

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

1-1-2006

Efficient test sequence generation for Web applications with frames.

Xiao Wang
University of Windsor

Follow this and additional works at: <https://scholar.uwindsor.ca/etd>

Recommended Citation

Wang, Xiao, "Efficient test sequence generation for Web applications with frames." (2006). *Electronic Theses and Dissertations*. 7145.
<https://scholar.uwindsor.ca/etd/7145>

This online database contains the full-text of PhD dissertations and Masters' theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Efficient test sequence generation for web applications with frames

by

Xiao Wang

A Thesis

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

Windsor, Ontario, Canada

2006

© 2006 Xiao Wang



Library and
Archives Canada

Published Heritage
Branch

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque et
Archives Canada

Direction du
Patrimoine de l'édition

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence
ISBN: 978-0-494-42336-3
Our file Notre référence
ISBN: 978-0-494-42336-3

NOTICE:

The author has granted a non-exclusive license allowing Library and Archives Canada to reproduce, publish, archive, preserve, conserve, communicate to the public by telecommunication or on the Internet, loan, distribute and sell theses worldwide, for commercial or non-commercial purposes, in microform, paper, electronic and/or any other formats.

The author retains copyright ownership and moral rights in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

AVIS:

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, publier, archiver, sauvegarder, conserver, transmettre au public par télécommunication ou par l'Internet, prêter, distribuer et vendre des thèses partout dans le monde, à des fins commerciales ou autres, sur support microforme, papier, électronique et/ou autres formats.

L'auteur conserve la propriété du droit d'auteur et des droits moraux qui protègent cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

■ ■ ■
Canada

ABSTRACT

Finite state machine based conformance testing techniques have been widely used in various hardware and software systems. Such techniques can be applied to web applications to check if the implementation demonstrates the desired navigational behavior. We explain here how to specify the desired navigational behavior of a web application in terms of extended finite state machine, and under which situations can such a specification be directly given as a finite state machine. Our emphasis is on the specification of web frames which allows the web browser to display more than one frame pages of the same application at the same time. Based on this formalization, we present a solution to generate test sequences from a given finite state machine using the well-known transition coverage criterion, taking into account the reduction of testing equivalent transitions, i.e. transitions describing the navigations triggered by the same input event from the same frame page. Empirical study has been conducted to demonstrate the efficiency of the proposed method in terms of the lengths of the generated test sequences compared to the one without considering the reduction on testing equivalent transitions.

Keyword: Hyperlink, Web Navigation, Web Frame, Conformance Testing,

T-method, Finite State Machine, Graph Theory

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 Hu, Dr. Ezeife, and Dr. Lu, for spending their precious time to read this thesis and putting on their comments, suggestions on the thesis work.

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

Finally, I would like to give many thanks to family, my wife Guan Jing, my mother Guangqi Liu, my father Yongju Wang, my younger brother Yan Wang and younger sister Miao Wang for their patience, understanding and endless supporting.

TABLE OF CONTENTS

ABSTRACT.....	iii
ACKNOWLEDGEMENTS.....	iv
TABLE OF CONTENTS.....	v
LIST OF TABLES.....	vii
LIST OF FIGURES.....	viii
I. INTRODUCTION AND PROBLEM DESCRIPTION.....	1
II. BACKGROUND.....	12
2.1 WEB APPLICATIONS.....	12
2.1.1 URL.....	12
2.1.2 SERVER PAGE, STATIC PAGE AND CLIENT PAGE	13
2.1.3 SESSION AND COOKIE.....	14
2.1.4 HYPERLINK AND DYNAMIC HYPERLIN.....	14
2.1.5 WEB FRAME.....	15
2.1.6 THE PROPERTIES OF A CLIENT PAGE.....	16
2.1.7 ARCHITECUTRE OF WEB APPLICATIONS.....	17
2.2 MODELING FORMALISMS	18
2.2.1 FINITE STATE MACHINE (FSM).....	18
2.2.2 EXTENDED FINITE STATE MACHINE (EFSM).....	19
2.3 CONFORMANCE TESTING	21
III. RELATED WORKS	23
IV. MODELING WEB APPLICATIONS WITH EFSM.....	28
4.1 A SPECIAL CASE.....	33

4.2 ADDING FRAMES	36
V. OUR TESTING METHOD	41
VI. IMPLEMENTATION	42
6.1 THE IMPLEMENTATION OF THE CPP ALGORITHM.....	42
6.2 IMPLEMENTATION OF THE ALGORITHM OF RPP	45
6.2.1 HEURISTIC ALGORITHM FOR RPP	46
6.2.2 AHO'S ALGORITHM FOR RPP	49
6. 3 THE IMPLEMENTATION OF A GPP ALGORITHM	51
VII. EXPERIMENTS AND EVALUATION	53
7.1 LENGTH OF TEST SEQUENCE	53
7.1.1 NUMBER OF VERTICES	54
7.1.2 NUMBER OF EDGES	55
7.1.3 NUMBER OF SUBSETS	57
7.2 TIME TO GENERATE TEST SEQUENCE.....	58
VIII. CONCUSION AND FUTURE WORK	59
8.1 CONCLUSION.....	59
8.2 FUTURE WORK.....	59
REFERENCE.....	61
APPENDIX A.....	64
VITA AUCTORIS.....	123

LIST OF TABLES

TABLE4.1.1 THE DETAILED INFORMATION OF T	32
TABLE7.1	55
TABLE7.2	56
TABLE7.3	57

LIST OF FIGURES

FIGURE1.1 SOME WEB PAGES OF THE WEBSITE 1	8
FIGURE1. 2 PART OF THE CORRESPONDING FSM.....	9
FIGURE2.1 SAMPLE CODE FOR WEB FRAME	16
FIGURE2.2 AN EXAMPLE OF A CLIENT PAGE	17
FIGURE2.3 THE STRUCTURE OF WE APPLICATION	18
FIGURE2. 4 A SAMPLE OF FSM.....	19
FIGURE4.1 AN EXAMPLE OF EFSM OF THE WEB APPLICATION	31
FIGURE4.2 AN EXAMPLE OF TIMETABLE SEARCH.....	35
FIGURE4.3 THE EFSM OF THE WEB APPLICATION	35
FIGURE4.4 AN EXAMPLE OF MODELING THE WEBSITE OF COURSE 280.....	39
FIGURE6.1 AN EXAMPLE OF THE CPP	45
FIGURE6.2 A HEURISTIC ALGORITHM TO SOLVE THE RPP.....	49
FIGURE6.3 EXAMPLE OF AHO'S ALGORITHM	51
FIGURE6.4 AN EXAMPLE TO ILLUSTRATE THE GPP ALGORITHM	52

CHAPTER I

INTRODUCTION AND PROBLEM DESCRIPTION

In recent years, information on the Internet has grown extraordinarily fast around the world. Web systems have become the primary device for information sharing and retrieval. Web technologies have been applied to all major areas. The contents of web applications is increasing rapidly, which means there are much more pages and hyperlinks in a web application than before. The quality and reliability of the web applications become more and more important. It is necessary to find a suitable way to check the correctness of web applications.

A web system can be as simple as a web site that consists of a set of static pages like a personal homepage. It can also be a complicated commercial web application like an online-store system that can handle various kinds of transaction requests from different clients (computers) everywhere in the world. In such a web application, web pages must be generated dynamically. The diversity and the intensive use of the web systems are supported by the advance of the emerging technologies, such as cookies/sessions, dynamic web pages, and web frames.

A cookie is a piece of small text that record identification and some related information of a certain client and 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 will be sent along with the request message until its expiration. Based on cookie technique, a web application can overcome the shortcoming of HTTP protocol to recognize a certain client by establishing a session between the web browser and the web server and thus make the web application stateful.

Dynamic web pages are generated dynamically by web servers based on the requested messages. A dynamic web page is the combination of a predefined page template and dynamic contents that are obtained after a web server receives a specific web page request. The server customizes the page content based on the client attributes and thus dynamic web pages facilitate users to interact with web servers. A web page can be generated from two different kinds of template pages: static page and server page, both located in the web server. A server page can produce a dynamic web page while static web page cannot because both its template and its contents are predefined. If a hyperlink could be dynamically displayed on the web pages generated by the same server page, we call such a hyperlink *dynamic hyperlink*. Here we are only interested in the texts and hyperlinks of the client pages. For a client page p generated by a static or server page, we use $O(p)$ to denote that the observable characteristics in p that we are interested in. $O(p)$ is represented as a pair $\langle T, H \rangle$ of a set T of text symbols and a set H of hyperlink symbols in p .

Web frame is the technology that divides a web page into two or more different frames. A web frame is a rectangular area in a static HTML page where navigation can take place independently. Moreover, the different web frames into which a page is decomposed can interact with each other, in the sense that a hyperlink in a page loaded into a frame can force the loading of another page into a different frame. Web frame subdivision can be recursive. Each web frame has a default page. We call a page that contains web frames a *frame page*. A frame page is a list of static or server pages and its characteristic is based on all pages. For example, a client page p of a frame page is a list of pairs $\langle T_i, H_i \rangle$ where $i \in [1, k]$ and k is the number of pages in the frame page.

We can model the specification a web application in terms of Extend Finite State Machine (EFSM) [10]. An EFSM is a basic mechanism in modeling reactive systems. It

is a six-tuple: $EFSM = (S, s_0, I, O, T, V)$, where s_0 is the initial state and S, I, O, V and T are finite set of states, input symbols, output symbols, (state) variables, transitions respectively. Each transition t in the set T is a six-tuple: $t = (s_t, e_t, i_t, o_t, P_t, A_t)$ where s_t, e_t, i_t and o_t are the start (current) state, end (next) state, input symbols, and output symbols, respectively. $P_t(\vec{v})$ is a predicate on the current variable values and $A_t(\vec{v})$ gives an action on variable values. When there is no state variable in the EFSM and no transition of the EFSM includes a predicate or post action, the EFSM is simplified as a Finite State Machine (FSM) [8].

Now, we show how to model web applications in EFSM. We assume that the forward/backward buttons of the browsers are disabled and the user cannot type in a URL in the address bar of the browsers but can reset to homepage from any page using the “home” button of the browsers. When a *web application under test* (WAUT) has web frames, we can use a state, transition, input symbol, output symbol and state variables to represent a list of server or static pages, a page navigation from one page to another page, the user’s action on clicking the hyperlink symbol with some inputs that satisfy certain conditions, a list of pairs $\langle T_i, H_i \rangle$ where $i \in [1, k]$ and k is the number of the pages in the page list of the corresponding state, and variables in a web application respectively. Note that a state may represent either a frame page, where its list of pages contains all pages that are loaded into it, or a static or server page where its list of pages only contains this page itself. If there is no cookie/session in the WAUT, we can reduce the EFSM model to an FSM model.

The EFSM model of WAUT can be used for conformance testing with certain test purposes. Conformance testing is to check if a given implementation that can be viewed as a “black box” conforms to a given specification. In our context, the test purpose is to

check if all navigations conform to the specification of WAUT. Here we only discuss the testing about the web application without cookies/sessions that can be modeled as an FSM. For convenience, we call WAUT as implementation FSM and the FSM derived from the specification of WAUT as specification FSM.

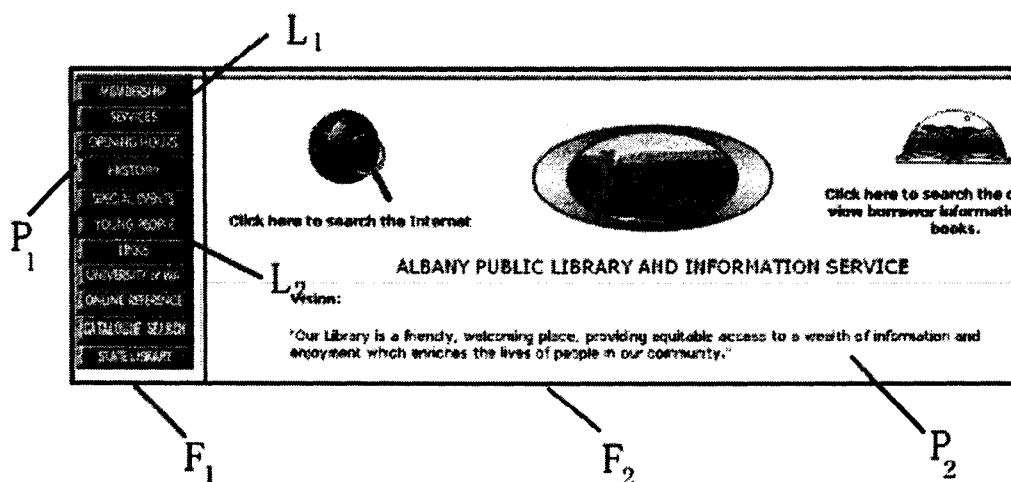
A test sequence is generated from the specification FSM. In general, we apply test sequence that satisfies a certain test criterion to the implementation to check its correctness by comparing the actual output sequence with the expected one. Based on the test purpose, a test criterion can be used to generate a test sequence to check for examples, the correctness of the output, the start or the end states of each transition of the implementation FSM triggered by a specific input. In the following, we adopt the well-known transition coverage criterion, i.e., to test each transition at least once. Very often, we apply results in graph theory to the specification FSM to generate a test sequence because an FSM can be viewed as a directed graph $G = (V, E)$ where V is a finite set of vertices and E is a finite set of edges. Each vertex stands for a state of the FSM and an edge represents a transition. Given a directed graph representing a specification FSM, we can use an algorithm to resolve the Chinese Postman Problem (CPP) [7] to generate a test sequence. The CPP is to find a least-cost walk to travel all edges in a directed graph.

Although a test sequence generated by the CPP satisfying the transition coverage criterion can accomplish our test purpose, such a test sequence could cause *redundant-testing phenomenon* when WAUT has web frame pages. A static or server page may be contained in two or more web frame pages and represented by different states. Thus, navigation from this page may be represented by two or more different transitions in the corresponding FSM. For such transitions representing a same navigation, we assume that the correct implementation of one of them implies that of all of others. However, the test

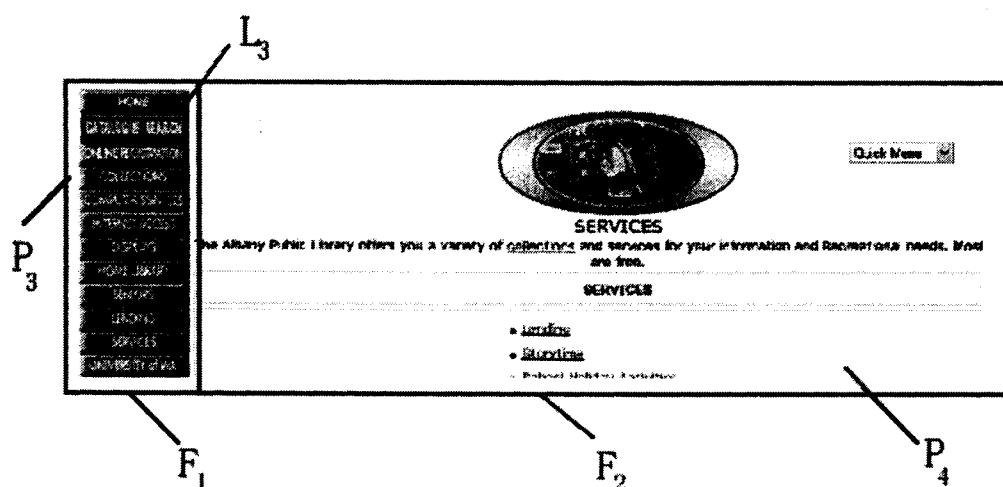
sequence based on the CPP requires that we check all of such transitions and therefore introduces redundancy into the constructed test sequence.

Here we use the website of Albany public library as an example to show redundancy-testing phenomenon. This website is used to provide some on-line library services. In the website, each frame page contains two frames: the right frame and the left one. Figure 1.1 shows some frame pages of this web site and Figure 1.2 gives an FSM for it. Figure 1.1(a) is the home page of the website and we call it Frame page 1. The left frame of Frame page 1 shows a static page P_1 that provides some hyperlinks of main functions. Its right frame shows a static page P_2 which gives a simple introduction about the website. L_1 and L_2 in Frame page 1 are the hyperlinks to the frame pages for basic services and young people respectively. Figure 1.1(b) is Frame page 2 for basic services. The left frame of Frame page 2 shows a static page P_3 that provides some hyperlinks to frame pages for related functions where L_3 in particular is the hyperlink to load static page P_7 for catalogue searching in the right frame. The right frame of Frame page 2 shows a static page P_4 which gives an introduction together with some hyperlinks about basic services. Figure 1.1(c) is Frame page 3 that also has two frames. The right one displays the static page P_5 that shows some hyperlinks to frame pages for related functions and L_4 is the hyperlink to load the web page P_7 for catalogue searching in the right. The left frame of Frame page 3 shows a static page P_6 that gives some interesting contents and hyperlinks for young people. After clicking the hyperlink L_3 , Frame page 4 is shown in Figure 1.1(d). The left frame of Frame page 4 shows static page P_3 and the right frame of Frame page 4 displays static page P_7 which provides some different ways to make catalogue searching by author's name and the name of the article, etc. Frame page 5 is generated as shown in Figure 1.1(e) after clicking the hyperlink of L_4 . The right frame shows the static page P_5 and the left frame displays the static page P_7 . The hyperlink L_5 in P_7 of

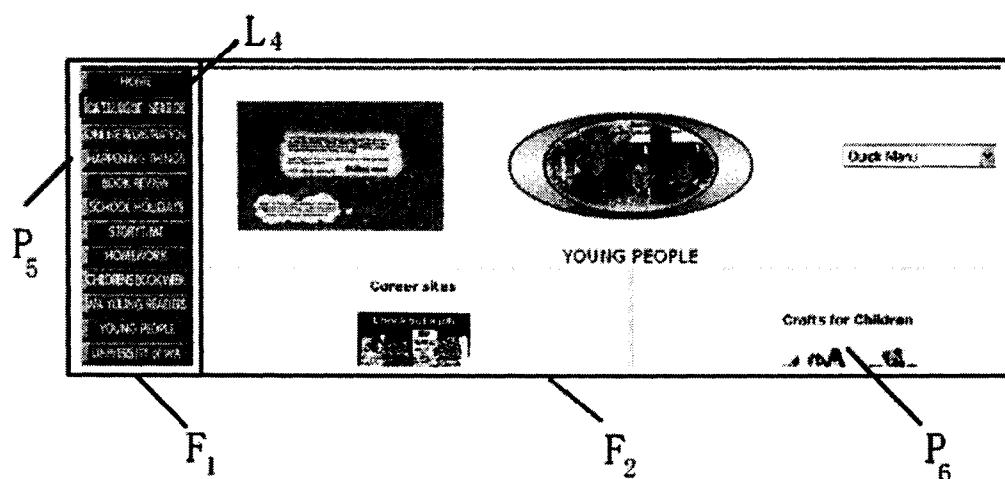
Frame page 4 and 5 is to load a server page in the right frame to provide searching function by typing key words and turns Frame page 4, 5 into Frame page 6 and 7 respectively.



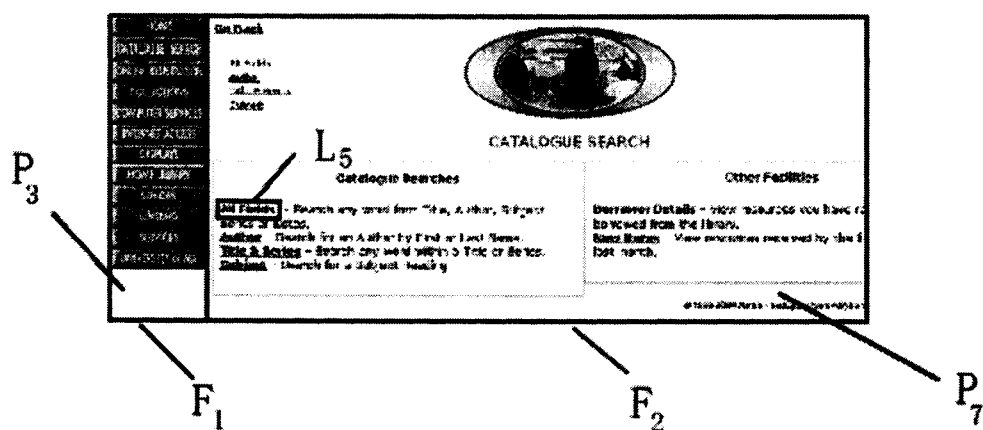
(a)



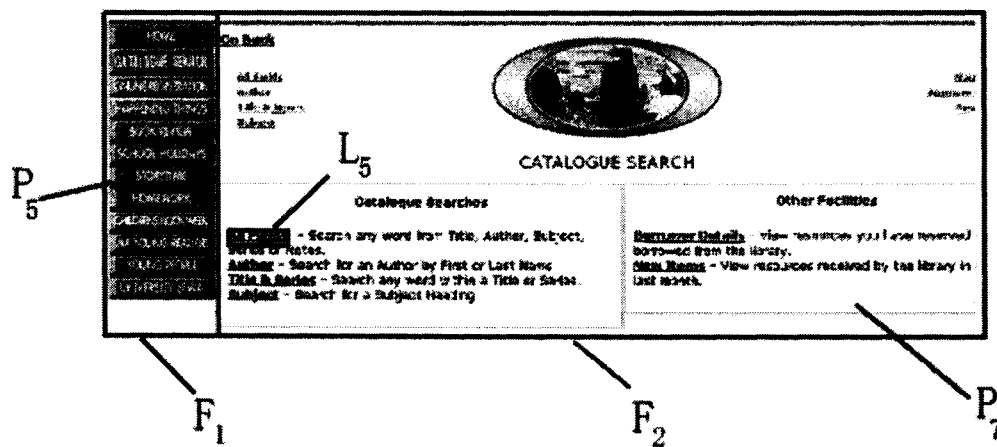
(b)



(c)



(d)



(e)

Figure 1.1 some web pages of the website 1

Frame page 1, 2, 3, 4, 5, 6 and 7 are represented by states $s_0, s_1, s_2, s_3, s_4, s_5$, and s_6 respectively. The transitions t_1, t_2, t_3, t_4, t_5 , and t_6 represent six different navigations in the web site triggered by clicking on hyperlinks L_1, L_2, L_3, L_4, L_5 and L_5 in states $s_0, s_0, s_1, s_2, s_3, s_4$ respectively.

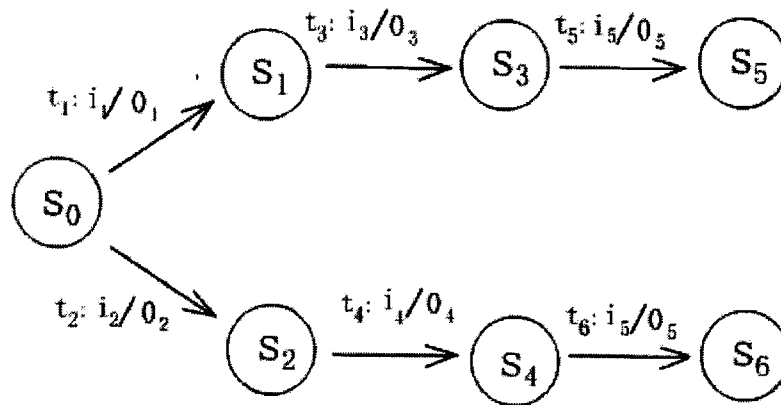


Figure1.2 part of the corresponding FSM

With the corresponding FSM, if we use the algorithm of CPP to generate a test sequence to test each transition at least once, transitions t_5 and t_6 should both be tested. But actually it is sufficient to test only one of them because they represent the navigations caused by the same hyperlink in the same client page.

The above example is illustrative. There may exist many server pages or static pages in many different frame pages and a test sequence based on the algorithm of CPP may cause serious redundancy in testing.

Note that the redundant-testing phenomenon is caused by web frames. From the viewpoint of specification-based testing, we need to obtain a “good” model that can represent the characteristic of web frames effectively. Second, we need a “good” testing criterion to generate a testing sequence that can avoid the redundant-testing phenomenon.

There are two main steps in our work. The first step is to model a web application in terms of FSM; The second step is to generate a test sequence with reduced length based on the FSM model.

In the first step, under certain assumptions, we give general discussions on the EFSM formalization of generic web applications with most of the characteristics such as sever or static pages, hyperlinks, cookies/sessions, client pages etc. Then we focus on modeling web applications with web frames that has no cookie/session and dynamic hyperlinks in terms of FSM.

In the second step, we show how to search for optimal solution to generate an efficient test sequence from a given FSM, avoiding the redundant-testing. We show how to use GPP [4] solution to reach this. The GPP is to find a closed walk in a directed graph G with minimum weight which traverses at least one edge of each subset of the edges from a given set of subsets of edges. We put all transitions in the specification FSM that represent a same navigation in the web application into a same subset of transitions and then make use of the algorithm of GPP to generate an efficient test sequence that can avoid the redundant-testing phenomenon.

To evaluate the proposed method, we compare our testing method with the one based on the algorithm of CPP. To carry out this comparison, a java program has been developed to provide experimental result.

The rest of this thesis is organized as follows: Section 2 gives a brief introduction to web applications including web servers, dynamic hyperlinks. Section 3 introduces some related works on modeling web applications and testing methods. Section 4 presents our EFSM and FSM models for web applications. Section 5 presents our test method. Section 6 introduces main algorithms for CPP, RPP and GPP. Section 7 analyzes and evaluates the result of experiments based on two test methods. Section 8 gives the conclusion and future work.

CHAPTER II

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 EFSM and conformance testing.

2.1 WEB APPLICATIONS

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. They may come from the input of the client and/or 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 server receives a request message that contains a URL and cookie information(if provided) from a browser, which can be triggered by clicking on a hyperlink or typing a URL in the address bar of browser, it sends back a 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 are 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 be uniquely identified in the web application. In practice, a web page is identified by its URL including web server address, and 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. There are many cases that 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 tracking the user's history.

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

2.1.4 HYPERLINK AND DYNAMIC HYPERLINK

All pages of a web application are interconnected by hyperlinks. These hyperlinks form the navigation of 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 the file name. Here we give each hyperlink in a certain static or server page a unique linkID which is used to identify the link within the same page.

According to the different information of cookies/sessions, database or request parameters, a hyperlink could be dynamically shown on a client page which is generated from the same server page. Such a hyperlink is called a *dynamic 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 login.

2.1.5 WEB FRAME

A web frame (simply called frame below) is a rectangular area in a static HTML page where navigation can take place independently. Moreover, the different frames a page is decomposed into can interact with each other, since a link in a page loaded into a frame can force the loading of another page into a different frame. Frame subdivision can be recursive. Each frame has a default page within the page that contains this frame initially loaded by the user.

```
<title>homepage</title>
<frameset cols=50%, 50%>
<frame src = "A.html" NAME="Hi">
<frame src = "B.html NAME="HELLO">
</frameset>
```

(a)

```
<title>A</title>
<a href="C.html" target="HELLO">Link1</a>
```

(b)

Figure2.1 Sample code for web frame

The two pieces of code above show an example of a web frame. The code in (a) illustrates that a page entitled “homepage” has two frames. One is “Hi”; the other is “HELLO”. A frameset is defined in a page with the frame tag “frameset” and “/frameset”. When the homepage is initially loaded by the browser, two pages in A.html and B.html are individually loaded into frames Hi and HELLO, respectively. The code in (b) illustrates that when A.html is loaded into the frame of the homepage, the user’s click on the hyperlink named Link1 can force the frame HELLO to load C.html. This activity will not change the URL of the page.

2.1.6 THE PROPERTIES OF A CLIENT PAGE

On abstract level, a client page can be viewed as a list of pairs $\langle T_i, H_i \rangle$ and $i \in [1, k]$ of a set T_i of text symbols and a set H_i of hyperlink symbols where k is the number of the server or static pages that generate the client page. If the client page is generated by a single server or a static page, $k = 1$; otherwise, the client page is generated by a frame page and $k > 1$. A text symbol represents the information of some text. A hyperlink symbol represents the existence of a hyperlink: it means the user’s clicking on the hyperlink symbol on a client page can trigger a request message to the server.

Note that although the formats of a hyperlink symbol and 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. It may contain however cookie information on this page.

Figure 2.2 shows an example of a client page that is generated by a single server page. It is the homepage of the student web mail system of University of Windsor. In this client page, the black lines point out all the text and hyperlink symbols: the pair $\langle T, H \rangle$. The hyperlink symbol "Sign in" cannot contain any input of the clients because the input of current page has not yet been given.

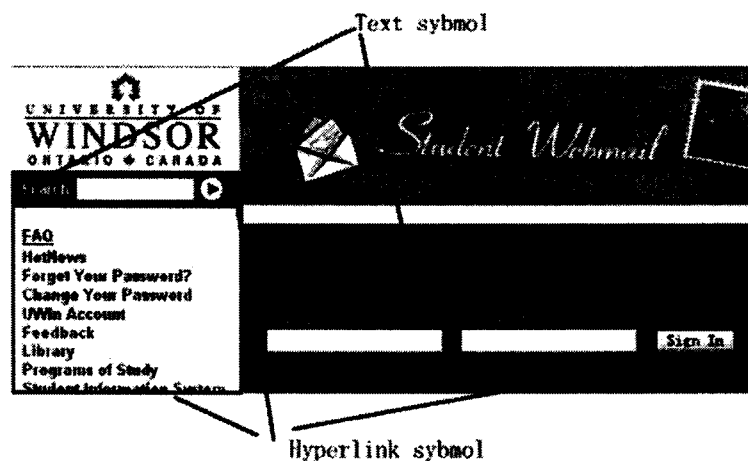


Figure2.2 an example of a client page

2.1.7 ARCHITECTURE OF WEB APPLICATIONS

Figure 2.3 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 one.

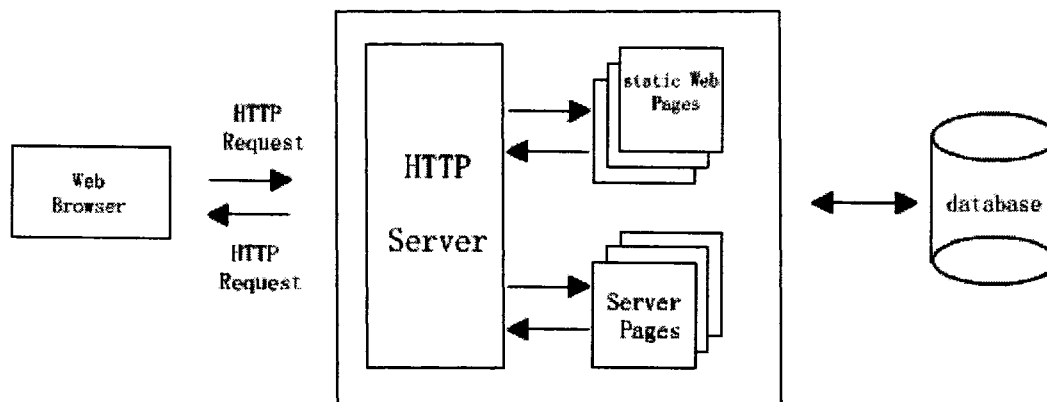


Figure 2.3 the structure of we application

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

2.2 MODELING FORMALISMS

2.2.1 FINITE STATE MACHINE (FSM)

A finite state machine [8] can be represented as 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.

$\delta: S \times I \rightarrow S$ is the state transition function and

$\lambda: S \times I \rightarrow O$ is the output function.

When the machine is in current state s and receives an input a from I , it moves into the next state specified by $\delta(s, a)$ and produces an output given by $\lambda(s, a)$.

Figure 2.4 shows an example of an FSM where $I = \{a, b\}$, $O = \{0, 1\}$, $S = \{s_0, s_1, s_2\}$, $\delta(s_0, a) = s_1$, $\delta(s_1, b) = s_2$, $\lambda(s_0, a) = 1$ and $\lambda(s_1, b) = 0$. The initial state of the FSM is s_0 .

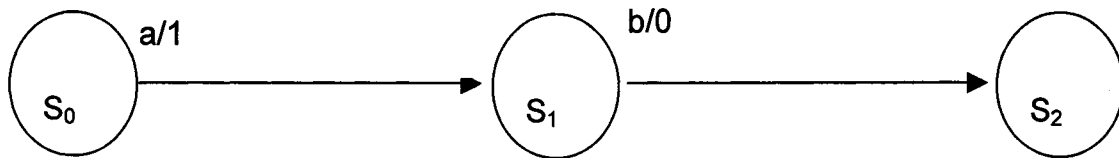


Figure 2.4 a sample of FSM

2.2.2 EXTENDED FINITE STATE MACHINE (EFSM)

Finite state machines are widely used as modeling formalism for many applications such as sequential circuits and control portions of communication protocols. However, in practice the systems which we are interested in usually include variables and operations based on variable values; ordinary finite state machines are not powerful enough to model them in a succinct way. Extended finite state machines, which are finite state machines extended with variables, have emerged from the design and analysis of this kind of systems.

In the following we denote a finite set of variables by a vector: $\vec{v} = (v_1, \dots, v_k)$. A predicate P on variable values $P(\vec{v})$ returns True or False; a set of variable values \vec{v} is

valid if $P(\vec{v}) = \text{True}$. An action is an assignment: $\vec{v} := A(\vec{v})$ where A is a function of \vec{v} .

An extended finite state machine (EFSM) is a six-tuple:

$$EFSM = (S, s_0, I, O, T, V)$$

where s_0 is the initial state and S, I, O, V and T are finite sets of states, input symbols, output symbols, variables, and transitions respectively. Each transition t in the set T is a six-tuple

$$t = (s_i, e_i, i_i, o_i, P_i, A_i)$$

where s_i, e_i, i_i and o_i are the start (current) state, end (next) state, input, and output, respectively. $P_i(\vec{v})$ is a predicate on the current variable value and $A_i(\vec{v})$ gives an action on variable values.

Initially, the machine is in its initial state $s_0 \in S$ with initial variable values: \vec{v}_{init} .

Suppose that the current state is s and the current variable values are \vec{v} . Upon input i , the machine follows a transition $t = (s, e, i, o, P, A)$ if \vec{v} is valid for $P: P(\vec{v}) = \text{True}$. In this case, the machine outputs o , changes the current variable values by action $\vec{v} := A(\vec{v})$, and moves to state e .

For each state $s \in S$, let all the transitions with start state s and input $j \in I$ be: $t_j = (s, e_j, j, o_j, P_j, A_j)$, $1 \leq j \leq r$ and $r \in \mathbb{N}$. In a deterministic extended finite state

machine (EFSM), the sets of valid variable values of these r predicates are mutually disjoint, i.e., $V_{P_k} \cap V_{P_j} = \emptyset$, $1 \leq k, j \leq r, k \neq j$. Otherwise, the machine is nondeterministic. In a deterministic EFSM there is at most one possible transition to follow from a given state with given input. In the present work, we consider only deterministic EFSM.

2.3 CONFORMANCE TESTING

Given a specification and an implementation which can be regarded as a “black box”, we need to check if the implementation conforms to the specification. This is called the conformance testing. The specification of a web application can be modeled as an FSM. To conduct a conformance testing, a test sequence can be generated based on the FSM. The test sequence can be applied on the web application to check whether the output is consistent with the specification of the web application.

In an FSM, two states s_i and s_j are equivalent if and only if for any input sequence the finite state machine will produce the same output sequence. For two states in different machines with the same input and output sets, equivalence is defined similarly. Two machines M and M' are equivalent if and only if for every state in M there is a corresponding equivalent state in M' and vice versa. Given an FSM, there may exist two or more states equivalent. A machine is minimal if and only if no two states are equivalent. All FSMs we refer to here are minimal. During the procedure of test, the WAUT itself can be considered as an FSM. Our conformance testing is to check whether the two FSMs (the specification and implementation FSMs) are equivalent.

Generally speaking, each FSM can be considered as a directed graph. A state in FSM can be represented as a vertex and a transition can be represented as an edge. If a directed

graph has a path from each node to every other node, it is strongly connected. We say an FSM is strongly connected if its corresponding directed graph is strongly connected. For conformance testing, we assume that a) the specification FSM A is strongly connected and minimal; b) the implementation FSM B has the same input and output alphabets as A . If machine A is not strongly connected, there may exist some states that cannot be reached during the test, thus we will not be able to tell with certainty if machine B is correct.

Given a specification FSM, an essential task to conduct conformance testing is to generate an effective test from this FSM. The effectiveness of a test sequence is judged by the criterion used to generate the test sequence. Here we adopt the well-used transition coverage criterion which requires that the test sequence of the FSM covers each transition at least once. According to the literature [1], the problem of constructing a minimal-length test sequence satisfying this criterion is resolved by solving CPP in the underlying digraph of the given FSM when it is strongly connected.

CHAPTER III

RELATED WORKS

Our work is divided into three steps. The first step is to model the specification of a WAUT in terms of EFSM; The second step is to generate a test sequence with reduced length based on the given model; The third step is to provide empirical study to evaluate the performance of our testing method. There are several approaches to modeling the specifications of web applications for the purpose of testing providing the corresponding testing methods. We show the related modeling work and their corresponding testing methods and present their key ideas here.

In [17], it has been presented a testing method and testing tools based on the analysis of the structure of a web application. In the first part, the authors consider that the central entity in a web application is the web page. Based on this concept, they defined most of the related properties and associations of the entities by UML, and an UML-based model is built which can be exploited for analysis and testing. Two tools, ReWeb and Retest, have been developed to support the analysis and testing of the web applications. ReWeb downloads and analyzes the pages of a web application with the purpose of building a UML model of it. TestWeb generates and executes a set of test cases for a web application whose model was computed by ReWeb. A test case generation engine inside TestWeb is used to generate test cases, and the generation is based on a reduced graph by removing static web pages without forms in a navigation paths.

In [11], the authors presented a methodology to support web application testing based on the Web Test Model, called WTM, which can capture both structural and behavioral test objects of the web applications created by forward and reverse engineering tools. The testing work is divided into three parts according to: object perspective, behavior perspective and structure perspective. Each part is tested separately. From the object

perspective, entities of a web application are described using ORD (Object Relation Diagram) in terms of objects and dependent relationships. From the behavior perspective, the authors used PND (page navigation diagram) to depict the navigation behavior of a web application and PND can be derived from ORD. At the same time, a set of OSDs (Object State Diagram) is used to represent the state-dependent behavior of interacting objects. From the structure perspective, the authors use the BBD (Block Branch Diagram) and FCD (Function Cluster Diagram) to capture the control and data flow information of the scripts and functions in a web application. In the behavior testing, a navigation tree from the home page is constructed to generate test cases.

In [15], the authors proposed an approach to integrating existing testing techniques with a state-based testing which is to discover inconsistencies introduced by interactions with web browser buttons. The authors call the approach as base-line testing strategy. The authors considered the influence of the web browser on the navigations among web pages. A user can force the browser to display and reload the previously visited page or refresh the current web page by clicking on the related buttons, such as back, forward and fresh buttons. In order to test a web application with the browsers, a state chart of back and forward and refresh buttons is constructed with four states, BDFD(Back Disable, Forward Disable), BEFD(Back Enable, Forward Disable), BEFE(Back Enable, Forward Enable), BDFE(Back Disable, Forward Enable). After flattening each baseline test case with the state chart, the testing model is generated as a transition tree. The root of the tree is the home page of the web application, and each path of the tree is tested separately.

In [14], the authors exploited an object-oriented test model of a web application in terms of UML class diagrams. The test model includes each component of the web application relevant for testing purposes together with the relationship among them. The paper proposed a definition of the unit level for testing web applications. Based the model, the authors separately discussed both unit level testing and integration testing. For unit testing, the authors used decision table to generate test cases for client and server page.

Test case is selected in order to assure that the decision table variants are assumed both true or false. For integration testing, test case can be derived by use case diagram. For each use case of a web application, web pages collaborating to its implementation are taken into account and functional test cases are generated and executed for each of them.

In [13], the authors extended the data flow testing techniques to web applications and used web application test model (WATM) to capture the test object of web applications. The key of this paper is to test the dataflow of HTML and XML document in web applications so that the elements of an HTML or XML document are considered as structured variables within an object of WATM. In the WATM, test cases of the data flow are derived from three different perspectives: intra-object, inter-object and inter-client. From the intra-object perspective, test paths are selected for the variables that have def-use chains within an object. A test case can be obtained from the dataflow graph of related functions in an object. From the inter-object perspective, test path should traverse across objects. It means that a test case can be derived from the data interactions across objects in WATM. From the inter-client perspective, an object often contains variables that are shared by multiple clients so that test cases should be derived from the data graph which depicts the data interactions among the clients.

In [1], the authors describe a technique for generating an optimal (minimal in time) test sequence for an implementation of a protocol. A protocol can be specified as a deterministic finite state machine, where the state of the protocol is defined as a stable condition in which the protocol rests until an input is applied and to undergo a transition from the current state to a new state, where it stays until the next input. The purpose of this paper is to present a method to test whether there is a discrepancy between the specification and implementation of an FSM. In this paper, the technique is based on an optimization technique that has been developed to solve a class of problem, such as the CPP (Chinese Postman Problem) on directed graphs. The algorithm described in this paper finds a minimum-cost input sequence for exercising a given set of transitions of an

FSM. Furthermore, a concept called a Unique Input/Output Sequence is used in place of an more cumbersome distinguishing sequence, so that both the controllability and observability problems of the protocol testing problem are resolved simultaneously.

In [3] and [19], although the authors don't mention modeling work for web applications, they offer algorithms that are accessible to implement and produce reduced testing sequences based on most of models. Especially in [3], although many pseudo-code descriptions of the CPP exist, no executable algorithm for it is available. The aim of this paper is to motivate and exhibit a clear working algorithm rather than efficient algorithm and the author make use of negative canceling algorithm replacing matching algorithm to improve the feasibility of the CPP algorithm significantly. The RPP (Rural Postman Problem) is NP-complete problem and the author [3] presents a branch and bound algorithm (an heuristic algorithm) for the exact solution of the RPP based on bounds computed from Lagrangean Relaxation and on the fathoming of some of the nodes by the solution of minimal cost problem.

We do our work following previous pattern: modeling web applications first and then producing test sequences based on test methods. In most of previous papers [17][11][15][14], the authors consider web pages and components as central entities and then define the relationship between them in models, here we follow this idea, but the authors don't use an formal method to model web applications, on the contrary, we use FSM as [1], even EFSM, to model web applications in our work. Most of testing methods in previous work are simple to exhaust all possible paths in models but don't mention that web frame can cause redundant-testing phenomenon. For example, in [11], the authors presented a method to transform the behavior model from PND to FSM with a corresponding testing method that generates test case based on the derived navigation test tree, but they did not discuss that web frames may introduce the redundant-testing phenomenon. In [17], web frame is discussed but such discussion only aims at analyzing the structure of web applications instead of improving the efficiency of testing with the

property in web frames. In our work, we put an emphasis on solving this problem by generating a reduced test sequences based on a heuristic algorithm.

CHAPTER VI

MODELING WEB APPLICATIONS WITH EFSM

We assume that the forward/backward buttons of the browsers are disabled and the user cannot type in a URL in the address bar of the browsers but can reset to homepage from any page using the “home” button of the browsers.

For a web application without web frame, a state represents a server page or a static page. In practice, the homepage of the web application is represented as an initial state. An input symbol represents the user’s action on clicking the hyperlink symbol with some inputs that satisfy certain condition. When there is no input, we simply use the hyperlink symbol as input. An output symbol is a pair $\langle T, H \rangle$ of a set T of text symbols and a set H of hyperlink symbols. Let O be a mapping from the set of client pages to the set of $\langle T, H \rangle$ pairs.

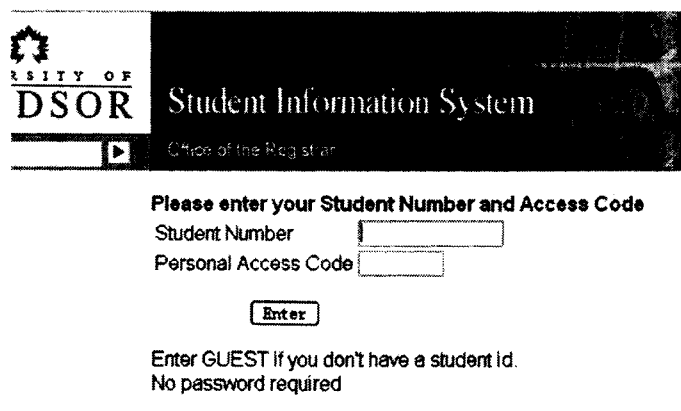
The variables in an EFSM of a web application are put into three different categories: cookie variable, database variable and hyperlink variable. A variable used to record login status of a user in a certain session is a cookie variable.

A transition $\langle s_1, s_2, h, \langle T, H \rangle, P(\vec{v}), A \rangle$ represents a page navigation from the page represented by s_1 to the page represented by s_2 , triggered by the user’s action on hyperlink h . The observation of this navigation is the texts in T and hyperlinks in H appeared in the page represented by s_2 . This navigation is enabled when $P(\vec{v})$ returns true with the values of \vec{v} in s_1 . A is the post action on the variables \vec{v} . For example, we use a Boolean cookie variable x to represent the login state of a user in a certain session. If x is true, it means that the user has logged in. Otherwise, the user has not. When a user clicks the hyperlink symbol “logout” on a client page, a transition can be triggered. The input of this transition is the hyperlink symbol “logout”. The predicate of the transition is $x == true$. The post action of the transition is an assignment: $x = false$.

A dynamic hyperlink is a hyperlink whose presence in a page depends on the history of reaching the page. A transition for clicking on a dynamic hyperlink symbol is not always enabled. Here we use a Boolean hyperlink variable v to record such status of a transition and we add $v = true$ to the predicate of the transition. Thus, if a hyperlink variable is true, the corresponding transition can be triggered.

Recall that we have both hyperlink and hyperlink symbols in a web application, and that a hyperlink symbol may contain cookie information. In our setting, i) a hyperlink symbol is only used for checking the output of a transition, i.e., the existence of a hypertext in the pair of the properties of the target page of the transition. ii) Cookies are represented separately as state variables and thus to check the correctness of a cookie in the target page of a navigation, we need to check the value of the corresponding cookie variable.

We use an example of a part of SIS to explain how to model a web application with EFSM. Figure 4.1(a) is a client page generated by the login page (static page). Each user uses this client page to input her/his ID and password to login the system. After login, the web application records some relative identity and navigation information and goes to a client page generated by the function panel page (a server page). Figure 4.1(b) illustrates the client page of the function panel page. Figure 4.1(c) illustrates the function of viewing transcript where various students can view the transcript information.



UNIVERSITY OF
DSOR

Student Information System
Office of the Registrar

Please enter your Student Number and Access Code

Student Number

Personal Access Code

Enter GUEST if you don't have a student id.
No password required

(a) Login page of SIS

What's New

- [View Student Evaluation of Teaching \(SET\) Results](#)

~~Next~~

Personal Information

- [Change your Access Code](#)
- [Update your address](#)

Registration Information

- [Check your registration time](#)
- [Register / Add or Drop Courses](#)
- [Search for a course to take](#)
- [Show details about a course](#)
- [View your transcript](#)
- [Application to Graduate](#)
- [Course/Exam Timetables](#)
- [View your Schedule & Grades](#)
- [View your exam details](#)
- [Check progress towards your degree](#)
- [Exam Numbers \(Law only\)](#)
- [Request printed transcript](#)
- [Other Post Secondary School Transcripts](#)
- [Request an Enrollment Certificate for this term](#)

(b) Function panel page of SIS

Web Services

[Register](#)

[Course Details](#)

[Your schedule
and Grades](#)

[Your transcript](#)

[Degree Audit](#)

[Your Account](#)

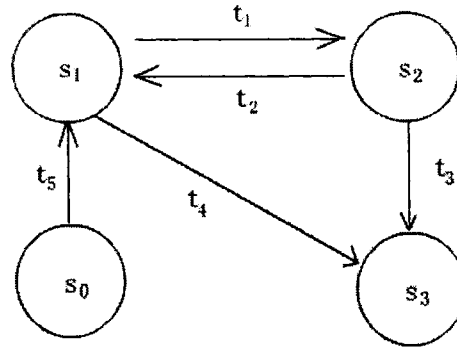
[Main SISweb menu](#)

[End Session](#)

View Transcript

2005 Winter-----			
BSC- Computer Science			
COMPSCI 510 Literature Review and Survey	IP	(3.00)	
COMPSCI 520 Presentations and Tools	IP	(3.00)	
COMPSCI 539 Emerging Non-traditional Database System	B	3.00	
COMPSCI 569 Semantic Web	A-	3.00	

(c) Viewing transcript page of SIS



s_0 : login page (static page)
 s_1 : function panel page(server page)
 s_2 : transcript page (server page)
 s_3 : logoff page(server page)

(d) The corresponding EFSM of SIS

Figure4 1 an example of EFSM of the web application

First of all, note that the hyperlink to the transcript page is a dynamic hyperlink. To model this part of the SIS, we need three Boolean variables. The first one is a cookie variable *Session* to record user's login status; the second one is a hyperlink variable *T_transcript* to represent the availability of the transition corresponding to the hyperlink to the transcript page; the third one is a cookie variable *status* to record the user's status: current or prospective student. If the variable *status* and *T_transcript* are true, it means the user's login is successful and the transition to the transcript page can be enabled. If variable *status* is false, it means that the use of the transition to the transcript page is not available.

The corresponding EFSM of the SIS is machine M (Figure4.1). $S_M = \{s_0, s_1, s_2, s_3\}$, s_0 is initial state, $I = \{a, b, c, d\}$, $O = \{o_1, o_2, o_3\}$, $T = \{t_1, t_2, t_3, t_4, t_5\}$, $V = \{Session, T_transcript, status\}$ and the initial value of *Session*, *T_transcript* and *status* are false, false and null respectively. s_0 is the login page (a static page), s_1 is the

function panel page (a server page), s_2 is the transcript page (a server page) and s_3 is the logoff page (server page). a is clicking on the hyperlink of 'view transcript', b is clicking on the hyperlink of 'Main SISweb menu', c is clicking on the hyperlink of 'logoff', d is clicking on the hyperlink of 'Enter' with the input of user's ID, password satisfying the condition such that the user is a current student. o_1 is the pair of a set of the hyperlink symbols in the client transcript page and the related transcript content (text symbol). o_2 is the pair of a set of the hyperlink symbols in the client function page and some text symbols. o_3 is the pair of a set of the hyperlink symbols in the client logoff page (here the set is empty) and the related text content symbols of logoff. Table4.1.1 shows the detailed information of T .

Name	Sstate	Estate	Input	Output	Predicate	Action
t_1	s_1	s_2	a	o_1	Session= true Status = current T_transcript=true	
t_2	s_2	s_1	b	o_2	Session = true	
t_3	s_2	s_3	c	o_3	Session = true	Session = false
t_4	s_1	s_3	c	o_3	Session = true	Session = false
t_5	s_0	s_1	d	o_2		Session = true; Status = current; T_transcript = true.

Table4.1.1 the detailed information of T

4.1 A SPECIAL CASE

When a web application server does not have sessions/cookies, web frames or databases that are used to record the history of the path, nor dynamic hyperlinks, we can test each server page or static page separately by providing a set of URLs with parameters as input. Instead of writing a specific test driver, we can browse all server pages to find all those containing a hyperlink to this *server page under test*, type in those hyperlinks and get to the *server page under test* by providing all typical request parameters. As we do not assume the availability of the navigation diagram, we need to analyze the typical request parameters in order to gain confidence in all possible behavior of this *server page under test*.

Alternatively, we can assume that the abstract behavior of the web application is described as an EFSM. We also assume that hyperlinks are predefined and these hyperlinks cannot be generated dynamically. According to this assumption, all hyperlinks contained in static or server pages are defined at the design stage of a web application.

We use a simple timetable search example to illustrate a web application that meets the above conditions. Figure 4.2(a) is a homepage of an airline company. In this homepage, a user can fill in the form to look up the related timetable. The information of departure city, arrival city and departure date should be given by the user in the form. After clicking 'submit' hyperlink, a result is shown in another client page of the *result page*. There are two kinds of results, one contains the related timetable and the back hyperlink as shown in the client page (Figure 4.2(b)); the other contains sorry information and the back hyperlink in the client page because no timetable satisfies the user's requirement (Figure 4.2(c)). Figure 4.3 shows the corresponding EFSM M of the web application. In the EFSM, $S_M = \{s_0, s_1\}$, s_0 is the initial state, $I_M = \{a, b, c\}$, $O_M = \{o_1, o_2, o_3\}$ and $T = \{t_1, t_2, t_3\}$. Input a is clicking on the hyperlink of 'submit' with the input variables of departure city, arrived city and departure date that can make the condition on the

existence of the timetable true. b is clicking on the hyperlink of 'submit' such that that input request results in no information found, c is clicking on the hyperlink of 'back'. o_1 is the pair of the form of the related timetable (text symbols) and the hyperlink symbol of 'submit', o_2 is the pair of the sorry information and the hyperlink symbol of 'back', o_3 is the pair of the user form and the hyperlink symbol of 'submit'. $V = \emptyset$ and no predicate and post action exists in any transition of the EFSM. If no predication, post action or state variable exists in an EFSM, such EFSM is in fact an FSM. Figure 4.3 shows the corresponding FSM of the web application.

Online timetable

Departure city: Arrival city:

Departure date: Mar 26

departure date departure city Arrival city

Printable timetable submit

(a)

Timetables

Flight: Albany County (NY) - ALB to Beijing (CN) - PEK
Date: Monday March 26, 2007

PRINT THIS PAGE

Flight	Aircraft	Departure	Arrival	Travel Time	Stops	Via
AC 7385	737	18:00	17:30	35 hr 05 min	0	Toronto Pearson (ON)
AC 101	320	22:45	01:00 +1		0	Vancouver (BC)
AC 629	737	12:40 +1	18:05 +2		0	

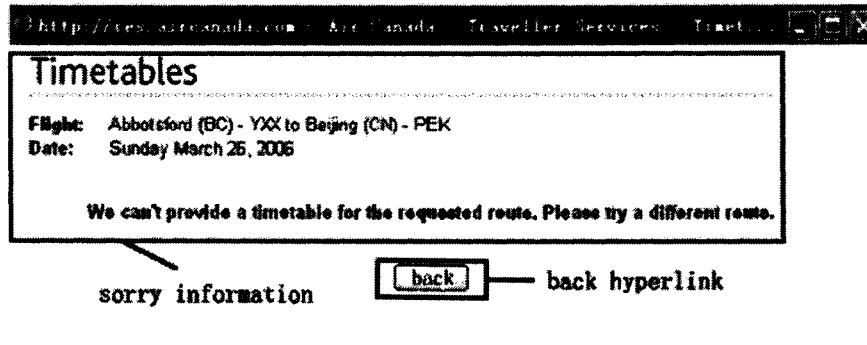
* AC7385 OPERATED BY/EXPLOITE PAR-AIR GEORGIAN

Text Content

Back hyperlink

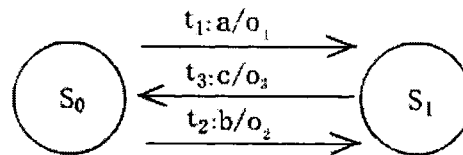


(b)



(c)

Figure4.2 an example of timetable search



S_0 : homepage (static page)

S_1 : resultpage (server page)

a : submit hyperlink with input S.T.
timetable does not exists(condition)

b : submit hyperlink with input S.T.
timetable exists(condition)

c : back hyperlink

O_1 : the related timetable and back hyperlink

O_2 : the sorry information and back hyperlink

O_3 : the user form and the hyperlink of submit

Figure4.3 the EFSM of the web application

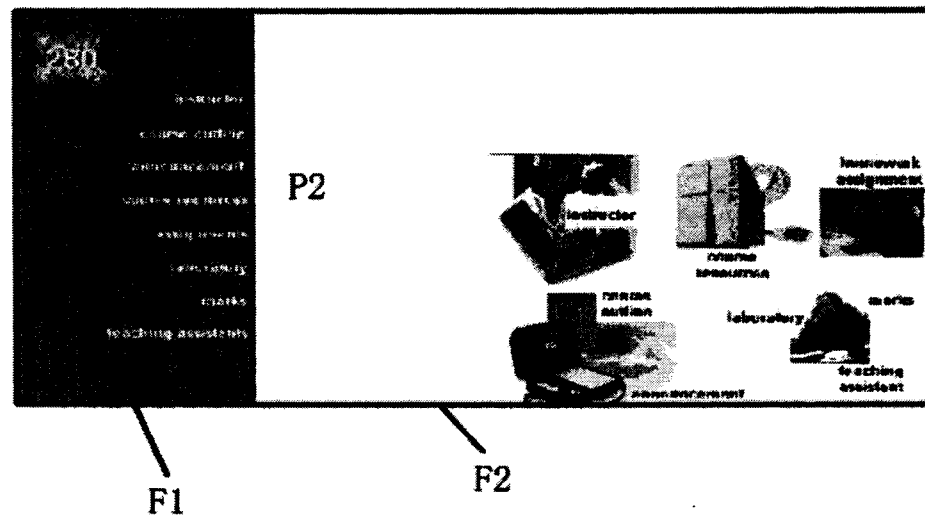
4.2 ADDING FRAMES

Here a web application we refer to stands for a web application without cookies/sessions, dynamic hyperlinks or databases. If a page contains a frameset, actually the page consists of two or more static or server pages. Each frame of the frameset displays the client pages in a certain area of the page containing the frameset. We call a page that contains frameset as *initial frame page* which corresponds to a URL. As we explained before, each frame of an initial frame page can load any page by hyperlink. After loading another page into any frame by hyperlink, an initial frame page turns into a *derived frame page*. An initial frame page is from a static or server page but a derived frame page does not really exist on server side. Here we use the term “frame page” to stand for both of initial and derived frame page.

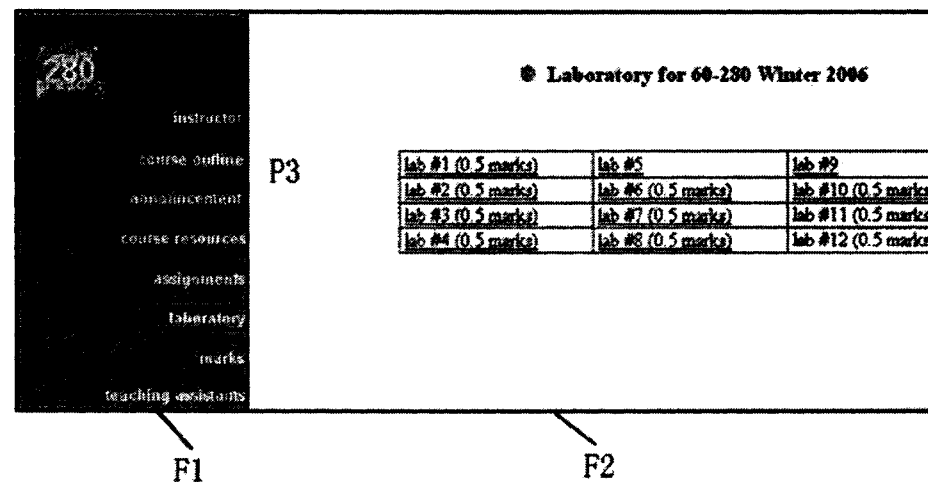
For FSM derived from a web application with frame, we consider each state representing a list of pages. If a server or static page does not have a frameset, a state represents the server or static page and its list of pages only contain this page itself. If a page has a frameset, it should be a frame page and a state representing it has a list of pages contains all pages that are loaded into the frame page.

We use the website of University of Windsor course 60-280 as an example to explain how to model a web application with frame into an FSM. Figure 4.4(a) shows the initial frame page1 and its URL is <http://cs.uwindsor.ca/~xjchen/60280>. Page1 has two frames: index, info window. Each frame contains one page. Index and info window frames contain the pages ‘frame1.html’ and ‘home.html’ respectively. Figure 4.4(b) shows a derived frame page2 and still has the same frames as page1. Index and info window frames contain the pages ‘frame1.html’ and ‘lab.html’ respectively. Figure 4.4(c) shows another derived frame page3 with the same frames. Index and info window frames contain pages ‘frame1.html’ and ‘assignment.html’. The hyperlink ‘laboratory’ is one of the hyperlinks in ‘frame1. html’, which allows the frame ‘info window’ to load page ‘lab.html’. For

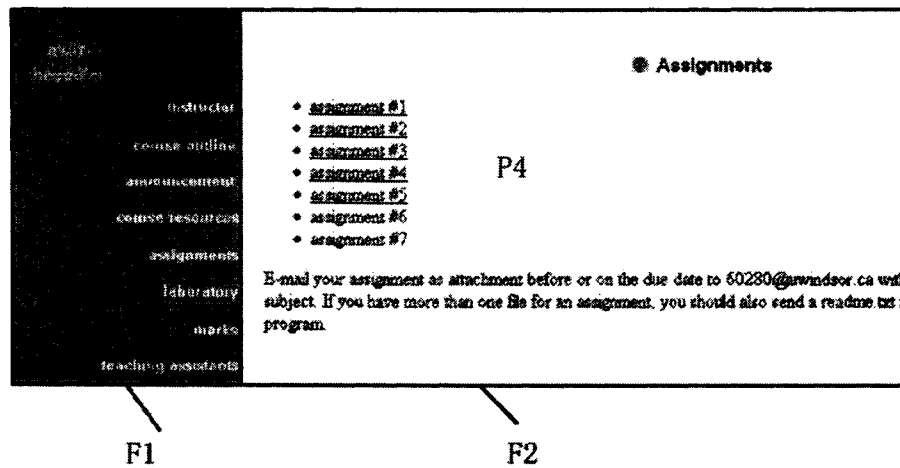
simplicity, we call 'frame1.html', 'home.html', 'lab.html', 'assignment.html', frame index, frame info window, hyperlink 'laboratory' as P1, P2, P3, P4, F1, F2, L1 respectively. We use P1:P2, P1:P3, and P1:P4 to represent Frame page1, Frame page2 and Frame page3. Figure 4.4(d) shows the abstract structure of the part of the web site.



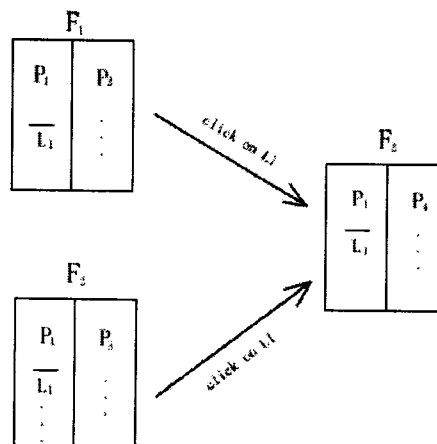
(a) Frame page1



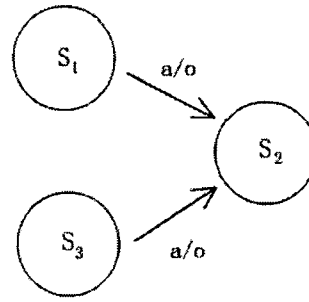
(b) Frame page2



(c) Frame page3



(d)



a : Clicking on L_1
 o : The properties of the
 client pages of P_1 and P_4
 S_1 : corresponding to F_1 ;
 page list:page1,page2
 S_2 : corresponding to F_2 ;
 page list:page1,page3
 S_3 : corresponding to F_3 ;
 page list:page1,page4

(e)

Figure4.4 an example of modeling the website of course 280

Figure 4.3.1(e) shows the corresponding part of FSM M of the web application. Each state in the FSM corresponds to a frame page or a list of pages. Input a ($a \in I_M$) is clicking on the hyperlink of L_4 , Output o ($o \in O_M$) is a pair of a set of hyperlink symbols and a set of related text symbols in P_4 , s_1, s_2, s_3 ($s_1, s_2, s_3 \in S_M$) are states in FSM M , $\delta(s_1, a) = s_2, \delta(s_3, a) = s_2$, $\lambda(s_1, a) = o$ and $\lambda(s_3, a) = o$. The initial state of the FSM is s_1 .

Clearly, given such an FSM as the specification of a web application with web frames, we can use CPP algorithm on its digraph to generate a test sequence that covers each transition at least once. Note that in such a formalism, transitions from the same web page (but possibly in different frame page and thus represented by different states in the FSM) triggered by the same action event actually represent the same hyperlink. We call such transitions equivalent. Figure 4.3.1(b) shows two equivalent transitions in the above example. It is natural to assume that the covered implementation of one transition implies

the covered implementation of all others equivalent to it. Under this hypothesis, the traditional method of using CPP algorithm to generate a test sequence that covers each transition at least once apparently introduces redundant testing. Our goal is to remove such redundant testing in the test sequence generation

CHAPTER V

OUR TESTING METHOD

Let $G = \langle V, E \rangle$ be a strongly-connected digraph. $Q = \{E_1, E_2, \dots, E_k\}$ where $E_i \subseteq E$ for $i \in [1, k]$. The generalized Chinese Postman Problem is to find a minimal-weight tour τ in G such that for each E_i , τ covers at least one edge in E_i at least once. The complexity of this problem is known as NP-hard [4] while heuristic solutions have been discussed in the literature [6].

As we explained before, a specification FSM A can be regarded as a strongly connected digraph and a state in FSM A can be represented as a vertex and a transition can be represented as an edge. Now given an FSM describing the navigational behavior of a web application with web frames, let G be its digraph and $Q = \{E_1, E_2, \dots, E_k\}$ for $E_i \subseteq E (i \in [1, k])$ corresponds to the partition of the transitions in the FSM induced by the equivalence relation defined on the transitions. Then we can adopt the GPP algorithm to find minimal-length test sequence that covers one equivalent transition at least once by setting the weight of each edge in E to be 1. Empirical study has been conducted to compare the lengths of the test sequences generated by our method and the one using CPP algorithm in order to quantitatively evaluate the effectiveness of our solution.

CHAPTER VI

6. IMPLEMENTATION

The purpose of this section is to discuss the implementation of the main algorithms in our work: CPP, Rural Postman Problem (RPP) [12], GPP.

We view a directed graph as $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_i\}$, $i \in V$ is a finite set of vertices and $E = \{e_1, e_2, \dots, e_i\}$, $i \in E$ is a finite set of edges. For each $e_{ij} \in E$, $e_{ij} = \langle \text{lable}, v_i, v_j, c, u \rangle$ where *lable* is an identifier for an edge from v_i to v_j , c_{ij} and u_{ij} are the cost and *capacity* [18] associated with it. Let $i \rightarrow j$ denote a least cost path from v_i to v_j . If there are multiple edges between adjacent vertices, the least cost path will obviously take the cheapest edge of those available. We use (i, j) to denote the cheapest edge from vertex v_i to v_j .

6.1 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 written $d^-(v)$, and the number of edges pointing out of a vertex v is the out-degree written $d^+(v)$. Let δ be

the difference between the in and out degrees: $\delta(v) = d^+(v) - d^-(v)$. 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* [7] 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 $\kappa = \sum_{v \in D^+} \delta(v)$. For an optimal CPT, the extra path taken between D^- and D^+ will be chosen to have least total cost. In general, a CPT may take some of the $D^- \rightarrow D^+$ paths more than once. Let f_{ij} be the number of times the path $i \rightarrow j$ 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 least cost. We use the algorithm of cycle canceling [9] to

find the optimal extras paths. This algorithm starts with an approximate solution, and then iteratively improves it.

In [19], an algorithm of CPP can be sketched as:

Step 1: determine δ of each vertex in graph G . If $\delta = 0$, 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 [18].

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 [20] based on least cost paths $i \rightarrow j$ 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 6.1 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 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 $2 \rightarrow 0$, then the other path is $3 \rightarrow 1$; the alternative is to use the paths $2 \rightarrow 1$ (from vertex 2, pass by vertex 3, 0, to vertex 1) and $3 \rightarrow 0$. 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. Finally, we

choose the paths $2 \rightarrow 1$ and $3 \rightarrow 0$. In Step 4, an Eulerian circuit is found: 0, 1, 3, 0, 1, 2, 3, 0, 2, 3, 0.

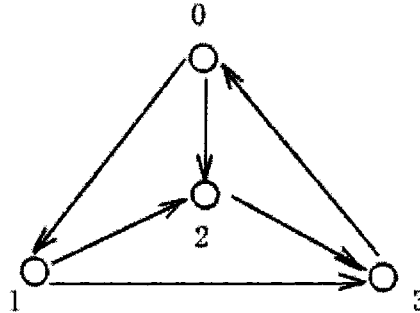


Figure 6.1 an example of the CPP

6.2 IMPLEMENTATION OF THE ALGORITHM OF RPP

The RPP is to find a shortest close tour for a postman who has to travel along some specified roads to deliver letters, i.e., in a directed strongly-connected graph $G = (V, E)$, we need to find a closed walk in G within minimum weight which traverses at least once each edge in a specified subset R of A . A simple case occurs when the subset R is equal to A and we can resolve it by the algorithm of CPP.

The difference between the directed RPP and the directed CPP is that the desired tour in the directed CPP is required to include all edges of G whereas the desired tour in the directed RPP is only required to include the edges of R . Finding the optimal tour that solves RPP is a harder problem than finding the optimal tour that solves the directed CPP. The directed CPP can be solved in polynomial time but the directed RPP is NP-complete so there are only some heuristic algorithms for the directed RPP. Aho gave an algorithm [1] that can find an optimal rural postman tour in G based on R in polynomial time based on the following two assumptions: two assumptions: 1) R is a spanning subgraph

whose vertex set is V ; 2) R is a weakly connected subgraph and its undirected graph is connected. Here we introduce a heuristic algorithm of RPP and the algorithm of RPP for finding optimal tour based on the algorithm of CPP.

6.2.1 HEURISTIC ALGORITHM FOR RPP

One of the heuristic algorithms of RPP is proposed by Christofides [3]. It starts with the graph $G'=(V',R)$ where $V'=\{all\ v_i, v_j \in V\ and\ (v_i, v_j) \in R\}$. The graph G' contains the basic edges and vertices. We need to add some additional edges to G' and find a shortest closed tour to cover each edges of R at least once in the augmentation graph G'' . There are two steps to add additional edges. Very often, in the graph G' , the subset R induces p connected components G_1, \dots, G_p with respective vertex sets V_1, \dots, V_p forming a partition of N' . The first step of adding additional edges to G' is to construct a shortest spanning tree T based on G that can connect G_1, \dots, G_p with minimal $\phi = \sum C_{ij} f_{ij}$. In the second step, if the graph $G' \cup T$ is Eulerian, we do nothing on it. Otherwise, we need to add some edges to $G' \cup T$ between D^- and D^+ of $G' \cup T$ based on G , and then obtain an augmentation graph G'' that is Eulerian. Finally, we can find a Eulerian circuit in G'' as final result.

This algorithm for the directed RPP can be sketched as:

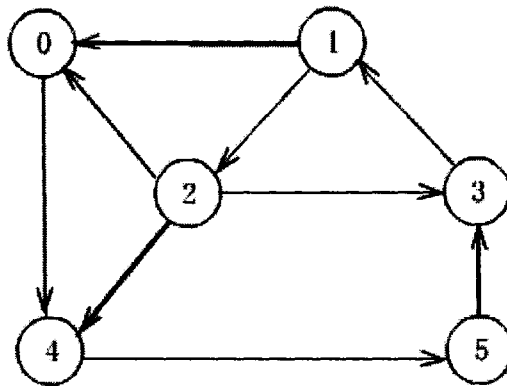
Step 1: Construct the directed minimum spanning tree (DMST) [5] T , rooted at an arbitrary vertex, and connecting G_1, \dots, G_p . Let $\bar{G} = T \cup R$ be the resulting graph.

Step 2: As for the directed CPP, derive an Eulerian graph from \bar{G} by adding edges in a least-cost manner so that the number of incoming edges of each vertex is equal to the number of outgoing edges.

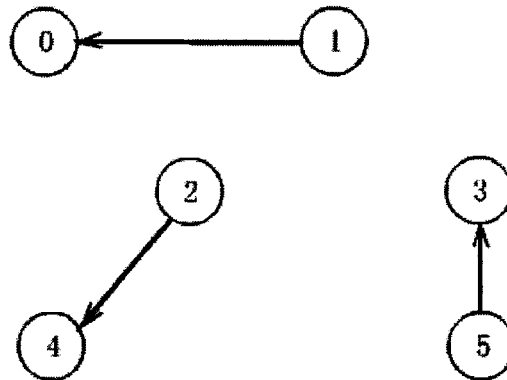
Step 3: Determine an Eulerian circuit on the augmented graph.

In the first step, we use the algorithm of Depth First Search (DFS) [18] to find the strongly-connect components and the algorithm of DMST (directed minimum spinning tree) to construct a directed shortest spinning tree T . In the second step, we use the algorithm of cycle canceling to find the additional edges to make the augmented graph Eulerian as we mentioned before. All the algorithms associated with the RPP are shown in Appendix A. As the complexity of algorithm of cycles canceling is dominant, the complexity of this heuristic algorithm is $O(n^2 m^3 \{\log n\})$ [9].

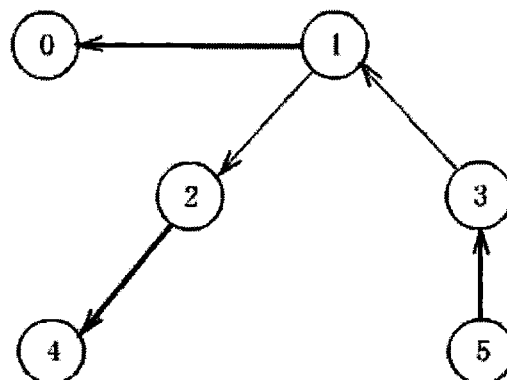
Here we use Figure 6.2 to illustrate the execution of the heuristic algorithm to solve the RPP. Figure 6.2(a) shows the graph $G = (V, E)$ and the edges of R are shown by bold line. Figure 6.2(b) shows the graph $G = (V', R)$ and the subset R induces 3 connected components G_1, G_2, G_3 with respective vertex sets $V_1 = \{0, 1\}$, $V_2 = \{2, 4\}$ and $V_3 = \{3, 5\}$ forming a partition of V' . Figure 6.2(c) shows the shortest spanning tree T rooted at V_3 and the subset R . In the graph $R \cup T$, there are $D^+ = \{1, 5\}$ and $D^- = \{0, 4\}$. We find that there are two ways to choose the set of extra paths. If one path is $0 \rightarrow 1$ (from vertex 0, pass by vertex 4, 5, and 3, to vertex 1), then the other path is $4 \rightarrow 5$; the alternative is to use the paths $0 \rightarrow 5$ (from vertex 0, pass by vertex 4, to vertex 5) and $4 \rightarrow 1$ (from vertex 4, pass by vertex 5 and 3, to vertex 1). As it happens, the choices have the equal cost ($c_{01} + c_{45} = 4 + 1, c_{05} + c_{41} = 2 + 3$), and both can be used. We choose the extra path $(0, 1)$ and $(4, 5)$ to $R \cup T$ and obtain the final result graph. The CPT in the graph is 0, 4, 5, 3, 1, 2, 4, 5, 3, 1.



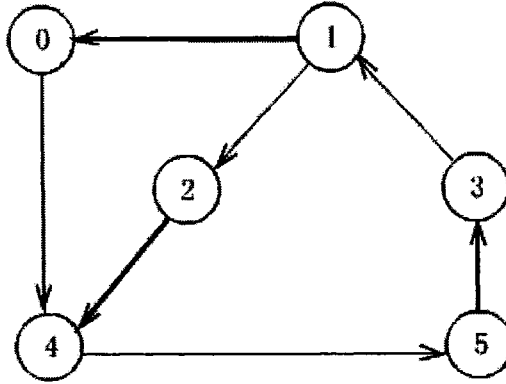
(a) The graph $G = (V, E)$



(b) The graph $G' = (V', R)$



(c) The graph $R \cup T$



(d) The resulting graph

Figure 6.2 a heuristic algorithm to solve the RPP

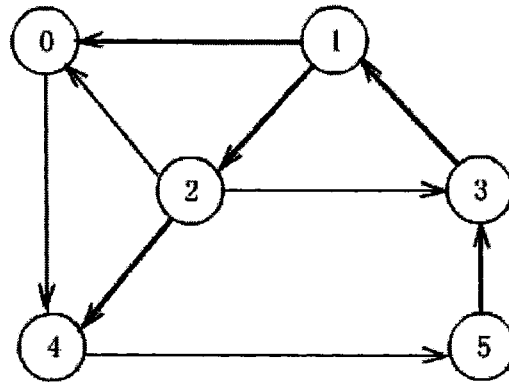
6.2.2 AHO'S ALGORITHM FOR RPP

Aho's algorithm is used to find an optimal RPP tour in G based on the two assumptions we mentioned before. In [1], it is shown that if these assumptions hold then finding optimal RPP is equivalent to finding a rural symmetric augmentation graph \hat{G} , which can be reduced to a minimum-cost maximum flow problem on a graph $G_F = (V_F, E_F)$ constructed from G . Define the *index* ξ of a vertex $v_i \in G$ to be the difference of the number of edges in R going into v_i and the number of edges in R leaving v_i . Let $V_F = V \cup \{s, t\}$, where s and t are the *source* and *sink* of G_F , and let $E_F = E \cup \{(s, t) : v_i \in C\} \cup \{(v_j, t) : v_j \in D\}$, where $C \subset V_F$ ($D \subset V_F$) is the set of vertices in G with positive (negative) indices. Let each edge (s, v_i) have cost zero and capacity $\gamma(s, v_i) = \xi(v_i)$. The remaining edges in E_F have the same cost in G_F as in G and infinite capacity. For simplicity, self-loop in E are omitted in E_F , and when there are multiple edges between v_i and v_j in G , only an edge with the minimum cost is included in E_F .

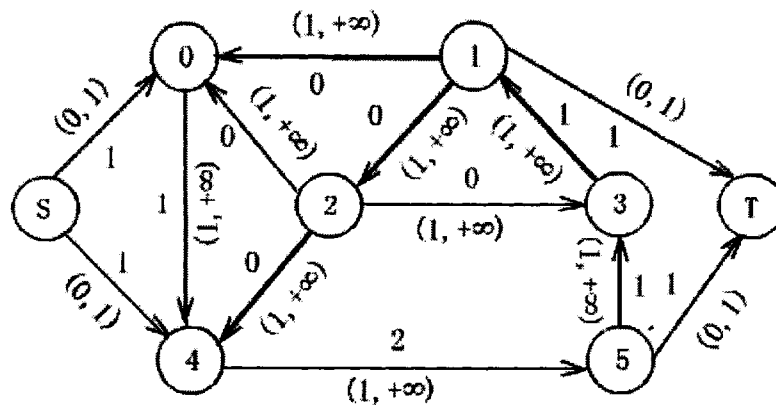
According to [1], given a minimum-cost maximum flow F on G_F , we can obtain symmetric augmentation graph \hat{G} based on F and define a function χ on $E \cup R$:

$$\chi(v_i, v_j) = \begin{cases} F(v_i, v_j) + 1, & \text{if } (v_i, v_j) \in R \\ F(v_i, v_j) & \text{if } (v_i, v_j) \in E \end{cases}$$

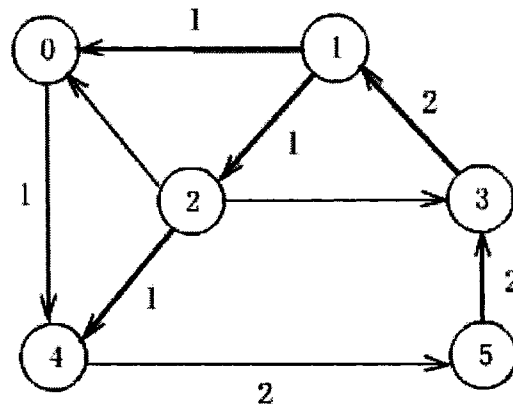
The graph \hat{G} formed by replicating each edge (v_i, v_j) in $E \cup R$ $\chi(v_i, v_j)$ times is symmetric and contains each edge in R at least once. Here we use cycle canceling algorithm [2] to find a minimum-cost maximum flow F on G_F . In Aho's algorithm, the complexity of cycle canceling is dominant so the complexity of this algorithm is $O(nm^3C)$ ($C = \max\{c_{ij} : C < \infty \text{ and } (i, j) \in A\}$)



(a) An graph $G = (V, E)$



(b) Graph $G_F = (V_F, E_F)$



(c) Graph \hat{G}

Figure 6.3 Example of Aho's algorithm

We use Figure 6.3 to show Aho's algorithm for RPP. Figure 6.3(a) shows a strongly-connected graph $G = (V, E)$ and the cost of each edge's is 1. The bold lines are the edges of R. Figure 6.3(a) and (c) show the graphs G_F and \hat{G} . In Figure 6.3(c), the number beside each edge shows how many times the edge is covered in RPP tour.

6. 3 THE IMPLEMENTATION OF A GPP ALGORITHM

For a directed weighted graph $G = (V, E)$, all edges are partitioned into different edge subsets E_1, \dots, E_k ($k \in N$). The GPP is to find a closed walk in G and with minimum weight which traverses at least one edge from each of the subsets E_i ($i = 1, \dots, k$) at least once. Since the complexity of GPP is NP-hard, we present a heuristic algorithm for the problem: we simply pick up one edge from each subset and then this problem is reduced to RPP. Here we use random way to select one edge from each subset. That is, for each subset, one edge is selected from it randomly.

This algorithm can be sketched as followings:

Step 1: Choose one edge from each subset randomly.

Step 2: follow the heuristic algorithm of RPP we presented in Section 6.2.1.

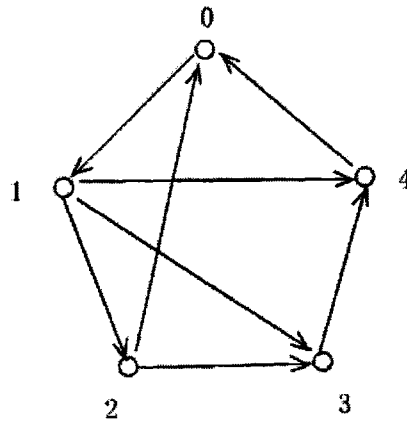


Figure 6.4 an example to illustrate the GPP algorithm

Here we use Figure 6.4 to show the algorithm of GPP. There are three subsets of edges in graph G : $E_1 = \{(0,1), (4,0)\}$, $E_2 = \{(1,3), (1,4)\}$, $E_3 = \{(2,0), (3,4)\}$ and $E_4 = \{(1,2), (2,3)\}$. In the first step, we calculate the numbers of edges of each vertex and the result is that numbers of v_0, v_1, v_2, v_3 and v_4 are 3, 4, 3, 3 and 3, respectively. In the second step, the edges (0,1), (1,3), (2,0) and (1,2) are selected from each subset. After the third step (follow the heuristic algorithm of RPP), the path is 01201340.

CHAPTER VII

EXPERIMENTS AND EVALUATION

To evaluate our method, we compare our testing method with the one based on Aho's algorithm for RPP. Given a strongly-connected digraph G from a specification FSM of a web application, our method is to put all edges in G that are equivalent into a same subset and then make use of the algorithm of GPP to generate a reduced-length test sequence. On the other hand, the method of using Aho's algorithm will generate an optimal test sequence which can cover all required edges (do not contain the edges from 'back home' button) at least once. The reason we compare with Aho's algorithm is that the assumptions subset R used in this algorithm hold in our setting: we assume we can reach all web pages from home page in web applications, and as a result, the subset R of G from a specification FSM is a weakly connected spanning subgraph. Here we compare the lengths of test sequences that are generated by the two different methods and the time to generate the test sequences. The FSM is input into our Java program as a directed strongly connected graph to generate a test sequence. The length and time can be affected by the number of vertices, edges and subsets. In this section, we present our experimental result for the effect of the three factors on the length and time by fixing the value of the other two factors.

7.1 LENGTH OF TEST SEQUENCE

Any of the three factors: the number of vertices, (selected) edges (in R) and subsets can affect the lengths of test sequences generated by the algorithm of RPP and GPP so we change the value of one of them and fix the others to observe the lengths of test sequences based on RPP and GPP and the ratio of the lengths of test sequences generated by RPP to the lengths of test sequences generated by GPP.

7.1.1 NUMBER OF VERTICES

When the numbers of the edges (in R) and the subsets are fixed, the lengths of test sequences, no matter they are generated by RPP or GPP, increase with the augmentation of the number of vertices. For the lengths of test sequences generated by GPP, if the number of vertices increase (the number of the edges in R and the subsets are fixed), it makes the subgraph formed by selected edges from each subset more sparse so that more edges is added into the subgraph to keep the augmentation of the subgraph Eulerian. This means that more edges are required to be traverses and thus the test sequences become longer. This is also true for the lengths of test sequences generated by RPP.

Table 7.1 shows the trend of the ratio of the lengths of test sequences generated by GPP to the lengths of test sequences generated by RPP. The result indicates that the lengths of test sequences generated by GPP are generally shorter than the lengths of test sequences generated by RPP.

Vertices	Edges	Subsets	Times	LGPP	LRPP	Tratio	Lratio
6	30	5	100	812	4138	1/1	0.1962
8	30	5	100	940	4348	1/1	0.2169
10	30	5	100	1062	4932	1/1	0.2153
12	30	5	100	1215	5339	4/4	0.2275
14	30	5	100	1362	5748	4/4	0.2369
16	30	5	100	1530	6360	4/3	0.2404

18	30	5	100	1628	6384	3/2	0.2550
20	30	5	100	1769	6890	6/4	0.2567
22	30	5	100	1890	7224	12/6	0.2616
24	30	5	100	2074	7443	14/6	0.2786

Table7.1

* Vertices: the number of vertices in G ; Edges: the number of edges in R ; Subsets: the number of subsets; Times: the number of graphs which meet the related requirements in each tuple and are generated randomly; LGPP: the total lengths of testing sequences generated by GPP in all cases; LRPP: the total number of testing sequence generated by RPP in all cases. Tratio: the time (unit is second) to generate test sequence by GPP divided by the time to generate test sequence by RPP; Lratio: the ratio of the lengths of test sequences generated by GPP to the lengths of test sequences generated by RPP.

7.1.2 NUMBER OF EDGES

When the numbers of vertices and subsets are fixed, the augmentation on the number of edges causes the increase of the lengths of test sequences generated by RPP while it has little effect on the test sequence generated by GPP. Thus, the length ratio decreased. The reason is that each additional edge can cause an additional edge required to traverse under the algorithm of RPP but it does not do so under the algorithm of GPP. Namely, the number of the edges required to traverse under the algorithm of GPP is equal to the number of subsets.

Table 7.2 shows the trend of the ratio of the lengths of test sequences generated by GPP to the lengths of test sequences generated by CPP with different number of edges.

Vertices	Edges	Subsets	Times	LGPP	LRPP	Tratio	Lratio
10	18	5	100	1153	3344	1/1	0.3447
10	20	5	100	1157	3696	1/1	0.3138
10	22	5	100	1166	3901	2/2	0.2988
10	24	5	100	1141	4163	1/2	0.2740
10	26	5	100	1115	4567	2/3	0.2452
10	28	5	100	1109	4673	2/4	0.2373
10	30	5	100	1132	4898	1/4	0.2311
10	32	5	100	1062	5163	2/4	0.2056
10	34	5	100	1067	5324	1/4	0.2004
10	36	5	100	1065	5511	3/4	0.1932

Table7.2

7.1.3 NUMBER OF SUBSETS

When the number of edges and vertices in R are fixed, the lengths of test sequences generated by RPP only have little changes with the augmentation of the subsets that contain equivalent edges. On the other hand, the lengths of test sequences generated by GPP increase with the augmentation of the number of edges because the augmentation of the number of subsets makes the number of selected edges from each subset increase under the algorithm of GPP and more edges is required to be traversed. However, the edges required to be traversed under the RPP is fixed. Thus the ratio of lengths increases. Table 7.3 shows the trend of the ratio with different number of subsets.

Vertices	Edges	Subsets	Times	LGPP	LRPP	Tratio	Lratio
10	20	3	100	857	3642	1/1	0.2353
10	20	5	100	1203	3702	1/1	0.3249
10	20	7	100	1423	3678	1/1	0.3868
10	20	9	100	1687	3631	2/1	0.4646
10	20	11	100	1882	3689	2/1	0.5101
10	20	13	100	2121	3699	2/1	0.5733
10	20	15	100	2344	3643	3/1	0.6434
10	20	17	100	2638	3652	3/1	0.7223
10	20	19	100	2840	3721	3/1	0.7632

Table7.3

7.2 TIME TO GENERATE TEST SEQUENCE

Here we use the same method to show the effect of the three factors on time to generate test sequences. We fix the numbers of two factors to observe the time consumption to generate test sequence caused by one factor. From the Table 7.1, Table 7.2 and Table 7.3, we can obviously realize that the trend of time consumption to generate test sequence is basically the same as that of the lengths of testing sequence. This confirms that the time complexities of the two methods are comparable.

CHAPTER VIII

CONCUSION AND FUTURE WORK

8.1 CONCLUSION

In this thesis work, we have presented an approach for representing navigational behavior of web applications with web frames in terms of EFSM and FSM. Focusing on web applications without cookies/sessions, databases that are used to record the navigation information and dynamic hyperlinks, we have presented an FSM-based navigation model considering web frames and it can model most of the elements of web applications, such as web frames, hyperlinks, client pages, server/static pages, navigations.

Based on the FSM model, we have defined equivalent navigations (transitions) in the FSM models of the web applications. We have explained how to apply an algorithm of GPP to the FSM model with equivalent transitions to generate test sequences with reduced length.

A Java program is implemented to compare our method based on the algorithm of GPP to the one that is based on Aho's algorithm for RPP. The experiment results show that the efficiency of our method is high.

8.2 FUTURE WORK

The future work can be considered in three aspects:

- Looking for a better way to define equivalent navigations in terms of equivalent states following the literature and compare it with existing well-known equivalence notions.

- Improve the method that reduce the GPP into the RPP: It means that we need a more efficient algorithm to pick up “good” arcs from each subset of G .
- Extend the work on efficient test sequence generation to EFSMs.

REFERENCES

- [1] A. V. Aho, A.T. Dahbura, D. Lee, and M. U. Uyar, *An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours*, IEEE Trans. on Communications, Vol. 39, No. 11, Pages 1604-1615, Nov. 1991.

- [2] R K. Ahuja, T. L. Magnanti and J. B. Orlin. *Network flow*. Prentice-Hall, 1993

- [3] N., V. Christofides, C, A. Corberan and E. Mota. *An algorithm for the Rural Postman Problem on a Directed Graph*. Math. Prog, Study 26, 1986 pages 155-166

- [4] M. Dror and M. Haouari, *Generalized Steiner Problems and other variants*, Journal of Combinatorial Optimization, vol.4, No.4, pages 415-436, 2000.

- [5] J. Edmonds, *Optimum Branching*. J. Res. Natl. Bur. Stand., Section B. 71, pages 233-240, 1967

- [6] H. A. Eiselt, M. Gendreau and G. Laporte, *Arc routing problems, part II: The rural postman problem*, Operations Research 43 (1995), pages 399-414.

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

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

[9] A. V. Goldberg, R. E. Tarjan, Finding Minimum-cost Circulations by Canceling Negative Cycles, *Journal of the Acm*, 36(4) pages: 873-886, 1989.

[10] D. Hogrefe, *ESFTELLE, LOTOS and SDL*, Springer-Verlag, 1989

[11] D. C. Kung, C. Liu and P. Hsia, An Object-Oriented Web Test Model for Testing Web Applications, in *The 1st Asia-Pacific Conference on Quality Software (APAQS 2000)*, pages 111-120, 2000.

[12] J. K. Lenstra and A. H. G. Rinnooy Kan. *On General Routing Problem*. *Networks* 6, pages 273-280, 1976.

[13] C. Liu, D. Kung, P. Hsia, and C. Hsu. *Structural testing of web applications*, in *Proceedings of the 11th IEEE International Symposium on Software Reliability, Engineering*, pages 84-96, Oct. 2000.

[14] G. Di Lucca, A. Fasolino, F. Faralli, U. de Carlini, *Testing Web Application*, In *IEEE International Conference On software Maintenance (ICSM' 02)*, 2002, pages: 0310-0319.

- [15] G. D. Lucca and M. D. Penta, *Considering Browser Interaction in Web Application Testing*, in Proceedings of the 5th IEEE International Workshop on Web Site Evolution (WSE'03), 2003.
- [16] S. Naito and M. Tsunoyama, *Fault detection for sequential machines by transitions tours*, in Proceeding of IEEE Fault Tolerant Comput. Symp., IEEE Computer Soc. Press, pages 238-243, 1981
- [17] F. Riccva and P. Tonella, *Analysis and testing of web applications*, in Proceeding Of the 23rd Internal conference on Software Engineering, 2001, Toronto, Ontario, Canada, pages: 25-34.
- [18] R.E. Tarjan, *Data Structures and Network Algorithm*. Philadelphia, PA:Society for Industrial and Applied Mathematics, 1983.
- [19] Harold W. Thimbleby. *The directed Chinese Postman Problem*. journal of Software - Practice and Experience, 33(11) pages:1081--1096, September 2003.
- [20] S. Skiena, *The Algorithm Design Manual*, Springer Verlag, 1998.

APPENDIX A THE PROGRAM FOR GPP

```
import java.io.*;
import java.util.*;

public class GPP
{
    int N; // number of vertices
    int ND; // number of divisions
    int NSV; // number of selected vertice
    float FBC;
    OpenCP G;
    int v;
    int vv;
    int vvv;
    int seqDFS[];
    int part1[];
    int tcost;
    int knumber;
    int Fnumber;
    int cnumber;
    int VChange[];
    Vector nVChange[];
    int visit[];
    int DC[];
    int CD;
    int cv;
    int r1[];
    int r2[];
    int DN[];
    int DS[];
    int DE[];
    int[][] AA;
    int temple[];
    int S2I[];
    int CNofSA[]; // connection number of selected arcs
    int CN[]; // connection number of selected arcs
    GPP GS;
    GPP CG;
    GPP CompleteG;
    GPP MST[];
    GPP Final;
    GPP OpenFinal;
    GPP OpenbestGraph;
    GPP Templ;
    boolean overarcs;
```

```

int NameNo[][];// for multiple arcs between two nodes.
String SelectedArcs[];
String selectedArcs[]; //selected Arcs
int selectedVertex[]; //selected vertetice
int delta[]; // deltas of vertices
int neg[], pos[]; // unbalanced vertices
int arcs[][]; // adjacency matrix, counts arcs between vertices
Vector label[][]; // vectors of labels of arcs (for each vertex pair)
Vector length[][];//recode the length of each arcs
int f[][]; // repeated arcs in CPT
float c[][]; // costs of cheapest arcs or paths
String cheapestLabel[][]; // labels of cheapest arcs
boolean defined[][]; // whether path cost is defined between vertices
int path[][]; // spanning tree of the graph
float basicCost; // total cost of traversing each arc once

```

```

int timestart;
int timeend;
int timecost;
int gv[];

```

```

void timecost(String tag)
{
    String t = tag;

    java.util.Date m_date = new java.util.Date();
    int nowSecond = m_date.getSeconds();
    int nowMinute = m_date.getMinutes();
    int nowHour = m_date.getHours();
    if (t == "start")
    {
        timestart = nowHour*3600 + nowMinute*60 + nowSecond;
    }
    if (t == "end")
    {
        timeend = nowHour*3600 + nowMinute*60 + nowSecond;
        timecost = timeend - timestart;
    }
}

```

```

GPP(int vertices)
{
    if( (N = vertices) <= 0 ) throw new Error("Graph is empty");
    delta = new int[N];
    defined = new boolean[N][N];
    label = new Vector[N][N];
    length = new Vector[N][N];
    c = new float[N][N];
    f = new int[N][N];
    arcs = new int[N][N];
    cheapestLabel = new String[N][N];
    path = new int[N][N];
    NSV = 0;
    v = 0;
    vv = 0;
    vvv = 0;
    CD = 0;
    cv = 0;
    cnumber = -1;
    NameNo = new int [N][N];
    CNofSA = new int [135];
    CN = new int [N];
}

```

```

GPP addArc(String lab, int u, int v, int cost)
{
    //      String templ = StringtoInt(cost);

    if(label[u][v] == null)
    {
        label[u][v] = new Vector();
        length[u][v] = new Vector();
    }
    basicCost += cost;
    label[u][v].addElement(lab);
    length[u][v].addElement("1");

    if( !defined[u][v] || c[u][v] > cost )
    {
        c[u][v] = cost;
        cheapestLabel[u][v] = lab;
        defined[u][v] = true;
        path[u][v] = v;
    }
}

```

```

    }
    arcs[u][v]++;
    delta[u]++;
    delta[v]--;
    CN[u]++;
    CN[v]++;
    return this;
}

```

```

void getselectedArcs()
{

```

```

    int Number = 135;
    boolean a[][];
    selectedArcs = new String[Number];

```

```

    a = new boolean[N][N];

```

```

    for(int i = 0; i < Number; i++)
    {
        selectedArcs[i] = "-1";
    }

```

```

    Random r1=new Random();
    int v1;
    int v2;
    int j = 0;

```

```

    while(j < Number)
    {
        v1 = r1.nextInt(N);
        v2 = r1.nextInt(N);

        if(arcs[v1][v2] > 0 && !a[v1][v2])
        {
            selectedArcs[j] = cheapestLabel[v1][v2];
            a[v1][v2] = true;
            j++;
        }
    }

```

```

    }

    for(int h = 0; h < Number; h++)
    {
        //      System.out.print(selectedArcs[h]);

    }
//      System.out.println("");

    /*
    selectedArcs = new String[9];
    selectedArcs[0] = "21";
    selectedArcs[1] = "12";
    selectedArcs[2] = "02";
    selectedArcs[3] = "14";
    selectedArcs[4] = "01";
    selectedArcs[5] = "03";
    selectedArcs[6] = "42";
    selectedArcs[7] = "10";
    selectedArcs[8] = "43";
    */
}

////////// pick up all subsets and their arcs
void getArc1(int Number, int Subsets, int Vertices, int[][] A)
{

    boolean a[][];
    boolean used[][][];
    SelectedArcs = new String[Number];
    used = new boolean[N][N][Number];

    AA = new int[Vertices][Vertices];
    for(int k = 0; k < Vertices; k++)
        for(int m = 0; m < Vertices; m++)
            AA[k][m] = A[k][m];

    a = new boolean[N][N];

    for(int i = 0; i < Number; i++)
    {

```

```

        SelectedArcs[i] = "-1";
    }

    Random r1=new Random();
    int v1;
    int v2;
    int v3;
    int j = 0;
    int unit;

    DN = new int[Subsets];
    DS = new int[Subsets];
    DE = new int[Subsets];

    while(j < Number)
    {
        v1 = r1.nextInt(N);
        v2 = r1.nextInt(N);

        if(arcs[v1][v2] > 0)
        {
            v3 = r1.nextInt(NameNo[v1][v2]);
            if(!used[v1][v2][v3])
            {
                SelectedArcs[j] =
String.valueOf(v1)+String.valueOf(v2)+String.valueOf(v3);
                CNoofSA[j] = CN[v1] + CN[v2];
                a[v1][v2] = true;
                used[v1][v2][v3] = true;
                AA[v1][v2]--;
                j++;
            }
        }
    }
}

```



```

unit = Number/Subsets;

// put the number of arcs to each subset.
for(int i = 0, t = 0; i < Subsets; i++)
{
    if(i == Subsets-1)
    {
        DN[i] = Number-t;
        DS[i] = t;//?
        DE[i] = Number-1;    //?
    }
    else
    {
        DS[i] = t;
        DN[i] = unit;
        t = unit+t;
        DE[i] = t-1;
    }
}

}

//////////pick up one possibility
String[] getArc2(int Number, int Subsets,int Vertices, int Arcs, int[][] AA)
{
    int a, b,at;
    int value = 1;

    selectedArcs = new String[Subsets];
    for(at = 0; at < Subsets; at++)
    {
        for(int i=DS[at]; i<= DE[at]; i++)
        {
            int maxCN = 0;
            if(CNofSA[i] > maxCN)
            {
                selectedArcs[at] =
SelectedArcs[i];
            }
        }
    }
}

```

```

//
at++;
/*
for(int n = at; n < Arcs-Number+Subsets;)
{
    for(int zz = 0; zz < Vertices; zz++)
        for(int xx = 0; xx < Vertices;
xx++)
        {
            if(AA[zz][xx] > 0)
            {
                selectedArcs[n]=String.valueOf(zz)+String.valueOf(xx);
                n++;
            }
        }
    }
}
*/
return selectedArcs;
}

```

```

void getMselectedArcs()
{

    int Number = 135;
    int a[][];
    selectedArcs = new String[Number];

    a = new int[N][N];

    for(int i = 0; i < Number; i++)
    {
        selectedArcs[i] = "-1";
    }
}

```

```

Random r1=new Random();
int v1;
int v2;
int v3;
int j = 0;

while(j < Number)
{
    v1 = r1.nextInt(N);
    v2 = r1.nextInt(N);

    if(arcs[v1][v2] > 0 && a[v1][v2] <= arcs[v1][v2])
    {
        v3 = r1.nextInt(arcs[v1][v2]);
        selectedArcs[j] =
label[v1][v2].elementAt(v3).toString();
        a[v1][v2]++;
        j++;
    }
}

for(int h = 0; h < Number; h++)
{
    //
    System.out.print(selectedArcs[h]);

    //
    System.out.println("");
}

/*
selectedArcs = new String[14];
selectedArcs[0] = "41";
selectedArcs[1] = "23";
selectedArcs[2] = "40";
selectedArcs[3] = "10";
selectedArcs[4] = "30";
selectedArcs[5] = "21";
selectedArcs[6] = "32";
selectedArcs[7] = "02";
selectedArcs[8] = "12";
selectedArcs[9] = "43";

```

```

        selectedArcs[10] = "42";
        selectedArcs[11] = "03";
        selectedArcs[12] = "13";
        selectedArcs[13] = "24";
        */
    }

```

```

void selectedGraph()
{
    int flag = 0;
    selectedVertex = new int[N];

    for(int q = 0; q < N; q++)
    {
        selectedVertex[q] = -1;
    }

    ////////////count the number of selectedvetice
    for(int i=0; i < selectedArcs.length; i++)
        for (int j =0; j < label.length; j++)
            for (int k = 0; k < label[j].length; k++)

                if(label[j][k]!= null)
                {

                    for(int m = 0; m < label[j][k].size(); m++)
                    {

```

```

                                if
(selectedArcs[i].equals(label[j][k].elementAt(m).toString()))
                                {

                                if ( selectedVertex[j]
                                {
                                NSV++;
                                }
                                if ( selectedVertex[k]
                                {
                                NSV++;
                                }
                                }
                                }
                                }

visit = new int[NSV];
for(int z = 0; z < NSV; z++)
{
visit[z] = -1;
}

DC = new int[200];
for(int z = 0; z < 200; z++)
{
DC[z] = -1;
}

r1 = new int[NSV];

```

```

        for(int z = 0; z < NSV; z++)
        {

                r1[z] = -1;

        }

seqDFS = new int[NSV];
        for(int z = 0; z < NSV; z++)
        {

                seqDFS[z] = -1;

        }

r2 = new int[NSV];
        for(int z = 0; z < NSV; z++)
        {

                r2[z] = -1;

        }

//////////vertex change

        VChange = new int[NSV];
        for(int i = 0, j = 0; i < N; i++)

                {

                        if

                        {

                                VChange[j]

                                j++;

                        }

                }

        = i;

//////////change end

```

//////////construct the graph which just include the selected arcs.

```

GS = new GPP(NSV);
int a = 0, b = 0;
for(int i=0; i < selectedArcs.length; i++)
    for (int j =0; j < label.length; j++)
        for (int k = 0; k <
label[j].length; k++)

            if(label[j][k]!= null)
            {

                for(int m = 0; m <
label[j][k].size(); m++)

                    {

                        if
                        {

                            for(int f = 0; f < VChange.length; f++)

                                {

                                    }

                                }

                            for(int f = 0; f < VChange.length; f++)

                                {

                                    }

                                }

                            GS.addArc(selectedArcs[i], a, b, 1);

                        }

                    }

            }

```

```

    }

//////////construction end

////////// deep first research
int DFS(int x)
{

    visit[v] = x;
    r1[x] = 1;
    v++;

    for(int j = 0; j < NSV; j++)
    {
        if ((GS.arcs[x][j] > 0 || GS.arcs[j][x] > 0)
        && r1[j] == -1)
        {
            DFS(j);
        }
    }

    seqDFS[NSV-vv-1] = x;
    vv++;
    DC[CD] = x;
    CD++;

    return x;

}

//////////DFS end

////////// reverse deep first research
void RDFS(int x)
{

    r2[x] = 1;

    for(int j = 0; j < NSV; j++)
    {
        if (GS.arcs[j][x] > 0 && r2[j] == -1 && r1[j] == 1)
        {
            RDFS(j);
        }
    }
}

```



```

    }

    DC[CD] = x;
    CD++;
}

//////////RDFS end

//////////strongly connected component
void CC()
{
    if (NSV <= 0) throw new Error("select arcs firstly!");
    boolean q = true;
    int i;

    while(q)
    {
        for( i = 0; i < NSV; i++) //try each vertice which is selected
        {
            if(r1[i] == -1) break;
        }

        DFS(i);
        DC[CD]= -2;
        CD++;

        /*
        for(int l = 0; l < NSV; l++)
        {
            if(seqDFS[l] != -1 && r2[seqDFS[l]] != 1
&& r1[seqDFS[l]] == 1)
            {
                RDFS(seqDFS[l]);
                DC[CD]= -2;
                CD++;
            }
        }
        */
    }
}

```

```

        q = false;
        for(int j = 0; j < NSV; j++)
        {
            if(seqDFS[j] == -1) q = true;
        }

        //1      System.out.println("1111");

    }

    //////////cc end

    //////////compacted graph
    void CG()
    {

        int c1 = 0, c2 = 0;
        int k = 0;
        int m = 0;
        int tf1 = 1;
        int tf2 = 0;
        boolean tag = false;
        S2I = new int[N];

        for(int t = 0; t < N; t++)
            S2I[t] = t;

        for(int i = 0; i < 2*NSV; i++)
        {
            if(DC[i] != -2 && DC[i] != -1) c1++;
            if(DC[i] == -2) c2++;
        }

        cv = N-c1+c2;

        nVChange = new Vector[cv];
        CG = new GPP(cv);

        for(int s = 0; s < cv; s++)
        {

```

```

nVChange[s] = new Vector();

}

for(int i = 0; i < N; i++)
{
    for (int j = 0; j < VChange.length; j++)
        if(i == VChange[j])
        {
            tag = true;
        }
    if(tag != true)
    {

nVChange[k].addElement(String.valueOf(i));
        k++;
    }
    else tag = false;
}

knumber = k;

while(DC[m] != -1 && k < cv)
{
    if (DC[m] != -2)
    {

nVChange[k].addElement(String.valueOf(VChange[DC[m]]));
        m++;
    }
    else
    {
        k++;
        m++;
    }
}

for(int n = 0; n < N; n++)
    for(int b = 0; b < N; b++)
    {

        if(arcs[n][b] >= 1)

```

```

        {
            for(int h = 0; h <
nVChange.length; h++)
                for(int f = 0; f <
nVChange[h].size(); f++)
                    {
                        if(nVChange[h].elementAt(f).toString().equals(String.valueOf(n)))
                            tf1 = h;

                        if(nVChange[h].elementAt(f).toString().equals(String.valueOf(b)))
                            tf2 = h;
                    }
                for(int vi = 0; vi < arcs[n][b];
vi++)
                    if( tf1 != tf2)
                    {
                        CG.addArc(label[n][b].elementAt(vi).toString(), tf1, tf2, 1);
                    }
            }
        }
    }
    CG.leastCostPaths();

}

//////////cg end
//////////complete graph
void CompleteG()
{
    int n = nVChange.length - knumber;
    CompleteG = new GPP(n);
    for(int i = 0; i < n; i++)
        for(int j = 0; j < n; j++)
        {
            if(i != j)
            {
                String name = String.valueOf(i)
+ String.valueOf(j);

```

```

CompleteG.addArc(name, i, j,
(int)CG.c[knumber+i][knumber+j]);
    }
    }
    CompleteG.leastCostPaths();
}

```

```

//////////complete graph end
//////////MST
void MST()
{
    if(CompleteG.N > 1)
    {
        MST = new GPP[100];
        boolean circle = false;
        boolean inter = false;
        boolean M = false;
        boolean FF = false;
        boolean inter1 = false;
        boolean inter2 = false;
        int a;
        int j = 0;
        int kb = 0;
        int ka = 0;
        int m = 0;
        int temp = -1;
        int temp1 = -1;
        int x1 = -1, x2 = -1;
        int r = -1;
        int q = -1;
        int recoder[];
        int MUV[];
        int MN = 0;
        int rmn = 0;
        int v;

        cnumber = -1;
        recoder = new int[CompleteG.N];
        MUV = new int[CompleteG.N];
        for(int g = 0; g < CompleteG.N; g++)
        {

```

```

        MUV[g] = -1;
    }
    for(int i = 0; i < 100; i++)
        MST[i] = new GPP(CompleteG.N);

    temp = (int)CompleteG.basicCost;

    for(int b = 1; b < CompleteG.N; b++)
    {
        for(a = 0; a < CompleteG.N; a++)
        {
            if(CompleteG.c[a][b] < temp)
            {
                temp = (int)CompleteG.c[a][b];
                kb = b;
                ka = a;
            }
        }

        MST[j].addArc(CompleteG.label[ka][kb].toString(), ka, kb, temp);
        temp = (int)CompleteG.basicCost;
    }

    MST[j].leastCostPaths();

    for(int p = 0; p < CompleteG.N; p++)
    {
        if(MST[j].c[p][p] != 0)
        {
            circle = true;
            r = p;
        }
    }

    while(circle)
    {
        circle = false;
        FF = false;
        recoder[m] = r;
        rmn = MN;

        for(int tt = 0; tt < CompleteG.N; tt++)
        {
            if(MUV[tt] == r) FF = true;
        }
    }

```

```

if(!FF){MUV[MN] = r;MN++;}

q = MST[j].path[r][r];
while(q != r)
{
    M = false;
    for(int gg = 0; gg < CompleteG.N; gg++)
    {
        if(q == MUV[gg])
        {
            M = true;
        }
    }

    if(!M){MUV[MN] = q;MN++;}

    m++;
    recoder[m] = q;
    q = MST[j].path[q][r];
}

temp1 = (int)CompleteG.basicCost;

for(int w = 0; w <= m; w++)
    for(int e = 0; e < CompleteG.N; e++)
    {
        inter = false;
        inter1 = false;
        inter2 = false;
        if(temp1 >

CompleteG.c[e][recoder[w]])

        {
            for(int p = 0; p <= m;

p++)

            {
                if(recoder[p]

== e)

                {
                    inter = true;

                }
            }
        }
    }

```

```

rmn; pp++)
    == e)
        inter1 = true;
    == recoder[w])
        inter2 = true;

inter2)))
(int)CompleteG.c[e][recoder[w]];
recoder[w];

for(int pp = 0; pp <=
{
    if(MUV[pp]
    {
    }
    if(MUV[pp]
    {
    }
}
if(!(inter||(inter1 &&
{
    temp1 =
    x1 = e;
    x2 =
}
}
}

for(int ii = 0; ii < CompleteG.N; ii++)
    for(int jj = 0; jj < CompleteG.N; jj++)
    {
        if(MST[j].arcs[ii][jj] == 1 &&
        {
            MST[j+1].addArc(MST[j].label[ii][jj].toString(), ii, jj, MST[j].path[ii][jj]);
        }
        if(MST[j].arcs[ii][jj] == 1&&jj
        {

```



```

MST[j+1].addArc(CompleteG.label[x1][jj].toString(), x1, jj, temp1);
        }

    }

    j++;
    MST[j].leastCostPaths();

    for(int pp =0; pp < CompleteG.N; pp++)
    {
        if(MST[j].c[pp][pp] != 0)
        {
            circle = true;
            r = pp;
        }
    }
    m = 0;
    //MN++;

}
MN = 0;
cnumber = j;

}

}

//////////MST end
//////////UNZIP end
void UNZIP()
{
    int q;
    int jj = 0;
    int z = 0;
    part1 = new int[CG.N*CG.N];
    int def[][];
    String Strpart1[];

    def = new int[N][N];

    for(int bb = 0; bb < CG.N*CG.N; bb++)
        part1[bb] = -1;

    if(cnumber != -1)
    {
        for(int i = 0; i < CompleteG.N; i++)

```

```

for(int j = 0; j < CompleteG.N; j++)
{
    if(MST[cnumber].arcs[i][j] == 1)
    {
        q = knumber + i;
        part1[vvv] = q;
        vvv++;
        while( knumber+j != q)
        {
            q =
            part1[vvv] = q;
            vvv++;
        }
        part1[vvv] = -2;
        vvv++;
    }
}

CG.path[q][knumber+j];

Final = new GPP(N);
OpenFinal = new GPP(N+1);
OpenbestGraph = new GPP(N+1);
Strpart1 = new String[CG.N*CG.N];
for(int Q = 0; Q < CG.N*CG.N; Q++)
    Strpart1[Q] = "-1";

while((part1[z+1] != -2 || part1[z+2] != -1) && z+1 <= part1.length)
{
    if(part1[z+1] == -2) z = z+2;
    Strpart1[jj] = CG.cheapestLabel[part1[z]][part1[z+1]];
    z++;
    jj++;
}

for(int i=0; i < selectedArcs.length; i++)
    for (int j =0; j < label.length; j++)
        for (int k = 0; k < label[j].length;
k++)

        if(label[j][k] != null)
        {
            for(int m = 0; m <
label[j][k].size(); m++)
            {

```

```
(selectedArcs[i].equals(label[j][k].elementAt(m).toString()))
```

```
OpenFinal.addArc(selectedArcs[i], j, k, 1); ////1 de wen ti!
```

```
def[j][k] = 1;
```

```
if(label[j][k]!= null)
{
```

```

                                if
(Strpart1[i].equals(label[j][k].elementAt(m).toString()) && def[j][k] != 1)
                                {

```

```
OpenFinal.addArc(Strpart1[i], j, k, 1); ////1 de wen ti!
```

```

        if(cnumber == -1)
        {
            Final = new GPP(N);
            OpenFinal = new GPP(N+1);
            OpenbestGraph = new GPP(N+1);
            for(int i=0; i < selectedArcs.length; i++)
                for (int j =0; j < label.length;
j++)
                    for (int k = 0; k <
label[j].length; k++)

                        if(label[j][k]!= null)
                        {
                            for(int m = 0;
m < label[j][k].size(); m++)
                                {
                                    if
                                    {
                                        Final.
                                        Openl
                                        def[j]
                                    }
                                }
                        }
                    }
                }
            }

```

```

        Final.leastCostPaths();
        OpenFinal.leastCostPaths();
        FBC = Final.basicCost;
    }

```

//////////UNZIP end

```

////////// leastCostPath
void leastCostPaths()
{
    for( int k = 0; k < N; k++ )
        for( int i = 0; i < N; i++ )
            if( defined[i][k] )
                for( int j = 0; j < N; j++ )
                    if( defined[k][j]
                        && (!defined[i][j] || c[i][j] >
c[i][k]+c[k][j]) )
                    {
                        path[i][j] = path[i][k];
                        c[i][j] = c[i][k]+c[k][j];
                        defined[i][j] = true;
                        if( i == j && c[i][j] < 0 ) return;
                    }

// stop on negative cycle

}

////////// end leastCostPath
////////// find Unbalanced vertetice
void findUnbalanced()
{
    int nn = 0, np = 0; // number of vertices of negative/positive
delta
                                for( int i = 0; i < N; i++ )
                                    if( delta[i] < 0 ) nn++;
                                    else if( delta[i] > 0 ) np++;

                                neg = new int[nn];
                                pos = new int[np];
                                nn = np = 0;
                                for( int i = 0; i < N; i++ ) // initialise sets
                                    if( delta[i] < 0 ) neg[nn++] = i;
                                    else if( delta[i] > 0 ) pos[np++] = i;
                                }
////////// end findUnbalanceed

//////////find a Feasible way
void findFeasible()
{
    // delete next 3 lines to be faster, but non-reentrant
    int delta[] = new int[N];
    for( int i = 0; i < N; i++ )
        delta[i] = this.delta[i];
}

```

```

        for( int u = 0; u < neg.length; u++ )
        {
            int i = neg[u];
            for( int v = 0; v < pos.length; v++ )
            {
                int j = pos[v];
                f[i][j] = -delta[i] < delta[j]? -delta[i]: delta[j];
                delta[i] += f[i][j];
                delta[j] -= f[i][j];
            }
        }
    }

//////////end findFeasible

//////////improvement
boolean improvements()
{
    int t = 0;
    GPP residual = new GPP(N);
    for( int u = 0; u < Final.neg.length; u++ )
    {
        int i = Final.neg[u];
        for( int v = 0; v < Final.pos.length; v++ )
        {
            int j = Final.pos[v];
            residual.addArc(null, i, j, (int)c[i][j]);
            if( Final.f[i][j] != 0 )
                residual.addArc(null, j, i, -(int)c[i][j]);
        }
    }

    residual.leastCostPaths(); // find a negative cycle

    for( int i = 0; i < N; i++ )
        if( residual.c[i][i] < 0 ) // cancel the cycle (if any)
        {
            int k = 0, u, v;
            boolean kunset = true;
            u = i; do // find k to cancel
            {
                v = residual.path[u][i];
                if( residual.c[u][v] < 0 && (kunset || k >
Final.f[v][u]) )
                {
                    k = Final.f[v][u];
                    kunset = false;
                }
            } while( (u = v) != i );
            u = i; do // cancel k along the cycle
            {
                v = residual.path[u][i];

```

```

        if( residual.c[u][v] < 0 ) Final.f[v][u] -= k;
        else Final.f[u][v] += k;
    } while( (u = v) != i );
    return true; // have another go
}

return false; // no improvements found
}

//////////end improvement

//////////improvement
boolean Openimprovements(GPP x)
{
    int t = 0;
    GPP residual = new GPP(N);
    for( int u = 0; u < x.neg.length; u++ )
    {
        int i = x.neg[u];
        for( int v = 0; v < x.pos.length; v++ )
        {
            int j = x.pos[v];
            residual.addArc(null, i, j, (int)c[i][j]);
            if( x.f[i][j] != 0 )
                residual.addArc(null, j, i, -(int)c[i][j]);
        }
    }

    residual.leastCostPaths(); // find a negative cycle

    for( int i = 0; i < N; i++ )
        if( residual.c[i][i] < 0 ) // cancel the cycle (if any)
        {
            int k = 0, u, v;
            boolean kunset = true;
            u = i; do // find k to cancel
            {
                v = residual.path[u][i];
                if( residual.c[u][v] < 0 && (kunset || k >
x.f[v][u]) )
                {
                    k = x.f[v][u];
                    kunset = false;
                }
            } while( (u = v) != i );
            u = i; do // cancel k along the cycle
            {
                v = residual.path[u][i];

```

```

        if( residual.c[u][v] < 0 ) x.f[v][u] -= k;
        else x.f[u][v] += k;
    } while( (u = v) != i );
    return true; // have another go
}

    return false; // no improvements found
}

//////////end improvement

//////////findpath
int findPath(int from, int f[][]) // find a path between unbalanced vertices
{
    for( int i = 0; i < N; i++ )
        if( f[from][i] > 0 ) return i;
    return NONE;
}

//////////End findpath

//////////CPT
void printCPT(int startVertex)
{
    int v = startVertex;
    // delete next 7 lines to be faster, but non-reentrant
    int arcs[][] = new int[N][N];
    int f[][] = new int[N][N];
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
        {
            arcs[i][j] = this.arcs[i][j];
            f[i][j] = this.f[i][j];
        }

    while( true )
    {
        int u = v;
        if( (v = findPath(u, f)) != NONE )
        {
            f[u][v]--; // remove path
            for( int p; u != v; u = p ) // break down path into its
arcs

```



```

//1
"+cheapestLabel[u][p]
//1
{
    p = path[u][v];
    System.out.println("Take arc
        + " from "+u+" to "+p);
}
}
else
{
    int bridgeVertex = path[u][startVertex];
    if( arcs[u][bridgeVertex] == 0 )
        break; // finished if bridge already used
    v = bridgeVertex;
    for( int i = 0; i < N; i++ ) // find an unused arc,
using bridge last
        if( i != bridgeVertex && arcs[u][i] > 0 )
        {
            v = i;
            break;
        }
    arcs[u][v]--; // decrement count of parallel arcs
    System.out.println("Take arc
        + " from "+u+" to "+v); // use each arc
//1
"+label[u][v].elementAt(arcs[u][v])
//1
label in turn
}
}
}

//////////end CPT

////////// Be a Euler graph
void EG()
{
    for(int i = 0; i < N; i++)
        for(int j = 0; j < N; j++)
        {
            if(Final.f[i][j] > 0)
            {
                //Final.addArc(cheapestLabel[i][j], i, j,
(int)c[i][j]); //1 de wen ti!
                for(int s = Final.f[i][j]; s > 0; s--)
                {
                    Final.f[i][j]--;
                    int q = i;
                    Final.addArc(cheapestLabel[i][path[i][j]], i, path[i][j], (int)c[i][path[i][j]]);
                    q = path[q][j];
                }
            }
        }
}

```



```
//////////End OpenEG
```

```
//////////OpenEG1
```

```
void OpenEG1(GPP x)
```

```
{
```

```
    for(int i = 0; i < N; i++)
```

```
        for(int j = 0; j < N; j++)
```

```
        {
```

```
            if(x.f[i][j] > 0)
```

```
            {
```

```
                for(int s = x.f[i][j]; s > 0; s--)
```

```
                {
```

```
                    x.f[i][j]--;
```

```
                    int q = i;
```

```
                    x.addArc(cheapestLabel[i][path[i][j]], i,
```

```
path[i][j], (int)c[i][path[i][j]]);
```

```
                    q = path[q][j];
```

```
                    while(q != j)
```

```
                    {
```

```
                        x.addArc(cheapestLabel[q][path[q][j]], q, path[q][j], (int)c[q][path[q][j]]); //1 de wen ti!
```

```
                        q = path[q][j];
```

```
                    }
```

```
                }
```

```
            }
```

```
        }
```

```
        x.leastCostPaths();
```

```
    }
```

```
//////////OpenEG1
```

```

//////////CPP
void CPP()
{
    //leastCostPaths();
    // checkValid();
    Final.findUnbalanced();
    Final.findFeasible();
    while( improvements() );
    tcost = (int)cost();
    EG();
    int r = -1;
    r = VChange[0];
    if(r != -1)
        Final.printCPT(r);
}

//////////end cpp

//////////OpenCPP
void OpenCPP()
{
    //leastCostPaths();
    // checkValid();
    OpenFinal.findUnbalanced();
    OpenFinal.findFeasible();
    while( improvements() );
    EG();
    OpenFinal.printCPT(0);
}

//////////end openCPP

float cost()
{
    return FBC+phi();
}

float phi()
{
    float phi = 0;
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
            phi += c[i][j]*Final.f[i][j];
    return phi;
}

float Opencost(GPP x)
{
    return FBC+Openphi(x);
}

```

```

    }

float Openphi(GPP y)
{
    float phi = 0;
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
            phi += c[i][j]*y.f[i][j];
    return phi;
}

float OpenbestGraphphi()
{
    float phi = 0;
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
            phi +=
c[i][j]*OpenbestGraph.f[i][j];
    return phi;
}

void checkValid()
{
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
            if(!defined[i][j]) throw new Error("Graph is not
strongly connected");
            if(c[i][i] < 0)      throw new Error("Graph has a negative
cycle");
    }
}

void solve(int check)
{
    //
    getselectedArcs();
    if(check == 1)
    {
    }
}

```

```

        selectedGraph();
        CC();
        CG();
        CompleteG();
        MST();
        UNZIP();
        CPP();
    }

    void result()
    {
        for(int i = 0; i < N; i++)
        {
            for(int j = 0; j < N; j++)
            {
                //1          System.out.print(arcs[i][j]);

            }
            //1          System.out.println("");
        }

        for(int u = 0; u < selectedArcs.length; u++)
        {
            //1          System.out.print(" " +selectedArcs[u]+"

        );
        }
        //1          System.out.println("");
    }
}

```

```

float OpenprintCPT(int startVertex)
{
    GPP OpenbestGraph = null;
    int bestCost = 0, cost;
    int i = 0;
    int k = 0;
    int ng[];
    ng = new int[N];

    do

```

```

{
    GPP g = new GPP(N+1);
    GPP Templ = new GPP(N+1);
    for(int ii = 0; ii < N; ii++)
        for(int j = 0; j < N; j++)
        {
            if(OpenFinal.arcs[ii][j] > 0)
                for(int s = OpenFinal.arcs[ii][j]; s > 0; s--)
                {

g.addArc(OpenFinal.label[ii][j].elementAt(s-1).toString(), ii, j, 1); ///1 de wen ti!

Templ.addArc(OpenFinal.label[ii][j].elementAt(s-1).toString(), ii, j, 1); ///1 de
wen ti!

                }
            }
        }
    g.leastCostPaths();
    Templ.leastCostPaths();

    int b = 0;
    cost = (int)FBC;
    g.findUnbalanced();
    Templ.findUnbalanced();
    OpenFinal.findUnbalanced();

    for(int kk = 0; kk < g.neg.length; kk++)
        ng[kk] = g.neg[kk];

    g.addArc("virtual start", N, startVertex, cost);
    Templ.addArc("virtual start", N, startVertex, cost);

    if (k == 0)
    {
        g.addArc("virtual end",
//
graph is Eulerian if neg.length=0

        g.neg.length == 0? startVertex: g.neg[0], N, (int)cost);

        Templ.addArc("virtual end",
//
graph is Eulerian if neg.length=0

```

```

Templ.neg.length == 0? startVertex: Templ.neg[0], N, (int)cost);

        }
        if (k > 0)
        {

g.addArc("virtual end",

Templ.addArc("virtual end",

        }

        g.findUnbalanced();
        Templ.findUnbalanced();
        g.findFeasible();
        Templ.findFeasible();
        g.leastCostPaths();
        Templ.leastCostPaths();
        while( Openimprovements(g) );
        while( Openimprovements(Templ) );
        OpenEG(g);
        OpenEG1(Templ);

        k = k+1;
        if( OpenbestGraph == null || bestCost >
Opencost(g) )
        {
                bestCost = (int)Opencost(g);
                OpenbestGraph = Templ;
        }

        i++;
    }
    while(i < OpenFinal.neg.length);

//1
    System.out.println("Open CPT from "+startVertex+" (ignore virtual
arcs)");
    //System.out.println(N);

```



```

OpenbestGraph.leastCostPaths();
OpenbestGraph.printCPT(N);
return bestCost;

```

```

}

```

```

int OpenGPP()
{
    int besti = 0;
    float bestCost = 0;
    // for( int i = 0; i < N; i++ )
    // {
    //     System.out.println("Solve from " + i);
    //     float c = OpenprintCPT(0);
    //1    System.out.println("Cost = " + c);
    //     if( i == 0 || c < bestCost )
    //     {
    //         bestCost = c;
    //         besti = 0;
    //     }
    // }
    // OpenbestGraph.printCPT(0);

    //1    System.out.println("the best way: solve from " + besti);
    //1    System.out.println("bestcost = " + bestCost);
    return (int)bestCost;
}

```

```

//////////original constructed graph
GPP OCG(int vertices, int Arcs)
{
    GPP g = new GPP(vertices);
    g.G = new OpenCP(vertices);
    g.overarcs = false;
    Random r1 = new Random();
}

```

```

int N = vertices;
int[][] define;
int v1;
int v2;
int a = 0;

define = new int[N][N];
for(int ab = 0; ab < N; ab++)
    for(int ba = 0; ba < N; ba++)
    {
        define[ab][ba] = -1;
        g.NameNo[ab][ba] = 0;
    }

boolean unfinished = true;

while(unfinished || a < Arcs)
{
    v1 = r1.nextInt(N);
    v2 = r1.nextInt(N);
    if(!g.defined[v1][v2] && v1 != v2) // add arcs to a
        g.addArc(String.valueOf(v1)+String.valueOf(v2)+String.valueOf(g.NameNo[v1][v2]), v1, v2, 1);
    //g.G.addArc(String.valueOf(v1)+String.valueOf(v2)+String.valueOf(g.NameNo[v1][v2]), v1, v2, 1);
    //g.G.addArc(String.valueOf(v1)+String.valueOf(v2), v1, v2, 1);
    define[v1][v2] = 1;
    g.NameNo[v1][v2]++; // for naming.
    a++;
    System.out.println(a);
}
unfinished = false;
g.leastCostPaths();
for(int i = 0; i < N; i++)
{
    for(int j = 0; j < N; j++)

        if(!g.defined[i][j]) unfinished = true;
}

```

```

    }

    if(v1 != v2 && !unfinished && a < Arcs)// after
constructing DCG,adding the rest arcs.

    {

        g.addArc(String.valueOf(v1)+String.valueOf(v2)+String.valueOf(g.NameNo[v
1][v2]), v1, v2, 1);

        g.G.addArc(String.valueOf(v1)+String.valueOf(v2)+String.valueOf(g.NameNo
[v1][v2]), v1, v2, 1);

        define[v1][v2] = 1;

        g.NameNo[v1][v2]++;// for naming.

        a++;

        //1

        System.out.println(a);

    }

    g.leastCostPaths();

    if( a > Arcs)
    {
        g.overarcs = true;
    }

}
/*
    g.addArc("01", 0, 1, 1).addArc("02", 0, 2, 1).addArc("03", 0,
3, 1).addArc("10", 1, 0, 1).addArc("12", 1, 2, 1).addArc("14", 1, 4, 1).addArc("21", 2, 1,
1).addArc("34", 3, 4, 1).addArc("42", 4, 2, 1).addArc("43", 4, 3, 1);
    g.G.addArc("01", 0, 1, 1).addArc("02", 0, 2, 1).addArc("03",
0, 3, 1).addArc("10", 1, 0, 1).addArc("12", 1, 2, 1).addArc("14", 1, 4, 1).addArc("21", 2,
1, 1).addArc("34", 3, 4, 1).addArc("42", 4, 2, 1).addArc("43", 4, 3, 1);

    g.leastCostPaths();
    g.overarcs = false;
    */
    return g;

}

```

//////////OCG end

```
GPP ROCG(int Vertices, int[][] A)
{
    int Aa[][];

    GPP g = new GPP(Vertices);
    g.G = new OpenCP(Vertices);
    Aa = new int [Vertices][Vertices];

    for (int o = 0; o < Vertices; o++)
        for(int p = 0; p < Vertices; p++)
        {
            Aa[o][p] = A[o][p];
        }

    for(int i= 0; i< Vertices; i++)
        for(int j= 0; j< Vertices; j++)
        {

            while(Aa[i][j]>0)
            {
                Aa[i][j]--;

                g.addArc(String.valueOf(i)+String.valueOf(j)+String.valueOf(Aa[i][j]), i, j, 1);
                g.G.addArc(String.valueOf(i)+String.valueOf(j)+String.valueOf(Aa[i][j]), i, j,
1);
            }

        }

    return g;
}
```

```

//////////MOCG
//////////original constructed graph
GPP MOCG(int vertices, int Arcs)
{
    GPP g = new GPP(vertices);
    g.G = new OpenCP(vertices);
    g.overarcs = false;
    Random r1=new Random();
    String QQ;
    int N = vertices;
    int[][] define;
    int v1;
    int v2;
    int a = 0;

    define = new int[N][N];
    for(int ab = 0; ab < N; ab++)
        for(int ba = 0; ba < N; ba++)
            define[ab][ba] = -1;

    boolean unfinished = true;

    while(unfinished || a < Arcs)
    {
        v1 = r1.nextInt(N);
        v2 = r1.nextInt(N);
        if(!g.defined[v1][v2] && v1 != v2)
        {
            QQ = String.valueOf(a);

            g.addArc(String.valueOf(v1)+String.valueOf(v2)+QQ, v1, v2, 1);
            g.G.addArc(String.valueOf(v1)+String.valueOf(v2)+QQ, v1, v2, 1);
        }
    }
}

```

```

//g.G.addArc(String.valueOf(v1)+String.valueOf(v2), v1, v2, 1);
                define[v1][v2] = 1;
                a++;
                //      System.out.println(a);
            }
            unfinished = false;
            g.leastCostPaths();
            for(int i = 0; i < N; i++)
            {
                                for(int j = 0; j < N; j++)
                                if(!g.defined[i][j])

unfinished = true;

                                }

                                if(v1 != v2 && !unfinished && a < Arcs)
                                {
                                        QQ =
String.valueOf(a);

                                g.addArc(String.valueOf(v1)+String.valueOf(v2)+QQ, v1, v2, 1);

                                g.G.addArc(String.valueOf(v1)+String.valueOf(v2)+QQ, v1, v2, 1);
                                define[v1][v2] = 1;
                                a++;
                                //      System.out.println(a);
                                }

                                g.leastCostPaths();

                                if( a > Arcs)
                                {
                                        g.overarcs = true;
                                }

                                }

                                /*
                                g.addArc("02", 0, 2, 1).addArc("03", 0, 3, 1).addArc("10", 1, 0,
1).addArc("12", 1, 2, 1).addArc("13", 1, 3, 1).addArc("21", 2, 1, 1).addArc("23", 2, 3,
1).addArc("24", 2, 4, 1).addArc("30", 3, 0, 1).addArc("31", 3, 1, 1).addArc("32", 3, 2,
1).addArc("40", 4, 0, 1).addArc("41", 4, 1, 1).addArc("42", 4, 2, 1).addArc("43", 4, 3, 1);
                                g.G.addArc("02", 0, 2, 1).addArc("03", 0, 3, 1).addArc("10", 1, 0,
1).addArc("12", 1, 2, 1).addArc("13", 1, 3, 1).addArc("21", 2, 1, 1).addArc("23", 2, 3,
1).addArc("24", 2, 4, 1).addArc("30", 3, 0, 1).addArc("31", 3, 1, 1).addArc("32", 3, 2,
1).addArc("40", 4, 0, 1).addArc("41", 4, 1, 1).addArc("42", 4, 2, 1).addArc("43", 4, 3, 1);

```

```

g.leastCostPaths();
g.overarcs = false;
*/
return g;

```

```

}

```

```

//////////MOCG end

```

```

//////////CPP functions

```

```

void CPPsolve()
{
    CPPleastCostPaths();
    CPPcheckValid();
    CPPfindUnbalanced();
    CPPfindFeasible();
    while( CPPimprovements() );
}

```

```

void CPPleastCostPaths()
{
    for( int k = 0; k < N; k++ )
        for( int i = 0; i < N; i++ )
            if( defined[i][k] )
                for( int j = 0; j < N; j++ )
                    if( defined[k][j]

```

```

c[i][j] > c[i][k]+c[k][j]) )
                                {
                                path[i][j] = path[i][k];
                                c[i][j] = c[i][k]+c[k][j];
                                defined[i][j] = true;
                                if( i == j && c[i][j] <
                                0 ) return; // stop on negative cycle
                                }

}

void CPPcheckValid()
{
    for(int i = 0; i < N; i++)
    {
        for(int j = 0; j < N; j++)
            if(!defined[i][j]) throw new
Error("Graph is not strongly connected");
            if(c[i][i] < 0)        throw new
Error("Graph has a negative cycle");
    }
    //add end
}

// </tex><tex file="cost.tex">
float CPPcost(int cost)
{
    return cost+CPPphi();
}

float CPPphi()
{
    float phi = 0;
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
            phi += c[i][j]*f[i][j];
    return phi;
}
//</tex><tex file="degrees.tex">

void CPPfindUnbalanced()
{
    int nn = 0, np = 0; // number of vertices of negative/positive delta

    for( int i = 0; i < N; i++ )

```



```

        if( delta[i] < 0 ) nn++;
        else if( delta[i] > 0 ) np++;

neg = new int[nn];
pos = new int[np];
nn = np = 0;
for( int i = 0; i < N; i++ ) // initialise sets
    if( delta[i] < 0 ) neg[nn++] = i;
    else if( delta[i] > 0 ) pos[np++] = i;
    }
//</tex><tex file="greedy.tex">

void CPPfindFeasible()
{
    // delete next 3 lines to be faster, but non-reentrant
    int delta[] = new int[N];
    for( int i = 0; i < N; i++ )
        delta[i] = this.delta[i];

    for( int u = 0; u < neg.length; u++ )
    {
        int i = neg[u];
        for( int v = 0; v < pos.length; v++ )
        {
            int j = pos[v];
            f[i][j] = -delta[i] < delta[j]? -delta[i]: delta[j];
            delta[i] += f[i][j];
            delta[j] -= f[i][j];
        }
    }
}

// </tex><tex file="iterate.tex">
boolean CPPimprovements()
{
    GPP residual = new GPP(N);
    for( int u = 0; u < neg.length; u++ )
    {
        int i = neg[u];
        for( int v = 0; v < pos.length; v++ )
        {
            int j = pos[v];
            residual.addArc(null, i, j, (int)c[i][j]);
            if( f[i][j] != 0 ) residual.addArc(null, j, i, -
(int)c[i][j]);
        }
    }
    residual.leastCostPaths(); // find a negative cycle
    for( int i = 0; i < N; i++ )
        if( residual.c[i][i] < 0 ) // cancel the cycle (if any)
        {
            int k = 0, u, v;
            boolean kunset = true;

```

```

        u = i; do // find k to cancel
        {
            v = residual.path[u][i];
            if( residual.c[u][v] < 0 && (kunset || k >
f[v][u]) )
                {
                    k = f[v][u];
                    kunset = false;
                }
        } while( (u = v) != i );
        u = i; do // cancel k along the cycle
        {
            v = residual.path[u][i];
            if( residual.c[u][v] < 0 ) f[v][u] -= k;
            else f[u][v] += k;
        } while( (u = v) != i );
        return true; // have another go
    }
    return false; // no improvements found
}

// </tex><tex file="printCPT.tex">
static final int NONE = -1; // anything < 0

int CPPfindPath(int from, int f[][]) // find a path between unbalanced vertices
{
    for( int i = 0; i < N; i++ )
        if( f[from][i] > 0 ) return i;
    return NONE;
}

void CPPprintCPT(int startVertex)
{
    int v = startVertex;

    // delete next 7 lines to be faster, but non-reentrant
    int arcs[][] = new int[N][N];
    int f[][] = new int[N][N];
    for( int i = 0; i < N; i++ )
        for( int j = 0; j < N; j++ )
            {
                arcs[i][j] = this.arcs[i][j];
                f[i][j] = this.f[i][j];
            }

    while( true )
    {
        int u = v;
        if( (v = findPath(u, f)) != NONE )
            {
                f[u][v]--; // remove path
                for( int p; u != v; u = p ) // break down path into its
arcs
                    {
                        p = path[u][v];

```

```

//1
"+cheapestLabel[u][p]
//1
}
else
{
    int bridgeVertex = path[u][startVertex];
    if( arcs[u][bridgeVertex] == 0 )
        break; // finished if bridge already used
    v = bridgeVertex;
    for( int i = 0; i < N; i++ ) // find an unused arc,
        if( i != bridgeVertex && arcs[u][i] > 0 )
        {
            v = i;
            break;
        }
    arcs[u][v]--; // decrement count of parallel arcs
    System.out.println("Take arc
        "+" from "+u+" to "+v);
}
}
}

using bridge last

//1
"+label[u][v].elementAt(arcs[u][v])
//1
label in turn
}
}
}

```

```

//////////////////////////////////CPPfunctions end
//////////////////////////////////main
static public void main(String args[])
{

    int[] presetup_Arcs;
    int[] presetup_Vertices;
    int[] presetup_Subsets;
    int[] presetup_number;
    int[] presetup_times;
    int presetupofnumber = 1; //number of trying

    presetup_Arcs = new int[presetupofnumber];
    presetup_Vertices = new int[presetupofnumber];
    presetup_Subsets = new int[presetupofnumber];
    presetup_number = new int[presetupofnumber];
    presetup_times = new int[presetupofnumber];

    //definitaion the data for running;
    presetup_Arcs[0] = 40;
    presetup_Vertices[0] = 10;
    presetup_Subsets[0] = 20;
    presetup_number[0] = 20;
    presetup_times[0] = 100;

    /*
    presetup_Arcs[1] = 20;
    presetup_Vertices[1] = 6;
    presetup_Subsets[1] = 5;
    presetup_number[1] = 20;
    presetup_times[1] = 10;

    presetup_Arcs[2] = 20;
    presetup_Vertices[2] = 8;
    presetup_Subsets[2] = 5;
    presetup_number[2] = 20;
    presetup_times[2] = 10;

    presetup_Arcs[3] = 20;
    presetup_Vertices[3] = 10;
    presetup_Subsets[3] = 5;
    presetup_number[3] = 20;
    presetup_times[3] = 10;

```

```
presetup_Arcs[4] = 20;  
presetup_Vertices[4] = 12;  
presetup_Subsets[4] = 5;  
presetup_number[4] = 20;  
presetup_times[4] = 10;
```

```
presetup_Arcs[5] = 20;  
presetup_Vertices[5] = 14;  
presetup_Subsets[5] = 5;  
presetup_number[5] = 20;  
presetup_times[5] = 10;
```

```
presetup_Arcs[6] = 20;  
presetup_Vertices[6] = 16;  
presetup_Subsets[6] = 5;  
presetup_number[6] = 20;  
presetup_times[6] = 10;
```

```
presetup_Arcs[6] = 30;  
presetup_Vertices[6] = 24;  
presetup_Subsets[6] = 5;  
presetup_number[6] = 30;  
presetup_times[6] = 10;
```

```
presetup_Arcs[7] = 20;  
presetup_Vertices[7] = 10;  
presetup_Subsets[7] = 5;  
presetup_number[7] = 20;  
presetup_times[7] = 10;
```

```
presetup_Arcs[8] = 22;  
presetup_Vertices[8] = 10;  
presetup_Subsets[8] = 5;  
presetup_number[8] = 22;  
presetup_times[8] = 10;
```

```
presetup_Arcs[9] = 24;  
presetup_Vertices[9] = 10;  
presetup_Subsets[9] = 5;  
presetup_number[9] = 24;  
presetup_times[9] = 10;
```

```
presetup_Arcs[10] = 26;  
presetup_Vertices[10] = 10;  
presetup_Subsets[10] = 5;  
presetup_number[10] = 26;  
presetup_times[10] = 10;
```

```
presetup_Arcs[11] = 28;  
presetup_Vertices[11] = 10;  
presetup_Subsets[11] = 5;  
presetup_number[11] = 28;  
presetup_times[11] = 10;
```

```
presetup_Arcs[12] = 30;  
presetup_Vertices[12] = 10;  
presetup_Subsets[12] = 5;  
presetup_number[12] = 30;  
presetup_times[12] = 10;
```

```
presetup_Arcs[13] = 32;  
presetup_Vertices[13] = 10;  
presetup_Subsets[13] = 5;  
presetup_number[13] = 32;  
presetup_times[13] = 10;
```

```
presetup_Arcs[14] = 34;  
presetup_Vertices[14] = 10;  
presetup_Subsets[14] = 5;  
presetup_number[14] = 34;  
presetup_times[14] = 10;
```

```
presetup_Arcs[15] = 10;  
presetup_Vertices[15] = 20;  
presetup_Subsets[15] = 5;  
presetup_number[15] = 20;  
presetup_times[15] = 10;
```

```
presetup_Arcs[16] = 10;  
presetup_Vertices[16] = 20;  
presetup_Subsets[16] = 7;
```

```

presetup_number[16] = 20;
presetup_times[16] = 10;

presetup_Arcs[17] = 10;
presetup_Vertices[17] = 20;
presetup_Subsets[17] = 9;
presetup_number[17] = 20;
presetup_times[17] = 10;

presetup_Arcs[18] = 10;
presetup_Vertices[18] = 20;
presetup_Subsets[18] = 11;
presetup_number[18] = 20;
presetup_times[18] = 10;

presetup_Arcs[19] = 10;
presetup_Vertices[19] = 20;
presetup_Subsets[19] = 13;
presetup_number[19] = 20;
presetup_times[19] = 10;

presetup_Arcs[19] = 10;
presetup_Vertices[19] = 20;
presetup_Subsets[19] = 15;
presetup_number[19] = 20;
presetup_times[19] = 10;

presetup_Arcs[20] = 10;
presetup_Vertices[20] = 20;
presetup_Subsets[20] = 17;
presetup_number[20] = 20;
presetup_times[20] = 10;

presetup_Arcs[21] = 10;
presetup_Vertices[21] = 20;
presetup_Subsets[21] = 19;
presetup_number[21] = 20;
presetup_times[21] = 10;

presetup_Arcs[22] = 10;
presetup_Vertices[22] = 20;
presetup_Subsets[22] = 20;
presetup_number[22] = 20;
presetup_times[22] = 10;
*/
for(int v = 0; v < presetupofnumber; v++)

```

```

{

    long OpenCPP = 0;
    long OpenGPP = 0;
    long TOpenCPP = 0;
    long TOpenGPP = 0;
    double result = 0;

    System.out.println("No." + v);
    System.out.println("Arcs =" + presetup_Arcs[v] + "
Vertices =" + presetup_Vertices[v] + " Subsets =" + presetup_Subsets[v]);

    for(int utral = 0; utral < presetup_times[v]; utral++)
    {

        int tem;
        int ln = 1;
        int bests=1000000;
        String[] SS;
        int Arcs = presetup_Arcs[v];
        int Vertices = presetup_Vertices[v];
        int Subsets = presetup_Subsets[v];
        int Number = presetup_number[v]; // the arcs
        which does not belong to subsets.

        int A[][];
        String SA[];

        SS = new String[Arcs-Number+Subsets]; //the arcs

        in RPP

        A = new int[Vertices][Vertices];
        SA = new String[Number];
        GPP PP = new GPP(Vertices);
        PP.overarcs = true;
        PP.timecost("start");
        int timetempl = PP.timestart;

        while(PP.overarcs)
        {
            PP = PP.OCG(Vertices,Arcs); //consturct

            a fit graph.

        }
    }
}

```



```

for(int a = 0; a < Vertices; a++)
    for(int b = 0; b < Vertices; b++)
    {
        A[a][b] = PP.arcs[a][b];
    }
PP.getArc1(Number, Subsets, Vertices, A);

```

```

//1
System.out.println("hh =" + hh);
SA =
PP.getArc2(Number, Subsets, Vertices, Arcs, PP.AA);
GPP GG = new GPP(Vertices);

GG = GG.ROCG(Vertices, A);
GG.selectedArcs = SA;
GG.overarcs = PP.overarcs;

if(!GG.overarcs)
{
    GG.leastCostPaths();
    GG.checkValid();
    // GG.leastCostPaths();
    GG.result();
    GG.solve(1);
    GG.OpenGPP();

    tem = GG.OpenGPP();
    if(tem < bests)
    {
        SS = SA;
        bests = tem;
    }
}
GG.timecost("end");
GG.timecost = GG.timeend - timetempl;
TOpenGPP = GG.timecost + TOpenGPP;

GPP QQ = new GPP(Vertices);

```

```

QQ = QQ.ROCG(Vertices,A);
QQ.selectedArcs = SS;

QQ.leastCostPaths();
QQ.checkValid();
QQ.leastCostPaths();
//1 System.out.println("");
//1 System.out.println("");
//1 System.out.println("");
//1
System.out.println("*****");
//1 System.out.println("the bestone of
OpenGPP:");

QQ.result();
QQ.solve(1);

//1 OpenGPP = QQ.OpenGPP()+OpenGPP;
System.out.println("//////////OpenCPP");
//QQ.timecost("start");
OpenCPP = OpenCPP + QQ.G.test();
//QQ.timecost("end");
TOpenCPP = QQ.timecost + TOpenCPP;

}

System.out.println("Final result!");
result =(float)OpenGPP/(float)OpenCPP;

System.out.println("OpenGPP:"+OpenGPP+", "+"OpenCPP:"+OpenCPP);
System.out.println(result);
float Tresult
=(float)TOpenGPP/(float)TOpenCPP;

System.out.println("TOpenGPP:"+TOpenGPP+", "+"TOpenCPP:"+TOpenCPP);
System.out.println(Tresult);
//1 System.out.print("succeed!");

}

}

```

```
}
```

```
class OpenCP
{
    class Arc
    {
        String lab; int u, v; float cost;
        Arc(String lab, int u, int v, float cost)
        {
            this.lab = lab;
            this.u = u;
            this.v = v;
            this.cost = cost;
        }
    }
    Vector arcs = new Vector();
    int N;

    OpenCP(int vertices)
    {
        N = vertices;
    }

    OpenCP addArc(String lab, int u, int v, float cost)
    {
        if( cost < 0 ) throw new Error("Graph has negative costs");
        arcs.addElement(new Arc(lab, u, v, cost));
        return this;
    }

    float printCPT(int startVertex)
    {
        GPP bestGraph = null, g, templ = null;
        float bestCost = 0, cost;
        int i = 0;
        int k = 0;
        int ng[];
        int length;
        ng = new int[N];
        do
        {
            int b = 0;
            g = new GPP(N+1);
            for( int j = 0; j < arcs.size(); j++ )
            {
                Arc it = (Arc) arcs.elementAt(j);
                g.addArc(it.lab, it.u, it.v, (int)it.cost);
            }
            cost = g.basicCost;
            g.CPPfindUnbalanced(); // initialise g.neg on original graph
        }
    }
}
```

```

length = g.neg.length;
for(int kk = 0; kk < g.neg.length; kk++)
    ng[kk] = g.neg[kk];

g.addArc("virtual start", N, startVertex, (int)cost);

if (k == 0)
{
    g.addArc("virtual end",
// graph is
Eulerian if neg.length=0
== 0? startVertex: g.neg[0], N, (int)cost);
}
if (k > 0)
{
    g.addArc("virtual
end",
// graph is
Eulerian if neg.length=0
== 0? startVertex: ng[i], N, (int)cost);
}

g.CPPsolve();
k = k+1;
if( bestGraph == null || bestCost > g.CPPcost((int)cost) )
{
    bestCost = g.CPPcost((int)cost);
    bestGraph = g;
}
} while( ++i < length );

//1
System.out.println("Open CPT from "+startVertex+" (ignore virtual
arcs)");
bestGraph.CPPprintCPT(N);
return cost+bestGraph.CPPphi();
}
//</tex>

int test()
{

```

```

// create a graph of four vertices
// add the arcs for the example graph

int besti = 0;
float bestCost = 0;
// for( int i = 0; i < N; i++ )
// {
//1      System.out.println("Solve from "+0);
//      float c = this.printCPT(0);
//1      System.out.println("Cost = "+c);
//      if( i == 0 || c < bestCost )
//      {
//          bestCost = c;
//          besti = 0;
//      }
// }

//1      System.out.println("// <tex file=\"open.tex\">");
//      this.printCPT(besti);

//1      System.out.println("Cost = "+bestCost);
//1      System.out.println("// </tex>");
//      return (int) bestCost;

    }
// <tex file="open.tex">

}

```

VITA AUCTORIS

Name: Wang, Xiao

Place of Birth: Chengdu, China

Year of Birth: 1974

Education: University of Windsor, Windsor, Ontario, Canada

2002-2006 M.Sc. in Computer Science

Tsinghua University, Beijing, China

B.Eng. in Computer Science and Engineering

Working Experience:

Software Developer, IBM , Southbury CT, USA, 2006 ~ Present

Software Developer, Iconomics Inc, Toronto, Canada, 2004~2006

Software Developer, JCS Ltd. Yasu, Japan, 1998~2002