generating all possible test paths as discussed at 1). To optimize generated test paths, we applied genetic algorithm with following rules.

1) All paths P = {p1, p2, p3, ...} from start node to final node of SYTG become a coverage input.
2) Assign weight for each node in every path. Normal weight is defined as one, while weight of parent node is sum of all child nodes.
3) Cost for each path is the total weight of all nodes inside the path.
4) Apply genetic algorithm with:
   - 40% truncation operator,
   - order based crossover operator,
   - insertion mutation operator,
   - population size of 75,
   - child percentage of 50% of each generation,
   - 0.33 mutation rate,
   - maximum stagnancy of 20 as a termination condition.
5) Best path is chosen based on the fitness value.

The following section will describe systematically our proposed method with simple case of login system.

4. **Case Study.** This section provides an experiment result of test case generation from activity graph, sequence graph, and SYTQ from one example case which is login case. Figure 2 and Figure 3 are activity diagram and sequence diagram that we use in this case from the previous work [1].

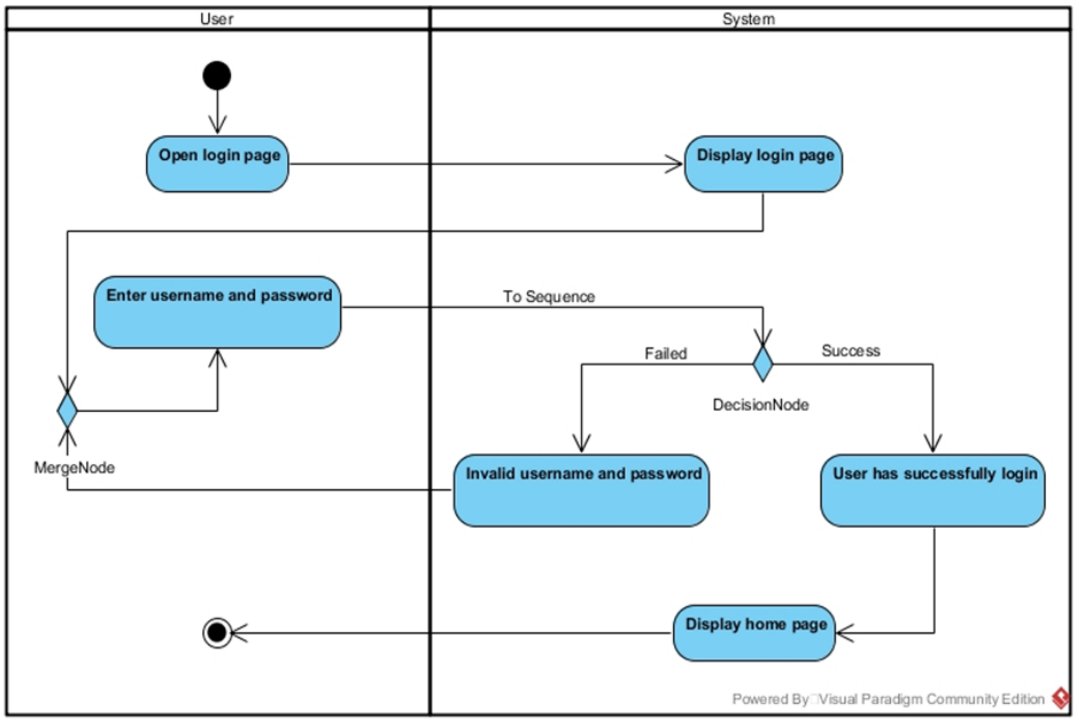

FIGURE 2. Activity diagram

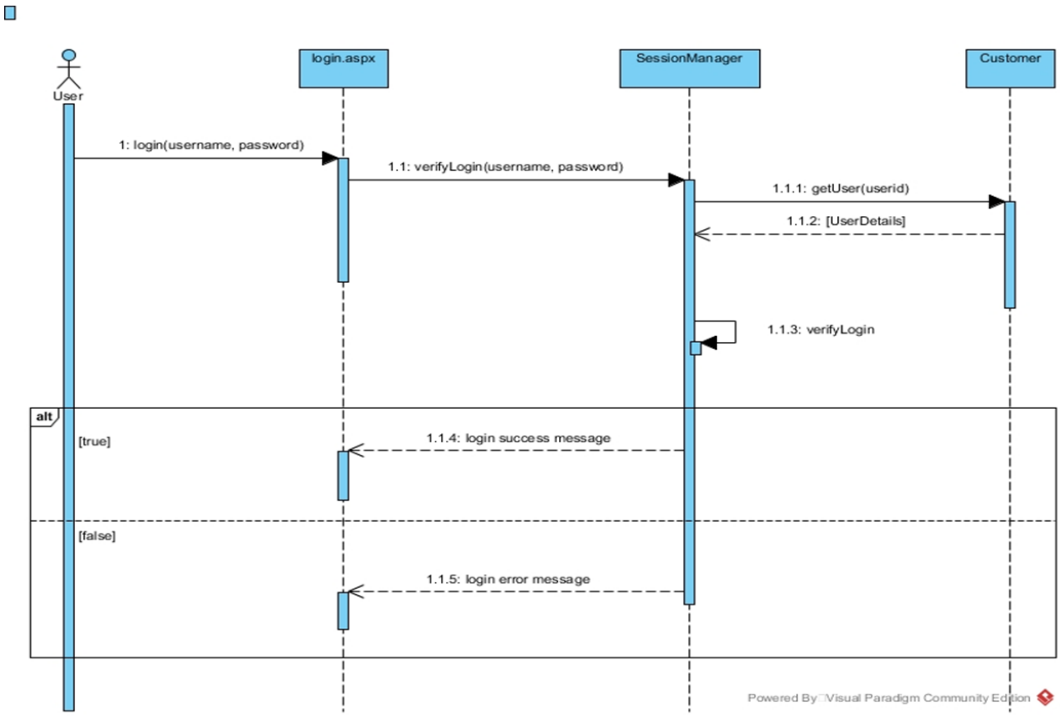

FIGURE 3. Sequence diagram

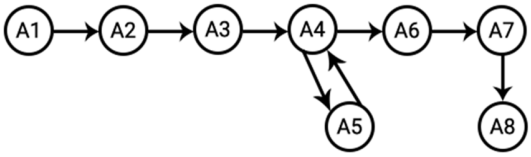

FIGURE 4. Activity diagram graph

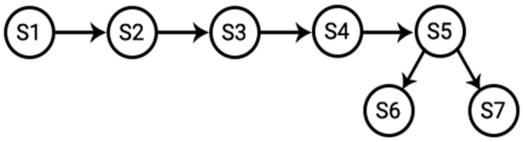

FIGURE 5. Sequence diagram graph

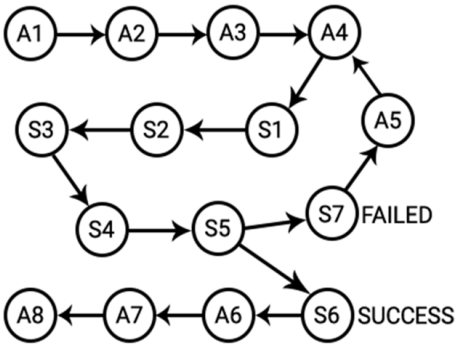

FIGURE 6. System testing graph

Both diagrams are converted into graph diagram as shown in Figure 4 and Figure 5, and then combined into system testing graph in Figure 6.

The possible test paths generated from system testing graph are:

- P1: A1 ⇒ A2 ⇒ A3 ⇒ A4 ⇒ S1 ⇒ S2 ⇒ S3 ⇒ S4 ⇒ S5 ⇒ S6 ⇒ A6 ⇒ A7 ⇒ A8, fitness value = 0.55
- P2: A1 ⇒ A2 ⇒ A3 ⇒ A4 ⇒ S1 ⇒ S2 ⇒ S3 ⇒ S4 ⇒ S5 ⇒ S7 ⇒ A5 ⇒ A4 ⇒ S1 ⇒ S2 ⇒ S3 ⇒ S4 ⇒ S5 ⇒ S6 ⇒ A6 ⇒ A7 ⇒ A8, fitness value = 0.62

Test path with maximum fitness value will be the optimum path. From this experiment, path 2 shows higher fitness value (0.62) compared with path 1 (0.55). Thus, we could conclude that path 2 is the optimum path that has maximum coverage.

5. **Conclusion and Future Works.** Testing is an important process in software development life cycle to ensure software quality. However, the huge resource for conducting complete testing most likely could not be afforded. Therefore, automate testing needs to be conducted to facilitate this challenge. One way to have efficient testing is by automating test case generation and optimizing the possible test cases. This paper presented a method to generate test case from system testing graph as combination of UML activity and sequence diagram. All possible test paths are optimized by applying genetic algorithm. From the aforementioned experiment, diagrams conversion is correctly generated from both UML diagrams, as well as the integration process into system testing graph. The implementation of genetic algorithm is able to optimize generated test paths by showing the fitness value. This experiment is a preliminary work to implement GUI testing that produced huge number of combinatorial test cases. Unlike simple example presented in this paper, the real implementation of genetic algorithm will have significant

impact to optimize test cases and affect the testing efficiency process. The future work of this research is to compare other hybrid evolutionary algorithm that could improve the test cases generation and optimization.

REFERENCES

[1] Meiliana, I. Septian, R. S. Alianto, Daniel and F. L. Gaol, Automated test case generation from UML activity diagram and sequence diagram using depth first search algorithm, *Procedia Computer Science*, vol.116, 2017.

[2] S. Asad, A. Shah, R. K. Shahzad, S. Shafique, A. Bukhari and M. Humayun, Automated test case generation using UML class & sequence diagram, *British Journal of Applied Science & Technology*, vol.15, no.3, pp.1-12, 2016.

[3] N. Khurana and R. S. Chillar, Test case generation and optimization using UML models and genetic algorithm, *Procedia Computer Science*, vol.57, pp.996-1004, 2015.

[4] J. P. Faria, A. Paiva and Z. Yang, Test generation from UML sequence diagrams, *IEEE the 8th International Conference on the Quality of Information and Communications Technology*, pp.245-250, 2012.

[5] R. K. Swain, V. Panthi and P. K. Behera, Generation of test cases using activity diagram, *International Journal of Computer Science and Informatics*, vol.3, no.2, 2013.

[6] V. Panthi and D. P. Mohapatra, Automatic test case generation using sequence diagram, *Proc. of International Conference on Advances in Computing*, vol.2, no.4, pp.22-29, 2012.

[7] A. Hettab, A. Chaoui and A. Aldahoud, Automatic test cases generation from UML activity diagrams using graph, *The 6th International Conference on Information Technology*, 2013.

[8] V. M. Sumalatha, Object oriented test case generation technique using genetic algorithms, *International Journal of Computer Applications*, vol.61, no.20, pp.20-26, 2013.

[9] A. V. K. Shanthi, A novel approach for automated test path generation using TABU search algorithm, *International Journal of Computer Applications*, vol.48, no.13, pp.28-34, 2012.

[10] A. Tripathy and A. Mitra, Test case generation using activity diagram and sequence diagram, *Proc. of International Conference on Advances in Computing*, 2012.

[11] R. K. Sahoo, D. Ojha, D. P. Mohapatra and M. R. Patra, Automated test case generation and optimization: A comparative review, *International Journal of Computer Science & Information Technology*, vol.8, no.5, pp.19-32, 2016.

[12] H. Wu, C. Nie, F. C. Kuo, H. Leung and C. J. Colbourn, A discrete particle swarm optimization for covering array generation, *IEEE Trans. Evol. Comput.*, vol.19, no.4, pp.575-591, 2015.

[13] C. Patidar, Research paper test case generation using discrete particle swarm optimization algorithm, *International Journal of Scientific Research in Computer Science and Engineering*, no.1, pp.38-42, 2013.

[14] S. Singla, D. Kumar, H. M. Rai and P. Singla, A hybrid PSO approach to automate test data generation for data flow coverage with dominance concepts, *International Journal of Advanced Science and Technology*, vol.37, pp.15-26, 2011.

[15] A. Singh, N. Garg and T. Saini, A hybrid approach of genetic algorithm and particle swarm technique to software test case generation, *International Journal of Innovations in Engineering and Technology*, vol.3, no.4, pp.208-214, 2014.

[16] A. Ganjali, *A Requirements-Based Partition Testing Framework Using Particle Swarm Optimization Technique*, Master Thesis, University of Waterloo, 2008.

[17] T. J. Sahib, Z. Jano, I. H. Mohamed and U. Teknikal, Optimum allocation of distributed generation using PSO: IEEE test case studies evaluation, *Int. J. Appl. Eng. Res.*, vol.12, no.11, pp.2900-2906, 2017.

[18] M. M. Öztürk, A bat-inspired algorithm for prioritizing test cases, *Vietnam J. Comput. Sci.*, vol.5, pp.45-57, 2017.

[19] C. Ping and X. Min, Software testing case generation of ant colony optimization based on quantum dynamic self-adaptation, *International Journal of Hybrid Information Technology*, vol.8, no.9, pp.95-104, 2015.

[20] H. Li and C. P. Lam, Software test data generation using ant colony optimization, *Trans. Engineering, Computing and Technology*, vol.1, no.1, pp.1-4, 2004.

[21] C. Loiola, B. Maia, N. Ferreira and F. G. De, An ant colony based algorithm for test case prioritization with precedence, *Proc. of the 3rd Int. Symp. Search Based Softw. Eng.*, vol.6956, pp.10-12, 2011.