International Journal of the Physical Sciences Vol. 6(13), pp. 3232-3245, 4 July, 2011
Available online at http://www.academicjournals.org/IJPS
DOI: 10.5897/IJPS11.298
ISSN 1992 - 1950 ©2011 Academic Journals

Full Length Research Paper

User-session-based automatic test case generation using GA

Xuan Peng* and Lu Lu

Department of Computer Science and Engineering, South China University of Technology, Guangzhou Higher Education Mega Centre, Panyu District, Guangzhou, Guangdong, P.R. China, 510006.

Accepted 21 March, 2011

User-session-based testing which relies on capturing and replaying real user sessions is an effective approach to test web applications. However, as a black-box testing, test case generation totally based on user session data may not be qualified for ensuring the reliability of web applications. This paper proposed an approach in terms of gray-box testing, combining user session data with request dependence graph of web application, to generate test cases automatically. A model of RDG was first constructed according to the structural analysis of the application under test; and then transition relations between pages and requests were extracted based on the request dependence graph. Finally, a reasonable test suite was generated to cover as many fault sensitive transition relations as possible with the aid of genetic algorithm. Simulation results indicated that our approach was better than the traditional user-session-based testing, in that it attained a higher path coverage and fault detection rate within small size of the test suite.

Key words: Web application testing, automatic test case generation, user session, request dependence graph, genetic algorithm.

INTRODUCTION

In recent years, as web applications play increasingly important roles in software engineering area, how to assure the security and dependability of web applications becomes a significant issue for the engineering practitioners. A web application (Ammann and Offutt, 2008) is often composed of relatively small software components, created with multitudinous technologies (for example, JSPs, ASPs, servlets, XML, PHP, etc.). Thus, web applications are complex and changeable, and effective testing of web application is essential to guarantee the coordination among those components and technologies.

User-session-based testing (Sprenkle et al., 2005a) has been researched recently to make use of user session data to generate test cases from a functional standpoint. Session information transparently collects user interactions, and is stored in a server log. A user session is composed of a sequence of user requests. A request is generally in the form of user IP, timestamp, request pattern (GET/POST), URL, parameter-value pairs, data transmission protocol, etc. User-session-based testing automatically generates test cases based on user profiles (Sampath et al., 2008). By capturing and replaying user sessions, this technique reruns real user-induced events to meet the functional requirements of web application. When compared with traditional white-box testing, user-session-based testing reduces much human effort of manual test case generation. Additionally, test cases generated from user sessions are more representative of the real field usage application, thus are prone to detect faults effectively.

Although user-session-based testing utilizes capture and replay mechanism to test the functionality of web applications with little intervention and participation of testers, there are yet several issues existing. On the one hand, for the web application which has been deployed on the server for a long time and which has received a large number of hits, huge quantity of user sessions will be collected from the server and it is costive to replay all those user sessions. To tackle this problem, approaches for reducing original user sessions were proposed. These

*Corresponding author. E-mail: p.xuan02@mail.scut.edu.cn Tel: +86 159 890 50398. Fax: +86 755 8246 1479.

approaches addressed the selection of representative user sessions and the removal of redundant ones to meet some certain test requirements, and are mostly focused on the requirement of base request coverage (Sampath et al., 2008; Di Lucca et al., 2006). On the other hand, web application testing based on user sessions belongs to black-box testing and ignores the structure of the web application; therefore, test case generation totally based on user session selection may not be qualified for various coverage requirements, in terms of traditional white-box testing, such as block, function, path coverages, etc. In this context, user session data can be applied as an accessible baseline test suite which reflects the real user operations. However, testing that relied solely on user session selection may be excessively dependent on the quality of the collected and selected session data; thus, the size and quality of the test suite generated by the session selection are absolutely tied to the user sessions of the original set. In this way, the cost effectiveness and fault detection ability cannot be guaranteed.

In this paper, we present a test case generation approach named US-RDG in terms of gray-box testing, which combines user sessions with request dependence graph (RDG) of web application, to take both user session collection and structural analysis of application into account. In our approach, base request coverage is no longer the only criterion for test case generation, and we induct an additional conception of transition relation in the form of "page→request→page" to charge the generation process. Also, a model of RDG (request dependence graph) is first constructed and the transition relations existing in the application are extracted to reveal the transition relationship between pages and requests, before the optimized test cases are generated by the user session mixture in the unit of "page→request→page", with the aid of genetic heuristic. In addition, we make a further analysis of data and link dependence to transition relations to distinguish the fault-sensitive ones. The goal of this approach is to generate a reasonable test suite to cover as many transition relations as possible, especially the fault-sensitive ones.

Comparing the study's approach with the existing approaches of generating test cases, based on user session data, our approach has the following features:

1. Combine user-session-based testing with white-box web application testing techniques by inducting structural analysis to charge the test case generation process, so as to obtain a test suite with sufficient structural coverage.
2. Process user sessions from the perspective of transition relation in the form of "page→request→page" rather than "request" only, which is more accurate in user usage presentation.
3. Abandon the complex high-level structural analysi better effectiveness to produce test suite using user s of application, and evaluate user sessions in terms of a bottom-layer analysis by integrating transition relation with request dependence.
4. Generate test cases in the unit of transition relation instead of user session and mix different user sessions to form a test case using genetic heuristic, so as to generate a more comprehensive, flexible and fault detectable test suite.

RELATED WORK

Much related work has been done in the areas of web application testing, user-session-based testing and test case generation, using GA. Here, a brief overview was done first.

A number of approaches (Ricca and Tonella, 2001; Dai and Chen, 2007; Chen et al., 2007; Miao et al., 2008) have been proposed to generate test data relying on the structure analysis of web application in terms of traditional white-box testing. Ricca and Tonella (2001) proposed an approach to create a model based on Unified Modeling Language (UML), in which path expression was created to generate test cases for web application. Chen et al. (2007) partitioned web applications into logical components (LCs) at different levels of abstraction using Pages-Flow-Diagram (PFD), and then used an automaton to model those LCs, with which test cases could be generated automatically.

However, web application testing simply depends on white-box testing technology which has many limitations: (1) the complex deploying environment and changeability of web application result in relative limitations of white-box testing; (2) Each test case is generated one by one manually, which is labour-intensive and inefficient; (3) Test cases are created by testers and cannot represent real field usage of application, and thus perform poorly in fault detective capability.

According to the defects of traditional web application in the aforementioned testing, the technique of user-session-based testing which was based on the real user interactions of application was proposed. This technique is less dependent on the complex and fast changing technology underlying web application (Elbaum et al., 2005b), and can reduce the human effort in test data generation. Much work has been done to address test case generation and optimization, based on user sessions (Di Lucca et al., 2006; Elbaum et al., 2003, 2005a, b; Sampath et al., 2004, 2008; Sampath and Sprenkle, 2007; Sprenkle et al., 2005; Luo et al., 2009).

Elbam et al. (2005b) presented three strategies to generate test cases based on user session data: directly reusing user sessions as test cases, combining different user sessions to form a test case and reusing user sessions with form modifications. In addition, they proposed two hybrid techniques to combine user-session-based approaches with structured white-box web application testing techniques. Their results showed the

session data than the white-box techniques considered. Nevertheless, it was indicated that the faults detected by the user-session-based testing and traditional white-box testing differed, suggesting that these two techniques were complimentary. Besides, we find that by inducting an unconstrained and complete random operation to mix or modify user sessions, the size of generated test suite was similar to the original user session set and the results of their empirical studies did not reveal quite an improvement in source code coverage, block coverage and fault detective ability.

In allusion to the big size issue of the test suite generated from user sessions, Sreedevi and Sampath (2007) addressed user-session-based test suite reduction in several papers (Di Lucca et al., 2006; Sampath et al., 2007; Sprenkle et al., 2005). In their research, user sessions which had the common base request patterns were grouped into the same cluster by concept analysis and one user session was randomly selected from each cluster to be a test case. Consequently, a set of additional heuristics was proposed to control the test case amount. Their empirical study indicated that there is a trade-off existing between test suite size and fault defective effectiveness. Lucca et al. (2006) proposed a technique to identify equivalent user behaviors included in user sessions and remove user sessions which provided the same page coverage to reduce the test suite. These two approaches are emphasized on the user session selection according to base request coverage, and the user session which contained more base requests would have bigger probability to be selected as the test data. However, there are two limitations in generating test data with this mode:

(1) It generates a test case with the unit of the user session, which is entirely dependent on the original user sessions. For example, if each user session in the original set contains only a small number of base requests, a large number of user sessions may be selected and each selected one may be too simple to be fault detectable.
(2) Selecting user sessions with the aim of request coverage ensures that each request is executed at least once, while it is not sufficient for some other structural coverage, such as block coverage, page path coverage and request transition coverage. In this case, structural analysis of the under-test web application is necessary to combine with the user-session-based testing.

Luo et al. (2009) presented a technique to generate test cases by combining user session selection with a high-level structural analysis. They constructed a service profile of application to cluster user sessions and then selected a set of representative user sessions according to structural analysis of the web application. The results demonstrated that this technique exerted better fault detective ability than those techniques leaving application structure out of account. Yet the accuracy of service profile construction and user session clustering is crucial to the implementation process of the technique, since any deviation may affect the quality of the generated test cases.

A number of approaches (Doungsa-ard et al., 2007; Ghiduk et al., 2007; Kalaji et al., 2009; Khor and Grogono, 2004; Rauf et al., 2010) for generating optimal test cases using GAs have been proposed, all of which showed the superiority of GA in problem optimization; whereas, a majority of these approaches were based on white-box testing and they used GA to generate test cases on the basis of random or manual created initial populations.

METHODOLOGY

Here, our methodology of US-RDG which combines user session data with request dependence graph (RDG) to generate test cases automatically is presented. The aim of this methodology is to create a reasonable test suite based on the collected user sessions with genetic heuristic. A test suite is considered as reasonable if it meets the following criteria: (1) Proper test suite size; (2) Proper test case length; (3) Cover as many fault-sensitive transition relations in the form of “page→request→page” as possible; and (4) Integrate conflicting user usages to provide more powerful test data.

Methodology overview

Here, an overview of this study’s methodology will be observed. As seen previously, a set of requests sent from clients are recorded in an access log of server. Three portions of the request were taken into consideration for the user-session-based testing. The first portion is composed of user IP and timestamp, which can be used to identify a user session. In general, a user session is said to have began when a new IP address sends a request to the server and ends when the user leaves the web site, or the session is timed out (Elbaum et al., 2005b; Luo et al., 2009; Sampath et al., 2006, 2008; Sprenkle et al., 2005). Another portion is composed of request pattern (GET/POST) and URL, which is called base request, while the third portion is the parameter-value pairs carried by base request. An identified user session can be simplified as a sequence of base requests and parameter-values to describe users’ sequential actions for web resources. A request is generally sent by a certain page through a trigger of user operation, such as clicking a button; and in a sequence of base requests, the URL of the previous request is the corresponding page which sends the next request to the server. Thus, it is simple to convert the form of user session from base request sequence, such as “request→request→…→request”, into transition sequence of pages and requests, such as “request→page→request→page→…→request→page”. Most approaches evaluated user sessions from the standpoint of base request coverage. However, evaluating user sessions purely by the state of request is not sufficient for recognizing users’ real operations; as such, transition relations between pages and requests should be taken into consideration:

192.168.0.100 - - [03/Dec/2010:10:18:50 +0800] "GET /AdvSearch.jsp HTTP/1.1" 200 5231
192.168.0.100 - - [03/Dec/2010:10:18:55 +0800] "GET /Books.jsp?name=&author=& HTTP/1.1" 200 14882
172.19.153.224 - - [05/Dec/2010:15:14:30 +0800] "GET /Books.jsp?category_id=3&name= HTTP/1.1" 200 13084

172.19.153.224 - - [05/Dec/2010:15:15:08 +0800] "GET /AdvSearch.jsp HTTP/1.1" 200 5231

The foregoing session is a segment intercepted from the access log of an e-business application named Book Store. Two user sessions are identified by user IP. From the request state point of view, the two user sessions cover the same base requests of "GET AdvSearch.jsp" and "GET Books.jsp", which cannot distinguish any difference in between them, while from a transition angle, the two user sessions are represented as "GET AdvSearch.jsp →AdvSearch.jsp→ GET Books.jsp→ Books.jsp" and "GET Books.jsp→ Books.jsp → GET AdvSearch.jsp →AdvSearch.jsp", which intuitively reveals that these two sessions have sent the same requests in reverse orders and through disparate elements of different pages. Thus, we induct the "page→request→page" mode to represent each session from a transition point of view, so as to reflect requests' transition process exactly and get a more accurate evaluation of user sessions.

In our approach, we generate test cases based on the user session prototype. During the generation, we adopt coverage of transition relation in the form of "page→request→page" as a major criterion to evaluate user sessions. To attain a more objective and comprehensive user session evaluation from the application itself, we reveal the structure of the under-test application by listing all the existing transition relations between pages and requests, which is in the form of "page→request→page" as well. Each transition relation refers to a potential action from one page to another. The source code covered by a transition relation differs from one another; thus, executions of different transition relations can detect multiple faults. The goal of this study's approach is to process user sessions to be the test cases for covering as many transition relations as possible.

Since user sessions record real operations of users, replaying them directly as test cases without modification may not be quite fault detectable because they have been already executed previously by the users when operating. On the other hand, as a major requirement for test case generation is to provide most of the coverage in functions, blocks and paths in order to detect faults as many as possible, the technique of test suite generation purely based on user session selection relies too much on the original user sessions, which is not flexible to retain, to a maximum extent, the original coverage with a small size of the test suite. In this context, generating test data in unit of user session is not preferable to form an optimized test suite. In our approach, we make use of transition relation which can be corresponded conveniently with user sessions, as a relative small unit, and generate a more powerful test data of multi-usage by mixing different user sessions. Thus, we use genetic heuristic to control the generation process with the three genetic processors of selection, crossover and mutation.

The rest of the study's methodology is to present the concrete realization of our methodology. Subsequently, a request dependence graph is constructed from a structural point of view to reveal transition relations existing between pages and requests. In addition, a further analysis of link dependence and data dependence is made by transition relations in order to distinguish the significance of each transition relation in fault detection, after which a genetic heuristic was presented to generate test data to cover as many transition relations as possible.

Request dependence graph construction

A web application generally contains a set of correlative static or dynamic web pages and other components, and these components integrate pages to form a system. To reveal the structure of a web application, dependence relationship between web pages can be extracted by source code analysis of application. Chen et al. (2008) developed a software tool, WebMTA (Chen et al., 2008), which constructs system dependence graphs in terms of data and link dependence for web application. However, in order to cater for the testing based on user sessions, in our approach, we propose a model to analyze application's dependence relationship from the standpoint of base requests, for the purpose of extracting all possible transition relations between pages and requests.

Excluding requests of irrelevant files, such as .jpg, .gif, etc., a request is made in general for the client to visit another page via some elements of the current page. Thus, there exists a request dependence relationship between pages, and this relationship can be attained from the developers, that is, a specification document or source code analysis of the application. A web page A is request dependent on a web page B if an execution of B will trigger a request for A. For example, in a Default.jsp page, when a hyperlink named "Login" is clicked, a request as "GET /Login.jsp" will be constructed and sent to the server. After receiving the server's response, the client browser will be directed to a login page named Login.jsp; thus, page Login.jsp is request dependence on Default.jsp through request "GET /Login.jsp", which can be converted to a transition relation as "Default.jsp → GET /Login.jsp → Login.jsp".

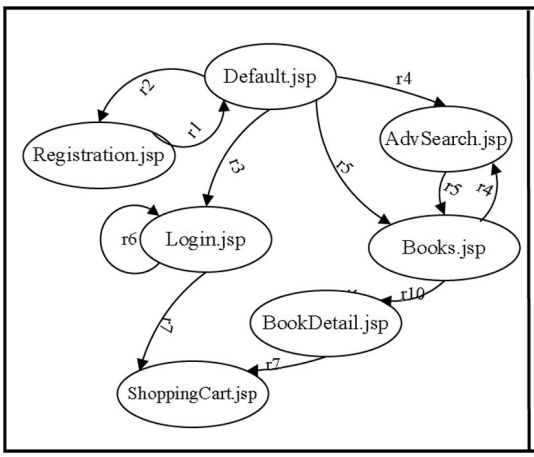
To depict the request dependence relationship clearly, we construct a request dependence graph. In this graph, a node represents a web page, a directed edge represents a request dependence relationship between two pages, and the request itself is presented to identify each directed edge. As the parameter-value pairs of a request are uncertain, we enumerate all the possible parameters of requests without values. Thus, the request labeled in the graph can be formatted as: "GET/POST PAGE <p1, p2, …,pn>", where p1, p2,…, pn are possible parameters carried by the request. Figure 1 is an example of request dependence graph for a part of the Book Store application which will also be used in the empirical studies.

Each directed edge in the request dependence graph can be corresponded to a transition relation. Take the edge identified by request r1 for example, we can see that a corresponding transition relation can be extracted as "Registration.jsp → r1 → Default.jsp". Since there are 11 edges in Figure 1, an equal number of transition relations can be extracted correspondingly. In addition, we induct a further analysis of data dependence and link dependence relationship to each transition relation. A transition relation of "page A → request R → page B" is identified as data dependence if there is any data transition or data operation from A to B through R. For example, in the Book Store application, a web page "Books.jsp" offers a list of books with introductions and hyperlinks. When clicking the hyperlink of a certain book, a request like "GET /BookDetail.jsp?item_id=34" will be sent to the server, and another page "BookDetail.jsp" is dynamically generated according to the value of "item_id" to offer the detailed information of a certain book. In this context, transition relation of "Books.jsp → GET /BookDetail.jsp?item_id=34 → BookDetail.jsp" is data dependence because there is a data transition within it. Otherwise, if there is no data transition or data operation in the request transition process, the relation is identified as link dependence. A link dependence transition relation is in general a simple direction from one page to another.

Test case generation using GA

As transition relations are extracted from the structural analysis of the web application in request dependence graph construction, test cases should be generated to cover as many of those relations as possible. In our US-RDG, we generate test cases using GA based on user scenarios. User session data captured from server logs are used as the initial population of GA, and on this basis, a reasonable test suite is generated.

GAs (Genetic Algorithms), developed by Holland (1975) with the inspiration from Darwin's evolution theory, are adaptive heuristic



r1. Get Default.jsp < FormRecommended_Page >

r2. Get Registration.jsp < member_login, member_password, member_password2, first_name, last_name, email, address, phone, card_type_id, card_number, FormName, FormAction, member_id, PK_member_id >

r3. Get Login.jsp < querystring >

r4. Get AdvSearch.jsp <>

r5. Get Books.jsp < name, author, category_id, pricemin, pricemax>

r6. Post Login.jsp <>

r7. Get ShoppingCart.jsp <>

r10.Get BookDetail.jsp <quantity, FormName, FormAction, order_id, item_id, PK_order_id, rating, rating_count,Trn_item_id, PK_item_id>

Figure 1. A. Partitial request dependence graph and; B. labeled requests.

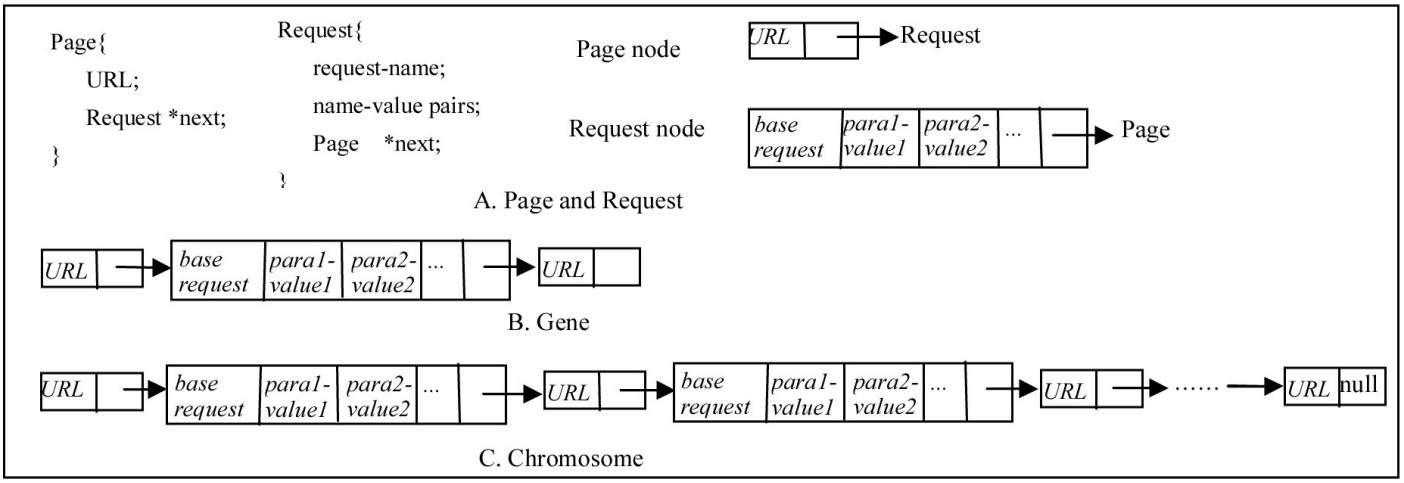


Figure 2. Genetic encoding and chromosome.

search algorithms premised on the evolutionary ideas of natural selection and genetic. Heuristics of GA are broadly applied to generate useful solutions for optimization and search problems with natural evolution. These heuristics were started with a set of solutions called the initial population, and then new populations were created gradually through three genetic processors of selection, crossover and mutation.

In our genetic heuristic, we encode each gene as a combination of requests and pages. We use two structured nodes to present pages and requests in Figure 2A, in which one is page node with a member variable of URL, and the other is a request node with a set of member variables of base requests and parameter-value pairs; and then chromosome is constructed by linking page and request nodes alternately with a unidirectional chained list (Figure 2B). The gene in a chromosome is structured in the form of "page→request→page" (Figure 2C), which can be corresponded with the transition relation; and each two adjacent genes share a common page node.

In the following, the steps of our genetic heuristic will be applied to generate a test case.

(A) Initial population generating and genetic encoding: In our heuristic, we treat each user session as a chromosome. To encode chromosome, pre-analysis is processed to the request sequence of each user session. For a request, separate portions are used as base request (action and URL), while name-value pairs are extracted, and the request node (Figure 2A) can be easily constructed. A related page node with the same URL as the request is later constructed, and the next pointer of the request node is fixed to the page node, since the URL of the request is the next page which will be visited in the client end. After that, the next request of the sequence is analyzed to construct another request node, and the pointer of the previous page node is fixed to it. The above process will be repeated until the request sequence has no request left. Since the first request of a user session has no previous request to fix a page which triggers it, the first page node of a chromosome is generally with URL of ε.

(B) Fitness Function and Selection Strategy: We evaluate fitness of each chromosome based on the transition relation coverage analysis. For a chromosome, we calculate how many transition relations were covered in it. In addition, we induct the dependence property of each covered transition relation into fitness calculation, because there is difference between data dependence transition

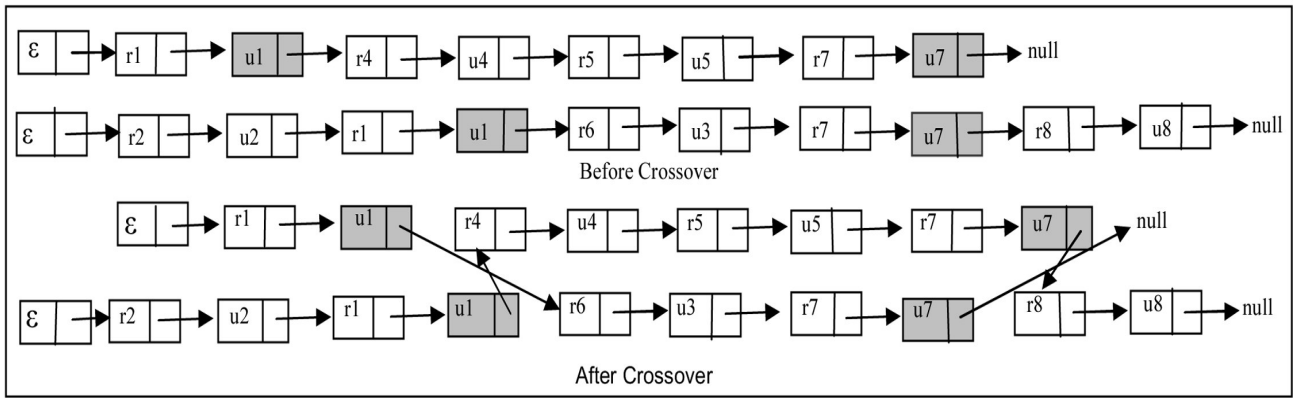


Figure 3. Crossover process.

relation and link dependence transition relation in fault detection. A transition relation identified by data dependence triggers more background program segments to process data, sometimes interacting with database, and may be more fault-prone than that identified by link dependence. Thus, we assign 1 to be the coefficient of the link dependence transition relation and introduce a parameter α which is defined as much greater than 1 to be the coefficient of the data dependence transition relation, so as to increase the proportion of the data dependence transition relation in fitness calculation. In this context, the chromosome which covers more data dependence transition relations will have a relatively bigger fitness. The fitness function of a chromosome is seen as:

$$\text{Fitness value} = (\alpha * |CDTR| + |CLTR|) / (\alpha * |DTR| + |LTR|) \quad (1)$$

Where $|CDTR|$ and $|CLTR|$ separately denotes the number of data and link dependence transition relations covered in the chromosome; $|DTR|$ and $|LTR|$ separately denotes the number of data and link dependence transition relations existing in the application which are attained from the structural analysis in request dependence graph construction. From the fitness function, we can see that the highest fitness value is top achieved as 1 when a chromosome covers all the data dependence transition relations and link dependence transition relations existing in the application.

We use the typical roulette wheel selection (Michalewicz, 1999) to select chromosomes for new population generation. The chromosome which has better fitness will have a greater chance of being selected.

(C) Crossover: The two-point crossover method is taken to reproduce chromosomes, because a request can be only sent by some certain pages; thus, the dependence relationship between pages and requests should be taken into account. For the two parent chromosomes, pc1 and pc2, presented in Figure 2B, page nodes will be first compared between them, after which each node with the same URL will be identified as a pair successively. If there are two or more than two identified pairs in these two chromosomes, random probabilities will be generated for each pair and two pairs of nodes with the highest probability will be selected to be the crossover points. Finally, the crossover points' *next pointer of each pair will be exchanged with each other. Otherwise, if no crossover points are selected, then no crossover process will be carried out. We take Figure 3 to illuminate the crossover process. Supposing the page nodes with URL of u1 and u7 in shadow are the crossover points, for simplicity, the name-value pairs are left out temporarily.

(D) Mutation: The mutation takes place to change the request node of a gene. In the mutation process, a mutation probability is first predefined, and for each chromosome a mutation score is randomly generated to compare with the mutation probability to decide whether or not a mutation will be processed. For a mutating chromosome, a mutation gene in the form of "page1→request→page2" is randomly selected; and another random selected chain, which has a head page node with the same URL as page1 and a tail page node with the same URL as page2, is then cut from a random chromosome of the initial population. Finally, the common transition relations covered between the mutating chromosome and the selected chain are counted, and the common ratio in the selected chain is calculated. If the common ratio is smaller than a predefined common threshold, then mutation process will be carried by replacing the request node of mutation gene with the selected chain removing the head node and tail node; otherwise, another chain will be selected to be judged as the foregoing, until the mutation process is carried out or a limited number of times is reached. However, the mutation process is illuminated in Figure 4.

(E) Acceptance and Replacement: For the fact that the aforementioned two processors of crossover and mutation are fraught with uncertainty, it is not sure that the offsprings are superior to their parents. We will re-calculate the fitness values of the new ones with the fitness function in step B; and then their fitness values will be compared with that of the parents, after which two chromosomes with relatively high fitness value will be selected and placed in a new population. This process assures that in any case, the optimal chromosome is inherited. The new population is used for the next iteration of the heuristic.

(F) Stop: If the average fitness value reaches a steady state in recent several populations or a predefined maximum generation has been achieved, the mutation should be stopped, and the best solution should be returned in the current population.

(G) Loop: Go to step B.

EMPIRICAL STUDIES

Here, we conduct an empirical study to evaluate the validity and effectiveness of our US-RDG approach to generate test cases for web application testing. Several issues need to be dealt with and confirmed in the empirical study: (1) Can US-RDG perform well in web

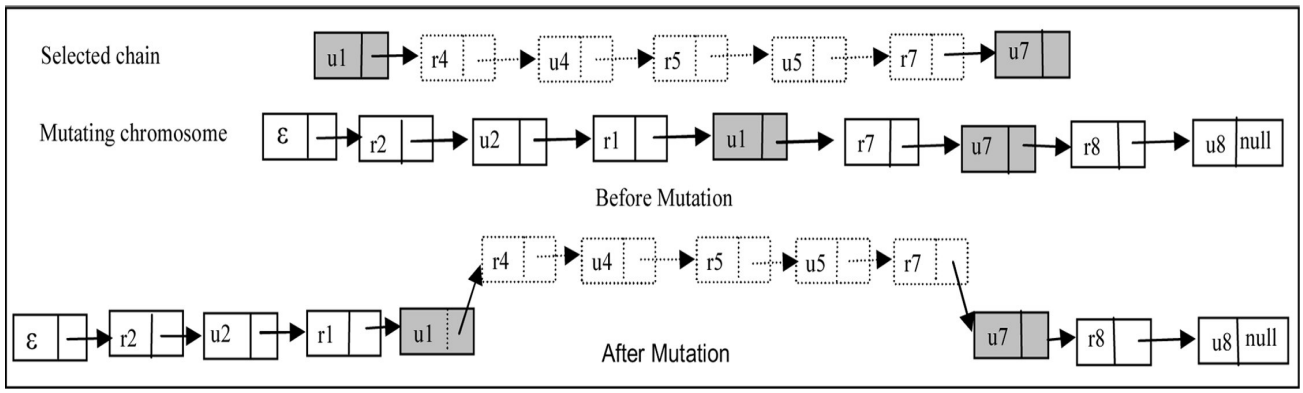


Figure 4. Mutation process.

application testing? (2) How effective are the test cases generated through our approach in terms of request coverage, transition relation coverage and fault detection? In addition, we apply some other existing approaches which also address test case generation based on user session to our empirical environment, in order to provide an explicit comparison.

In our empirical studies, we used an open source online Book Store available at gotocode.com. The online Book Store allows users to register, sign in/out, search books, order books, edit shopping cart and edit user profile. There is also a module for administrators; because the user data we collected were aimed at customer activities, we only address the test suite generation for the customer module. This application uses JSPs for its front end and a MSSQL database for the back end, and we deployed it on an environment combining Apache Http server with Tomcat server and JRE (Java Runtime Environment).

Request dependence graph construction of book store

The customer module of Book Store contains 9 JSPs, which are Default.jsp, Registration.jsp, Login.jsp, AdvSearch.jsp, BookDetail.jsp, Books.jsp, ShoppingCart.jsp, MyInfo.jsp and ShoppingCartRecord.jsp. By means of the specification document and structure analysis, 10 base requests are identified among these JSPs, after which request dependence relationships of the Book Store are constituted and request dependence graph is constructed in the left side of Figure 5A. The labeled request is figured out in the right side of Figure 5B.

As can be seen from Figure 5, there are 49 edges in the request dependence graph of the Book Store. In this context, 49 transition relations can be extracted correspondingly. From a further analysis of data dependence and link dependence, 15 transition relations among them can carry a data transition or data operation; thus, these 15 transitions presented in the following can be identified as data dependence transition relations.

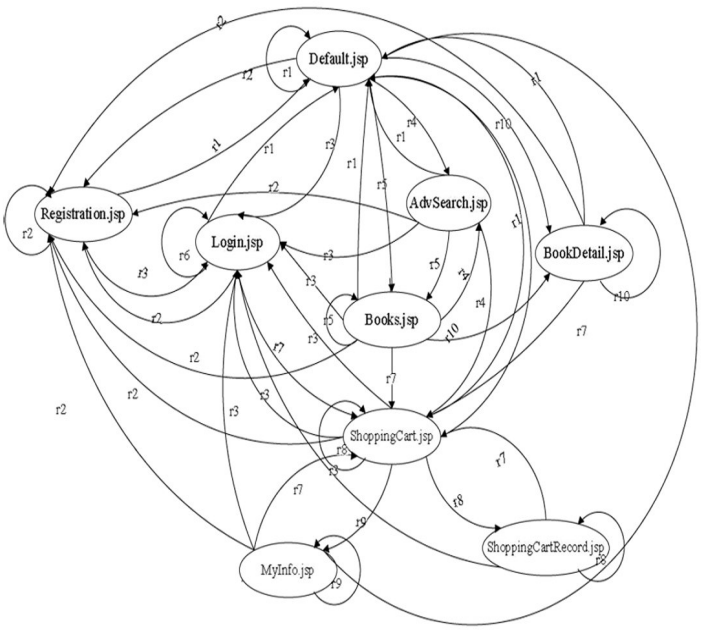
Default.jsp→r1→Default.jsp;
Default.jsp→r2→Registration;
Registration.jsp→r2→Registration.jsp;
BookDetail.jsp→r3→Login.jsp;
ShoppingCart.jsp→r3→Login.jsp;
Default.jsp→r5→Books.jsp;
AdvSearch.jsp→r5→Books.jsp;
Books.jsp→r5→Books.jsp; Login.jsp→r6→Login.jsp;
ShoppingCart.jsp→r8→ShoppingCartRecord.jsp;
ShoppingCartRecord.jsp→r8→ShoppingCartRecord.jsp;
MyInfo.jsp→r9→MyInfo.jsp;
Default.jsp→r10→BookDetail.jsp;
Books.jsp→r10→BookDetail.jsp;
BookDetail.jsp→r10→BookDetail.jsp.

User session collection and preprocessing

In order to collect adequate and available user sessions, we invited students who had online shopping experience and no prior knowledge on this project to visit a Book Store. We asked them not to use the browser's navigation features such as "back/forward", so as to ensure the accuracy of mapping requests to transition relations. In the period of our experiment, we had 49 students' participants and 3219 requests were recorded in the log file with a size of 1.14M. After removing the irrelevant data such as requests of .jpg, .ico, .gif, etc. files, 1451 requests were left. We developed an applet in C# to identify each user session based on user IP address and the visiting stamp. Finally, 87 user sessions were attained. Table 1 shows the characteristics of the collected user sessions.

Test case generation using GA

As it is indicated previously, we use our genetic heuristic



r1. Get Default.jsp < FormRecommended_Page >
r2. Get Registration.jsp < member_login, member_password, member_password2, first_name, last_name, email, address, phone, card_type_id, card_number, FormName, FormAction, member_id, PK_member_id >
r3. Get Login.jsp < querystring >
r4. Get AdvSearch.jsp <>
r5. Get Books.jsp < name, author, category_id, pricemin, pricemax>
r6. Post Login.jsp <>
r7. Get ShoppingCart.jsp <>
r8. Get ShoppingCartRecord.jsp <quantity, FormName, FormAction, order_id, member_id, PK_order_id>
r9. Get MyInfo.jsp <member_password, name, last_name, email, address, phone, notes, card_type_id, card_number, FormName, , member_id, PK_member_id>
r10. Get BookDetail.jsp <quantity, FormName, FormAction, order_id, item_id, PK_order_id, rating, rating_count,Trn_item_id, PK_item_id>

Figure 5. A. Request dependence graph of book store and; B. labeled requests.

Table 1. Characteristics of the collected user sessions.

Characteristic	Value
Total number of user sessions	87
Total number of requests accessed	1451
Largest user session in the number of requests	43
Average user session in the number of requests	16.7
Number of unique requests covered (coverage percentage)	10 (100%)
Number of data dependence transition relations covered (coverage percentage)	15 (100%)
Number of link dependence transition relations covered (coverage percentage)	26 (76.47%)
Number of transition relations covered (coverage percentage)	41 (83.67%)

to generate test cases based on transition relation analysis. At first, original user sessions were encoded to chromosomes through an applet, and then, our GA heuristic was applied.

We deploy the heuristic in C# with 6 modules, which are fitness function, filtering function, crossover function, mutation function, acceptation function and control function.

Fitness function

The fitness function is used for calculating the fitness value of each chromosome. The data dependence transition relations covered in a chromosome can be identified by a simple script. In our experiment, we view a transition relation as data dependence when the request in it carries any parameter with assigned value. After the data dependence transition relations and link dependence transition relations covered in a chromosome are separately counted, the fitness function, presented in formula (1) of "test case generation using GA", is applied to calculate fitness. However, α is a parameter of coefficient used to increase the proportion of data dependence transition relations in fitness calculation.

Filtering function

First, we sort chromosomes according to fitness from

high to low, and then select chromosomes in order. In addition, the filtering process follows a rule that the chromosome whose fitness is lower than a predefined percentage (fitness percentage threshold) of the parents' average fitness should not be selected. The chromosomes selected in this function are called a selected group.

Crossover function

Two chromosomes with the highest fitness and lowest fitness are respectively selected from the chromosome group; and then two pairs of crossover points are fixed by comparison and screening. Finally, the *next pointer of the crossover points (if they are present) will be exchanged with each other as presented in Figure 3.

Mutation function

A mutation probability is predefined to control whether or not a mutation operation will be carried out for a chromosome. If a chromosome is identified to be mutated, a mutation point is decided randomly; and then random chains are selected successively from chromosomes of the initial population, until the common ratio between the selected chain and the mutating chromosome is smaller than a defined common ratio threshold.

Acceptation function

Acceptation function is used to compare the fitness values of four chromosomes, the offsprings and their parents; while two chromosomes with the highest fitness values are chosen for the next generation.

Control function

Control function is used to control the coordination of the aforementioned 5 modules. These 5 modules work coherently and iteratively until the average fitness value reaches a steady state in recent several populations (fluctuation is less than the fluctuating range), or a predefined maximum generation is achieved. Furthermore, in the iterative process, the three modules of crossover, mutation and acceptation are in a nested loop to deal with chromosomes in each selected group.

GA performance in test case generation

The charts shown in Figures 6 and 7 present the results when running our GA heuristic under the condition of the given parameters as follows: (1) α: 10; (2) fitness percentage threshold: 0.92; (3) mutation probability: 0.25; (4) common ratio threshold of mutation: 0.5; (5) fluctuating range of resent several populations: 0.0001; and (6) maximum generation: 100. Due to the GA's the randomness, the heuristic was run for 20 times and all results were recorded so as to evaluate its performance objectively. In these 20 runs, the number of iterations used to generate an ultima test suite ranges from 12 to 23 with an average of 17.94, and the number of ultima generated test cases ranges from 3 to 10 with an average of 5.94. Since the goal of our genetic heuristic is not only to generate the test data as traditional gas, but also to optimize the original test data (user session) set in aspects of both size reduction and quality improvement, first we reveal the average performance of chromosomes in each generation. Figure 6 presents the chromosomes' average performance of each generation in five aspects of fitness, base request coverage, data dependence transition relations coverage and transition relations coverage. In addition, the worst, average and best-case scenarios were outlined for each aspect to offer an objective evaluation of the heuristic's performance. Simplification of a dot in a broken line indicates chromosomes' average performance in a certain case and a certain generation. As can be seen, all the broken lines in the charts are on the rise with increasing generation. Figure 6A reveals the variation of fitness, from which we can see the average fitness increase with generation steadily and the average fitness value of the last (23rd) generation which ranges from 0.7716 to 0.9157 with a mean value of 0.8594; thus, the distance between the best and the worst case of each generation is less than 0.15, which indicates that our heuristic is relatively stable. The reason why the average fitness of the last generation is less than 1 is that the transition relation coverage of the original user session set did not reach 100%, and the fitness value is to some extent bound by it. Figure 6B presents the average base request coverage in each generation: the average base request coverage increased from 62.96 to almost 100% in the 2nd generation, which suggests that the requirement of the base request coverage is easy to be satisfied, and the user session analysis, totally based on request coverage, is not sufficient for test data generation. Figures 6C, D and E present the average coverage performance of each generation in data dependence transition relations, link dependence transition relations and total transition relations, respectively. Our heuristic obtained a good effect in the data dependence transition relation coverage, which achieved an average of nearly 100% ultimately. Although the ultima average link dependence coverage transition relation and total transition relation coverage seem not to be so good in Figures 6D and E, yet it is because the original user session set has just covered 76.47% link dependence transition relations and 83.67% transition relations;

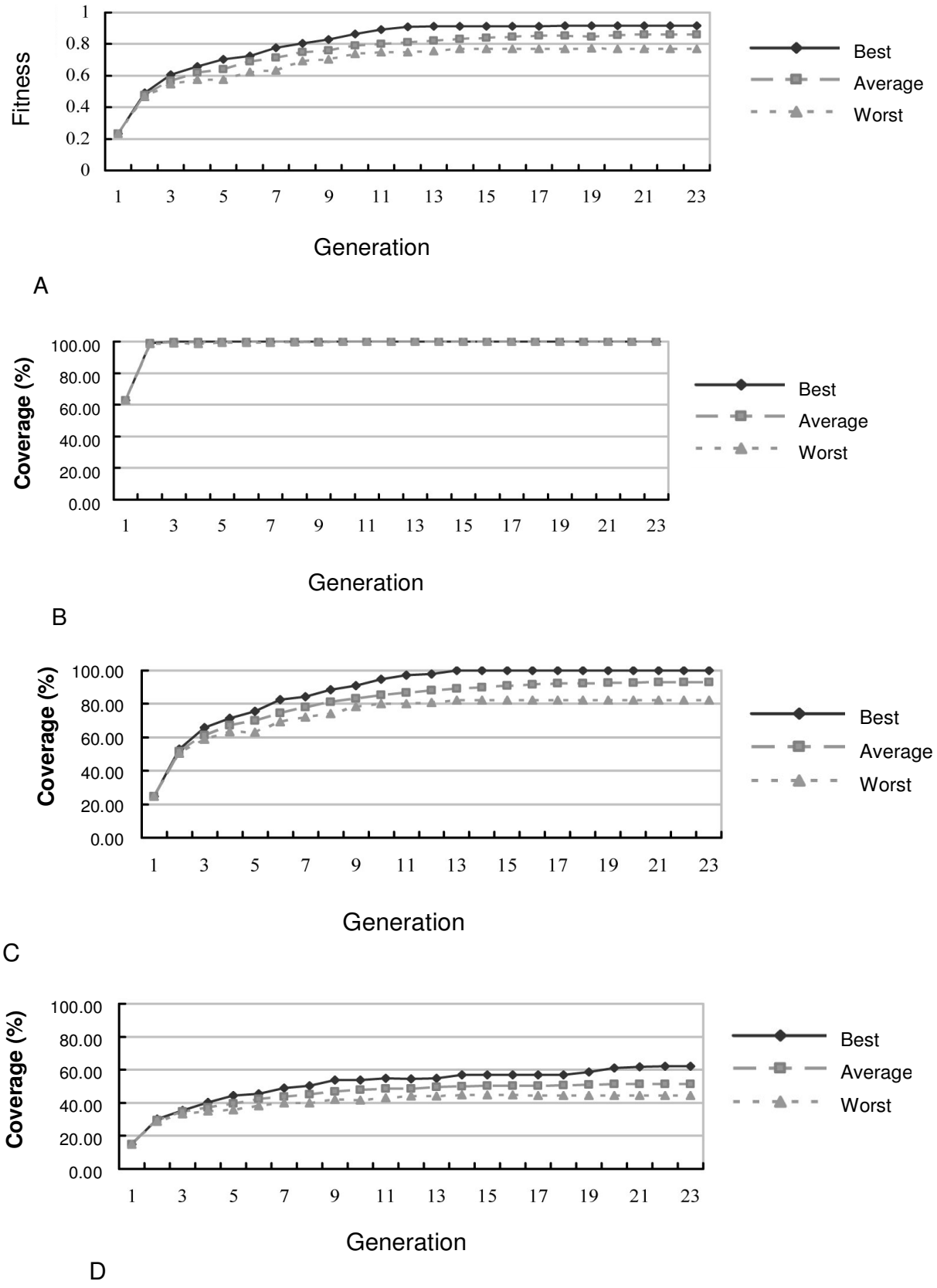


Figure 6. Average performances of chromosomes in each generation. A. Average fitness; B. Average request coverage; C. Average data dependence transition relation coverage; D. Average link dependence transition relation coverage; E. Average transition relation coverage of chromosomes.

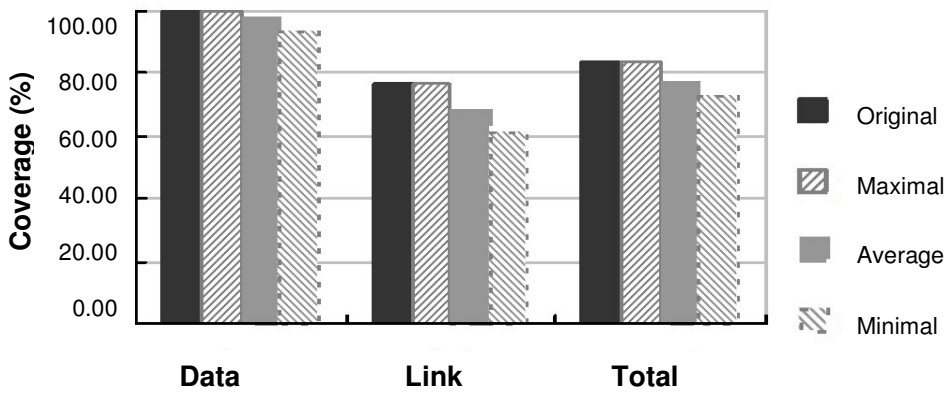


Figure 7. Performance of generated test suite.

moreover, in our heuristic, each test case can achieve an average link dependence transition relation coverage of 44.32 to approximately 62.25% and an average transition relation coverage of 59.06 to approximately 72.39%. In this case, the results of this study are acceptable. The result reported in the foregoing is from the standpoint of each chromosome's performance on average. In the following, we will analyze the coverage performance of the ultimate test suite generated by our genetic heuristic. The chart in Figure 7 also presents the three kinds of transition relation coverage from maximal, average and minimal levels. Of the 20 runs, 15 have generated test cases to cover all the data dependence transition relations, while the remaining 5 covered 14 transition relations. As can be seen from the chart, data dependence transition relation coverage of the generated test suite ranges from 93.33 to 100% with an average of 98.33%, which indicates the good performance of our heuristic in data dependence transition relation coverage. Furthermore, the link dependence transition relation coverage ranges from 61.76 to 76.47% with an average of 68.38% and the total transition relation coverage ranges from 73.47 to 83.67% with an average of 77.96%. When compared with the original user sessions, the generated test suite relatively covers 80.77~100% link dependence transition relations and 87.81~100% total transition relations of the original relations. The results indicated that our heuristic can achieve up to 100% of the original transition relation coverage; even in minimal cases, our heuristic did not lost much coverage with a small test suite.

Replay mechanism

Sapmpath et al. (2007) implemented a customized replay tool using HTTPClient (available in http://www.innovation.ch/java/HTTPClient/) to replay user sessions on the application. For the simulation experiment of our US-RDG, two factors were taken into consideration when replaying the generated test cases: one is the web application state, and the other is the original user session state. The web application state was already considered both in Elbaum et al. (2005b) and Sampath et al. (2007), for the reason that the state of the application might affect the normal operation of some specific requests. Sapmpath et al. (2007) proposed a with_state replay method to attain the web application state of each original user session and restore the respective state when replaying a certain one. In addition, we introduce a new factor named user session state to our replay mechanism. Since the offsprings are generated by combining two individuals in both crossover and mutation processes of our genetic heuristic and the ultimate form of our generated test case is a mixture of different user sessions, the user session state is also crucial for the normal operation of the test case. To tackle the two issues discussed in the foregoing, for each test case, we identify all the crossover and mutation points in it, and record their respective session states and web application states from the corresponding original user session. Subsequently, these session states and web applications were restored on the points of those corresponding points of test cases which are about to be executed in the replaying process.

Fault seeding

Faults that belong to the following 3 types are seeded into the online Book Store:

F1 (GUI faults): Faults that occur during the generation or form validation of a page and influence the correct display or normal event handler of pages.

F2 (Database operation faults): Faults that occur when operations such as query, insert, update and delete are performed on the server database through the client browser.

Table 2. Results of test suite reduction.

Test suite	Originality	A_{CON}	A_{SER}	US-RDG (Average)
The number of test cases	87	2	13	3~10 (5.94)
Total requests	1451	49~68	292	162~432 (270.7)
Unique requests	10	10	10	10
Data dependence transition relations coverage	15	6~9	13	14~15(14.75)
Link dependence transition relations coverage	26	9~13	21	21~26(23.45)
Transition relations coverage	41	17~22	34	36~41(38.2)

F3 (Navigation faults): Faults that occur when the target URL of a hyperlink or button is incorrect, and the resource is not available or a page is unreachable, which influence the normal direction of pages and the integration of the application. Hence, 40 faults were totally seeded into the application, of which 10 are for F1, 15 are for F2 and 15 are for F3.

Result comparison

In this study, two other approaches were applied to generate test cases based on user sessions: A_{CON}, proposed by Sampath et al. (2007), applied concept analysis to cluster user session and it presented a set of heuristics for test case selection; while A_{SER}, proposed by XingminLuo et al. (2009), clusters user sessions based on the service profile and it selects a set of representative user sessions from each cluster.

To apply A_{CON}, a relational table was first established to reveal the relationship between sessions and requests, and then a concept lattice was constructed through a concept analysis tool, ConExp (Yevtushenko, 2000), by inputting the established relational table. We used the test-all-exec-request heuristic, in which test cases were selected from the bottom node and the next bottom node, to select user sessions as test cases. In our study, the bottom node contained no user sessions, and there were two next-to-bottom nodes, in which one contained four user sessions and the other contained one user session. Thus, we randomly selected one user session from each of the two next-to-bottom nodes. In this case, 2 test cases were attained for A_{CON}.

To apply A_{SER}, service profile was first constructed, and user sessions were then classified into different service clusters according to the common paths contained in user sessions and service. Finally, user sessions were selected from each cluster based on dependence relation. In this study, 7 services of login, registration, searching, advance searching, book ordering, shopping, cart editing and user profile editing were identified from the Book Store application; and then those 87 sessions were partitioned into 7 clusters. The respective user session count of the 7 clusters is 16, 4, 1, 13, 20, 5 and 3, and there are still 15 sessions which are not associated with any services. Finally, 13 user sessions were selected as test cases from those 7 clusters with count of 2, 1, 1, 3, 3, 2 and 1.

The US-RDG approach was implemented in GA performance in test case generation and replay mechanism. Due to the randomness of GA, test cases generated in those 20 GA runs were adopted to reveal the experiment results.

Table 2 shows the test suite reduction results of the three approaches. We list the performance of each approach from six aspects of the test suite size, total requests, unique requests, and three types of transition relations coverage. Since the test suite generated through both A_{CON} and US-RDG were uncertain, all the possible cases were listed. In addition, we offer brackets to present the average performance of US-RDG in each aspect. From the table, it is seen that all the approaches generated a test suite with small size, A_{CON} reduced to 2, A_{SER} reduced to 13, and our US-RDG ranged from 3 to 10 with an average of 5.94. A_{CON} performed best in test cases and requests reduction, while the coverage of transition relations was not that good. When compared with the original user session set, A_{CON} lost 19~24 (46.34~58.54%) transition relations of its originality, especially the data dependence transition relations which were fault sensitive (with 40~60% lost). A_{SER} generated 13 test cases with an average request number of 22.46, while US-RDG generated 5.94 test cases in average, with an average request number of 45.57. Although the average test case length of US-RDG was bigger than that of A_{SER}, US-RDG performed better than A_{SER} in transition relation coverage with a mixture of user sessions.

Table 3 presents the results of fault detection. The faults seeded for F1 were detected best by the three approaches, which were consistent with our testing experience that the GUI faults were more prone to be detected; even so, A_{CON} missed two faults of F1. For F2 and F3, both A_{CON} and A_{SER} did not perform well in fault detection, in that A_{CON} detected 7~8 faults of F2 with a detective rate less than 54% and 6~9 faults of F3 with a detective rate of 40~60%, while A_{SER} detected 12 faults of F2 with a detective rate of 80% and 10 faults of F3 with a detective rate of 66.67%. Since faults of these two types are closely associated with the different transmissions between pages, the probability of fault detection is relatively

Table 3. Results of fault detection.

Fault detection	Originality	A_{CON}	A_{SER}	US-RDG
F1 (GUI faults)	10	8	10	10
F2 (Database operation faults)	15	7~8	12	13~15
F3 (Navigation faults)	15	6~9	10	12~15
Total	40	21~25	32	36~40

more dependent to the coverage of transition relations. In our US-RDG approach, we achieved 86.67~100% detective rate for F2, and 80~100% for F3. The detective rate of US-RDG for total faults is 90~100%. As can be seen in the result, our approach reached a relative high fault detection probability.

Conclusion

In this paper, an approach named US-RDG for generating test cases based on user sessions was presented, in which a gray-box testing combining structural testing and user-session-based testing was inducted to generate test cases automatically. We proposed a new conception named transition relation in the form of "page→request→page" to present the transition relationship between pages and requests from a structural standpoint. In addition, user sessions were correspondingly represented in the form of transition relations.

In our approach, structural analysis was first made with the application under test, and a graph named RDG was constructed based on the request dependence relationship between pages. Transition relations were then extracted according to the RDG, and a further analysis of data dependence and link dependence was made to identify the significance of each transition relation. Finally, a GA heuristic was proposed to generate test cases by mixing different user sessions so as to cover as many fault sensitive transition relations as possible. Our empirical studies were compared with the results of the other two approaches with our methodology of US-RDG. The results showed that our approach performed well for test case generation of web application, and the generated test cases were effective and efficient in requests coverage, path coverage and fault detection.

However, even though our US-RDG achieved similar transition relation coverage as the original user session set in a maximum level, the coverage of the generated test suite is dependent on the original coverage; that is to say, the transition relations which are not covered in the original user session set would not be covered in our US-RDG. In the future, we plan to investigate the argumentation of the generated test suite to meet a full coverage from a standpoint of structure analysis.

ACKNOWLEDGEMENTS

This paper is supported by Guangdong Technology Project (2009B010800048), GuangDong National Science Fund (10151064101000011) and Fundamental Research Funds for the Central Universities, SCUT.

REFERENCES

Ammann P, Offutt J (2008). Introduction to Software Testing. China Machine Press, Beijing, China, 246-267.

Chen SB, Miao HK, Qian ZS (2007). Automatic generating test cases for testing web applications. In: CISW 2007: Int. Conf. Comput. Intell. Secur. Workshops Harbin, China, pp 881-885.

Chen MH, Song C, Luo XM, Zheng XY (2008). WebMTA. <http://www.cs.albany.edu/~mhc/WebMTA/docs/tool.pdf>.

Dai ZY, Chen MH (2007). Automatic test case generation for multi-tier web applications. In: WSE 2007: Proc. 9th IEEE Int. Workshop Web Site Evol. Paris, France. pp 39-43.

Di Lucca GA, Fasolino AR, Tramontana P (2006). A technique for reducing user session data sets in web application testing. In: WSE '06: 8th IEEE Int. Symp. Web Site Evol. Philadel. USA, pp 7-13.

Doungsa-ard C, Dahal K, Hossain A, Suwannasart T (2007). Test data generation from UML state machine diagrams using Gas. In: ICSEA 2007: 2nd Int. Conf. Softw. Eng. Advances Cap Esterel, France. pp 47-52.

Elbaum S, Karre S, Rothermel G (2003). Improving web application testing with user session data. In: ICSE'03: Proc. 25th Int. Conf. Software Eng. Portland, Oregon, USA, pp 49-59.

Elbaum S, Karre S, Gibson E, Pollock L (2005a). An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. In: ICSM05: Proc. 21st IEEE Int. Conf. Softw. Mainten. Budapest, Hungary, pp .587-596.

Elbaum S, Rothermel G, Karre S (2005b). Leveraging user-session data to support web application testing. IEEE Tran. Softw. Eng., 31(3): 187-202.

Ghiduk AS, Harrold MJ, Girgis MR (2007). Using genetic algorithms to aid test-data generation for data-flow coverage. In: APSEC 2007: 14th Asia-Pacific Software Eng. Conf. Nagoya, Japan, pp. 41-48.

Holland J (1975). Adaptation in Natural and Artificial Systems. Uni. Michigan Press, Ann Arbor, USA, 66-72.

Kalaji AS, Hierons RM, Swift S (2009). Generating feasible transition paths for testing from an extended Finite State Machine. In: ICST'09: Inte. Conf. Softw. Test. Verification Validation Denver, Colora., pp 230-239.

Khor S, Grogono P (2004). Using a genetic algorithm and formal concept analysis to generate branch coverage test data automatically. In: ASC'04: Proc. 19th Int. Conf. Automated Softw. Eng. Lina, Austria, pp. 346-349.

Luo XM, Ping F, Chen MH (2009). Clustering and tailoring user session data for testing web applications. In: ICST '09: 2nd Int. Conf. Softw. Test. Verification Validation Denver, Colorado, pp. 336-345.

Miao HK, Qian ZS, Song B (2008). Towards automatically generating test paths for web application testing. In: TASE'08: 2nd IFIP/IEEE Inte. Symp. Theor. Aspects Softw. Eng Nanjing, China, pp. 211-218.

Michalewicz Z (1999). Genetic algorithms + data structures = evolution programs. 3rd ed., Springer, London, UK, 33-44.

Rauf A, Anwar S, Jaffer MA, Shahid AA (2010). Automated GUI test coverage analysis using GA. In: ICIT 2010: 7th Int. Conf. Inform.Technol. Las Vegas, Nevada, USA, pp. 1057-1062.

Ricca F, Tonella P (2001). Analysis and testing of web applications. Proc. 23rd Int. Conf. Softw. Eng., pp. 25-34.

Sampath S, Mihaylov V, Pollock L (2004). A scalable approach to user-session based testing of web applications through concept analysis. In: ASE'04: Proc. 19th Int. Conf. Automated Softw. Eng. Washington DC, USA, pp. 132-141.

Sampath S, Bryce RC, Viswanath G, Kandimalla V, Koru AG (2008). Prioritizing user-session-based test cases for web applications testing. In: ICST 2008: 1st Int. Conf. Softw. Test. Verifi. Validation Lillehammer, Norway, pp. 141-150.

Sprenkle S, Sampath S, Gibson E, Pollock L, Souter A (2005). An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. In: ICSM05: Proceedings of the 21st IEEE Int. Conf. Softw. Mainten. Budapest, Hungary, pp. 587-596.

Sampath S, Sprenkle S (2007). Applying concept analysis to user-session-based testing of web applications. IEEE Tran. Softw. Eng., 33 10: 643-658.

Yevtushenko SA (2000). System of data analysis "Concept Explorer" (in Russian). In: KII-2000: Proc. 7th national conf. Artif. Intell. Russia, pp. 127-134.