1-1-2007

Enhanced traverse of Web pages.

Yan Wang
University of Windsor

Enhanced Traverse of Web Pages

by

Yan Wang

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2007

Your file Votre référence
ISBN: 978-0-494-35049-2
Our file Notre référence
ISBN: 978-0-494-35049-2

ABSTRACT

Correct navigational behavior of a web application is essential to its reliability. An effective means to improve our confidence in the correct behavior of a web application is to test it by exploring the possible navigation among the web pages at client side: The tester carries out the testing by consecutively clicking the hyperlinks along with some possible search parameters and checking whether the returned web pages are as expected. Traditional conformance testing techniques based on finite state machines can be adopted in this setting to automatically generate suitable test sequences to traverse among client pages. In this thesis, we present an improvement on T-method for test sequence generation to reduce considerably its length by making use of the characteristics provided by the web browsers. Our method is sound and experiments show a 29%-68% saving on the test sequence lengths compared to the direct application of T-method.

Keyword: web applications, conformance testing, test sequence, finite state machines

DEDICATION

To the great Chinese soldiers who were dead in Korea war.

ACKNOWLEDGEMENTS

First of all, I would like to thank my supervisor, Dr. Jessica Chen, for her invaluable guidance and advices, for her enthusiastic encouragement and her great patience to me. Without her help, the work presented here would not have been possible.

Next, I would like to thank my committee members, Dr Lu, Dr. Wu, and Dr. Yuan, for spending their precious time to read this thesis and putting down their comments, suggestions on the thesis work.

My special thanks go to Miss. Lihua Duan for her ardent help.

Finally, I would like to give many thanks to my family and Nan Li for their patience, understanding and endless supporting.

Table OF CONTENT

List of Figures

List of Tables

1. INTRODUCTION AND PROBLEM DESCRIPTION

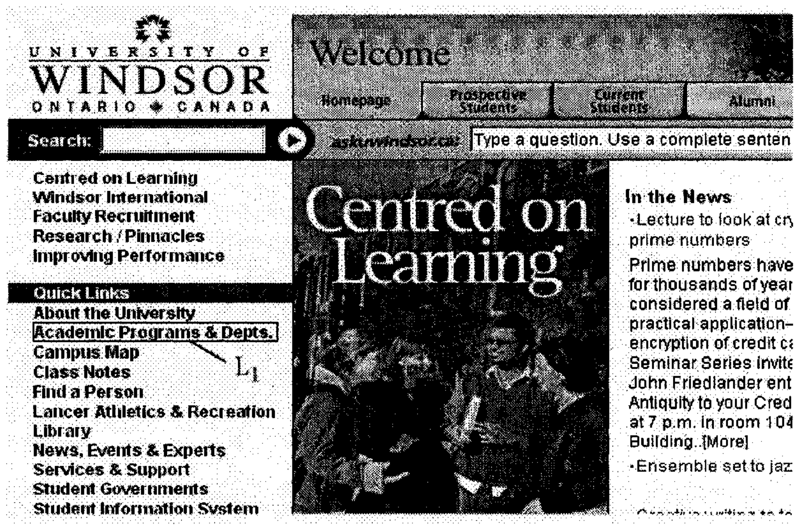
With the advent of networking and web technology, more and more information is being posted into and retrieved from the World Wide Web. Web-based software systems are becoming the primary device for information sharing and retrieval. Some of them now play so important roles that we can hardly tolerate any unexpected behavior from them. A tiny mistake in an online banking system may put thousands of people around the world in trouble. Even lasted for less than one hour, an unexpected error in an online store system may lead a business company to millions of dollars of revenue loss. The *reliability* of these critical web systems is directly hooked up to the quality of our daily life, and to assure their high reliability has become one of our major concerns.

Although existing rigorous software engineering methods and techniques can be adopted into the development procedure of the web systems to improve the quality of our final product, due to the special features and additional complexity brought by the emerging web technology, extra effort is much required when we *adapt* existing software engineering methods and techniques to this important application domain to achieve better effectiveness and higher efficiency. We have great interest in enhanced techniques for reliability control of web applications. As software testing has been well recognized as an essential part of software development techniques to gain our confidence in the reliability of the software product, we discuss the issues in conducting software testing on web application.

A key measurement of the reliability of web applications is their *correct navigational behavior*. That is, starting from any accessible client page (i.e. a web page displayed to the clients via web browsers), by clicking on any available hyperlink together with any possible input data as search parameters, the users should always access the *correct* information including both the text and the available hyperlinks on the displayed client page. To assure the correct navigational behavior of a web application is a non-trivial task due to the emergence of more and more complex web applications. Although a simple personal homepage may only consist of a small set of static web pages, a complicated commercial web application like an online-store system has to handle various kinds of transaction requests from millions of clients worldwide. One of the possibilities of checking the correct navigational behavior is to conduct conformance testing: to test the web application, viewed as Implementation Under Test (IUT), from the client side by browsing the client pages. Of course, here we need to assume the correct functionality of the web browser we use during the test procedure, just like we assume that the operating system is functioning well.

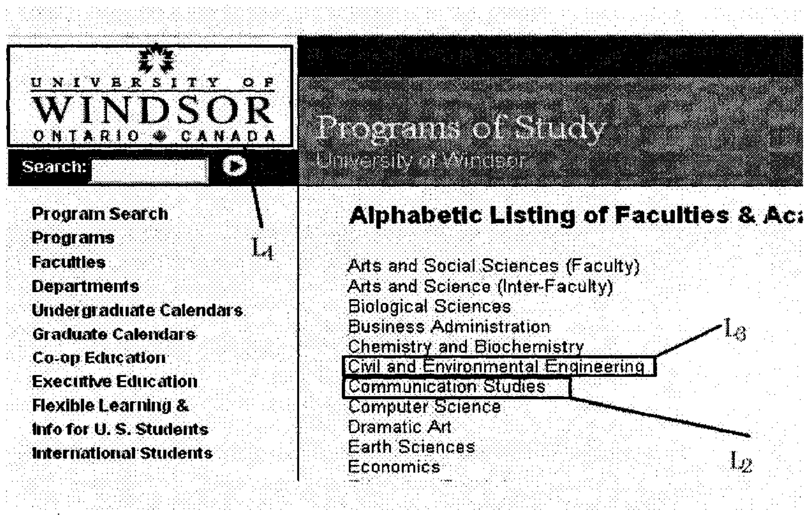
We also assume that the *expected* navigational behavior of the IUT is available in terms of a deterministic Finite State Machine (FSM) [11]. An FSM is a 5-tuple $M = (I, O, S, \delta, \lambda)$, where I, O and S are finite and nonempty sets of input symbols, output symbols and states respectively. Here we use each state to represent a client page and each transition to represent a possible transformation from one client page to another. The action of triggering a hyperlink, possibly together with some search parameters, is expressed as the input of the transition, while the characterization of the client page generated by the server page associated to the triggered hyperlink forms the output of the transition. This setting leads us to FSM-based conformance testing techniques that have been quite successful in both hardware and software testing (see e.g. [1, 8, 20, 28]).

We use part of the website of University of Windsor as a simple example to show how to model a web application in terms of FSM. This website is officially provided by the University of Windsor. People can enter the home pages of all departments from the homepage of the University website to find out more detailed information about each department. Figure 1.1(a) shows the homepage P_1 of the website that provide hyperlinks to all home pages of the main parts of the website where L_1 is the hyperlink to the web page for the list of the faculties and academic departments. Figure 1.1(b) shows web page P_2 that lists the hyperlinks to faculties and academic departments where hyperlinks L_2 , L_3 and L_4 refer to the homepages of communication study department, civil & environmental engineering department and University of Windsor separately. Figure 1.1(c) shows web page P_3 which is the home page of the communication study department. Here hyperlinks L_5 and L_6 refer to the home page of the University of Windsor and the web page that lists the hyperlinks to the faculties and academic departments respectively. Figure 1.1(d) shows web page P_4 which is the home page of the civil and environmental engineering department where hyperlinks L_7 and L_8 link to the home page of the University of Windsor and the web page that lists the hyperlinks to the faculties and academic departments respectively.

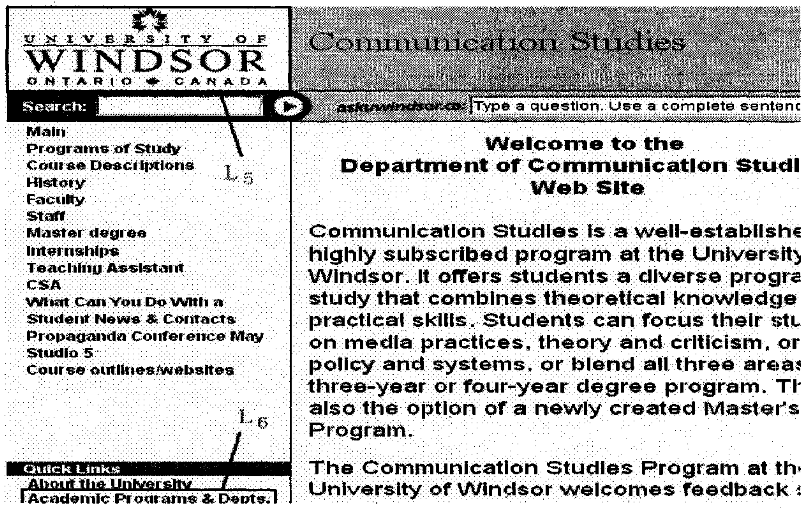


•Lecture to look at cry
prime numbers
Prime numbers have
for thousands of year
considered a field of
practical application–
encryption of credit ca
Seminar Series invite
John Friedlander ent
Antiquity to your Cred
at 7 p.m. in room 104
Building..[More]
•Ensemble set to jaz

(a)



(b)

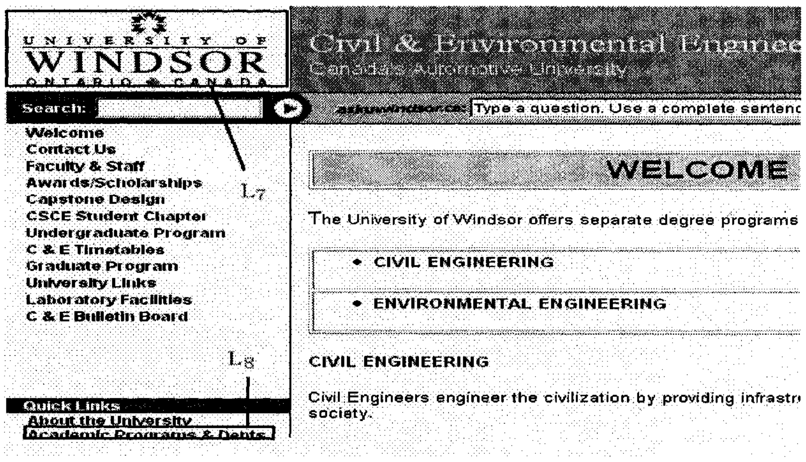


Welcome to the Department of Communication Studi Web Site

Communication Studies is a well-establishe highly subscribed program at the Universit Windsor. It offers students a diverse progra study that combines theoretical knowledge practical skills. Students can focus their stu on media practices, theory and criticism, or policy and systems, or blend all three area three-year or four-year degree program. Th also the option of a newly created Master's Program.

The Communication Studies Program at th University of Windsor welcomes feedback

(c)



The University of Windsor offers separate degree programs

CIVIL ENGINEERING

Civil Engineers engineer the civilization by providing infrastr society.

(d)

Figure 1. 1 THE WEBSITE OF THE UNIVERSITY OF WINDSOR

Figure 1.2 shows the FSM of the above selected part of the website. Web page p_1, p_2, p_3 and p_4 are represented by states s_1, s_2, s_3 and s_4 respectively. The transitions t_1, t_2, t_3, t_4, t_5, t_6, t_7 and t_8 represent eight different navigations in the web site triggered by clicking on hyperlinks L_1, L_2, L_3, L_4, L_5, L_6, L_7 and L_8 in the states s_1, s_2, s_3 and s_4 respectively.

There are two major tasks in FSM-based conformance testing: one is to generate an input/output sequence, called *test sequence*, that is both effective in terms of fault detestability and efficient in terms of its length; the other is to apply automatically the generated input sequence to the implementation and compare the output sequence with the expected one. Here we focus on the former problem of generating an efficient test sequence which represents a traverse among client side web pages.

There are several methods proposed in the literature for test sequence generation, typically known as T-method [21], U-method [1, 20 27], W-method [8] and D-method [13, 15, 28]. For testing web navigation, however they all converge to the simplest one: the T-method, with the test criterion to require that each transition in the given FSM be traversed at least once in the generated path of the test sequence. Given an FSM, we use the algorithm of Chinese Postman Problem (CPP) [10] to generate a test sequence to test each transition at least once. In the above example, a generated test sequence is $i_1/o_1, i_3/o_3, i_7/o_7, i_1/o_1, i_3/o_3, i_4/o_4,$ $i_5/o_5, i_8/o_8, i_1/o_1, i_5/o_5, i_6/o_6, i_2/o_2$ and its length is 12.

In regard of *efficient* test sequences in terms of their lengths, we show in this paper that by making use of the functionality of the web browsers, we can generate test sequences more efficient than those generated from T-method. We present an algorithm to considerably reduce the test sequence length based on some assumptions by using the *backward button* provided by the web browsers as an auxiliary means to transfer among client pages.

In the above example, suppose that we take into consideration the behavior of the backward button provided by the browser. A test sequence could be $i_1/o_1, i_3/o_3, i_7/o_7, i_{15}/o_{15}, i_4/o_4, i_5/o_5,$ $i_8/o_8, i_{16}/o_{16}, i_6/o_6, i_2/o_2$ whose length is10.

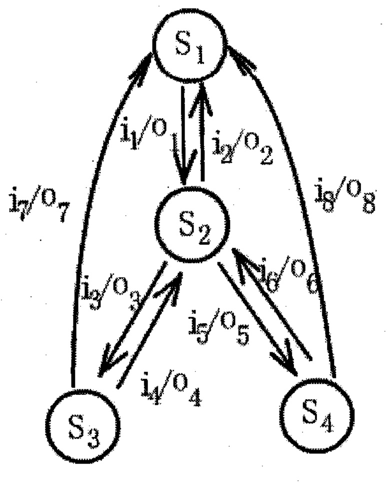


FIGURE 1. 2 THE FSM OF PART OF THE WEBSITE OF UNIVERSITY OF WINDSOR

Of course, the above example is illustrative and so the reduction on the test sequence length is not significant. In reality, if we can make use of the behavior of backward button to generate a test sequence, the test sequence could be reduced significantly.

The soundness of our algorithm is provided. The saving on the test sequence length compared to the one generated by T-method based on the algorithm of CPP however depends on the FSM structure and thus it is hard to be theoretically calculated. In this regard, we have carried out various kinds of experiments and we present our result together with our analysis on the quantity of the reduction of test sequence lengths according to various factors.

The rest of the thesis is organized as follows. We first introduce some notational background related to web applications in Chapter 2. Then we show how to model web applications in terms of FSM in Chapter 3. We discuss the major issues in the implementation of the existing algorithms related to our work in Chapter 4. In Chapter 5, we show that the backward button provided by the web browsers can be used to reduce the test sequence length and how to use this property to reach a solution for generating more efficient test sequences is presented. This is followed by our experimental results in Chapter 6 which demonstrates the significant reduction of test sequence length generated by our proposed method. Related work is then discussed in Chapter 7 and we give some concluding remarks at the end.

2. BACKGROUND

In this section, we introduce some related concepts on web applications, such as the architecture of web applications, URLs, web frames, web servers and dynamic hyperlinks. This is followed by a brief introduction to FSM and conformance testing.

2.1 WEB APPLICATION

2.1.1 URL

URL stands for Universal Resource Locator. It is used to locate any resources in the Internet. URL typically consists of a URL scheme, an authority, and a path. An URL scheme indicates a category of resources, such as http, https, and ftp. The authority typically consists of the name or IP address of a server and the port number. A path is to show the relative path of the resource inside the host server in the Internet. The format of an HTTP URL is often parameterized as: http://<host>: <port>/<path>?<request parameters>. Here "http://" indicates the resource type is http, and the communication protocol used to send this request is HTTP. <host> is the web server's IP address. It identifies a unique web server in the Internet address. <port> is the port number that the web server uses for HTTP communication. <path> specifies the relative storage position of the request resource. <request parameters> consists of request parameters a client passes to the web server. The request parameters can come from the input of the client and the cookies/sessions. These parameters could be used to compute and generate a client page. For example,

http://www.baidu.com/search?ie=gb2312

shows the basic elements of a URL. "http://" indicates the resource type is HTTP. "www.baidu.com" is the domain name and its corresponding IP address can be resolved by querying a DNS (Domain Name Service) server. The port number is implicit and it is 80 by default. "search" is a relative directory and "ie=gb2312" is the assignment of a request parameter.

2.1.2 SERVER PAGE, STATIC PAGE AND CLIENT PAGE

When a web application server receives a request message that contains a URL and cookie (if there exists) information from a browser, which can be triggered by clicking on a hyperlink or typing a URL in the address bar of the browser, it sends back the corresponding HTML web page according to the request. We call such corresponding HTML web page a *client page*. A client page is an HTML document with embedded scripts and is rendered by the web browser on the client side. A client page can be generated from two different template pages: static page and server page. Static pages have predefined page templates that are stored in a local directory of the web server. The web server can access these static pages with their file name and file path. For a static page request, the web server reads that HTML file, packs the file into the response message and sends the message back to the requested web browser as a client page. A server page can be a Common Gateway Interface (CGI) script, an Active Server Page (ASP), a Java Server Page (JSP), or a servlet. If a server page is requested, the web server dynamically creates a client page according to the information of the request message. After a client page has been generated, it is packed in a response message like a client page in the form of a static page and sent back to the requesting web browser.

Here we use the term "page" to stand for both of static and server page. Each web page has a page ID, to uniquely identify it in the web application. In practice, a web page is identified by its URL which includes the web server address and the relative directory. For simplicity, we use numeric page ID instead of the URL of the page.

2.1.3 SESSION AND COOKIE

Since HTTP is a stateless protocol, an HTTP server cannot recognize two different requests from the same client. After a response has been sent to a requested browser that issued the request, the connection is closed. The web server does not keep any information about the client. In many cases, some sort of relationship is required between two requests made by the same client. In order to resolve the problem, the concepts of *session* and *cookie* are introduced into web technology.

A *session* is the time duration used to uninterruptedly browse client pages spent by a certain user at a web site (application) from starting to browse the first client page to closing the web browser. In some web applications, a session is setup after the login of a client and closed when the client logouts. The concept of session puts a strong emphasis on recording of a certain user's state.

A cookie is a piece of small text that record identification and some related information of a certain client. It is stored in the web browser. Once the cookie is created on the client's side, for every further request made by the same user, the cookie is sent along with the request message until the expiration of the cookie. Based on the cookie technique, a web application can overcome the shortcoming of HTTP protocol by recognizing a certain client and establishing a session between a web browser and a web server.

2.1.4 HYPERLINK AND DYNAMIC HYPERLINK

Normally all pages of a web application are interconnected by hyperlinks. These hyperlinks form the navigational behavior of the web application. A hyperlink identifies a unique page that will be requested by a web browser. The format of a hyperlink includes web server address, relative directory that stores the page, and file name. Here we give each hyperlink in a certain static or server page a unique linkID which is used to identify the link to a particular page.

According to the different information of cookies/sessions, database or request parameters, a hyperlink could be dynamically shown on a client page. Such a hyperlink is called a d*ynamic hyperlink*. For example, in the online student information system of University of Windsor (SIS), after a current student logged into the SIS, the student can see a hyperlink to view his/her transcript but it is impossible for a prospective student to see such a hyperlink after his or her login.

2.1.5 THE PROPERTIES OF A CLIENT PAGE

On abstract level, a client page can be viewed as a pair $< T , H >$ of a set T of text symbols and a set H of hyperlink symbols. A text symbol represents the information of the text. A hyperlink symbol represents the existence of a hyperlink: it means that user's clicking on the hyperlink symbol on a client page can trigger a request message to the server.

Note that although the format of a hyperlink symbol and that of a hyperlink are the same, a hyperlink symbol is however different from a hyperlink used to retrieve a client page. A hyperlink to retrieve a client page contains host address, request parameter which may come from cookie or input variables. A hyperlink symbol is a hypertext displayed in a web page. It does not contain the input from the client to this page.

Figure 2.1 shows an example of a client page that is generated by a single server page. It is the homepage of the student webmail system of University of Windsor. In this client page, the black lines point out all the text and hyperlink symbols: the pair $<T,H>$. The hyperlink symbol "Sign in" cannot contain any input of the clients because the input of current page has not yet been given.

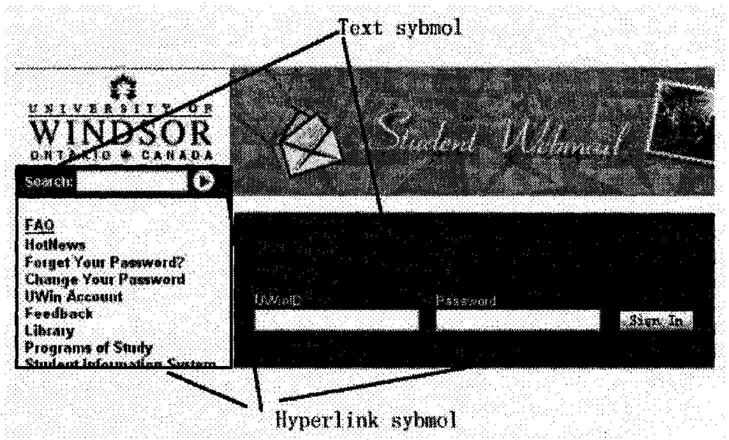


FIGURE 2. 1 AN EXAMPLE OF A CLIENT PAGE

2.1.6 ARCHITECTURE OF WEB APPLICATIONS

Figure 2.2 illustrates a generic architecture of web applications. A web application is an interactive system which contains web browser, the web application server and the database. It is a typical 3-tier model. While realizing more complicated functions, the modern web applications have expanded from a 3-tier model to an N-tier model.

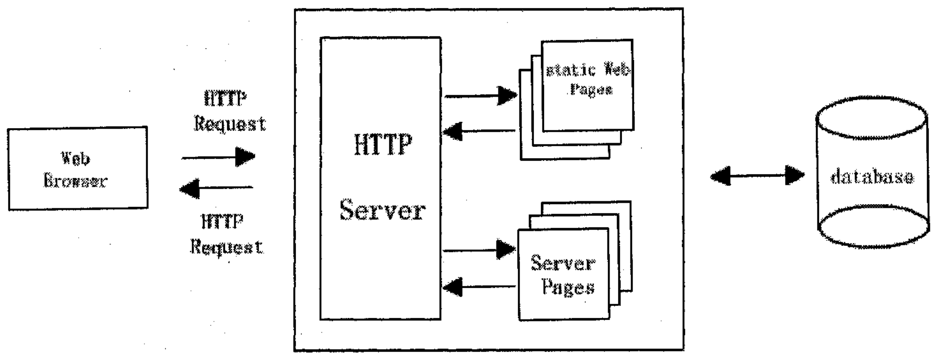


FIGURE 2. 2 THE STRUTURE OF WEB APPLICATION

A web browser is a standard window application and users use it to visit a web application by clicking hyperlinks or typing in URLs. After sending out a request message for a client page, the browser will receive a response message from the web application server. The message contains the requested client page.

2.1.7 HISTORY STACK

A browser's history stack [13] stores the previously visited URLs, and the stack is maintained by the browser module. Generally speaking, the history stack is a kind of stack with similar operations as normal stacks. A history stack maintains a stack pointer, top position and bottom position. The values of these variables determine whether a backward or forward button can be enabled. Compared with normal stack, the stack pointer of the browser's history stack can be moved back and forth if more than one item is stored in it.

When a browser starts up, its history stack is empty. The stack pointer points to nothing and the top position and bottom position are set to a null value. At this time, the backward and forward button are disabled. After a URL is requested and a web page is received, the URL is pushed into the stack and the stack pointer points to the newly inserted URL. The top position and bottom position are set to their corresponding values.

The browser's backward and forward buttons are enabled or disabled according to how many items are stored and according to the position of stack pointer. If the stack has more than two items and the stack pointer does not point to the first URL stored in the history stack, the backward button is enabled. This means that there exists an item before the current item that the stack pointer points to. At this time, a user can click the backward button, and the stack pointer will move from current position to its previous one. The URL stored in the position that the stack pointer points to is retrieved and sent to network interface for requesting its corresponding HTML web page. The page might be returned from local cache or the web server that sent the page before. The use of local cache depends on the cache policy set in the browser or HTTP cache controls that came with the received web page.

Similarly, the forward button is enabled when the stack has more than two items and the stack pointer does not point to the top item in the history stack. The clicks on the forward button can force the stack pointer moving from current position to its next one. The URL that the stack pointer points to is used to request its corresponding web page.

When a user clicks on a *backward* or *forward* button, the stack pointer moves back or forth from current position. If a user clicks on a hyperlink or types a URL in the browser's address textbox, the URL of newly requested web page is pushed into the history stack after the browser receives the web page successfully. Before the push operation, all URLs above stack pointer's current position are popped up, and after the push operation, the stack pointer points to the top position of the stack.

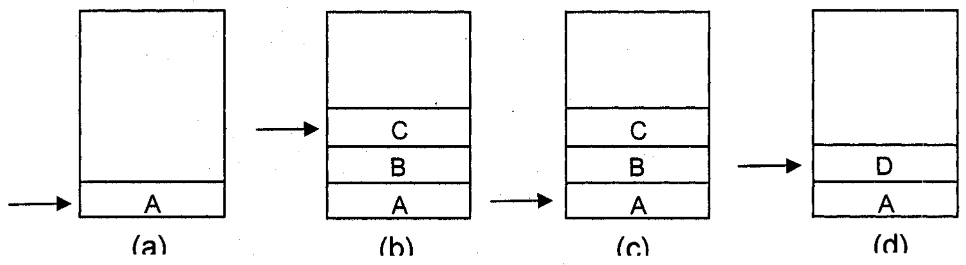


FIGURE 2. 3 HISTORY STACK OPERATION

Figure 2.3 gives an example to show the operations on the history stack. Figure 2.3(a) shows that page A's URL A is pushed into the stack after receiving page A. The stack pointer points to A. In Figure 2.3(b), page B and C have been received and their URLs B and C are pushed into the stack. In Figure 2.3(c), the user clicks *back* button twice, and the stack pointer points to the first URL A. Because page A contains hyperlinks, after a user clicks a link to page D in page A, page D's URL is pushed into the stack on the position above A. Before D is pushed into the stack, previous URLs B and C are popped up.

3. MODELING WEB APPLICATIONS WITH FSM

In our context, a web application refers to its server side implementation, which consists of a set of *server pages.* Typical server pages include HTML files, Java Server Pages, Java Servlets, ASPs, PHP files, etc. They can be generally viewed as a piece of program that dynamically generates client pages. Taking this set of server pages as a black-box implementation, its behavior is tested from the client side by navigating among the output client pages.

As we mentioned in the Introduction, we assume that the expected behavior of a web application is given in terms of a deterministic FSM [11]. A deterministic FSM M is defined by a tuple $(S, s_0, X, Y, \delta, \lambda)$ in which S is a finite set of *states* where $s_0 \in S$ is its *initial state*, X is a finite *input alphabet*, Y is a *finite output alphabet*, $\delta : S \times X \to S$ is a *next state function* and $\lambda : S \times Y \to Y$ is an output function. Note that δ and λ may be partial functions. Figure 3.1 shows an example of an FSM where $X = \{a, b\}$, $Y = \{0,1\}$, $S = \{s_0, s_1, s_2\}$, $\delta(s_0, a) = s_1$, $\delta(s_1, b) = s_2$, $\lambda(s_0, a) = 1$ and $\lambda(s_1, b) = 0$. The initial state of the FSM is s_0.

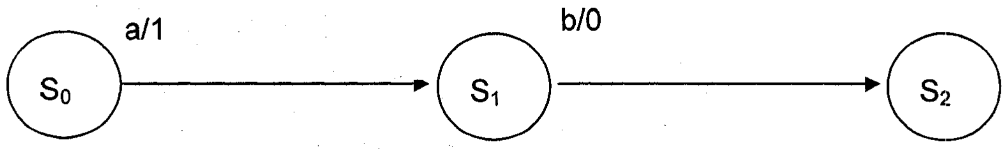


FIGURE 3. 1 AN EXAMPLE FSM

Each state uniquely represents one client page, which is an output of the execution of a server page. Apparently, we cannot exhaustively enumerate all client pages. For the purpose of testing, we only select some representations. This can be achieved by the following two techniques:

1) We consider a *closed* system in the sense that the hyperlinks all refer to the server pages within the same web application. An *open* system can be modeled in our setting by augmenting an additional page to represent all the Internet web pages not generated by the *web application under test*. If there is a hyperlink in a client page pointing to www.yahoo.com, for example, we can simply represent it as a hyperlink pointing to this special page.
2) We can use an *abstract representation* for a group of similar pages. For example, in a *student information system*, when student A and student B both have regular transcripts, a client page displaying the transcript of A and B both have regular transcripts. In this case, a client page displaying the transcript of A and a client page displaying the transcript of B can be considered as members of a same group with a single representative client page.

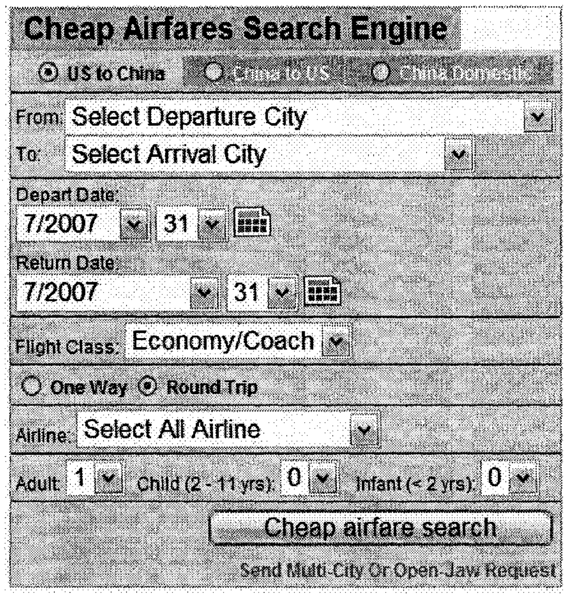
An input x represents the user's (or the tester's) input. It includes the triggering of a hyperlink possibly with some user's input data (for form submission) which is used as search parameters. In practice, people also use hyperlink to represent the hyperlink together with search parameters. Here we explicitly separate these two parts. A URL is associated with each hyperlink together with possible search parameters and points to unique server page to be executed to generate the next client page.

An output y represents the characteristics of a client page. For testing purpose, it can be understood as a multi-set of texts displayed and a multi-set of hyperlinks. The text can be normal text, images, some places for users to type in input data, the text portion of a hyperlink, or other non-hyperlink content. A hyperlink can be a normal hyperlink uniquely identified by its context and the server page it refers to. A client page is *correct* if the multi-set of texts and the multi-set of hyperlinks are all correct. Here we do not consider the test oracle problem on how to automatically check the correctness of various types of web content against the expected one. Readers interested in this issue are referred to [6, 19, 23, 29].

A transition t is defined by a tuple $(s_i, s_j, x/y)$ in which s_i is the *starting state*, x is the input, $s_j = \delta(s_i, x)$ is the *ending state*, and $y = \lambda(s_i, x)$ is the output. A transition $(s_i, s_j, x/y)$ represents the transformation from client page cp_i represented by state s_i to client page cp_j represented by state s_j triggered by user's input x.

Two states s_i and s_j of M are *equivalent* if applying any input sequence at s_i and s_j will yield the same output sequence. An FSM M is minimal if no FSM with fewer states than M is equivalent to M. A sufficient condition for M to be minimal is that every state can be reached from the initial state of M and no two states of M is equivalent. In this paper, the FSMs we consider are all minimal ones.

Let us consider the cheap airfare search engine of the online system USChinaTrip.com (www.uschinatrip.com) as an example web application. Here, we only focus on the major functionalities of this search engine. In home page cp_0, the user can issue a search by providing search criteria such as the date, departure city, destination city, and so on. If no such flight is available, a client page with information "the system didn't find any fares fitting your criteria" will be returned, and we use cp_1 to denote this client page. Otherwise, a client page with the detailed information about all the available flights will be displayed in cp_2. When the information for the search criteria is incomplete (e.g., no destination city is entered) or infeasible (e.g., the return date is before the departure date for a round trip), an error page cp_3 will be returned. In client page cp_2, the user can select any available flight for reservation by clicking the corresponding hyperlink. Then the personal information is required from the user in a new client page cp_4. Again, when the keyed-in personal information is incomplete or erroneous (e.g., telephone number contains alphabet letter), client page cp_5 will be prompted to user for correction. Client page cp_6 is generated with all the personal and flight information for the user to review. In cp_6, the user can choose "Send Request", "Modify Customer Information", or "Start New Search", which will result in client page cp_7 for request confirmation, client page cp_4, and cp_0, respectively. The "Frequently Asked Questions" page (denoted by cp_8) and the home page can be reached by special image hyperlinks from all the client pages. cp_0 and cp_3 are generated by server page s_0, cp_1 and cp_2 are generated by server page s_1, cp_4 and cp_5 are generated by server page s_2, cp_6 is generated by server page s_3, cp_7 is generated by server page s_4 and cp_8 is generated by server page s_5. Figure 3.2 shows the web pages of the online system.



Cheap Airfares Search Engine

[✓ US to China] China to US China Domestic

From: Select Departure City
To: Select Arrival City

Depart Date: 7/2007 31
Return Date: 7/2007 31

Flight Class: Economy/Coach

One Way [✓ Round Trip]

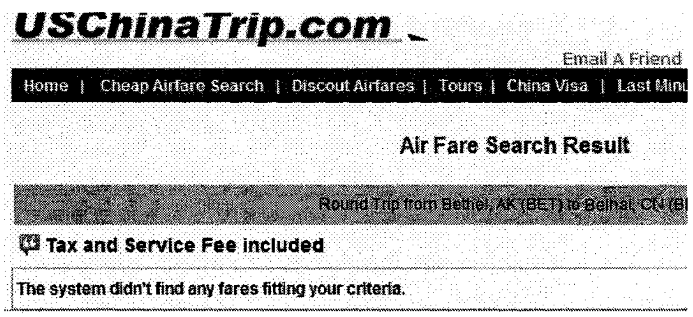
Airline: Select All Airline

Adult: 1 Child (2 - 11 yrs): 0 Infant (< 2 yrs): 0

Cheap airfare search

Send Multi-City Or Open-Jaw Request

(a) cp_0



USChinaTrip.com

Email A Friend

Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minu

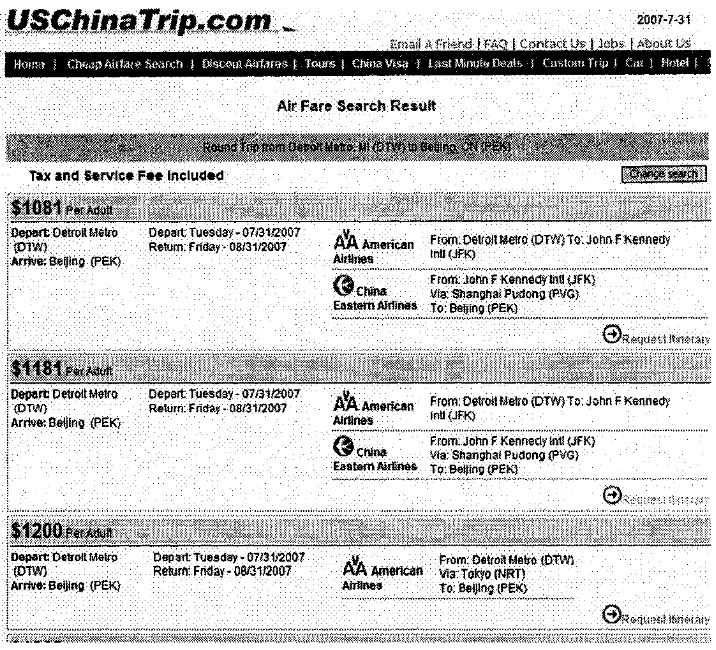
Air Fare Search Result

Round Trip from Bethel, AK (BET) to Beihai, CN (BI

Tax and Service Fee included

The system didn't find any fares fitting your criteria.

(b) cp_1



USChinaTrip.com 2007-7-31

Email A Friend | FAQ | Contact Us | Jobs | About Us

Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custom Trip | Car | Hotel |

Air Fare Search Result

Round Trip from Detroit Metro, MI (DTW) to Beijing, CN (PEK)

Tax and Service Fee Included Change search

$1081 Per Adult

Depart: Detroit Metro (DTW)
Arrive: Beijing (PEK)

Depart: Tuesday - 07/31/2007
Return: Friday - 08/31/2007

American Airlines — From: Detroit Metro (DTW) To: John F Kennedy Intl (JFK)

China Eastern Airlines — From: John F Kennedy Intl (JFK) Via: Shanghai Pudong (PVG) To: Beijing (PEK)

Request Itinerary

$1181 Per Adult

Depart: Detroit Metro (DTW)
Arrive: Beijing (PEK)

Depart: Tuesday - 07/31/2007
Return: Friday - 08/31/2007

American Airlines — From: Detroit Metro (DTW) To: John F Kennedy Intl (JFK)

China Eastern Airlines — From: John F Kennedy Intl (JFK) Via: Shanghai Pudong (PVG) To: Beijing (PEK)

Request Itinerary

$1200 Per Adult

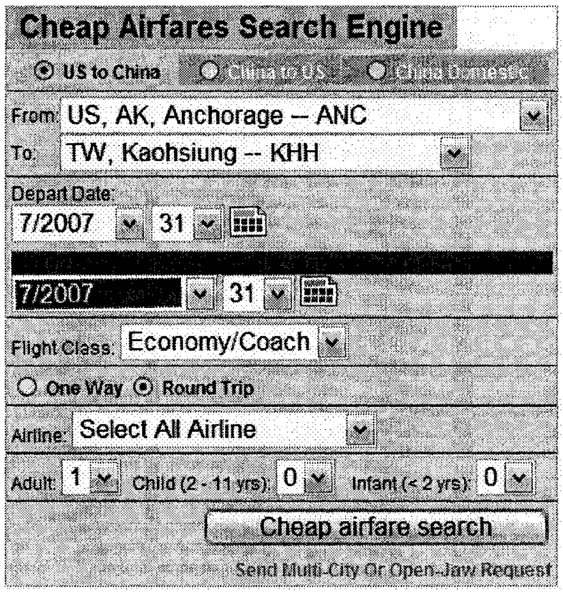
Depart: Detroit Metro (DTW)
Arrive: Beijing (PEK)

Depart: Tuesday - 07/31/2007
Return: Friday - 08/31/2007

American Airlines — From: Detroit Metro (DTW) Via: Tokyo (NRT) To: Beijing (PEK)

Request Itinerary

(c) cp_2



Cheap Airfares Search Engine

[✓ US to China] China to US China Domestic

From: US, AK, Anchorage -- ANC

To: TW, Kaohsiung -- KHH

Depart Date: 7/2007 31

7/2007 31

Flight Class: Economy/Coach

One Way [✓ Round Trip]

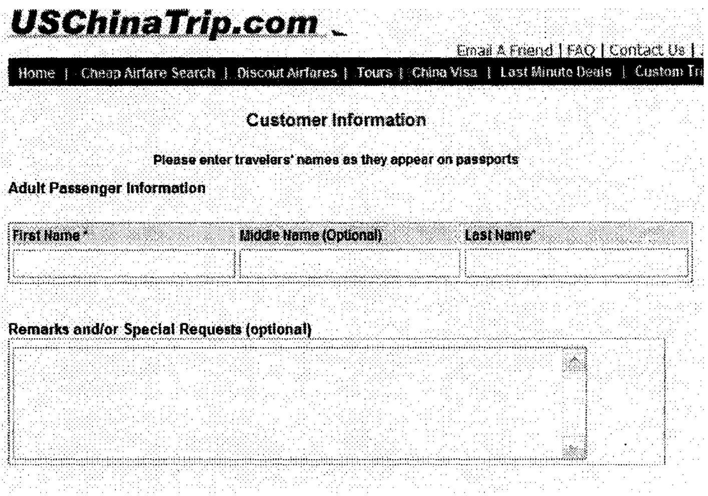
Airline: Select All Airline

Adult: 1 Child (2 - 11 yrs): 0 Infant (< 2 yrs): 0

Cheap airfare search

Send Multi-City Or Open-Jaw Request

(d) cp_3


USChinaTrip.com
Email A Friend | FAQ | Contact Us |
Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custom Tr
First Name *
Middle Name (Optional)
Last Name*

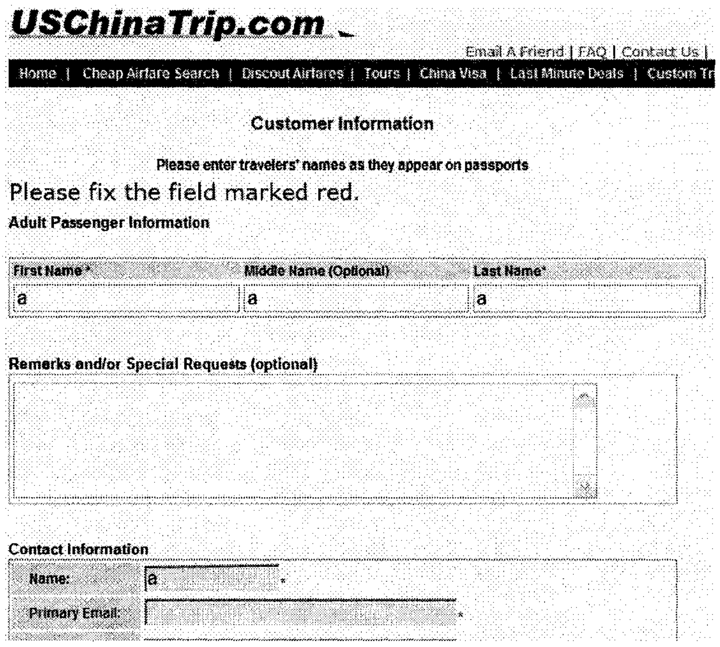
Customer Information

Please enter travelers' names as they appear on passports

Adult Passenger Information

Remarks and/or Special Requests (optional)

(e) cp_4


USChinaTrip.com
Email A Friend | FAQ | Contact Us |
Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custom Tr

Customer Information

Please enter travelers' names as they appear on passports

Please fix the field marked red.

Adult Passenger Information

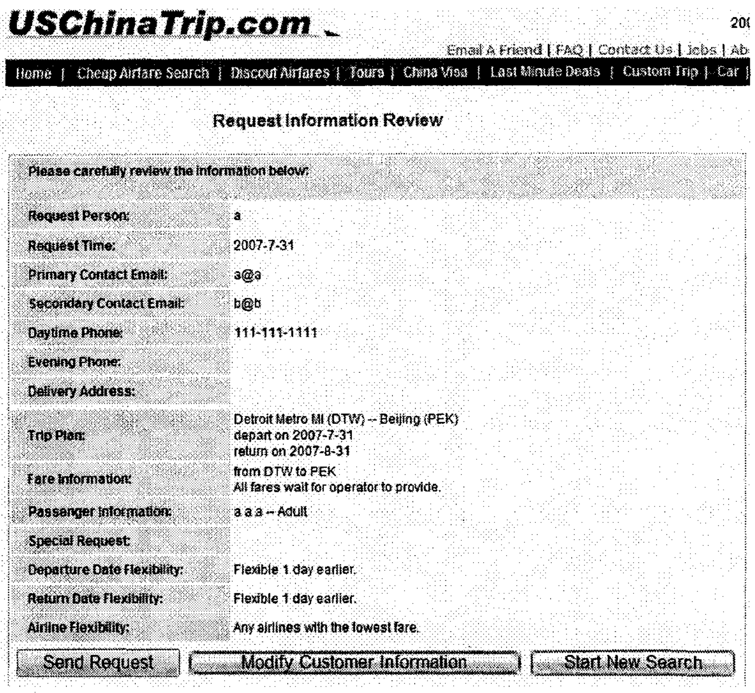
First Name *	Middle Name (Optional)	Last Name*
a	a	a

Remarks and/or Special Requests (optional)

Contact Information

Name:	a *
Primary Email:	*

(f) cp_5

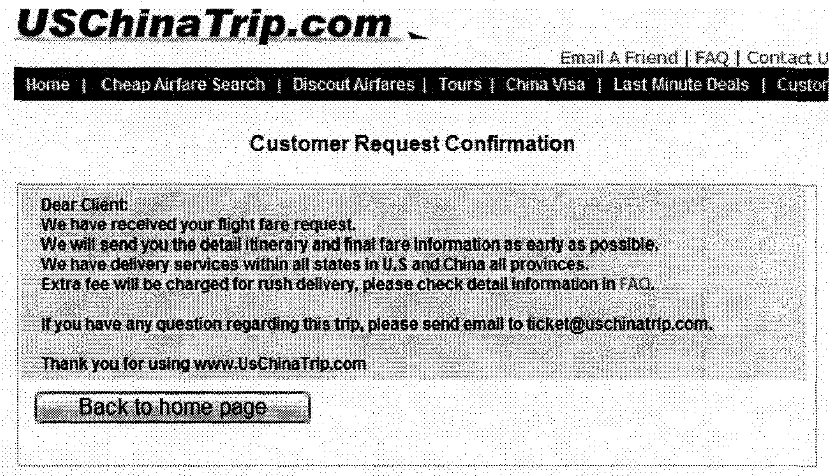

USChinaTrip.com
20(
Email A Friend | FAQ | Contact Us | Jobs | Ab
Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custom Trip | Car |
Send Request
Modify Customer Information
Start New Search

Request Information Review

Please carefully review the information below:

Request Person:	a
Request Time:	2007-7-31
Primary Contact Email:	a@a
Secondary Contact Email:	b@b
Daytime Phone:	111-111-1111
Evening Phone:	
Delivery Address:	
Trip Plan:	Detroit Metro MI (DTW) -- Beijing (PEK) depart on 2007-7-31 return on 2007-8-31
Fare Information:	from DTW to PEK All fares wait for operator to provide.
Passenger Information:	a a a -- Adult
Special Request:	
Departure Date Flexibility:	Flexible 1 day earlier.
Return Date Flexibility:	Flexible 1 day earlier.
Airline Flexibility:	Any airlines with the lowest fare.

(G) cp_6


USChinaTrip.com
Email A Friend | FAQ | Contact U
Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custor
Back to home page

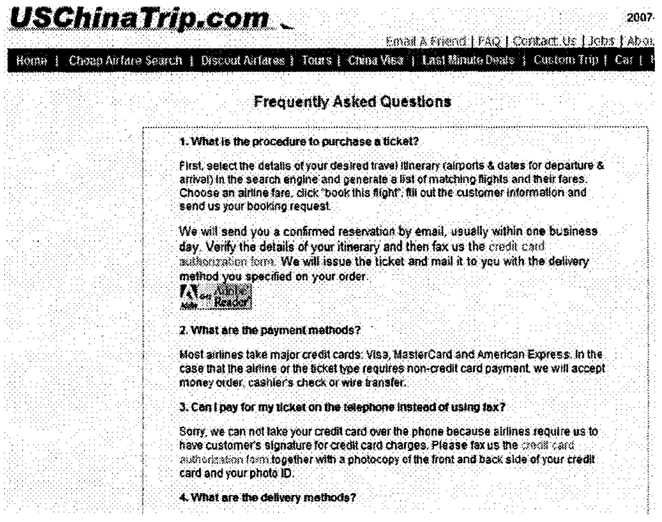
Customer Request Confirmation

Dear Client:
We have received your flight fare request.
We will send you the detail itinerary and final fare information as early as possible.
We have delivery services within all states in U.S and China all provinces.
Extra fee will be charged for rush delivery, please check detail information in FAQ.

If you have any question regarding this trip, please send email to ticket@uschinatrip.com.

Thank you for using www.UsChinaTrip.com

(H) cp_7


USChinaTrip.com 2007

Email A Friend | FAQ | Contact Us | Jobs | Abou

Home | Cheap Airfare Search | Discout Airfares | Tours | China Visa | Last Minute Deals | Custom Trip | Car |

Frequently Asked Questions

1. What is the procedure to purchase a ticket?

First, select the details of your desired travel itinerary (airports & dates for departure & arrival) in the search engine and generate a list of matching flights and their fares. Choose an airline fare, click "book this flight", fill out the customer information and send us your booking request.

We will send you a confirmed reservation by email, usually within one business day. Verify the details of your itinerary and then fax us the credit card authorization form. We will issue the ticket and mail it to you with the delivery method you specified on your order.

2. What are the payment methods?

Most airlines take major credit cards: Visa, MasterCard and American Express. In the case that the airline or the ticket type requires non-credit card payment, we will accept money order, cashier's check or wire transfer.

3. Can I pay for my ticket on the telephone instead of using fax?

Sorry, we can not take your credit card over the phone because airlines require us to have customer's signature for credit card charges. Please fax us the credit card authorization form together with a photocopy of the front and back side of your credit card and your photo ID.

4. What are the delivery methods?

(H) cp_8

FIGURE 3. 2 SOME CLIENT PAGE OF THE WEBSIT OF USCHINATRIP

This part of the web application is specified as FSM M_0 shown in Figure 3.3 and Table 3.1. Here, for $0 < i < 8$, state s_i represents cp_i. Since hyperlinks in different client pages or even within one client page can refer to the same client page, we use x_i^j to denote the input symbol that can produce y_i, where j is some integer. y_i denotes the characteristics of cp_i.

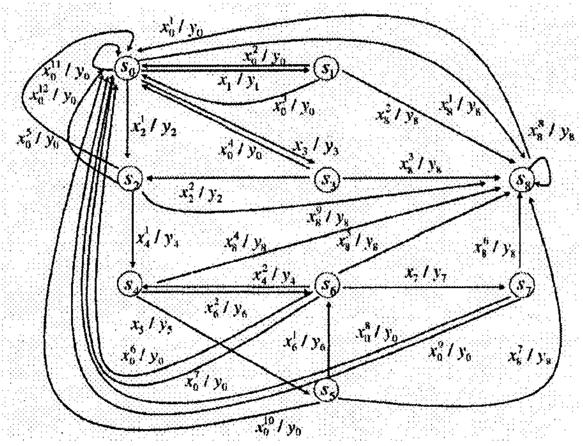


Figure 3.3 An example FSM for the cheap airfare search engine

Symbol	Meaning	Symbol	Meaning
y_0	The characteristic of client page cp_0	y_1	The characteristic of client page cp_1

y_2	The characteristic of client page cp_2	y_3	The characteristic of client page cp_3
y_4	The characteristic of client page cp_4	y_5	The characteristic of client page cp_5
y_6	The characteristic of client page cp_6	y_7	The characteristic of client page cp_7
y_8	The characteristic of client page cp_8	s_0	Client page cp_0
s_1	Client page cp_1	s_2	Client page cp_2
s_3	Client page cp_3	s_4	Client page cp_4
s_5	Client page cp_5	s_6	Client page cp_6
s_7	Client page cp_7	s_8	Client page cp_8
x_0^1	Clicking the hyperlink to s_0 with *null* (search parameters) in cp_8	x_1	Clicking the hyperlink to s_1 with the no-flight-return search parameters
x_0^2	Clicking one hyperlink to s_0 with *null* (search parameters) in cp_1	x_2^1	Clicking the hyperlink to s_1 with correct search parameters
x_0^3	Clicking the other hyperlink to s_0 with null (search parameters) in cp_1	x_2^2	Clicking the hyperlink to s_1 with correct search parameters
x_0^4	Clicking the hyperlink to s_0 with null (search parameters) in cp_3	x_3	Clicking the hyperlink to s_0 with incomplete search parameters
x_0^5	Clicking the hyperlink to s_0 with null (search parameters) in cp_2	x_4^1	Clicking one hyperlink to s_2 with flight information (search parameters)
x_0^6	Clicking one hyperlink to s_0 with null (search parameters) in cp_6	x_4^2	Clicking one hyperlink to s_2 with flight information (search parameters)
x_0^7	Clicking the other hyperlink to s_0 with null (search parameters) in cp_6	x_5	Clicking the hyperlink to s_2 with flight and customer's information (search parameters)
x_0^8	Clicking one hyperlink to s_0 with null (search parameters) in cp_7	x_6^1	Clicking the hyperlink to s_3 with flight and customer's information (search parameters)
x_0^9	Clicking the other hyperlink to s_0 with null (search parameters) in cp_7	x_6^2	Clicking the hyperlink to s_3 with flight and customer's information (search parameters)

x_0^{10}	Clicking the hyperlink to s_0 with null (search parameters) in cp_5	x_7	Clicking the hyperlink to s_4 with flight and customer's information (search parameters)
x_0^{11}	Clicking the hyperlink to s_0 with null (search parameters) in cp_0	x_8^1	Clicking the hyperlink to s_5 with flight and customer's information (search parameters)
x_0^{12}	Clicking the hyperlink to s_0 with null (search parameters) in cp_2	x_8^3	Clicking the hyperlink to s_5 with flight and customer's information (search parameters)
x_8^2	Clicking the hyperlink to s_5 with flight and customer information (search parameters)	x_8^5	Clicking the hyperlink to s_5 with flight and customer's information (search parameters)
x_8^4	Clicking the hyperlink to s_5 with flight and customer information (search parameters)	x_8^7	Clicking the hyperlink to s_5 with flight and customer's information (search parameters)
x_8^6	Clicking the hyperlink to s_5 with flight and customer information (search parameters)	x_8^8	Clicking the hyperlink to s_5 with flight and customer's information (search parameters)

TABLE 3. 1 EXPLAINATION OF THE SYMBOLS

Given an FSM describing the expected behavior of a *web application under test*, we would like to see, for each transition t, if we click on a hyperlink (possibly with some input data) corresponding to an input of the page represented by the starting state of t, then the resulting client page is correctly generated in the sense that both the displayed text content and hyperlinks are as characterized in the output of t. Furthermore, we would like to generate an input/output sequence (called *test sequence*) from a sequence of transitions such that each transition in the FSM is traversed at least once. This will give us continuous tests of each transition. In the following, all *test sequences* refer to those where each transition is traversed at least once. Optimal solution on generating a minimal-length test sequence in this setting is generally discussed in T-method [21]. It was based on a well-known algorithm in graph theory applied to the directed graph of a given FSM. We introduce below some notations on directed graphs before we present our improved algorithm.

For convenience to apply the graph theory on FSM, we need a transformation from FSM to a corresponding digraph. A directed graph (digraph) G is defined by a tuple (V,E,L) in which V is a set of vertices, E is a set of directed edges between the vertices. Each edge may have a label in L. An edge e from vertex v_i to vertex v_j with label l will be represented by (v_i,v_j,l), and we say edge e *leaves* v_i and *enters* v_j.

Each FSM M has a graph representation $G=(V,E,L)$, in which a state of M is represented by a vertex from V and a transition of M is represented by a vertex from E. We use G_M to denote the graph representation of FSM M, where state s_i is represented by vertex v_i, and transition from s_i to s_j with label x/y represented by edge $(v_i,v_j,x/y)$.

A *path* τ in a digraph G is either *null*, denoted by ε, or a finite sequence of edges $e_1e_2...e_k(k \geq 1)$ in G such that for $k \geq 2$, the ending vertex of e_i is the starting vertex of e_{i+1} for all $i \in \{1,2,...,k-1\}$. Given path $\tau = e_1e_2...e_k$ $(k \geq 1)$, we use $staring(\tau)$ and $ending(\tau)$ to denote the starting vertex e_1 and the ending vertex e_k respectively. A tour is a path starting from and ending at the same vertex.

A digraph is *strongly connected* if for any ordered pair of vertices (v_i,v_j) there is a path from v_i to v_j. The digraphs we consider here are all strongly connected. In fact, we assume that all client pages are reachable from the *home page* and that from any page we can reach the *home page* by finite steps of hyperlink clicks. When G is a strongly connected, a *postman Tour* of G is a tour which contains every edges of E at least once. Given digraph $G=(V,E,\text{cos}t)$, where *cost* is a cost function that associates each edge in E with a cost, we say G is a weighted digraph. The Chinese Postman Problem is to find the minimum-cost Postman Tour in a strongly connected (weighted) digraph.

Given FSM M, when its graph representation G_M is strongly connected, the optimization problem of finding a minimal-length test sequence corresponds to finding a minimal-length path that traverses each edge of G_M at least once and this problem is reduced to CPP in G_M where the cost of each edge in G_M is 1. It is possible to further reduce the length of the test sequence when the functionality of the web browsers is considered. In the following section, we consider how to make use of the *back* button of web browsers to reduce the length of the generated test sequence.

4. EXISTING ALGORITHMS

In this section, we introduce two algorithms for the CPP and for the minimum and maximum flow problem respectively in detail. They form the basis of our solutions.

4.1 THE IMPLEMENTATION OF THE CPP ALGORITHM

CPP is a very famous problem: finding the shortest route for the Chinese postman who wishes to travel along every road to deliver letters. i.e., in a directed graph (here we only discuss the directed weighted CPP), we need to find a closed walk with minimum weight which covers each edge at least once.

To well understand the algorithm to solve the CPP, we introduce some graph-theoretic terms first. The number of edges going into a vertex v is the in-degree denoted by $\deg_{in}(v)$, and the number of edges pointing out of a vertex v is the out-degree denoted by $\deg_{out}(v)$. Let δ be the difference between the in and out degrees: $\delta(v) = \deg_{out}(v) - \deg_{in}(v)$. If $\delta(v) = 0$, we say the vertex v is balanced. Otherwise, let $D^+ = \{v \mid \delta(v)>0\}$ be the set of unbalanced vertices with an excess of out-going edges, and $D^- = \{v \mid \delta(v)<0\}$ the set of unbalanced vertices with an excess of in-going edges.

An *Eulerian graph* [10] is a graph that has a circuit traversing each edge exactly once, and returning to the start vertex. A standard theorem is that a graph has an Euler circuit if and only if every vertex is balanced. If a graph is Eulerian, we can easily find an Euler circuit in the graph. An Euler circuit of a graph is an *optimal Chinese*

Postman Tour (CPT), since each edge is traversed exactly once. If a graph is not Eulerian, there should exist nonempty D^+ and D^- in the graph. Thus the CPT will have to walk extra paths to 'join up' the unbalanced vertices in D^- and D^+ to make each vertex balance and the number of extra paths is equal to $\kappa = \sum_{n \in D^+} \delta(v)$. For an optimal CPT, the extra path taken between D^- and D^+ will be chosen to have the least total cost. In general, a CPT may take some of the $D^- \to D^+$ paths more than once. Let f_{ij} be the number of times the path $i \to j$ must be taken to be added to the original graph as an edge to make the graph Eulerian. We use $\phi = \sum C_{ij} f_{ij}$ to represent the additional cost of traversing the chosen additional paths. The key idea of the algorithm of CPP is to find an optimal ϕ.

If we want an optimal ϕ in a directed graph, we need to find an optimal set of extra paths. Generally speaking, the number of extra paths is equal to $\kappa = \sum_{v \in D^+} \delta(v)$. The first of these κ paths might go to any of the (at most) κ vertices in D^+, the second can be any of the remaining $\kappa - 1$, and the third to any of the remaining $\kappa - 2$, and so on. In the worst case there are $\kappa!$ choices. Clearly it would be an inefficient algorithm to examine all the choices and pick the one with the least cost. We use the algorithm of cycle canceling [12] to find the optimal extras paths. This algorithm starts with an approximate solution, and then iteratively improves it.

In [33], an algorithm of CPP can be sketched as:

Step 1: determine δ of each vertex in graph G. If δ = 0, go to Step 5;

Step 2: determine D^+ and D^- in G.

Step 3: For each vertex in G, find the shortest paths p_{ij} and minimal costs c_{ij} to all vertices by the Floyd-Warshall algorithm [32].

Step 4: find f to minimize $\phi = \sum C_{ij} f_{ij}$ by the algorithm of cycle canceling, where $f_{ij} \geq 0$ should be integer, $\sum_{v \in D^+} f_{ij} = -\delta(i)$ and $\sum_{v \in D^-} f_{ij} = \delta(i)$.

Step 5: Construct an Eulerian circuit by Fleury's algorithm [34] based on the least cost paths $i \to j$ and each path repeated $f_{ij} \geq 0$ times.

The complexity of this algorithm is $O(n^2 m^3 \{\log n\})$. We have implemented a Java program for these five steps and the code is given in Appendix A.

Here we use Figure 4.1 to show the algorithm to solve the CPP. In this figure, the weight of each edge is '1'. In Step 1, $\delta(0)=1$, $\delta(1)=1$, $\delta(2)=-1$ and $\delta(3)=-1$ are computed. From the result of Step 1, we can determine that $D^+=\{0,1\}$ and $D^-=\{2,3\}$ in Step 2. In Step 3, we determine p_{ij} and the minimal costs c_{ij}. In Step 4, based on D^+, D^-, p_{ij} and c_{ij}, we find that there are two ways to choose the set of extra paths. If one path is $2 \to 0$, then the other path is $3 \to 1$; the alternative is to use the paths $2 \to 1$ (from vertex 2, pass by vertex 3, 0, to vertex 1) and $3 \to 0$. As it happens, the choices have the equal cost ($c_{21}+c_{30}=3+1, c_{20}+c_{31}=2+2$), and both can be used for an optimal CPT. Let us say that we choose the paths $2 \to 1$ and $3 \to 0$. In Step 4, an Eulerian circuit is found: $0, 1, 3, 0, 1, 2, 3, 0, 2, 3, 0$.

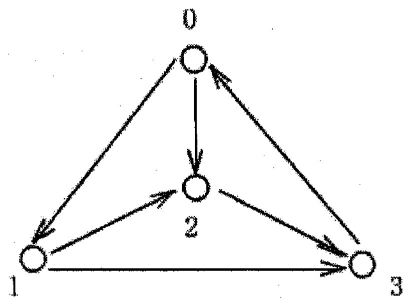


FIGURE 4. 1 AN EXAMPLE OF THE CPP

4.2 THE IMPLEMENTATION OF THE ALGORITHM OF NEGATIVE CYCLE-CANCELING FOR MINIMAL-MAXIMUM FLOW PROBLEM

A *network* $N=(V', E', capacity.\cos t, s, t)$ is a digraph, where

- V' is a finite set of vertices.
- E' is a finite set of edges.
- $capacity$: $E' \to N \cup \{0\}$, where N is the set of natural numbers, is the *capacity function*. We call the nonnegative $capacity(e)$ *capacity* on each edge

$e \in E$.

- $\cos t$: $E' \rightarrow N \cup \{0\}$ is the $\cos t$ *function*. We call the nonnegative $\cos t(e)$ $\cos t$ *per unit flow* on each edge $e \in E$.
- $s \in V'$ is the *source*.
- $t \in V'$ is the *sink*.

A *flow* $f : E' \rightarrow N \cup \{0\}$ is a function to assign a value $f(e)$ to each edge e. We use $f^{+}(v)$, $f^{-}(v)$ to denote the total flow on edges leaving v and edges entering v respectively. A flow is *feasible* if for any $e \in E'$, $0 \leq f(e) \leq capacity(e)$ and for any $v \in V' - \{s,t\}$, $f^{+}(v) = f^{-}(v)$. The *network flow* is the flow entering the sink t, i.e., $f^{-}(t)$. A *maximum-flow* and *minimum-cost* problem is to determine a feasible maximum network flow while $\sum_{e \in E'} \cos t(e) \times f(e)$ is minimum.

The algorithm of negative cycle-canceling is originally designed to solve the minimum cost flow problem, but if the flow in a given network is maximum, such algorithm also fits for min-max flow problem. Our case meets the requirement. This algorithm provides a feasible solution for maximum network flow. At each iteration, it attempts to improve its objective function value. The algorithm first establishes a feasible flow f in the network. Then it iteratively finds negative cost-directed cycles in the *residual network* and augments flows on these cycles [2]. The residual network $G(f)$ corresponds to a flow f and we replace each edge $(i,j) \in E$ by two arcs (i,j) and (j,i). The arc (i,j) has cost c_{ij} and *residual capacity* $r_{ij} = cap_{ij} - f_{ij}$ and the arc (j,i) has cost $c_{ji} = -c_{ij}$ and residual capacity $r_{ij} = f_{ij}$. The residual network consists only of arcs with positive residual capacity. The negative cycle-canceling algorithm terminates when the residual network contains no negative cost-directed cycle. When there is no negative cycle in the residual network, a minimum cost flow is found.

We use the example shown in Figure 4.2 to illustrate the cycle-canceling algorithm. Figure 4.2.1(a) depicts a feasible flow in the network and Figure 4.2.1(b) gives the corresponding residual network. Suppose that the algorithm first selects the cycle

4-2-3-4 whose cost is -1. The residual capacity of this cycle is 2. The algorithm augments 2 units of flow along this cycle. Figure 4.2.1(b) shows the modified residual network. In the next iteration, suppose that the algorithm selects the cycle 4-2-1-3-4 whose cost is -2. The algorithm sends 1 unit of flow along this cycle. Figure 4.2.1(d) depicts the updated residual network. Since this residual network contains no negative cycle, the algorithm terminates.

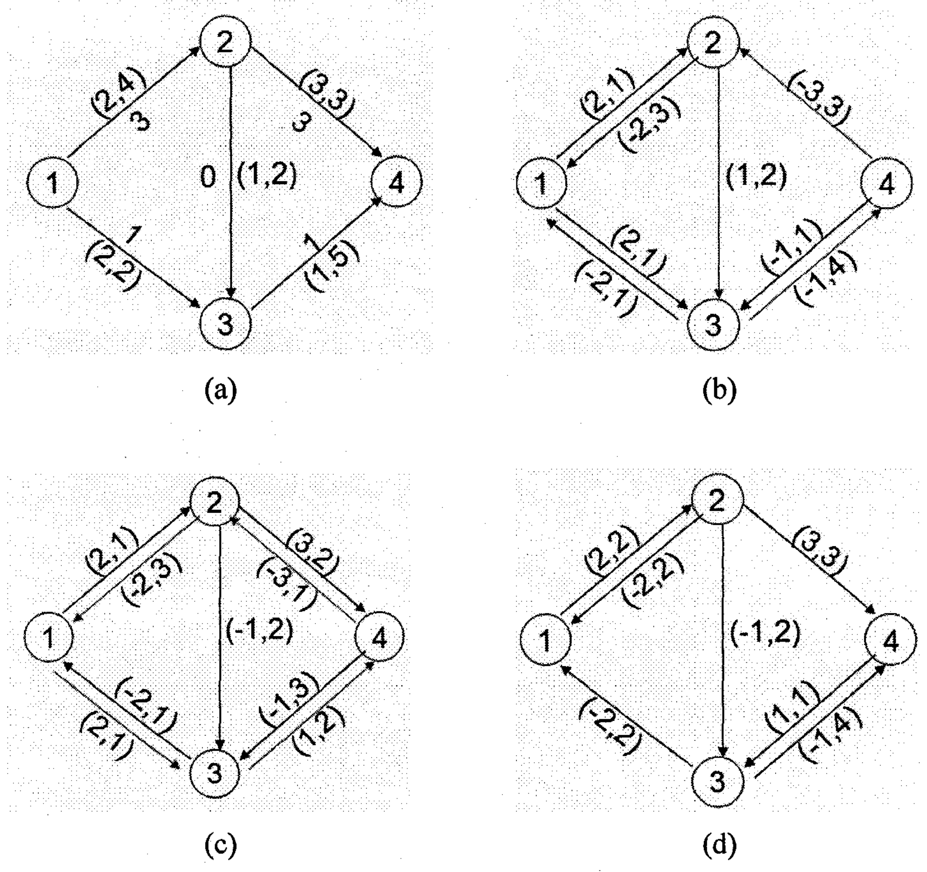


FIGURE 4. 2 EXAMPLE OF THE ALGORITHMS OF NEGATIVE CYCLE-CANCELING

The complex of this algorithm is $O(nm^2CU)$ where, for any c_{ij}, cap_{ij}, $c_{ij} \leq C$ (C is the maximum value in c_{ij}) and $cap_{ij} \leq U$ (here U is the total number of the units of the flow).[2]

5. OUR TESTING METHOD

5.1 BACK BUTTON OF THE WEB BROWSER

According to major web browsers we use nowadays, such as the Internet Explore, Mozilla Firefox, Safari, and Opera, each web browser maintains a *history stack* to keep the previously visited URLs and uses a stack pointer to record the URL of the current client page. The history stack can be accessed by making use of the *forward* button and the *back* button on the *navigation bar* of the browser.

For any two client pages cp_i, cp_j in the web application under test, let τ be the shortest path to navigate from cp_i to cp_j. With the use of the *back button,* it is possible that there exists a path τ' to navigate from cp_i to cp_j such that τ' is shorter than τ. Consequently, by making use of the *back button,* the total length of the test sequence may be reduced.

In the Example of www.uschinatrip.com, suppose that the current navigation history is cp_0 cp_2 cp_4 cp_6 cp_8. Now we want to visit client page cp_7. Without the *back button,* the shortest path to reach cp_7 is to visit the following client pages in sequence: cp_0 cp_2 cp_4 cp_6 cp_7. By making use of the *back button,* cp_7 can be reached by only two clicks: first click the *back button* to return to cp_6 and then click the hyperlink associated with cp_7 in cp_6, and Figure 5.1.1 shows the shortest path (bold lines)

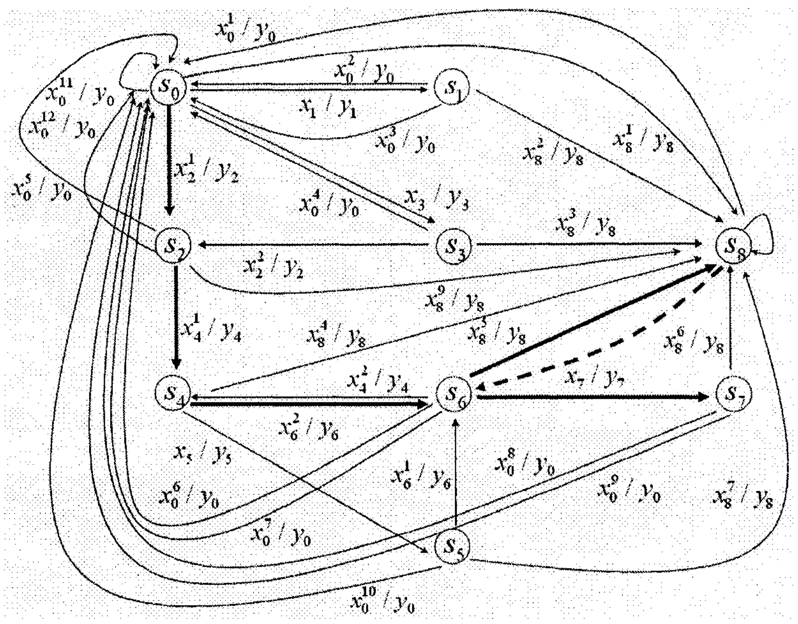


FIGURE 5. 1. 1 AN EXAMPLE TO SHOW HOW TO USE THE BACK BUTTON

Note that we do not consider the *forward* button because its use cannot save the steps of hyperlink clicks for the navigation between any two client pages.

When the test specification of a web application is given as an FSM M, traditionally, the minimal-length test sequence can be generated by applying CPP algorithm directly on G_M. Here, we propose the following optimization problem: find a minimal-length test sequence from a given FSM M with the use of the *back* button.

The following theorem claims that the optimal solution to this problem is better than the one produced by T-method in our setting.

Theorem 1 *Let M be a given minimal FSM where its graph representation* $G_M = (V,E,L)$ *is strongly connected. Let* χ *be a minimal-length test sequence generated from* M *with the use of the back button. For any test sequence* χ'*, we have* $|\chi| \leq |\chi'|$.

Proof: To consider the use of the *back* button for generating test sequences, we need to incorporate the operational behavior of the *back* button into the given test specification M. Suppose that $M = (S, s_0, X, Y, \lambda, \delta)$. We define another digraph $G_{M,back}$ that represents the navigation among client pages with *back* button.

$G_{M,back} = (V_{back}, E_{back}, L_{back})$ is defined as follows:

- Let *back* be a special symbol ($back \notin X$), which represents the user's action of triggering the *back* button on the web browser.
- Let $Q = \{q_i \mid s_i \in S\}$ be a set of state variables such that for each state $s_i \in S$, there is exactly one state variable $q_i \in Q$ corresponding to it.
- $V_{back} = \{r_i \mid s_i \in S\}$
- $L_{back} \subseteq X \cup \{back\} \times Y \times P \times A$ is a set of labels, where P and A represent the set of predicates and actions on state variables of Q.
- For any $s_i, s_j \in S$, if $s_j = \delta(s_i, x)$ and $y = \lambda(s_i, x)$, we add to E_{back} two edges $e = (r_i, r_j, (x, y, true, q_j := s_i))$ and $\bar{e} = (r_j, r_i, (back, y', q_j = s_i, q_j := null))$. Where y and y' are the characteristics of client pages represented by s_j and s_i respectively.

The third element of a label in an edge of E_{back} represents a predicate on the current variable values and the fourth element of a label in an edge of E_{back} represents the action of assigning new variable values. The intuitive meaning of such an edge $(r_i, r_j, (x, y, p, a))$ is: when the predicate p is evaluated to true in state s_i and x is prompted as an input, y will be produced as an output and the state is transferred to s_j where the new values of the state variables are obtained by executing a on the values of the state variables in state s_i. Clearly, when p is *true*, such an edge represents a transition that is executable in any circumstance. The paths we consider in G_M should satisfy the property that if $(r_i, r_j, (x, y, p, a))$ appears in the path, then when we reach the state represented by r_i, the evaluation of p in this state should return true.

Let $E'_{back} \subset E_{back}$ denote the set of edges representing the transitions with predicate *true*. Since χ is a minimal-length test sequence with the use of the *back* button, the path ρ of χ corresponds to a minimal-length tour in G_{back} that traverses each edge in E_{back} at least once.

On the other hand, any $e \in E$ is related to an edge in E_{back} with corresponding starting and ending vertices and with predicate *true*. Hence, if ρ' is the path of χ', ρ' corresponds to a tour in $G_{M,back}$ that traverses each edge in E_{back} at least once. Since ρ is a tour of minimal-length in $G_{M,back}$, we have $|\rho| \leq |\rho'|$. It follows that $|\chi| \leq |\chi'|$.

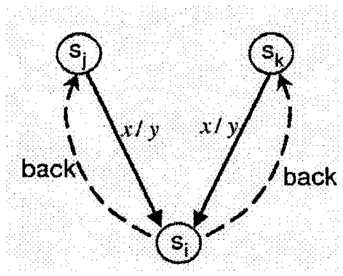


FIGURE 5. 1. 2 AN ILLUSTRATION OF THE RESTRICTION FOR THE USE OF THE BACK BUTTON

Theorem 1 suggests that a shorter test sequence is possible when the use of the *back* button is considered. A natural way to reach a better solution is to augment G_M into G'_M by adding edges representing the *back transitions*. However, restricted by the way that the history stack works, finding the minimal-length test sequence in G'_M is non-trivial: Suppose that both client page cp_j (represented by s_j) and client page cp_k (represented by s_k) contain a hyperlink referring to client page cp_i (represented by s_i) (see Figure 5.1.2). When the *back* button is clicked in cp_i, we cannot unconditionally get back to the page represented by s_j or that represented by s_k even though there exist two edges (s_i, s_j, back/y_j) and (s_i, s_k, back/y_j) in the digraph. Which one is allowed is dynamically determined by the browsing history: the client page whose URL is pushed into the history stack right before the one of cp_i will be returned when the *back* button is clicked. In other words, when using the *back* button, the browsing history should be taken into account: the sequence of client pages reached by continually clicking the back button should be exactly the reverse of the sequence of client pages previously visited by continuously clicking the hyperlinks.

We say a test sequence χ uses the back button *properly* if it conforms to the semantics of the web browsers. In the next section, we present an algorithm to reduce the length of the test sequence with a given FSM M when the *back* button is properly used.

5.2 REDUCING THE LENGTH OF TEST SEQENCE

Given a test specification $M = (S, s_0, X, Y, \delta, \lambda)$, our algorithm is divided into two steps in the same way as the above introduced solution to the CPP problem:

1) Construct a minimal symmetric and strongly connected digraph G^* from G_M such that the *proper* use of the back button is guaranteed. We say a digraph is symmetric if for any vertex v, the number of edges entering v is equal to the number of edges leaving v.
2) Find a Euler tour τ in G^*. Then the label of ρ represents the desired test sequence.

We first discuss step 1) of the algorithm in details: to construct a symmetric and strongly connected graph G^* from G_M. Augmenting a given digraph G_M to be symmetric in our setting is carried out as follows:

- identify all the vertices that are not symmetric;
- find *additional paths*, which consist of edges in G_M and edges representing the navigation triggered by clicking the *back* button, to connect these vertices such that each vertex in the resulting graph is symmetric.

When a given graph is strongly connected, its minimal symmetric graph can be obtained by augmenting the edges found by applying maximum-flow and minimum-cost network flow algorithm on the given graph. Our first step is based on the well-known solution to the maximum-flow and minimum-cost network flow problem.

The step 1) of algorithm is described below.

1.1) Construct a *network* N_M from G_M such that the use of the *back* button is considered.

1.2) Use the maximum-flow and minimum-cost network flow derived from step 1.1) to construct the desired digraph G^*.

In the following, we show how such a network is constructed in step 1.1). We will use $\deg_{in}(v)$ and $\deg_{out}(v)$ to denote the number of edges entering v and the number of edges leaving v respectively.

First, define a network $N_M = (V', E', capacity, \cos t, s, t)$ from $G_M = (V, E, L)$ as follows:

- $V' = \{s,t\} \cup W \cup Z$, where
  - $W = \{w_i \mid v_i \in V\}$;
  - $Z = \{z_i \mid v_i \in V\}$;
- $E' = E_1 \cup E_2 \cup E_3 \cup E_4 \cup E_5$, where
  - $E_1 = \{(s, w_i) \mid \deg_{in}(v_i) - \deg_{out}(v_i) > 0, v_i \in V\}$;
  - $E_2 = \{(z_i, t) \mid \deg_{in}(v_i) - \deg_{out}(v_i) < 0, v_i \in V\}$;
  - $E_3 = \{(w_i, w_j) \mid (v_j, v_i, x/y) \in E \textit{ for some input } x \textit{ and output } y\}$;
  - $E_4 = \{(w_i, w_j) \mid (v_j, v_i, x/y) \in E \textit{ for some input } x \textit{ and output } y\}$;
  - $E_5 = \{(w_i, z_i) \mid w_i \in W, z_i \in Z\}$;
- The capacity function is defined as: *capacity (e) =*

$$\begin{cases} \deg_{in}(v_i) - \deg_{out}(v_i), \textit{ if } e = (s, w_i) \in E_1 \\ \deg_{out}(v_i) - \deg_{in}(v_i), \textit{ if } e = (z_i, t) \in E_2 \\ 1, \qquad \textit{if } e \in E_3 \\ \infty, \qquad \textit{if } e \in E_4 \cup E_5 \end{cases}$$

- The cost function is defined as:

$$\cos t(e) = \begin{cases} 1, & \text{if } e \in E_3 \cup E_4 \\ 0, & \text{if } e \in E_1 \cup E_2 \cup E_5 \end{cases}$$

Here, each edge in $E_1 \subseteq \{s\} \times W$ represents a vertex v in G_M where the number of edges entering v is greater than the number of edges leaving v. Each edge in $E_2 \subseteq Z \times \{t\}$ represents a vertex v in G_M where the number of edges entering v is less than the number of edges leaving v. Edges in $E_3 \subseteq W \times W$ represent the connectivity among the client pages formed by clicking the back button. Without loss of generality, we assume all the transitions in M are *backable* in the sense that for any transition $(s_i, s_j, x/y)$ in M, where s_i and s_j represent cp_i and cp_j respectively, if cp_i is visited right before cp_j, then cp_i can be reached by clicking the *back* button in cp_j. In the case where some transitions are not *backable*, we can simply assign the capacity of the edges in E_3 that correspond to these *unbackable* transitions to be 0. Edges in $E_4 \subseteq Z \times Z$ represent the connectivity among the client pages specified by the test specification M. $w_i \in W$ and $z_i \in Z$ correspond to a same client page and we use an edge in E_5 to connect them.

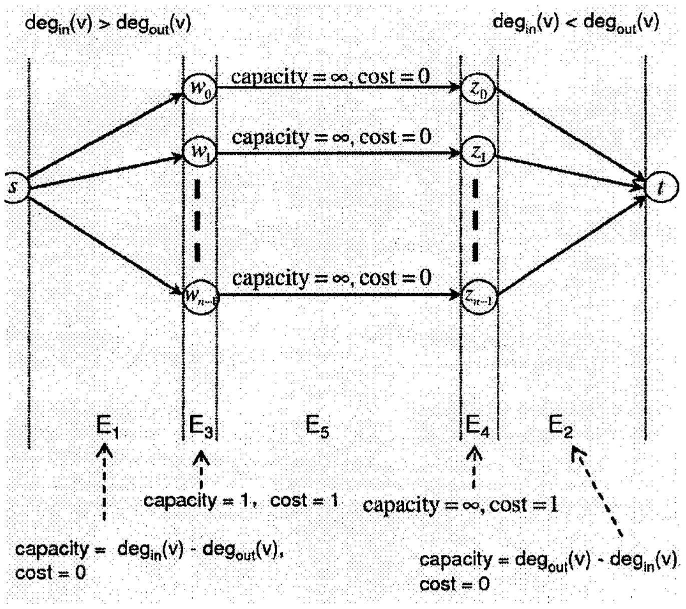


FIGURE 5. 2. 1 AN ILLUSTRATION FOR NETWORK CONSTRUCTION

Figure 5.2.1 illustrates how the network N_M is constructed. For clarity, we have only shown the edges in E_1, E_2, and E_5. Edges in E_3 and E_5 are omitted. The following lemma claims that the above procedure always yields a result as we expected: Part A. expresses that it is always feasible to find a maximum flow for the network defined above. Part B. says that adding the edges derived from the maximum flow to G_M will make it symmetric; and Part C. claims that the total number of edges added in B. is the minimal among all edges to be added to G_M to make it *symmetric*. We also give the proof of this lemma.

Lemma 1 *Let* $M = (S, s_0, X, Y, \delta, \lambda)$ *be a given minimal FSM where its graph representation* $G_M = (V, E, L)$ *is strongly connected. Let* $N_M = (V', E', capacity, \cos t, s, t)$ *be a network derived from* G_M *according to the above definition. Let maxflow denote the maximum feasible flow in* N_M. *Let* f *be the solution to the maximum-flow and minimum-cost network flow problem on* N_M and $\Gamma = \{\gamma_1, ..., \gamma_r\}$ *the set of paths derived by* f *such that for any* $1 < i < r$, γ_i *carries exactly one unit of flow. Let* E_i'' *be the set of edges in* E'' *in* γ_i, *and let* E'' *be the multiset union of* E_i'' *for* $1 < i < r$.

A. maxflow exists: maxflow = $\sum_{e \in E_1} capacity(e) = \sum_{e \in E_2} capacity(e)$.

B. Digraph $(V, E \cup E'')$ *is symmetric.*

C. Let E^+ *be any multiset of edges such that digraph* $(V, E \cup E^+)$ *is symmetric. We have* $|E'| \leq |E^+|$.

Proof: A. From graph theory, we have $\sum_{v \in V} \deg_{in}(v) = \sum_{v \in V} \deg_{out}(v)$ for G_M. v can be partitioned into three classes: v_1, v_2, and v_3, where

$V_1 = \{v \in V \mid \deg_{in}(v) > \deg_{out}(v)\}$;

$V_2 = \{v \in V \mid \deg_{in}(v) < \deg_{out}(v)\}$;

$V_2 = \{v \in V \mid \deg_{in}(v) = \deg_{out}(v)\}$.

Thus, we have

$$\sum_{v\in V}\deg_{in}(v)=\sum_{v\in V_1}\deg_{in}(v)+\sum_{v\in V_2}\deg_{in}(v)+\sum_{v\in V_3}\deg_{in}(v).$$

Similarly,

$$\sum_{v\in V}\deg_{in}(v)=\sum_{v\in V_1}\deg_{in}(v)+\sum_{v\in V_2}\deg_{in}(v)+\sum_{v\in V_3}\deg_{in}(v)$$

Since $\sum_{v\in V_3}\deg_{in}(v)=\sum_{v\in V_1}\deg_{out}(v)$, we have

$$\sum_{v\in V_1}(\deg_{in}(v)-\deg_{out}(v))=\sum(\deg_{out}(v)-\deg_{in}(v)).$$

According to the way we construct N_M, it follows that

$$\sum\nolimits_{e\in E_1} capacity(e)=\sum_{e\in E_2} capacitye(e).$$

Since the capacity of any edge in $E_4\cup E_5$ is infinite and the digraph (Z,E_4) is strongly connected, we have that the maximum feasible flow *maxflow* is $\sum\nolimits_{e\in E_1} capacity(e)=\sum\nolimits_{e\in E_2} capacity(e)$.

B. Adding all edges in a path ρ of a graph G into G will change the degrees of only two vertices of G: the out-degree of vertex $start(\rho)$ is increased by one, and the in-degree of vertex $end(\rho)$ is increased by one. Each $\gamma_i\in\Gamma$ will increase by one the out-degree of a vertex whose out-degree is less than its in-degree, and increase by one the in-degree of a vertex whose in-degree is less than its out-degree. According to A, $\sum\nolimits_{e\in E_1} capacity(e)=\sum\nolimits_{e\in E_2} capacity(e)$. So by adding E'' into G_M, the in-degree and out-degree of each vertex will become the same. That is, $(V,E\cup E'')$ is symmetric.

C. In general, when we want to find a shortest navigation path from one client page to another, we can i) go back to some previously visited client page by continually clicking the *back* button; and then ii) continually click the hyperlinks to reach the desired client page. Correspondingly, a path to increase the out-degree of one vertex by one and increase the in-degree of another vertex by one should have a sequence of edges representing the actions of clicking *back* buttons, followed by a sequence of edges that represent the actions of clicking hyperlinks.

Now, each path in Γ is denoted by a flow of one unit from the source s to the destination t. According to the way we construct N_M, it is in the form $e_1@\rho_1@e_2@\rho_2@e_3$, where $e_1 \in E_1$, $e_2 \in E_5$, $e_3 \in E_2$, ρ_1 and ρ_2 are possibly null path. ρ_1 is a path consisting of edges in E_3, and thus it corresponds to a navigation path obtained by continually clicking the *back* button. Similarly, ρ_2 is a path consisting of edges in E_4 and thus it corresponds to a navigation path specified in the test specification M.

Since the computed *maxflow* has minimum-cost, $(V, E \cup E^+)$ is symmetric implies that $|E''| \leq |E^+|$.

It is straightforward that N_M can be constructed from G_M in $O(|V| \leq |E|)$ time. According to [9, 11], the maximum-flow and minimum-cost network flow problem can be resolved in $O(|V'|^2 \leq |E'|^3 \log|V'|)$ time. Since $|V'|$ and $|E'|$ are of linear order of $|V|$ and $|E|$ respectively, the time complexity of our present algorithm for step 1.1) is $O|V|^2|E|^3 \log|V|$.

5.3 FINDING A TOUR

We have shown how to derive a symmetric digraph from G_M using network flow algorithm. An Euler tour can be easily obtained from this symmetric digraph using traditional algorithm. However, this will not yield a tour where the *back* button is *properly* used.

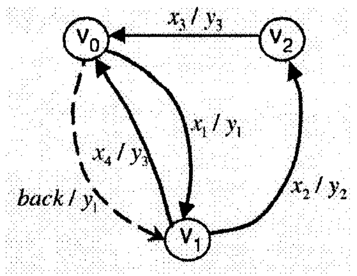


FIGURE 5. 3. 1 AN EXAMPLE OF G.

In Figure 5.3.1, edges with solid arrows represent transitions specified in FSM M_1 and the edge with dashed arrow represents the transition triggered by clicking the *back* button. It is obvious that this digraph G is symmetric and strongly connected and thus an Euler tour exists. However, not in any Euler tour in G, the *back* button is properly used. For example,

$$\tau = (v_0, v_1, x_1/y_1)(v_1, v_2, x_2/y_2)(v_2, v_0, x_3/y_3)(v_0, v_1, back/y_1)(v_1, v_0, x_4/y_3)$$

is an Euler tour in G. When the *back* button is clicked, instead of client page cp_1 (represented by v_1), client page cp_2 (represented by v_2) will be returned. This is due to the wrong order of those edges representing the transitions by the *back* action in the tour. In order to use the *back* button properly, edge $(v_1, v_0, x_4/y_3)$ should be traversed right before edge $(v_0, v_1, back/y_1)$.

In the following, we explain our modifications on the traditional algorithm for constructing Euler tour so that the *back* button is properly used.

Suppose that function f is the solution to the maximum-flow and minimum-cost flow problem of the constructed N_M. Let $\Gamma = \{\gamma_1, \ldots, \gamma_r\}$ be the set of paths derived from f such that for any $1 < i < r$, γ_i carries exactly one unit of flow. According to the way we construct N_M. we have that $\gamma_i = e_i^1 \rho_i^1 e_i^2 \rho_i^2 e_i^3$, where $e_i^1 \in E_1$, $e_i^2 \in E_5$, $e_i^3 \in E_2$, ρ_i^1 *and* ρ_i^2 contain edges only in E_3 and E_4 respectively. Let $\Omega^1 = \{\rho_1^1, \rho_2^2, \ldots, \rho_l^1\}$ and $\Omega^2 = \{\rho_1^2, \rho_2^2, \ldots, \rho_h^2\}$ for some integer l and h. Note that for any $\rho_i^1 \in \Omega^1$, ρ_i^1 is not a null path. Similarly, any $\rho_i^2 \in \Omega^2$ is not a *null* path. For each $\rho_i^1 \in \Omega^1$, let $\rho_i^1(\omega_i^1, \omega_i^2)(\omega_i^2, \omega_i^3)\ldots(\omega_i^k, \omega_i^{k+1})$, where $k = |\rho_i^1|$. We use $\theta(\omega_i^j)$ to denote the vertex corresponding to ω_i^j in V of G_M. A selfloop path induced by ρ_i^1 is defined as $\sigma_i^1 = (\theta(\omega_i^{k+1}), \theta(\omega_i^k), x_k / y_k) ... (\theta(\omega_i^2), \theta(\omega_i^1, x_1 / y_1)$ which is called a *forward* path; and $\sigma_i^2 = (\theta(\omega_i^1), \theta(\omega_i^2), back / y_2) ... (\theta(\omega_i^k), \theta(v_i^{k+1}), back / y_{k+1})$ which is called a *backward path.* Here, x_j denotes the input symbol that contains a hyperlink and the search parameters associated with the client page cp_j and y_j denotes the output characteristics of client page cp_j.

Given test specification M, and Ω^1 and Ω^2 constructed from $G_M = (V, E)$, the digraph $G^* = (V^*, E^*)$ is initially defined as follows:

- $V^* = \{v_i^* \mid v_i \in V\}$;
- $E^* = E_1^* \cup E_2^* \cup E_3^*, where$
  - $E_1^* = \{(v_1^*, v_1^*, label(\sigma_i)) \mid \exists \rho_i^1 \in \Omega^1, v_1 = start(\sigma_i)\}$.

    The *self-loop* edges are introduced here to guarantee the proper execution of the *back* button.
  - $E_2^* = \{(v_1^*, v_1^*, x / y) \mid \exists e = (v_1, v_2, x / y) \in E$ s.t. $e \notin \sigma_i^1$ for any $\rho_i^1 \in \Omega^1$ where σ_i^1 is the forward path of ρ_i^1 }. E_2^* is a subset of edges in E of G_M such that it only includes those edges not contained in the forward path of the selfloop path induced by $\rho \in \Omega^1$ for $1 \le i \le l$.
  - $E_3^* = \{(v_1^*, v_2^*, label(\rho_i^2)) \mid \rho_i^2 \in \Omega^2, v_1 \in V$ corresponds to $start(\rho_i^2), v_1 \in V$ corresponds to $end(\rho_i^2)\}$.

    Recall that each $\rho_i^2 \in \Omega^2$ contains edges only in E_4 and each edge in E_4 corresponds to an edge in E of G_M. So each ρ_i^2 corresponds to a path in G_M (also denoted by ρ_i^2).

With the above defined G^*, we remove the isolated vertices from V^*. If there are more than one disconnected component, select a vertex from each disconnected one. Since G_M is strongly connected, we can add a tour to connect all these selected vertices. Note that the problem of finding a tour connecting all disconnected components with minimal-length is NP-hard. Here, we just simply find any tour that can connect these components to reduce the complexity of the proposed algorithm.

Lemma 2 Let $M = (S, s_0, X, Y, \delta, \lambda)$ be a given test specification of a web application. Suppose M is minimal and its graph representation $G_M = (V, E)$ is strongly connected. Let N_M be the network constructed from G_M and f the solution to the maximum-flow and minimum-cost network flow problem on N_M. Let G^* be the digraph defined above. Then G^* is symmetric and strongly connected.

Proof: According to Lemma 1, the feasible maximum flow determined by f is equal to $\sum_{e \in E_1} capacity(e)$ and $\sum_{e \in E_2} capacity(e)$. That is, f determines a set of additional paths such that adding the edges of these additional paths to G_M, the resulting graph G' will be symmetric. For each additional path found by f, when the back button is used, we form a navigation path denoted by a *selfloop path induced by this additional path*, to guarantee the proper execution of the back button. This process will not change the symmetry of each vertex since for every internal vertex v on this selfloop path, we remove an edge entering v and an edge leaving v at the same time. Furthermore, adding a tour to connect vertices selected from each disconnected component will not change the symmetry of each vertex either. Therefore, G^* is symmetric.

From graph theory, we know that if a graph is connected and symmetric, then it is strongly connected. According to our definition, G^* has one component, i.e. it is connected. So G^* is strongly connected since we have proved that G^* is symmetric.

With the solution f to the maximum-flow and minimum-cost network flow problem on N_M, G^* can clearly be constructed in $O(|V| + |E|)$ time.

Since G^* is symmetric and strongly connected, we can find an Euler tour τ' in G^*. Then, we can derive τ from τ' by changing each edge that represents a self-loop path with that path itself. According to Lemma 2 and the way we define τ, it is straightforward that τ contains each transition in M at least once and the use of the *back* button in it is proper.

Theorem 2 *Let $M = (S, s_0, X, Y, \delta, \lambda)$ be a given test specification of a web application. Suppose M is minimal and its graph representation $G_M = (V, E)$ is strongly connected. Let τ be the path obtained by our proposed method.*

A. τ contains each transition in M at least once.

B. The back button is properly used in τ.

According to Theorem 2, the input/output sequence of τ is the desired test sequence.

A complete specification of *the cheap Airfares Search Engine* application in Example 1 consists of 25 states and 118 transitions. With this specification, the test sequence generated by our method is of length 162 while the one generated by T-method is of 236. The saving is 31%.

6. METHOD EVALUATION

We have conducted experiments to evaluate the performance of our proposed method in terms of the savings we gain on the test sequence length. This is in comparison to the *direct application* of T-method on test sequence generation. All experiments are performed on a PC with 2.80 GHz CPU. Both methods are implemented in Java and run under Java Runtime Environment 1.6.0 with 256 Megabytes maximum memory assigned for Java Virtual Machine.

We compare the performance of the two methods according to the following factors of digraphs G_M:

- the number of edges in G_M (denoted by q), also called the *size* of G_M;
- the *diameter* of G_M (denoted by d), i.e., the greatest value in the set $\{distance(v_i, v_j) \mid v_i, v_j \in V\}$, where $distance(v_i, v_j)$ is the length of a shortest path from v_i to v_j.
- the total unsymmetrical degrees of G_M denoted by u. That is,

$u = \sum_{v \in V^-} (\deg_{out}(v) - \deg_{in}(v))$, where V^- is the set of those vertices in V whose out-degrees are greater than their in-degrees.

6.1 SIZE OF G_M

In this subsection, we analyze how the change of the size of G_M affects the lengths of the generated test sequences for both methods. Let the number of vertices in G_M be 100 and let $d = 35$, and $u = 20$. We increase the value of size q. For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 6.1.1.

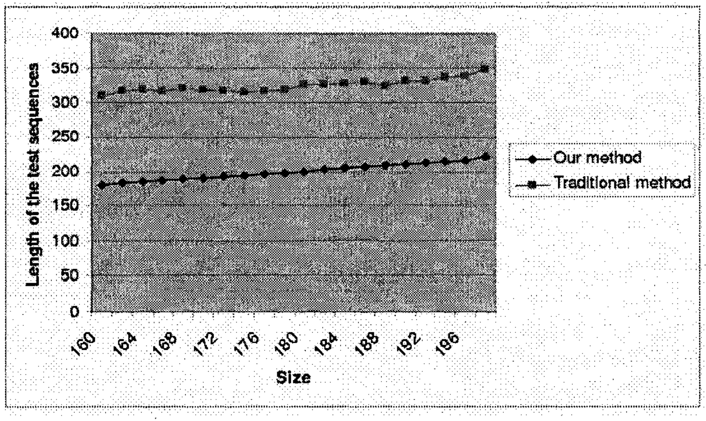


FIGURE 6. 1. 1 AN ILLUSTRATION OF THE CHANGES OF THE LENGTHES OF THE TEST SEQUENCES ACCORDING TO THE INCREASE OF THE SIZE OF G_M

According to this figure, we have the following observations:

O.1.1 Our method considerably reduces the lengths of the test sequences compared to T-method, leading to 36%-42% of savings;

O.1.2 With the increase of the *size* of G_M, the increasing rate of the lengths of the generated test sequences according to our method is similar to that of T-method.

The idea of both of the methods is the same: first, find some additional paths and add their edges to G_M to obtain a symmetric augmentation of G_M; and then find an Euler tour in the resulting symmetric graph. Consequently, the length of the generated test sequence depends on the length of each additional path found in the first step. Since our method can reduce the lengths of the additional paths by making use of the *back* button, it is straightforward that the lengths of the generated test sequences by our method are shorter than those generated by T-method. This conforms to our observation O.1.1.

When the diameter and the total asymmetric degree of G_M are fixed, as T-method and our method work in a similar way, the lengths of the generated test sequences are mainly determined by two factors: i) the number of edges to be traversed; and ii) the number of augmented edges. The latter is in turn determined by the lengths of the additional paths since the total number of additional paths is fixed for all the randomly generated G_M. With the increase of the *size* of G_M, the number of edges increases. As a consequence, the corresponding path of a desired test sequence needs to traverse more edges, yielding longer test sequences in general for both methods. Furthermore, the increasing rates of the lengths of the generated test sequences for both methods are similar (see O.1.2). This is because the increase of the *size* of G_M affects the lengths of the additional paths in a similar way for the case with the use of the *back* button and the case without the use of the *back* button.

6.2 DIAMETER OF G_M

Now we analyze how the increase of the diameter of G_M affects the length of the generated test sequence for both methods. Let the number of vertices in G_M be 50, and let $q = 75$, and . $u = 15$. We increase the value of diameter d. Again, for each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The experimental results are shown in Figure 6.2.1.1.

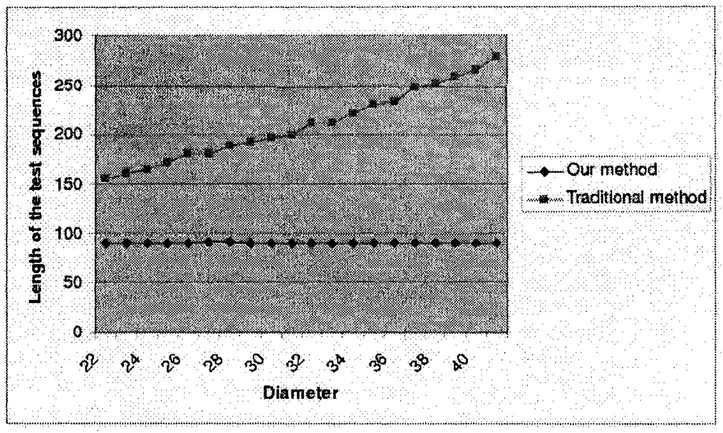


FIGURE 6. 2. 1 AN ILLUSTRATION OF THE CHANGES OF THE LENGTHES OF THE TEST SEQUENCES ACCORDING TO THE INCREASE OF THE DIAMETER OF G_M

According to this figure, we have the following observations:

O.2.1 Our method considerably reduces the lengths of the test sequences compared to T-method, leading to 42%-68% of savings;

O.2.2 With the increase of the diameter of G_M, the increasing rate of the lengths of test sequences generated by T-method is higher than that of ours.

Observation O.2.1 conforms to our similar analysis as for O.1.1.

When the size and the total asymmetric degrees of G_M are fixed, as T-method and our method work in a similar way, the lengths of the generated test sequences are determined mainly by the lengths of the additional paths. By the definition of the diameter ofG_M, the value of the diameter is directly related to the lengths of the shortest paths between vertices when only edges in G_M are considered, and thus it is related to the lengths of the additional paths. Consequently, with the increase of the diameter, the lengths of the additional paths will increase according to T-method where only edges in G_M are considered to generate additional paths. It follows that the lengths of generated test sequences increase significantly. On the other hand, our proposed method makes use of the *back* button when generating the additional paths. When the diameter is increased, the lengths of the additional paths are less likely to change because the *back* action provides us with the opportunity of using shorter additional paths. Thus, the lengths of test sequences generated by our proposed method remain more or less unchanged. This is demonstrated in Figure 6.

6.3 TOTAL UNSYMMETRIC DEGREES OF G_M

Now we analyze how the change of the total asymmetric degrees of G_M affects the length of the generated test sequence for both methods. Let the number of vertices in G_M be 50, and let $q = 75$, and $d = 28$. We increase the value of total asymmetric degrees u. Again, for each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. Our experimental results are reported in Figure 6.3.1.1.

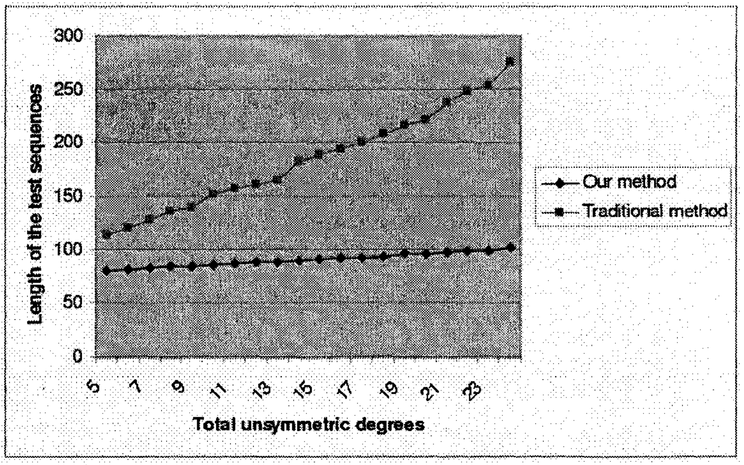


FIGURE 6. 3. 1AN ILLUSTRATION OF THE CHANGES OF THE LENGTHES OF THE TEST SEQUENCES ACCORDING TO THE INCREASE OF THE DEGREES OF G_M

According to this figure, we have the following observations:

O.3.1 Our method considerably reduces the lengths of the test sequences compared to T-method, leading to 29%-63% of savings;

O.3.2 With the increase of the total asymmetric degrees of G_M, the increasing rate of the lengths of the test sequences generated by T-method is higher than that of ours.

Again, O.3.1 conforms to our similar analysis as for O.1.1.

When the size and the diameter of G_M are fixed, as T-method and our method work in a similar way, the lengths of the generated test sequences are determined by the number of additional paths. With the increase of the total asymmetric degrees, the number of the needed additional paths will increase. Therefore, the lengths of the generated test sequences tend to increase with both methods. Again, since our method considers the *back* action, the additional paths are very often much shorter than those in T-method. It follows that when the number of additional paths are increased, the increasing rate of the lengths of the test sequences generated by T-method is higher than that of ours. This is demonstrated in Figure 6.3.1 as observation O.3.2.

7. RELATED WORKS

There are various types of generic testing techniques explored in the past. According to the testing methodology, there are *white-box* testing, *black-box* testing etc. Most of them can be used for testing web applications with proper adaptation. Some general discussions can be found in [18].

For *white-box testing*, a typical approach is to conduct data flow analysis on the source code of the server pages and generate proper test suites satisfying certain test selection criterion. A comprehensive survey on various test selection criteria on data flow testing proposed in the literature can be found in [31], and how to apply data flow analysis to testing web applications is discussed in [17].

Along the *black-box* testing approach, no information about the source code of the server pages is available: all what we can do with the implementation is to execute it. Depending on whether the expected behavior is available or not, different research directions have been taken.

When the expected behavior is unknown, we can first retrieve the design or test cases from the implementation via reverse engineering techniques. In Ricca and Tonella's approach [21, 25] and Benedikt, Freire, and Godefroid's approach [4], auxiliary testing tools called *ReWeb* and *VeriWeb* have been developed respectively to navigate automatically through web applications. ReWeb is applied on the implementation to gather the client pages of the web application under test so that the navigation relation among these client pages is derived and represented by a UML model. Based on this model, another tool called TestWeb can then used to these generated client pages by checking whether their response to different input data are as expected. VeriWeb is designed to explore client pages while performing testing at the same time. The derived test cases are then *reused* for *regression testing.*

When the expected behavior of a web application is available in terms of FSMs, our major task is to generate a test sequence from it. Due to the problem of limited resource to carry out software testing, much of our effort has been put on reducing the length of the generated test sequence. This can be accomplished in two ways: one is to divide-and-conquer, and another is to search for optimal solution for the minimal-length test sequence. Discussion on the former issue can be found in [3], where the authors proposed a hierarchical approach to model and test potentially large web applications. Our approach to checking of the correct navigational behavior of a web application falls into the latter category.

Of course, the specification of the expected behavior of a web application is not restricted to FSMs: it can take any format according to various test purposes. Chang and Hon [5] proposed a mechanism to generate test suit based on *statistical usage model* of the web applications for link validation. An agent-based framework to automatically generated and coordinate test agents proposed by Qi, Kung, and Wong [24] is based on *Belief-Desire-Intention model.* Lee and Offutt [16, 30] used mutation testing techniques to generate test cases from *XML-based documents.*

Most of the test procedure can be carried out either on the server side or on the client side. For the latter, there must be a web browser involved. A web browser introduces additional user interface such as the *back* and *forward* buttons, *the URL address bar.* This interface brings out additional difficulties in formal verification and testing [7] because it complicates the original model of the web applications. At the same time, we have shown in this thesis, that this new feature can also be well used to achieve better solution towards test sequence reduction.

8. CONCLUDING REMARKS

Web technologies have posed new challenges in our continued endeavor for further exploring effective and efficient testing techniques. For web applications, the user's interface comes not only from the one provided by the web applications themselves, but also from that of the web browsers. Making use of this feature, we have proposed a method for test sequence generation specially tailored for testing web applications.

The optimization problem of finding a minimal-length test sequence whose corresponding path traverses each transition in a given FSM M at least once is traditionally reduced to CPP on G_M. In the literature, the time complexity of the most efficient algorithms for CPP is $O(|V|^2|E|^3 \log|V|)$. With our proposed method, this time complexity remains the same, while our experimental result shows significant reduction in the length of the generated test sequences.

Some web browsers, such as Internet Explorer, now provide enhanced history browsing function where any client page whose URL is currently stored in the history stack can be selected to visit. It remains interesting to study on how to use this enhanced function to further reduce the test sequence length.

8.REFERENCE

[1] A. V. Aho, A. Dahbura, D. Lee, and M. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. *IEEE Trans Comm.*, 39(11): 1604-1615, Nov. 1991

[2] R. K. Ahuja, T. L. Magnanti, Netwrok flows, *Prentice Hall*, 1993

[3] A. Andrews, J. Offutt, and R. Alexander. Testing web applications by modeling with FSMs. *Software systems and Modeling*, 4(3): 326-345, 2005

[4] M.Benedikt, J. Freire, and P. Godefroid. VeriWeb: A platform for automating web site testing. In *Proc. Of 11th World Wide Web Conference*, pages 654-668, 2002.

[5] W. K. Chang, S. K. Hon, and C. Chu. A systematic framework for evaluating hyperlink validity in web environments. In *Proc. of 3rd International Conference on Quality Software 2003*, pages 178-185, 2003.

[6] J. Chen and S.Subramaniam. Specification-based testing for GUI-based applications. *Software Quality Journal*, 10(3): 205-224, 2002.

[7] J. Chen and X. Zhao. Formal models for web navigations with session control and browser cache. In *Proc. of International Coference on Formal Engineering Methods. LNCS 3235*, pages 46-60. Springer-Verlag, 2004.

[8] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, SE-4(3): 178-187, May 1987.

[9] J. Edmands and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248-264, 1972.

[10] A. Gibbons, Algorithmic Graph Theory, *Cambridge University Press*, 1985

[11] A. Gill, *Introduction to the theory of Finite-State Machines*, Mc Graw-Hill Book Company,Inc, 1962.

[12] A. V. Goldberg and R. E. Trajan. Finding minmum-cost circulations by canceling negative cycles. *Journal of the ACM*, 36(4):873-886, 1989.

[13] G. Gonenc. A method for the design of fault detection experiments. *IEEE Trans. Computers*, 19(6):551-558, June 1970.

[14] S. Greenberg, A. Cockburn, "Getting Back to Back: Alternate Behaviors for a Web Browser's Back Button", In Proc. of the 5th Annual Human factors and the Web conference , 1999.

[15] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proc. of 5th Ann. Symp. Switching Circuit Theory and Logical Design*, pages 95-110, 1964.

[16] S. C. Lee and J. Offutt. Generating test cases for XML-based web component interactions using mutation analysis. In *Proc. of the 12th IEEE International Synposium on Software Reliability Engineering (ISSRE'01)*, Pages 200-209, 2001

[17] C. Liu, D. Kung, P. Hsia, and C. Hsu. Structural testing of web applications. In *Proc. 11th IEEE International Symposium on Software Reliability*. Pages 84-96, 2000.

[18] G. D. Lucca, A. Fasolino, f. Faralli, and U. de Carlini. Testing web application. In *Proc. of IEEE international Conference On software Maintenance (ICSM' 02)*, pages 310-319, 2002.

[19] A. Memon, M. Pollack, and M. Soffa. Hierarchical GUI test case generation using automated planning. *IEEE Trans. Software Eng.*, 27(2): 144-155, 2001.

[20] R. E. Miller and S. Paul. On the generation of minimal length conformance tests for communications protocols. *IEEE/ACM Transactions on Networking*, 1(1): 116-129,

1993.

[21] S. Nation and M. Tsunoyama. Fault detection for sequential machines by transition tours. In *Proc. of 11th. IEEE Fult Tolerant Computing Symposium*, pages 238-243, 1981.

[22] M. Nottingham, "Caching Tutorial for Web Authors and Webmasters", *http://www.web-caching. com/mnot_tutorial/*, last access: July 2007.

[23] T. Ostrand, A. Anodide, H Foster, and T. Goradia. A visual test development environment for GUI systems. In *Proc. of ACM SIGSOFT Internation Symposium on Softeare Testing and Analysis*. Volume 23, pages 82-92, 1998.

[24] Y. Qi, D. Kung, and W. E. Wong. An agent-based data-flow testing approach for web applications. *Journal of Information and Software Technology*, 48(12):1159-1171, 2006.

[25] F. Ricca and P. Tonella. Testing processes of web applications. *Annals of Software Engineering*, 14:93-114, 2002.

[26] R. Ricca and P.Tonella. Analysis and testing of web applications. In *Proc. of the 23rd IEEE International Conference on Software Engineering (ICSE'01)*, pages 25-34, 2001.

[27] K. K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 4(15):285-297, 1988.

[28] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93-99, 1997.

[29] L. White and H. Almezen. Generating test cases for GUI responsibilities using complete interaction sequences. In *Proc. of 11th International Symposium on Software Reliability Engineering*, pages 110-121, 2000.

[30] W. Xu, J. Offutt, and J. Luo. Testing web services by xml perturbation. In *Proc. of IEEE International Symposium on Software Reliability Engineering*. pages 257-266, 2005.

[31] H. Zhu, P. A. V. Hall, and J.H.R. May. *Software unit test coverage and adequacy.* ACM Computing Surveys, 29:366-427, 1997

[32] R.E. Tarjan. Data Structures and network algorithm. *Philadelphia, PA: Society for Industrial and Applied Mathematics*, 1983.

[33] Harold W. Thimbleby. The directed Chinese Postman Problem. *Journal of Software Practice and Experience.* 33(11) pages: 1081-1096. September 2003.

[34] S. Skiena. The Algorithm Design Manual. *Springer Verlag.* 1998.

9. Vita Auctoris

NAME:	WANG, YAN
PLACE OF BIRTH:	CHENGDU, CHINA
YEAR OF BIRTH:	1980

EDUCATION: UNIVERSITY OF WINDSOR, WINDSOR, ONTARION, CANADA
2005-2007 M.SC. IN COMPUTER SCIENCE
BEIJING INFORMATION TECHNOLGY INSTITUTE, BEIJING, CHINA
1999-2003 B.ENG. IN COMPUTER ENGINEERING

WORKIING EXPERIENCE:
SOFTWARE DEVELOPER, STATE BUREAU OF SURVEYING AND MAPPING, BEIJING, CHINA, 2004~2006