IJICTR International Journal of Information & Communication Technology Research

Volume 6- Number 2- Spring 2014 (29-40)

Automatic Test Case Generation for Modern Web Applications Using Population-Based Automatic Fuzzy Neural Network

Mohammad Reza Keyvapour
Computer Engineering Department
Alzahra University
Tehran, Iran
keyvanpour@alzahra.ac.ir

Hajar Homayouni
Computer Engineering Department
Alzahra University
Tehran, Iran
Homayouni.hajar@student.alzahra.ac.ir

Received: November 9, 2013- Accepted: February 8, 2014

Abstract—**Automatic test case generation is an approach to decrease cost and time in software testing. Although there have been lots of proposed methods for automatic test case generation of web applications, there still exists some challenges which needs more researches. The most important problem in this area is the lack of a complete descriptive model which indicates the whole behaviors of web application as guidance for the generation of test cases with high software coverage. In this paper, test cases are generated automatically to test web applications using a machine learning method. The proposed method called RTCGW (Rule-based Test Case Generator for Web Applications) generates test cases based on a set of fuzzy rules that try to indicate the whole software behaviors to reach to a high level of software coverage. For this purpose a novel machine learning approach based on fuzzy neural networks is proposed to extract fuzzy rules from a set of data and then used to generate a set of fuzzy rules representing software behaviors. The fuzzy rule set is then used to generate software test cases and the generated test cases are optimized using an optimization algorithm based on combination of genetic and simulated annealing algorithms. Two benchmark problems are tested using the optimized test cases. The results show a high level of coverage and performance for the proposed method in comparison with other methods.**

Keywords- Automatic Test case generation; web applications; population-based Fuzzy Neural Network

I. INTRODUCTION

Using web is one of the most important, unavoidable, and advantageous ways for business, learning and information gathering. As web progresses, web applications are becoming more popular than desktop applications. Therefore, their quality verifications are of a higher importance. An efficient way for software verification is software testing which is applied during software development cycle repeatedly hence it is a costly process. An essential and costly step in software testing is test case generation [1]. An efficient test suite should be able to cover different features of software. Test case generation automation can effectively reduce the time and costs consumed for the whole software testing process. Although there have been lots of researches to produce test cases with high software coverage capability [2], there still exists challenges in this area especially for web applications with their particular properties. One of the most important problems is not presenting an all-around model which can be able to illustrate all web application features for generating test cases. Existing methods generate test cases based

on an incomplete model that lead to sets of test cases with a low level of software coverage. Accordingly, in this paper, a new learning method based on fuzzy neural networks [3] is presented to extract the web application behaviors as a fuzzy rule set, and then uses this rule set as a model for web application to generate test cases with high code coverage. The generated test cases are then optimized using an algorithm composed of genetic and simulated annealing. The optimized test cases are applied on two benchmark web applications called TUDUList and BlindTextGenerator and compared with other approaches. The evaluations show a higher performance, coverage and automation for the proposed method.

II. Related Works

Web applications have been evolved through last decade to satisfy requirements of different users. Its evolution process started from a simple static page-sequence client/server system into a dynamic medium of user-created content and rich interaction. The complete evolution steps are discussed in [4]. Modern web applications (web 2.0 applications) as dynamic medium of user-created content and rich interaction rely on asynchronous client/server communication. Rich Internet Applications are usually developed using Web 2.0 technologies, such as Ajax, Silverlight, or Flex [5]. Among the technologies that are being developed to implement its features, Ajax[1] is one of the most promising and mature. Ajax Web applications are heavily based on asynchronous messages and DOM manipulation. Accordingly the faults associated with these two features are relatively more common than in other kinds of applications. Thus, most Ajax testing techniques are directed toward revealing faults related to incorrect manipulation of the DOM [6]. For these applications, test cases are sequences of events that are potentially fault prone [7]. The testing techniques applied to traditional multipage web applications are not sufficient for testing modern single page web applications. In more detail, the properties of Ajax make them extremely difficult to test. Static analysis techniques are not able to reveal many of the dynamic dependencies present in modern web applications. The highly dynamic nature of Ajax user interfaces and their client/server delta communication adds an extra level of complexity to the traditional web analysis and testing challenges [4]. Furthermore, traditional web testing techniques are based on the traditional page request/response model and not taking into account client side functionality [8]. In [9] the adequacy and effectiveness of the most famous web testing methods are applied to these modern web applications and results shows that they are inadequate for testing AJAX-based web applications. An all-around classification framework for both traditional and modern web application test case generation techniques is presented in [10]. To bridge the gap between current web testing techniques and the main new features provided by AJAX, [9] proposed a Model-based testing technique represented by a FSM to describe the behavior of an AJAX Web page, according to its DOM structure and content in which nodes represent DOM states and edges are events which modify the DOM. To extract the FSM model of a web application, different methods have been proposed using execution traces [6], crawlers [11, 12] or user sessions [5]. WebMate [13] as a technique for generating test case for popular web 2 applications like social networks uses an initialized finite state automaton to generate test cases and tries to visit all states of the so-called usage model. The proposed mechanism in [14] makes use of a crawler to capture the client side fields and create a state-flow which is a basis for the completion of automatic testing. These techniques can automatically generate many test cases, whose effectiveness depends on the completeness of the initial model. When the initial model is obtained by stimulating the application under test with a simple sampling strategy that uses a subset of GUI actions to navigate and analyze the DOM, the derived model is partial and incomplete, and the generated test cases necessarily overlook the many interactions not discovered in the initial phase.

III. Proposed Method

Existing methods to test previous web applications are not applicable for testing the new generation of web applications which are based on web2 technology [9]. In other words, special features of these applications make their testing process much harder. Static analysis cannot discover the dynamic communications of these applications. On the other hand, their dynamic nature and particular client/server communications in these applications make them more complicated and cause new challenges for their test [4]. Moreover, the classical testing techniques were based on a request/response model between pages and didn't consider the functionality of client side [12]. Due to the fact that new web applications are based on asynchronies messages and DOM manipulations, errors related to these two features are more common. Therefore, most of techniques in this area consider detecting of errors caused by DOM manipulations directly [6]. Hereupon in this research test cases are generated using DOM. In other words, test cases are introduced as a sequence of events that change the DOM structure. Existing test case generation method for web2 applications are based on an incomplete model of application which is mostly described as a finite state graph that doesn't represent all software behaviors. Accordingly, the proposed method in this paper tries to first extract a complete model of web application and then generate test cases based on this model. To extract the web application model, a new neural network called PAFuNN is presented. This model is utilized for generating web applications test cases.

Figure 1 shows the overall architecture of Rule-based Test Case Generation for Web Applications (RTCGW). To extract DOM structure there is a need for client-side code of web application. Thus, in RTCGW, client-side code is defined as input and a test suite (a set of test cases) is defined as output.

[1] Asynchronous Javascript And XML

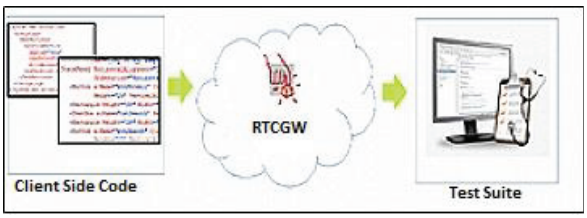


Fig. 1 The overall architecture of proposed method

According to figure 2 automatic test case generation system is composed of two subsystems. The first subsystem is responsible for generating an overall fuzzy rule set model of web application by dynamic analysis of software and the second one generates test cases based on his model.

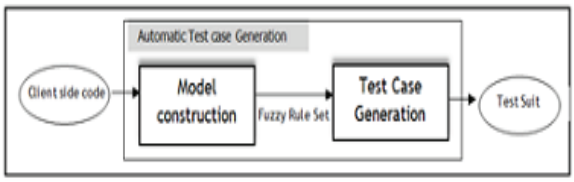


Fig. 2 The block diagram of RTCGW

3-1-Model construction subsystem

Due to the high complexity of modern web applications, accessing to all states are almost impossible [15]. On the other hand, the main goal in software testing is to verify all possible states of software. In recent researches [6, 15, 16] the web applications are modeled using a finite state graph in which nodes represent software states and edges illustrate the factors which cause a transition between states (like events, links, …). Most of the time this graph is generated according to some execution traces which lead to incomplete and inefficient model. Consequently, the generated test cases are also incomplete and don't cover all testing purposes. Accordingly, in this research a set of fuzzy rules is used to predict all application behaviors and states according to its current state. Therefore, the first goal of this paper is to generate a fuzzy rule set that model all application's states. The block diagram for this system is illustrated in figure 3.

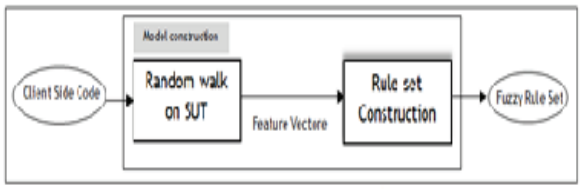


Fig. 3 Model construction block diagram

This subsystem has two components. In the first component, a random walk is applied in different states of web application to generate training data of PAFuNN. For this purpose, the client-side code of application is analyzed and different events are executed randomly. Information such as hash code of current state, events and input data, and hash code of next state are stored a vector and sent to the next component to generate fuzzy rule set from.

3-1-1- first component: random walk on system under test

The main purpose of this component is to generate input data for the next component. As mentioned before, due to the different nature of modern web applications, their navigation is different from moving between pages, but means moving from one DOM state to another. As a result, to crawl through different states of a modern web application, we need to execute it to see the changes in DOM states dynamically. To proceed, the method proposed by Ali Mesbah is used [12]. In this method a database is used to initialize the input elements of DOM [6]. Generally speaking, the processes needed for random walk on web application consists of: 1) DOM extraction from client-side code, 2) DOM analysis to select executable and clickable elements randomly. Table 1 shows some examples of executable elements and their events.

TABLE 1. EXECUTABLE ELEMENTS AND THEIR EVENTS

Component	Event
Button	Onclick
Image	Onclick, onMouseOver
Textarea	On key press
checkbox	Onselect, onclick
Link	Onclick
Radio	Onselect

3) Executing selected elements, and fill the input values by data in database. This leads to a change in DOM state. 4) Calculating hash code of current and next states as an indicator for each state. 5) Initializing input-output vector as table 2 shows.

TABLE 2. INPUT-OUTPUT VECTOR RESULTED FROM RANDOM WALK ON WEB APPLICATION

Input					Output
Element Types	Event Types	Input data Types	Current DOM State Index	Current DOM State hash code	Next DOM State hash code

Table 3 shows different types of input, events and elements with a code related to each one to be stored in the input-output vector. As an example, table 4 shows that if the state of DOM is 1 and we click on a link we go to state 7.

TABLE 3- CODES RELATED TO ELEMENTS, INPUTS AND EVENTS

Element Types	code	Event Types	Code	Input data	code
button	000	click	00	String	00
image	001	MouseOver	01	Numeral	01
Textarea	010	keypress	10	Both	10
checkbox	011	select	11	Non	11
Link	100				
Radio	101				

TABLE 4- AN EXAMPLE FOR INPUT-OUTPUT VECTORS

Element Types	Event Types	Input data Types	Current DOM State Index	Current DOM State hash code	Next DOM State hash code
100	00	11	1	10	7

3-1-2- second component: Fuzzy rule set generation

In this component a set of fuzzy rule set representing software states is generated. In next step, this set is utilized to predict different states of web application and consequently to generate test cases with high coverage capability. The (X, Y) input-output vector generated from previous component is now used to train PAFuNN and to generate a set of fuzzy rules.

- PAFuNN for extracting fuzzy rule set

In this part free from the paper goal, a novel method is presented generally for extracting fuzzy rules from a set of data. Population-based Automatic Fuzzy Neural Network is a five layer neural network like Evolving Fuzzy Neural Network (EFuNN) [17] which is appropriate for online knowledge discovery from large databases.

EFuNN has a five-layer structure including the rule layer, the condition-to-rule connection layer and the rule-to-action connection layer beside input and output layers.

The first layer of EFuNN is the input layer and the second layer represents fuzzy quantization for the input variables. The third layer (rule layer) evolves through supervised/unsupervised learning. There are two vector connection weights for each rule node r – $W_1(r)$ and $W_2(r)$. $W_1(r)$ is adjusted through unsupervised learning based on similarity measure within a local area of the problem space. $W_2(r)$ is adjusted through supervised learning based on the output error. The fourth layer represents fuzzy quantization for the output, and the fifth layer represents the real values for the output variables.

Each rule node r_j represents an association between a hyper sphere of fuzzy input space and that of fuzzy output space. R_j is the radius of the input hyper sphere of a rule node r_j which is defined as R_j=1-S_j, where S_j is the sensitivity threshold parameter and it defines the minimum activation of rule node r_j to a new example (x,y) in order the example to be considered for association with this rule node. Two conditions should be satisfied in order the pair of fuzzy input-output data vectors (X_f, Y_f) to be allocated to the rule node r_j: (1) a local normalized fuzzy difference between X_f and $W_1(r_j)$ is smaller than radius r_j, (2) the normalized output error is smaller than an error threshold.

As a new sample is presented to the network, it is first fuzzified at the fuzzy input layer. The fuzzy distance between the fuzzified input and the connections weights W_1 is then calculated as follows in order to determine whether the input example falls into input receptive field of some specific rule node or not.

$$FD(x_f, w_{1,j}) = \frac{\|x_f - w_{1,j}\|}{\|x_f + w_{1,j}\|} \quad (1)$$

$W_1 = [W_{1,1}\ W_{1,2} \dots W_{1,N}]$,

The rue node with the highest activation (Eq.2) or in other words with the lowest FD value is selected.

$$A_1 = [A_{1,1}\ A_{1,2} \dots A_{1,N}]^T \quad (2)$$

where $A_{1,j} = 1 - FD(x_f, w_{1,j})$

If the activation value of selected rule node is less than the sensitivity threshold, a new rule node is created and new connections weights are established for it as follows.

$$W_1(N+1) = X_f \quad , W_2(N+1) = Y_f \quad (3)$$

Otherwise W_1 is adjusted through unsupervised learning based on similarity between the fuzzy input and the previous connection weights for the jth rule node.

$$W_{1,j}(t+1) = W_{1,j}(t) + \eta_j (W_{1,j} - X_f) \quad (4)$$

Where η_j is the learning rate of the jth rule node which can be defined as

$$\eta_j = {}^1/_{ACC_j} \quad (5)$$

Where ACC_j is the accumulated number of accommodated examples for the jth rule node.

The output of fuzzy output layer A2 is computed via Eq.6

$$A_2 = satlin(W_2 \cdot A_1) \quad (6)$$

Where satlin() is the saturating linear transfer function.

If the fuzzy output error is larger than a pre-defined threshold value a new rule node will be created similarly.

$$FE_{out} = \|A_2 - y_f\| \quad (7)$$

Otherwise W_2 is updated according to the Widrow-Hoff Least Mean Square (LMS) algorithm in a supervised manner.

$$W_{2,j}(t+1) = W_{2,j}(t) + \eta_j (A_2 - Y_f) \cdot A_{1,j} \quad (8)$$

Finally output value Y_c can be derived by Eq.9

$$Y_c = W_3 \cdot A_2 \quad (9)$$

The proposed PAFuNN has the advantageous of EFuNN. However, this method tries to overcome

some existing challenges in EFuNNs. The most important challenges are:

- Lack of control on the number of rule nodes: EFuNN decides to add new neurons only based on the current input sample which leads to :
  - A high number of neurons in network which causes high level of complexity, so that some existing approaches like pruning or integrating nodes cannot be enough to optimize network architecture.
  - Rule insertion is based on a local view depending only on one example each time.
- Updating thresholds: sensitivity and error thresholds have effective roles in the functionality of network which are set to constant values in basic EFuNN. Improvement of these parameters during network learning can have a deep impact on the performance of network.

a) Control on number of neurons

In PAFuNN after some epochs, when a particular number of neurons are generated, if an input sample doesn't match with any of existing neurons (i.e. the two EFuNN conditions aren't satisfied for that input [14]), it is stored and finally neurons are generated based on a population of rejected stored examples. Otherwise, network parameters are updated in a supervised/unsupervised manner as same as EFuNNs. The stored examples are first sorted according to their fuzzy distance (Eq. (10)).

$$FD(E_{f1}, E_{f2}) = \frac{\|X_{f1} - X_{f2}\|}{\|X_{f1} + X_{f2}\|} \qquad (10)$$

Next, the sorted samples are classified to different subsets according to the sensitivity threshold why this threshold shows the radius of a neuron sphere. Finally, for each subset a neuron is inserted into the rule layer of network and its parameters are adjusted as same as EFuNN method.

b) Updating thresholds

In proposed method, two learning automata with fixed structures are utilized to adjust two network thresholds based on network response to each input sample. A learning automaton is decision maker which is placed in a random environment and choose the optimum activity by repeated interactions with its environment and based on the environment responses [18].

Two fixed structured learning automata (FSLA) are connected to rule layer and output layer of network to adjust sensitivity and error thresholds for those layers. It should be noted that PAFuNN is the environment for learning automata. The automata activities are choosing best values for thresholds. The environment response is an award to first automata if the activity value of selected rule node (the rule node with the highest activity) is more than sensitivity threshold, and the environment response is an award to second automata if fuzzy output error is less than error threshold. Figure 4 shows the internal connection of PAFuNN and FSLA.

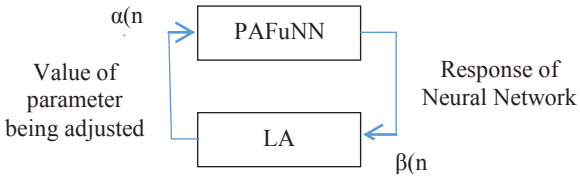


Fig. 4 Internal connection of learning automaton to PAFuNN

3-2- Test case generation subsystem

In this subsystem, test cases are generated using fuzzy rule set model without a need for executing web application. The main goal of this part is to generate test cases with high level of software coverage. According to the state explosion problem, there is a need for optimizing the generated test cases. Figure 5 shows the block diagram of test case generation subsystem.

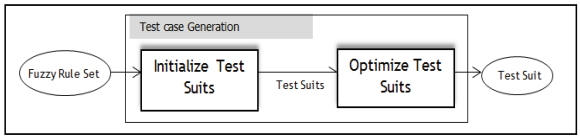


Fig. 5 test case generation subsystem block diagram

3-2-1- First component: initializing test suite

In this step test cases are initialized using a semi-random approach. Each test suite has at last k test cases. Each test case is a input-output vector like the one illustrated in table 5 in which software is in state *a* and do some action with *x* features and go to state *b*.

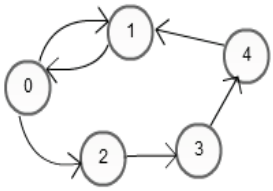
TABLE 5- THE STRUCTURE OF ONE TEST CASE

Element Types	Event Types	Input data	Current DOM state	Next DOM state
	X		A	b

A test suite is defined as a matrix with first state equal to the first state of application. An example of a test suite with 7 test cases is illustrated in figure 6 (a). Figure 6 (b) shows the state transitions for the test suite. In the semi-random approach, the current state of first test case of a test suite is set to the first state of the web application. Other elements of test case are initialized randomly. The next state of the test case is predicted by PAFuNN. For the second test case of the test suite the current state is set to the predicted value by PAFuNN and other elements are initialized randomly. These approaches are applied till all *k* test cases of a test suite are initialized. The process continues to fulfill *N* test suites semi-randomly.

Input data	Event Types	Element Types	Current state	Next state
100	00	11	0	1
010	01	11	1	0
001	11	00	0	2
101	10	11	2	3
000	00	01	3	4
010	10	10	4	1
000	11	11	1	0

(a)



(b)

Fig. 6 (a) Example of a test suite, (b) State transition for the test suit

3-2-2- Second component: Test suite optimization

The main goal of this component is to optimize test suites to achieve a test suit with a high level of coverage and performance. For this purpose various algorithms such as genetic, Memetic, simulated annealing and the combination of genetic and simulated annealing (GA+SA) is used to find the most effective one. In the proposed GA+SA algorithm, in each generation of genetic algorithm the solution is optimized using simulated annealing in two different ways: 1- in each generation of GA the solution with highest amount for fitness is optimized by SA, 2- in each generations the selected parents are optimized by SA after imposing GA operations.

Furthermore, the fitness function and operators of GA are as follows:

Fitness function: this function calculates the average of coverage related to test cases of a test suite. In this research the number of different states in a test suite is considered as the coverage criteria for that test suite. The more is the number of different states in a test suite, the more coverage it causes.

$$\text{Fitness(TS)} = number\ of\ distinct\ hash\ codes\ in\ the\ TS \quad (11)$$

GA operators: 1- selection: the chromosomes (i.e. test suites) are selected using roulette wheel in which solutions with higher fitness value have a higher chance for being selected. 2- crossover: two different crossovers are used in this research i.e. horizontal and vertical that incorporates chromosomes horizontally or vertically. It should be noted that PAFuNN is used for indicating the next state of test cases after each crossover.

IV. Evaluations

First the web applications to be evaluated are introduce and the evaluation criteria are indicated. Finally the results of applying proposed method for test case generation of web applications are illustrated and compared with other existing methods.

4-1- Web applications to be tested

Existing test case generation methods apply their method on benchmark problems i.e. famous web applications on this area [19, 20, 21]. For this purpose, web applications are run using generated test cases and the resulted coverage is analyzed and compared with results obtained from other methods. In this paper two benchmark web applications are chosen as follows to apply test cases and compare results.

- TUDULis [2] web application: Our first experimental subject is the Ajax-based open source TUDU web application for managing personal todo lists, which has also been used by other researchers [6, 12, 15]. The server-side is based on J2EE and consists of around 12K lines of Java/JSP code, of which around 3K forms the presentation layer we are interested in. The client-side extends on a number of Ajax libraries such as DWR and Scriptaculous, and consists of around 11k LOC of external JavaScript libraries and 580 internal LOC.
- Blind Text Generator[3] web application: the second experiment is an Ajax-based software which helps you create dummy text for all your layout needs, which has also been used by other researchers [12, 16]. Dummy text is useful for publication industry or web designers to be fulfilled by real text. It is especially useful when the real data is not accessible. Due to the fact that the number of states is high in this application, it is a good choice for our evaluations.

4-2- Evaluation Technique

To evaluate proposed method, the same technique in [15] is utilize. In this technique, the software under test is run using generated test cases. During this process all evaluation criteria are calculated. This is important to note that the input elements of a DOM is initialized randomly from a prepared database same as [12].

4-3- Evaluation Criteria

There have been various criteria to evaluate test cases generated for web applications. In a general view, we categorize the evaluation criteria into three general classes.

1) Performance: to calculate the performance of test suite different objects should be taken into consideration [22, 23] such as:

- DOM string size:

$$DOMStringSize = \sum_{TC=1}^{Size\,(TS)} Size(DOMString) \quad (12)$$

- Number of candidate elements executed during testing phase:

$$CandidateElements = \sum_{TC=1}^{size\,(TS)} (EntryPoints + ClickableElements) \quad (13)$$

- Number of detected states during test phase:

$$DetectedStates = \sum_{TC=1}^{Size\,(TS)} States \quad (14)$$

Accordingly the performance is calculated as:

$$Performance = DetectedStates + CandidateElements + DOMStringSize \quad (15)$$

[2] http://tudu.sourceforge.net
[3] www.blindtextgenerator.com

2) Automation level: This is analyzed by calculating manual and automatic efforts in terms of time for test case generation. To illustrate, time consumption for both automatic and manual works needed in whole test case generation process are considered to calculate the automation level of proposed method. For instance, in the random walk step of RTCGW the first state of DOMs is initialized manually by entering username and password for TUDUList problem. In this case, an average time of 1 minute is considered as a manual effort. Automation effort is then calculated by subtracting Manual efforts from the total time of test case generation process.

$$AutomationLevel = \frac{Automatic\ efforts}{Automatic\ efforts + Manual\ efforts} \quad (16)$$

3) Coverage: In this research client-side code coverage is taken into consideration as many other researches [25, 25].

$$CodeCoverage = \%Client\ Side\ Code\ Visited\ During\ Test \quad (17)$$

4-4- Methods to be compared with

- AUTUSA: the method is proposed by Ali Mesbah in 2011 [12], and is based on a graph model of web application. It uses a tool called CrawlAjax to generate state graph from client-side code. Test cases are generated using the generated graph. Due to its novelty and model-based nature of this approach, it can be an appropriate candidate to be compared with the proposed method in current research.
- US-CR: This is also a method based on graph model of web application [26]. Generally speaking, it generates test cases in three phases: 1) collecting a set of execution traces, 2) test case generation and 3) test case reduction. To collect execution traces it uses two methods consist of crawler (CR) and user session-based (US). Finally it uses a composition of two methods (CR+US) and compares the obtained results. Due to its complete results and model-based nature, it is also a suitable approach to be compared with the proposed method.

4-5- Running steps

- First Experiment: Test case generation for TUDUList web application

First step: random walk on software: in this step the number of runs is set to 30 runs.

Second step: PAFuNN training and generating fuzzy rule set: the result of this step is illustrated in figure 7 and table 6.

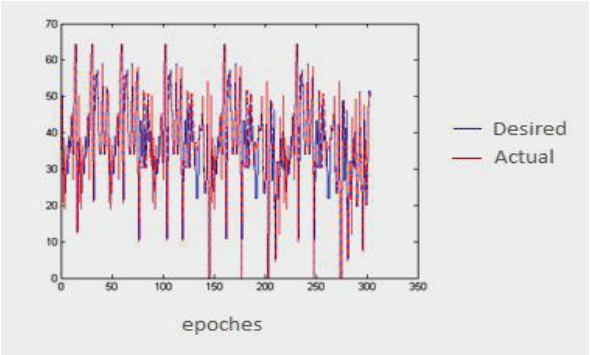


Fig. 7 PAFuNN predicted and real values during Training phase on input-output vectors obtained from random walk on TUDUList

TABLE 6- THE RESULTS OF TRAINING PAFUNN ON DATA FROM RANDOM WALK ON TUDULIST

Rule nodes	57
Epochs	**1**
Training time (CPUTime)	**3.2**
Root Mean Square Error	**5.8**

Third step: Test Case Generation: In this step test cases are initialized in a semi-random way. The result is 100 test suites with a maximum of 30 test cases in each test suite. Next the test suites are optimized using mentioned algorithms. The test suite with maximum fitness value is sent to next step.

Forth step: evaluations: In this part the results of applying generated test suite on TUDU problem is analyzed. First of all the results of executing optimization algorithm to obtain best test suite is illustrated in table 7 and figure 8. Then the results of applying RTCGW method when different optimization algorithms are used are shown in table 8 and figure 9. The number of test cases in a test suite is set to 30.

TABLE 7- FITNESS AND AVERAGE TIME RESULTED FROM USING DIFFERENT OPTIMIZATION ALGORITHMS

Algorithm	Statistics (fitness)	Avg. (time (sec))
GA	Min = 42 Max = 55.2 Avg. = 48	11
MA1	Min = 45 Max = 61.12 Avg. = 52.06	44.59
MA2	Min = 46 Max = 54 Avg. = 47	65
(GA+SA)1	Min = 47 Max = 51 Avg. = 49.04	53.75
(GA+SA)2	Min = 47 Max = 61.12 Avg. = 52.28	134.5
SA	Min = 32 Max = 50.16 Avg. = 32	7
HC	Min = 32 Max = 32.94 Avg. = 32	7

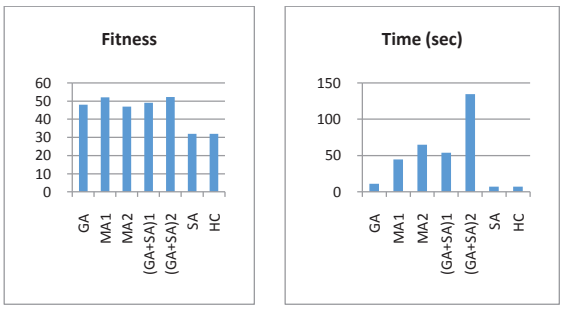


Fig. 8 Fitness and average time resulted from using different optimization algorithms

As it is obvious in figure 8, maximum fitness value is related to GA+SA with second policy for utilization of both algorithms features. While genetic algorithm is going to obtain best solutions, simulated annealing tries some bad solution to escape from local optimum. The minimum value is for hill climbing as a local search algorithm and it seems that it returns a local optimum as the final solution. Secondly, the Memetic algorithm could receive to a high fitness because of using a local optimization method beside a global one and successfully escape from local optimum. Time spent for GA+SA with second policy has the highest value, since in each generation during genetic evolution simulated annealing is called for two times (i.e. for two chromosomes). This time is less for first policy of composing genetic with simulated annealing because of one call for simulated annealing in each generation for this method. On the other hand, time spent for hill climbing has the minimum level of value and it is because of its simplicity.

TABLE 8- THE RESULTS OF APPLYING RTCGW METHOD WHEN DIFFERENT OPTIMIZATION ALGORITHMS ARE USED

Algorithm	#test cases	Performance	Automation level	Code Coverage
GA	30	158.79	0.97	**0.77**
MA1	30	169.80	0.97	**0.78**
MA2	30	166.38	0.97	**0.78**
(GA+SA)1	30	148.22	0.97	**0.77**
(GA+SA)2	30	180.79	0.97	**0.80**
HC	30	157.90	0.97	**0.76**
SA	30	152.68	0.97	**0.76**

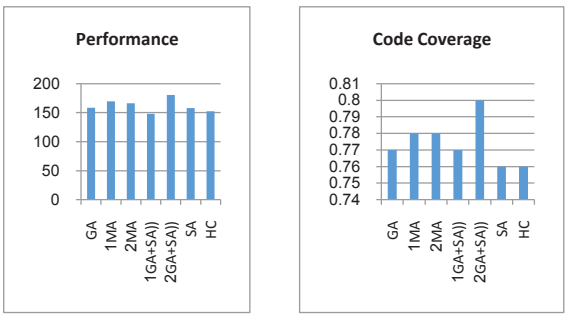


Fig. 9 The results of applying RTCGW method when different optimization algorithms are used

Clearly the results show that the more is the fitness of algorithm, the better are other evaluation criteria and this illustrates the correctness of fitness function selected in this study. While (GA+SA)2 leads to a test suite with the highest coverage and highest performance on TUDUList web application, hill climbing has the lowest results and it is exactly for the reasons described in previous section. The same analyze is also correct for Memetic algorithm with the second highest value. Due to the fact that the highest level of performance and code coverage is related to GA+SA with second policy, this algorithm is used as optimization method for test suites and the results are compared with US, CR and US+CR methods (table 9 and figure 10).

TABLE 9- COMPARE PROPOSED METHOD WITH CR, US AND CR+US

Method	#test case	Performance	Automation level	Code Coverage
RTCGW	30	40+(46+29)+65.79	46/(46+1	0.8
US	21	19+81+6.015	15/(15+20)	0.33
CR	203	14+42+6.015	15/(15+20	0.40
US+CR	224	19+81+6.015	15/(15+20	0.58

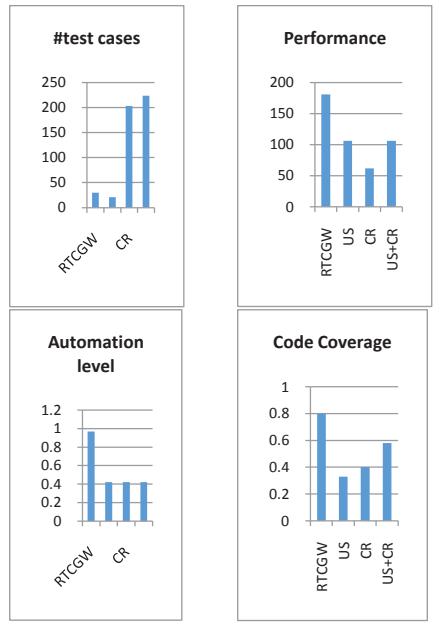


Fig. 10 Comparison between proposed method and US, CR, US+CR based on performance, code coverage and automation level

According to results, it's obvious that RTCGW has a better performance in comparison with others while having less number of test cases. Moreover, the level of code coverage is higher than other methods. The manual effort required for this method is the lowest among other methods because its automation level is the most. The three other methods have the same level of automation for using same strategies in reducing manual efforts. Minimum level of performance is for

CR and lowest code coverage is for US. The proposed method not only improves all of criteria but also made a tradeoff between them.

- Second Experiment: Automatic test case generation for "Blind Text Generator" software: This experiment is consisting of some steps to achieve results then a comparison with ATUSA method.

First Step: Random Software Navigation: This step consists of crawling on software states for preparing Input-Output vectors for learning PAFUNN network is necessary. In this experiment the number of runs is set to 30 runs.

Second Step: learning PAFUNN network and generating fuzzy roles set: Fuzzy network are learned with previous Input-Output data vectors in this step and experiments results are illustrated in Figure 11. Table 10 shows root mean square error, numbers of nods and CPU usage time.

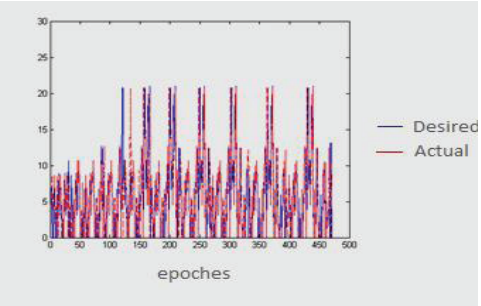


Figure 11 – Evolution and learning process of PAFUNN network on randomly navigated "Blind Text Generator" results data.

TABLE 10 – RESULTS OF LEARNING PAFUNN NETWORK ON RANDOMLY NAVIGATED "BLIND TEXT GENERATOR" DATA.

Rule nodes	67
Epochs	1
Training time (CPU-time)	5.3
RMSE	3.5

Third Step: Test Case Generation: Two step of initialing and optimizing data set made test case. a. initialing Test Sets: This step lead to initialing 100 results with a maximum of 30 test cases. b. Optimizing Test Sets: to achieve this goal used optimization Algorithms which mentioned in previous on data sets from initialing lead to a test sets with maximum fitness sent to next level to be analyzed.

Forth Step: Analyzing Data: This step will analyze resulted data from proposed method on "Blind Text Generator" as a case study. At first, results of different considered optimizing algorithms execution on TS Sets with fitness of each algorithm in 4 executions is illustrated in Table 11 and Figure 12.Then the results of applying RTCGW method when different optimization algorithms are used are shown in table 8 and figure 9. The number of test cases in a test suite is set to 30.

TABLE 11 – FITNESS VALUE AND CONSUMED TIME FOR EACH OPTIMIZING ALGORITHMS TEST CASE SETS

Algorithm	Statistics (fitness)	Avg. (time (sec))
GA	Min = 81.4 Max = 94.5 Avg. = 89.3	10
MA1	Min = 83.4 Max = 100 Avg. = 93.5	45
MA2	Min = 44 Max = 100 Avg. = 84.75	70
(GA+SA)1	Min = 57 Max = 100 Avg. =77	60
(GA+SA)2	Min = 88 Max = 100 Avg. = 97	130
SA	Min = 66 Max = 67 Avg. = 66.6	7
HC	Min = 17 Max = 69 Avg. = 51.74	6

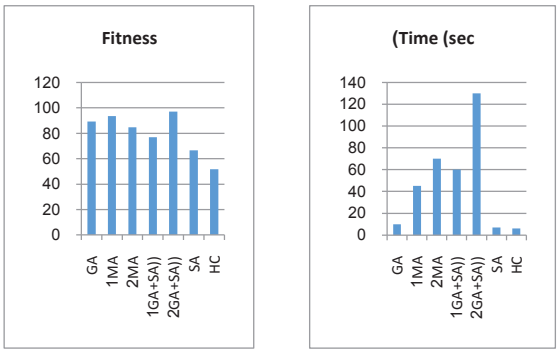


Fig. 12 Fitness value and average time of different

According to figure 12, maximum fitness value is related to GA+SA with second policy for utilization of both algorithms features. While genetic algorithm is going to obtain best solutions, simulated annealing tries some bad solution to escape from local optimum. The minimum value is for hill climbing as a local search algorithm and it seems that it returns a local optimum as the final solution. Secondly, the Memetic algorithm could receive to a high fitness because of using a local optimization method beside a global one and successfully escape from local optimum. Time spent for GA+SA with second policy has the highest value, since in each generation during genetic evolution simulated annealing is called for two times (i.e. for two chromosomes). This time is less for first policy of composing genetic with simulated annealing because of one call for simulated annealing in each generation for this method. On the other hand, time spent for hill climbing has the minimum level of value and it is because of its simplicity.

TABLE 12- THE RESULTS OF APPLYING RTCGW METHOD WHEN DIFFERENT OPTIMIZATION ALGORITHMS ARE USED

Algorithm	#test cases	Performance	Automation level (min)	Code Coverage
GA	30	171.3	0.97	0.76
MA1	30	176.84	0.97	0.78
MA2	30	174	0.97	0.78
(GA+SA)1	30	163.9	0.97	0.75
(GA+SA)2	30	183.60	0.97	0.80
HC	30	165.1	0.97	0.76
SA	30	170.27	0.97	0.76

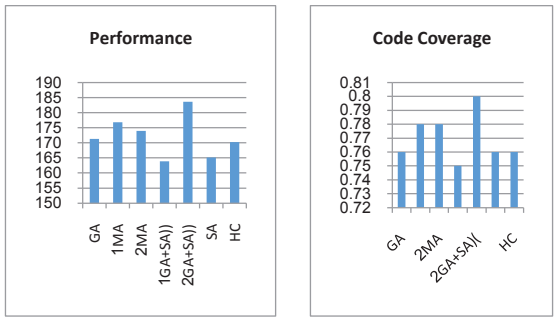


Fig. 13 The results of applying RTCGW method when different optimization algorithms are used

Clearly the results show that the more is the fitness of algorithm, the better are other evaluation criteria and this illustrates the correctness of fitness function selected in this study. While (GA+SA)2 leads to a test suite with the highest coverage and highest performance on Blind Text Generator web application, hill climbing has the lowest results and it is exactly for the reasons described in previous section. The same analyze is also correct for Memetic algorithm with the second highest value. Due to the fact that the highest level of performance and code coverage is related to GA+SA with second policy, this algorithm is used as optimization method for test suites and the results are compared with ATUSA (table 13 and figure 14).

TABLE 13- COMPARING PROPOSED METHOD WITH ATUSA

Method	#test cases	Performance	Automation level	Code Coverage
RTCGW	30	183.60	0.97	0.8
ATUSA	32	151.282	0.17	0.75

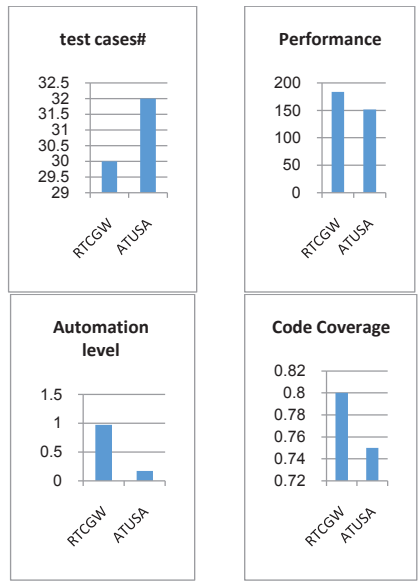


Fig. 14- Comparing proposed method with ATUSA

As the results show, the proposed RTCGW has a higher level of performance than ATUSA although having less number of test cases. The code coverage is also more than ATUSA. Automation level is higher and it leads to a lower level of manual effort needed for this method. Again the evaluations illustrate a tradeoff between all criteria. Using a complete model of web application in RTCGW better results obtained in comparison with ATUSA that uses a graph-based model which is generated by some incomplete execution traces. It means that model which describes web application has a deep impact on test cases generated. In other words, due to the fact that the model proposed in RTCGW is a kind of infinite model with the capability of predicting all the application behaviors, it results to test cases with high coverage of software.

V. CONCLUSION

This study tries to overcome some existing challenges in test case generation approaches for modern web applications. Due to the requirement of this research to a method for extracting an overall model which represents all aspects of software behaviors, a novel fuzzy neural network is proposed to extract a model based on fuzzy rule set and solve the problem of incomplete models of previous methods. The model is then used to generate test cases for web applications. Moreover, generated test cases are optimized using an algorithm composed of Genetic and Simulated annealing algorithms to overcome state explosion problem. Two benchmark web applications are tested using generated test suite and the results are compared with two well-known approaches in this area, which all shows the performance, high coverage and high level of automation comparing with other methods.

ACKNOWLEDGMENT

This work is supported by Research institute for ICT (ITRC) under Grant T/500/19236. The authors are grateful to anonymous referees of this paper for their constructive comments.

REFERENCES

[1] Keyvanpour M. R., Homayouni H., Shirazee H. Automatic Software Test Case Generation. Journal of Software Engineering 2011; **5**(3), 91-101.

[2] Keyvanpour M. R., Homayouni H., Shirazee H. Automatic Software Test Case Generation: An Analytical Classification Framework. International Journal of Software Engineering and its Applications 2012; **6**(4).

[3] Kasabov N. Foundations of Neural Networks, Fuzzy Systems and Knowledge, MIT Press 1996.

[4] Mesbah A. Applications, Analysis and Testing of Ajax-based Single-page Web, Karaj: TuDelft, 2009.

[5] Amalfitano D. Reverse Engineering and Testing of Rich Internet Applications. *Ph.D Thesis,* 2011.

[6] Marchetto A., Tonella P., Ricca F., State-Based Testing of Ajax Web Applications. in International Conference on Software Testing, Verification, and Validation 2008; Lillehammer.

[7] Turner D., Park M., Kim J., Chae J. An Automated Test Code Generation Method for Web Applications using Activity Oriented Approach. in 23rd IEEE/ACM International Conference on Automated Software Engineering 2008.

[8] Mesbah A., Deursen A. v., Roest D. Invariant-Based Automatic Testing of Modern Web Applications. Software Engineering Research Group, Department of Software Technology, Faculty of Electrical Engineering,Mathematics and Computer Science, Delft University of Technology 2011.

[9] Marchetto A., Ricca F., Tonella P. A case study-based comparison of web testing techniques applied to AJAX web applications. Int J Softw Tools Technol Transfer 2008; **10** (6), 477–492.

[10] Keyvanpour M. R., Homayouni H., Shirazee H. A Classification Framework for Automatic Test Case Generation Techniques for web applications. International Journal of Information Processing and Management(IJIPM) 2013; **4** (3), 26-39.

[11] Mesbah A., Deursen A. V., Lenselink S. Crawling AJAX-Based Web Applications through Dynamic Analysis of User Interface State Changes, ACM Trans. Web 6, 1, Article 3, 2012; DOI = 10.1145/2109205.2109208 http://doi.acm.org/10.1145/2109205.2109208.

[12] Mesbah A., Deursen A. Van, Roest D. Invariant-Based Automatic Testing of Modern Web Applications. IEEE Transactions on Software Engineering 2012; 38 (1), 35-53.

[13] Dallmeier V., Burger M., Orth T., and Zeller A. WebMate: Generating Test Cases for Web 2.0, Software Quality. Increasing Value in Software and Systems Development. Lecture Notes in Business Information Processing, 2013; **133**, 55-69.

[14] Babu M. R., Vasundra S. Enabling automatic testing of Modern Web Applications using Testing Plug-ins, International Journal of Computer Science & Engineering Technology, 2013; **4** (9), 1258-1262.

[15] Marchetto A., Tonella P. Using search-based algorithms for Ajax event sequence generation during testing. EmpirSoftwareEng 2011; **16** (1), 103–140.

[16] Mesbah A. Analysis and Testing of Ajax-based Single-page Web Applications. PhD Thesis, Faculty of Delf University, 2009.

[17] Kasabov N. Evolving Fuzzy Neural Networks for Supervised/ Unsupervised On-line, Knowledge-based Learning. IEEE Transactions of Systems, Man and Cybernetic 2001; **31** (6).

[18] Beigy H., Meybodi M. R., Menhaj M. B. Utilization of fixed structure learning automata for adaptation of learning rate in backpropagation algorithm. Pakistan Journal of applied sciences 2002; **2** (4).

[19] Jeevarathinam R., Thanamani A. Test Case Generation using Mutation Operators and Fault Classification. International Journal of Computer Science and Information Security, IJCSIS 2010; **7** (1), 190-195.

[20] Tracey N., Clark J., Mandar K., McDermid J. Automated Test-data Generation for Eception Conditions. Journal Software-Practice & Experience 2000; **30** (1).

[21] Prasanna M., Chandran K. R. Automatic Test Case Generation for UML Object Diagrams using Genetic Algorithm. Int. J. Advance. Soft Comput. App 2009; **1** (1).

[22] Dolby J., Artzi S., Jensen S. H., Moller A., Tip F. A framework for automated testing of javascript web applications. in International Conference on Software Engineering - ICSE, 2011.

[23] Saxena P., Akhawe D., Hanna S., Mao F., McCamant S., Song D. A Symbolic Execution Framework for JavaScript. in Proceedings of the 2010 IEEE Symposium on Security and Privacy, Washaington, 2010.

[24] Li J., Weiss D., Yee H. Code-coverage guided prioritized test generation. Information and Software Technology 2006; 48 (12), 1187-1198.

[25] Michael C., Mcgraw G., Schatz M. Generating Software Test Data by Evolution. IEEE Transactions on Software Engineering 2001; **27** (12), 1085-1110.

[26] Amalfitano D. Reverse Engineering and Testing of Rich Internet Applications. PhD. Thesis University of Napoli Federico, 2011.

MohammadReza Keyvanpour is an Assistant Professor at Alzahra University, Tehran, Iran. He received his B.Sc. degree in Software Engineering from Iran University of Science & Technology, Tehran, Iran. He received his M.Sc. and Ph.D. degrees in Software Engineering from Tarbiat Modares University, Tehran, Iran. His research interests include Software Engineering, Machine Learning and Multimedia

Hajar Homayouni received her B.Sc. degree in Software Engineering from University Of Kashan, Isfahan, Iran. She received her M.Sc. degree in Artificial Intelligence at Alzahra University, Tehran, Iran. Her research interests include Software Testing and Machine Learning.