

University of Windsor

Scholarship at UWindor

Electronic Theses and Dissertations

Theses, Dissertations, and Major Papers

2009

Improved web page traverse using genetic algorithm

Man Li

University of Windsor

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

Recommended Citation

Li, Man, "Improved web page traverse using genetic algorithm" (2009). *Electronic Theses and Dissertations*. 8076.

<https://scholar.uwindsor.ca/etd/8076>

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



Improved Web Page Traverse Using Genetic Algorithm

by

Man Li

A Thesis

Submitted to the Faculty of Graduate Studies

through Computer Science

in Partial Fulfillment of the Requirements for

the Degree of Master of Applied Science at the

University of Windsor

Windsor, Ontario, Canada

2009

© 2009 Man Li



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: 978-0-494-57565-9
Our file *Notre référence*
ISBN: 978-0-494-57565-9

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.

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

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

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.

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

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


Canada

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Correct navigational behavior of a web application is essential to its reliability. An effective means to improving our confidence in the correct behavior of a web application is to test it by exploring the possible navigation among the web pages at the client side: The tester carries out the testing by consecutively clicking the hyperlinks along with some possible search parameters and checking whether the returned web pages are as expected. Traditional conformance testing techniques based on graph can be adopted in this setting to automatically generate suitable test sequences to traverse among client pages. In this thesis, we present an improvement on T-method for test sequence generation to reduce considerably its length by making use of a genetic algorithm. Our experiments show a 34%-68% saving on the test sequence lengths compared to the direct application of T-method.

DEDICATION

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Jessica Chen, for her great ideas, invaluable guidance and enthusiastic encouragement. Without her help, the work presented here would not have been possible. Sincere thanks to Dr. Xiaobu Yuan, Dr. Kevin W. Li and Dr. Dan Wu for their precious time and ardent assistance in the development of this thesis.

Additional thanks to Dr. Lihua Duan for her contributions and assistance.

Finally, special thanks to my parents Zhigang Li and Xiangli Hu, my husband Kui jiao. Thank you for putting up with my difficult times and for understanding and endless supporting me while I complete one of my dreams.

TABLE OF CONTENTS

AUTHOR'S DECLARATION OF ORIGINALITY.....	iii
ABSTRACT.....	iv
DEDICATION.....	v
ACKNOWLEDGEMENTS.....	vi
LIST OF TABLES.....	x
LIST OF FIGURES.....	xii
1. INTRODUCTION AND PROBLEM DESCRIPTION.....	1
2. REVIEW OF LITERATURE AND BACKGROUND.....	10
2.1 RELATED WORK.....	10
2.1.1 WEB APPLICATION VERIFICATION MODELS.....	11
2.1.2 TESTING MODELS ON WEB APPLICATION.....	11
2.1.3 SUMMARY.....	14
2.2 BACKGROUND.....	15
2.2.1 URL.....	16
2.2.2 SERVER PAGE, STATIC PAGE AND CLIENT PAGE.....	17
2.2.3 SESSION AND COOKIE.....	18
2.2.4 HYPERLINK AND DYNAMIC HYPERLINK.....	19
2.2.5 THE PROPERTIES OF A CLIENT PAGE.....	19

2.2.6 ARCHITECTURE OF WEB APPLICATION.....	21
2.2.7 HISTORY STACK.....	22
2.2.8 USING BACK BUTTON.....	22
3. MODELING WEB APPLICATIONS WITH GRAPHS	26
4. IMPLEMENTATION	41
4.1 THE IMPLEMENTATION OF THE CPP ALGORITHM.....	41
4.2 THE IMPLEMENTATION OF THE GENETIC ALGORITHM.....	43
4.2.1 DESCRIPTION OF GA.....	43
4.2.2 CHROMOSOME AND GENOME.....	46
4.2.3 ENCODING.....	46
4.2.4 CROSSOVER.....	47
4.2.5 MUTATION	49
4.2.6 FITNESS FUNCTION	52
4.2.7 THE ALGORITHM.....	52
5. METHOD EVALUATION	58
5.1 The NUMBER OF BACK EDGES.....	59
5.2 SIZE OF G	60
5.3 DIAMETER OF G	62
5.4 RATIO OF COOKIE-AFFECTING TRANSTION IN G	64
5.5 SPECIAL CASE.....	66
6. CONCLUSIONS AND RECOMMENDATIONS	69
APPENDICES	71
REFERENCES	73

VITA AUCTORIS..... 77

LIST OF TABLES

Table 3.1 explanations of the symbols.....	35
Table 4.1 GA microbiological and model terminology	45

LIST OF FIGURES

Figure 1.1 the website of the eBay	4
Figure 1.2 the graph of part of the website of eBay	8
Figure 2.1 an example of a client page.....	20
Figure 2.2 the structure of web applications.....	21
Figure 2.3 history stack operation.....	23
Figure 3.1 some client page of the website of YahooTravel.....	30
Figure 3.2 an example graph for the cheap airfare search engine.....	34
Figure 3.3 an example to show how to use the back button.....	39
Figure 4.1 an example of the CPP.....	43
Figure 4.2 block diagram of basic GA cycle.....	44
Figure 4.3 crossover rule.....	49
Figure 4.4 mutation rule.....	50
Figure 4.5 (a) no mutually exclusion.....	51
Figure 4.5 (b) mutually exclusion.....	51
Figure 5.1 an illustration of the changes of the lengths of the test sequences according to the number of back edges of G	60
Figure 5.2 (a) the changes of the lengths of the test sequences according to the increase of the size of G	61
Figure 5.2 (b) the time changes with the lengths of the test sequences according to the increase of the size of G	61
Figure 5.3 (a) the changes of the lengths of the test sequences according to the increase of the diameter of G	63

Figure 5.3 (b) the time changes with the lengths of the test sequences according to the increase of the diameter of G63

Figure 5.4 (a) the change of lengths of the test sequences according to the ratio of the privilege transitions.....64

Figure 5.4 (b) the time changes according to the ratio of the privilege transitions.....65

Figure 5.5.1 a special case of the changes of the lengths of the test sequences according to the increase of the size of G67

Figure 5.5.2 a special case of the changes of the lengths of the test sequences according to the increase of the diameter of G68

CHAPTER I

INTRODUCTION AND PROBLEM DESCRIPTION

With the rapid advancement of networking and web technology, more and more information is being posted into and retrieved from the Web. Web systems are becoming the primary device for information sharing and retrieval. Some of them now play so important roles that they have been applied to all major areas. An exiguous mistake in an online store system may put thousands of people around the world in trouble and lead a business company to millions of dollars of income loss. Assuring the web system quality and reliability has become one of our major concerns. The quality of features and characteristics of a product or services determines its ability to meet stated or implied needs, and reliability is a measure of success on how the observed behavior of a system conforms to its specifications. They both directly affect the quality of our daily life.

The existing rigorous software engineering methods and techniques can greatly improve the quality of the software product. As we place very high expectations on software engineering methods and techniques to this important application domain to achieve better effectiveness and higher efficiency, we have great interests in enhanced techniques for reliability control of web applications. As software testing has been well recognized as an essential part of software development techniques to gain confidence in the reliability of the software product, we discuss issues in conducting software testing on web applications.

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

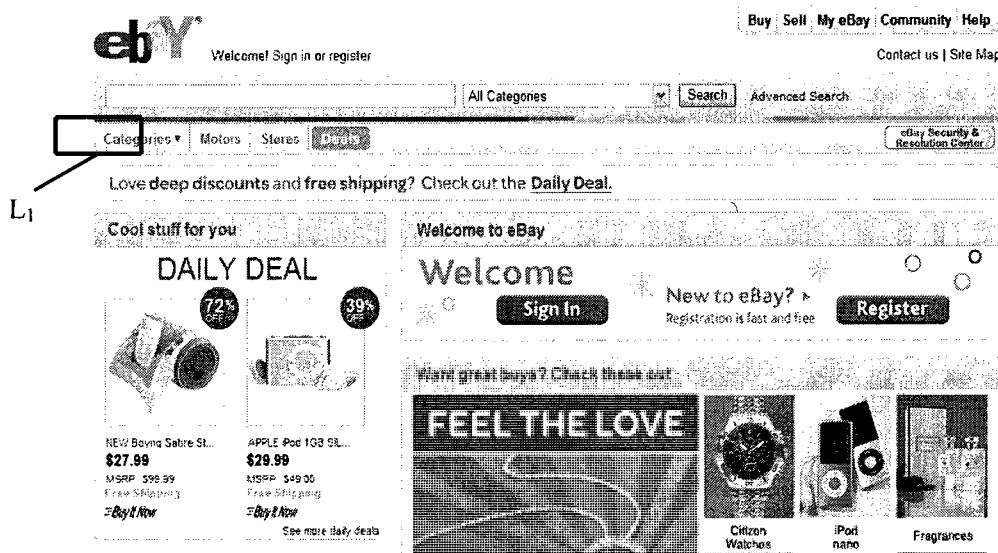
We also assume that the *expected* navigational behavior of the IUT is available in terms of a deterministic Finite State Machine (FSM) [10]. An FSM is a 5-tuple $M = (I, O, S, \delta, \lambda)$, where I , O and S are finite and nonempty sets of input symbols, output symbols and states, respectively. $\delta : S \times I \rightarrow S$ is the state transition function and $\lambda : S \times I \rightarrow O$ is the output function. Here we use each state to represent a client page and each transition to represent a possible transformation from one client page to another. The action of triggering a hyperlink, possibly together with some search parameters, is expressed as the input of the transition, while the characterization of the client page

generated by the server page associated with the triggered hyperlink forms the output of the transition.

Problem description: A test sequence is generated from the specification FSM. In general, we apply a 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 example, 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) [9] to generate a test sequence. The CPP is to find a least-cost walk to travel all edges in a directed graph.

We use part of the website of eBay as a simple example to show how to model a web application in terms of directed graph. People can enter the home pages of all categories from the homepage of the eBay to find out more detailed information about each category. Figure 1.1(a) shows the homepage P_1 of the website that provides hyperlinks to all home pages of the main parts of the website where L_1 is the hyperlink to the web page

for the list of the books. Figure 1.1(b) shows web page P_2 that lists the hyperlinks to categories where hyperlinks L_2 , L_3 and L_4 refer to the homepages of antiques genus, collectibles genus and eBay separately. Figure 1.1(c) shows web page P_3 which is the home page of the antiques genus. Here hyperlinks L_5 and L_6 refer to the home page of the eBay and the web page that lists the hyperlinks to all the categories respectively. Figure 1.1(d) shows web page P_4 which is the home page of the collectibles genus where hyperlinks L_7 and L_8 link to the home page of the eBay and the web page that lists the hyperlinks to all the categories respectively.



(a)

ebay L4 Welcome! Sign in or register

Search Advanced Search Buy Sell My eBay Community

Categories Motors Stores Deals eBay Security & Resolution Center

Home > Buy > All Categories

All Categories

Search

Search titles & descriptions

Browse Categories

Category: All Categories | Format: All Items | Listings: All Active | Location: Available on eBay.com | Show

Show number of items in category Show category numbers

<p>Antiques (212471) L2</p> <ul style="list-style-type: none"> Antiquities (6120) Architectural & Garden (14135) Asian Antiques (29497) Books & Manuscripts (3230) Decorative Arts (33047) Ethnographic (4772) Furniture (14184) Home & Hearth (651) Linens & Textiles (Pre-1930) (8713) Maps, Atlases & Globes (10366) 	<p>Collectibles (2089264) L3</p> <ul style="list-style-type: none"> Advertising (139007) Animals (97493) Animation Art & Characters (92533) Arcade, Jukeboxes & Pinball (10010) Autographs (7796) Banks, Registers & Vending (5773) Bankware (10013) Beads (1340) Bottles & Insulators (14779) Brevetiana, Best (34008) 	<p>Jewelry & Watches (1830474)</p> <ul style="list-style-type: none"> Children's Jewelry (11257) Designer Brands (37623) Engagement & Wedding (74013) Ethnic, Regional & Tribal (39180) Fashion Jewelry (736951) Fine Jewelry (98765) Handcrafted, Artisan Jewelry (68396) Jewelry Boxes, Cases & Display (22796) Jewelry Design & Repair (54615) Loose Beads (160652)
--	--	---

(b)

ebay L5 Welcome! Sign in or register

Buy Sell My eBay Community Help Site Map

Antiques Search Advanced Search

Categories Motors Stores Deals eBay Security & Resolution Center

Home > Buy > Antiques

Antiques

Opt out of the new search experience

Categories within Antiques

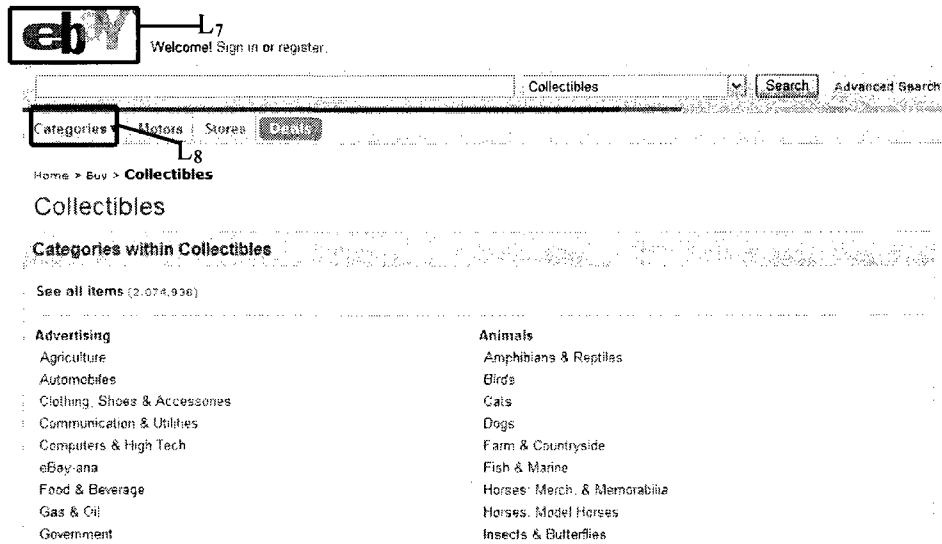
See all items (209,871)

<p>Antiquities</p> <ul style="list-style-type: none"> The Americas Byzantine Celtic Egyptian Far Eastern Greek Holy Land Islamic Near Eastern 	<p>Architectural & Garden</p> <ul style="list-style-type: none"> Balusters Barn Doors Beams Coiling Tins Chandeliers, Fixtures, Sconces Columns & Posts Corbels Doors Finials
---	---

Picks ending soon!

ANTIQUES

(c)



(d)

Figure 1. 1 the website of the eBay

Figure 1.2 shows the directed graph of the above selected part of the website. Web page P_1 , P_2 , P_3 and P_4 are represented by vertexes V_1 , V_2 , V_3 and V_4 respectively. The transitions t_1 , t_2 , t_3 , t_4 , t_5 , t_6 , t_7 and t_8 represent eight different navigations in the web site triggered by clicking on hyperlinks L_1 , L_2 , L_3 , L_4 , L_5 , L_6 , L_7 and L_8 in the vertexes V_1 , V_2 , V_3 and V_4 respectively.

There are two major tasks in graph-based conformance testing: one is to generate an input/output sequence, called *test sequence*, that is both effective in terms of fault detestability and efficiency in terms of its length; the other is to apply automatically the generated input sequence to the implementation and compare the output sequence with

the expected one. Here we focus on the former problem of generating an efficient test sequence which represents a traversal among client side web pages.

There are several methods proposed in the literature for test sequence generation, typically known as T-method [23], U-method [1, 22, 28], W-method [7] and D-method [11, 14, 33]. For testing web navigation, however, they all converge to the simplest one: the T-method, with the test criterion to require that each transition in the given FSM be traversed at least once in the generated path of the test sequence.

In the above example, a generated test sequence is $i_1/o_1, i_3/o_3, i_7/o_7, i_1/o_1, i_3/o_3, i_4/o_4, i_5/o_5, i_8/o_8, i_1/o_1, i_5/o_5, i_6/o_6, i_2/o_2$, and its length is 12.

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

In the above example, suppose that we take into consideration the behavior of the backward button provided by the browser. A test sequence could be $i_1/o_1, i_3/o_3, i_7/o_7, \textit{back}, i_4/o_4, i_5/o_5, i_8/o_8, \textit{back}, i_6/o_6, i_2/o_2$, whose length is 10.

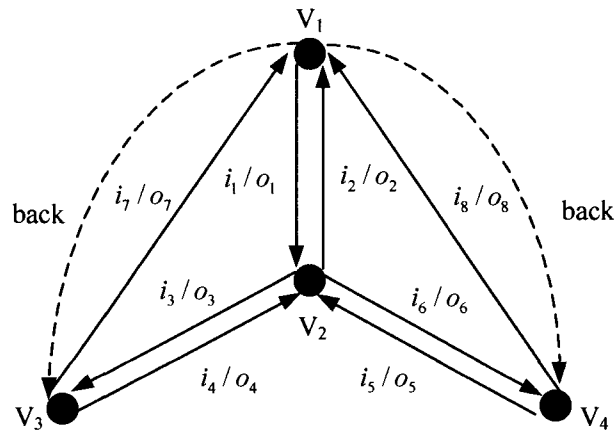


Figure 1. 2 the graph of part of the website of eBay

The above example is illustrative so the reduction on the test sequence length is not significant. In reality, if we can make use of the behavior of backward button and forward button to generate a test sequence, the test sequence could be reduced significantly. We apply genetic algorithm to provide a solution of generating efficient test sequences.

In [8], a network-flow algorithm was applied to generate efficient test sequences using back button. Sessions and cookies however are not allowed in the applications. In the present work we do consider sessions and cookies and their effect on the algorithm. In the special cases when sessions and cookies are not present, we have also carried out experiment to compare the lengths of the generated test sequence by using our proposed algorithm and that of [8].

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

The rest of the thesis is organized as follows. We first introduce some notational background related to web applications in Chapter 2. Then we show how to model web applications in terms of graphs in Chapter 3. We discuss the major issues in the implementation of the existing algorithms related to our work in Chapter 4. This is followed by our experimental results in Chapter 5 which demonstrates the significant reduction of test sequence length generated by our proposed method. Related work is then discussed in Chapter 6 and we give some concluding remarks at the end.

CHAPTER II

REVIEW OF LITERATURE AND BACKGROUND

2.1 RELATED WORK

In the last decade, there are various types of generic testing techniques explored. According to the testing methodology, there are *white-box* testing, *black-box* testing etc. Most of them can be used for testing web applications with proper adaptation. Some general discussions can be found in [19].

For *white-box testing*, a typical approach is to conduct data flow analysis on the source code of the server pages and generate proper test suites satisfying certain test selection criteria. A comprehensive survey on various test selection criteria on data flow testing proposed in the literature can be found in [36], and how to apply data flow analysis to testing web applications is discussed in [18].

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

2.1.1 WEB APPLICATION VERIFICATION MODELS

Sciascio et al. [29, 30] presents how to verify a web application with Symbolic Model Verifier (NuSMV) and Computation tree logic (CTL). NuSMV is an updated version of the SMV symbolic model checker. CTL is a branching-time temporal logic. The model of a web application is a web graph, which consists of nodes and arcs. In a web graph, nodes include pages, links, and windows, and the arcs connect the nodes. The browsing from one page to another following a hyperlink includes at least three nodes and two arcs. The first arc connects the start page to one of its hyperlinks, and the second connects the hyperlink to the destination page. All the requirement properties are written in CTL formulas. A symbolic model verifier, NuSMV, is applied after the web application model and requirement properties are ready. The final results and counter examples are reported. In order to make it easy to use model checking tool, a series of patterns are developed.

2.1.2 TESTING MODELS ON WEB APPLICATION

When the expected behavior is unknown, we can first retrieve the design or test cases from the implementation via reverse engineering techniques. Ricca and Tonella [26, 27] proposed testing strategy on web site analysis through web browser. A tool called ReWeb is developed to gather all web pages and their relations within one web site, and a UML-based model for the web site is constructed by this ReWeb. A testing tool, TestWeb, is responsible for testing the UML web application model. The authors considered white-box testing on a web application. The testing is mainly concentrated on web forms of a web application. 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. VeriWeb is designed to explore client pages while performing testing at the same time. The derived test cases are then *reused* for *regression testing*.

Kung, Liu and Hsia [16] use an ORD (Object Relation Diagram) based web application testing model (WTM) for testing web applications. A WTM model of a web application is created by reverse engineering on the source documents it. The testing work is divided into three parts: object perspective, behavior perspective and structure perspective. Each part is tested separately. The object perspective of the WTM describes the class structures of a web application including request, response, navigation and redirection. The behavior perspective of the WTM focuses on page navigation, and page navigation diagram (PND) derived from ORD is employed. Finally a navigation tree started from the home page is constructed and the testing of the navigation behavior is based on this PND. The structure perspective of the WTM is related to control flow and data flow information of a web application. Block branch diagram (BBD) and function cluster diagrams (FCD) are used respectively for describing control flow and data flow.

Lucca and Penta [20] proposed a base-line testing strategy that creates a testing model by adding browser statechart to a series of pages with inter-related hyperlinks. [20] considered the influence of web browser's behaviors on a web application. Each web browser contains buttons like back, forward and reload. A user's click on one of these

buttons can force the browser to display the previously visited page or refresh the current web page with the same URL. In order to test a web application with browser's behavior, a statechart of back and forward buttons is constructed with four states, BDFD (Back Disable, Forward Disable), BEFD (Back Enable, Forward Disable), BEFE (Back Enable, Forward Enable) and BDFE (Back Disable, Forward Enable). For each navigation path, e.g. a base line, the testing model is a navigation tree generated by adding the statechart of browser's behavior. The root of the tree is the home page of the web application, and each path of the tree is tested separately.

Graunke et al. adopted λ -Calculus to model web form related web applications in [12]. Each web application in this model is divided into a single server and a single client. The server contains a table that maps the requested URL to a process program. A client consists of a current form web page, and all previously visited web pages. Each form contains variables and the URL that the form data will be sent to. A set of rules is defined that regulates the transitions from one page to another.

Beek and Mauw [3] used labelled transitions systems to model web applications and the conformance testing is applied to this model. An MRRTS (Multi Request-Response Transition Systems), in which request is considered as input and response as output, is used. With this model, the navigation behaviour of a web application are modeled as URL label series.

2.1.3 SUMMARY

When the expected behavior of a web application is available in terms of graph, our major task is to generate a test sequence from it. Due to the problem of limited resource to carry out software testing, much of our effort has been put on reducing the length of the generated test sequence. This can be accomplished in two ways: one is to divide-and-conquer, and another is to search for optimal solution for the minimal-length test sequence. Discussion on the former issue can be found in [2], where the authors proposed a hierarchical approach to model and test potentially large web applications. Our approach to checking the correct navigational behavior of a web application falls into the latter category.

Of course, the specification of the expected behavior of a web application is not restricted to FSMs: it can take any format according to various test purposes. Chang and Hon [4] proposed a mechanism to generate test suit based on *statistical usage model* of the web applications for link validation. An agent-based framework to automatically generate and coordinate test agents proposed by Qi, Kung, and Wong [25] is based on *Belief-Desire-Intention model*. Lee and Offutt [17, 35] used mutation testing techniques to generate test cases from *XML-based documents*.

Most of the test procedure can be carried out either on the server side or on the client side. For the latter, there must be a web browser involved. A web browser introduces additional user interface such as the *back* and *forward* buttons, *the URL address bar*. This interface brings out additional difficulties in formal verification and testing [6] because it

complicates the original model of the web applications. At the same time, we will show in this thesis, that this new feature can also be well used to achieve better solution towards test sequence reduction.

Lucca and Penta [20] considered that the browser behavior has influences on the navigation behavior of a web application. According to our analysis on web browser, although Lucca and Penta proposed a testing method considering browser behavior, the browser model they used is not sufficient to describe the real browsers we use. It is necessary to consider the browser cache and history stack on testing a web application.

Duan and Wang [8] presented a network flow algorithm to solve the NP-hard problem of generating a minimal-length test sequence of hyperlink validation by making use of the *back* button provided by the web browsers. The present thesis work also applies *back* button feature of web browsers to reduce the length of the generated test sequence. While [8] does not consider the presence of cookies and sessions, the present work does. Thus, we are providing a more general solution.

2.2 BACKGROUND

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

2.2.1 URL

URL stands for Universal Resource Locator. It is used to locate any resources in the Internet. URL typically consists of a URL scheme, an authority, and a path. An URL scheme indicates a category of resources, such as `http`, `https`, and `ftp`. The authority typically consists of the name or IP address of a server and the port number. A path is to show the relative path of the resource inside the host server in the Internet. The format of an HTTP URL is often parameterized as: `http://<host>: <port>/<path>? <request parameters>`. Here “`http://`” indicates the resource type is `http`, and the communication protocol used to send this request is HTTP. `<host>` is the web server’s IP address. It identifies a unique web server in the Internet address. `<port>` is the port number that the web server uses for HTTP communication. `<path>` specifies the relative storage position of the request resource. `<request parameters>` consists of request parameters a client passes to the web server. The request parameters can come from the input of the client and the cookies/sessions. These parameters could be used to compute and generate a client page. For example, <http://www.baidu.com/search?ie=gb2312> shows the basic elements of a URL. “`http://`” indicates the resource type is HTTP. “`www.baidu.com`” is the domain name and its corresponding IP address can be resolved by querying a DNS (Domain Name Service) server. The port number is implicit and it is 80 by default. “`search`” is a relative directory and “`ie=gb2312`” is the assignment of a request parameter.

2.2.2 SERVER PAGE, STATIC PAGE AND CLIENT PAGE

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

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

2.2.3 SESSION AND COOKIE

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

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

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

2.2.4 HYPERLINK AND DYNAMIC HYPERLINK

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

According to the different information of cookies/sessions, database or request parameters, a hyperlink could be dynamically shown on a client page. Such a hyperlink is called a *dynamic hyperlink*. For example, in the online student information system of the 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 non-registered student to see such a hyperlink after his or her login.

2.2.5 THE PROPERTIES OF A CLIENT PAGE

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

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

Figure 2.1 shows an example of a client page that is generated by a single server page. It is the homepage of the student webmail system of the 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 the current page has not yet been given.

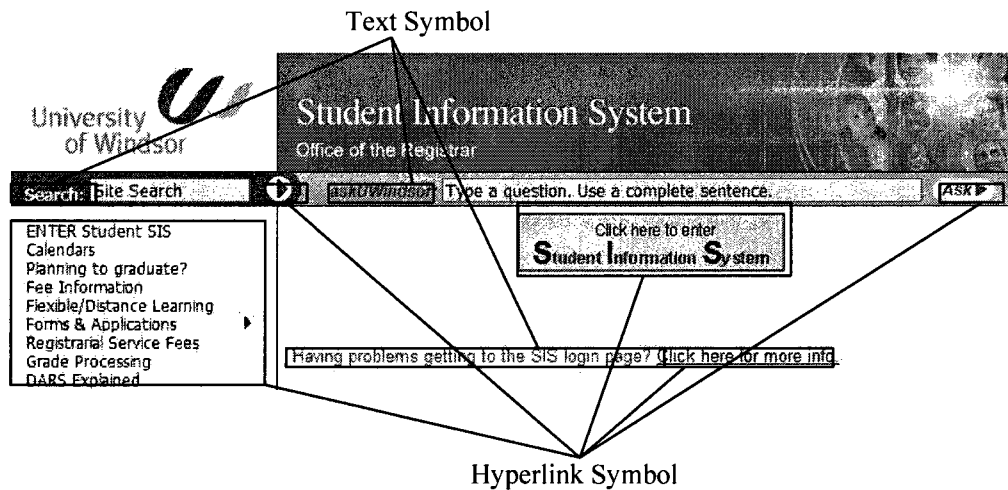


Figure 2.1 an example of a client page

2.2.6 ARCHITECTURE OF WEB APPLICATION

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

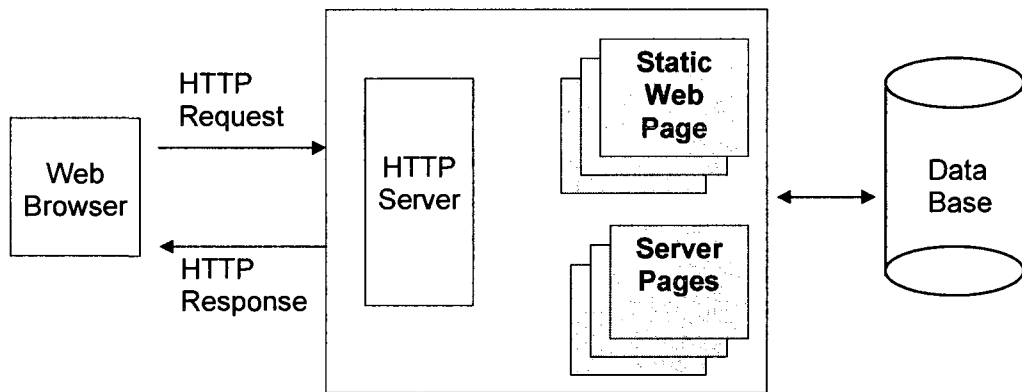


Figure 2.2 the structure of web applications

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

2.2.7 HISTORY STACK

A browser's history stack [11] stores the previously visited URLs, and the stack is maintained by the browser module. Generally speaking, the history stack is a kind of stack with similar operations as normal stacks. A history stack maintains a stack pointer, top position and bottom position. The values of these variables determine whether a backward or forward button can be enabled. Compared with normal stacks, the stack pointer of the browser's history stack can be moved back and forth if more than one item is stored in it.

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

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

from local cache or the web server that sent the page before. The use of local cache depends on the cache policy set in the browser or HTTP cache controls that came with the received web page.

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

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

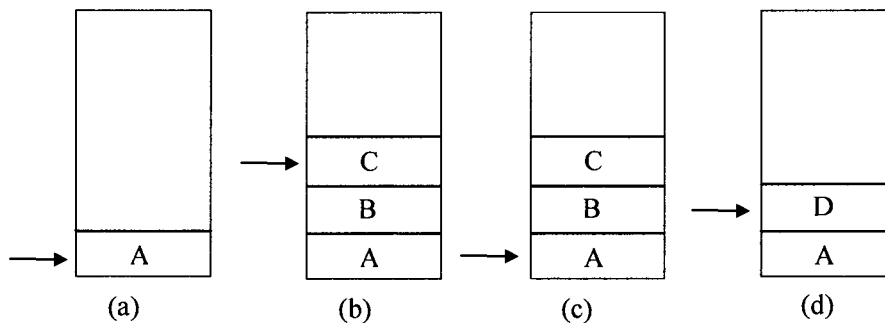


Figure 2.3 history stack operation

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

2.2.8 USING BACK BUTTON

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

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

The history stack only stores the URLs of previously visited web pages, so it is possible that a different web page, rather than the previously visited one, is visited when we select

to *back* a web page in the history stack due to the existence of the cookies. We say a test sequence χ uses the *back* button *properly* if it conforms to the semantics of the web browsers.

CHAPTER III

MODELING WEB APPLICATIONS WITH GRAPHS

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

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

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

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

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

We consider a *closed* system in the sense that the hyperlinks all refer to the server pages within the same web application. An *open* system can be modeled in our setting by augmenting an additional page to represent all the Internet web pages not generated by the *web application under test*. If there is a hyperlink in a client page pointing to www.uwindsor.ca, for example, we can simply represent it as a hyperlink pointing to this special page.

We can use an *abstract representation* for a group of similar pages. For example, in a *student information system*, when student *A* and student *B* both have regular transcripts, a client page displaying the transcript of *A* and *B* both have regular transcripts. In this case, a client page displaying the transcript of *A* and a client page displaying the transcript of *B*

can be considered as members of a same group with a single representative client page. A label of a graph is an input/output pair in our context.

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

An output y represents the characteristics of a client page. For testing purpose, it can be understood as a multi-set of texts displayed and a multi-set of hyperlinks. The text can be normal text, images, some places for users to type in input data, the text portion of a hyperlink, or other non-hyperlink content. A hyperlink can be a normal hyperlink uniquely identified by its context and the server page it refers to. A client page is *correct* if the multi-set of texts and the multi-set of hyperlinks are all correct. Here we do not consider the test oracle problem on how to automatically check the correctness of various types of web content against the expected one. Readers interested in this issue are referred to [5, 21, 24, 34].

A transition t is defined by a tuple $(v_i, v_j, x/y)$ in which v_i is the *starting vertex*, x is the input, v_j is the *ending vertex*, and y is the output. A transition $(v_i, v_j, x/y)$ represents the

transformation from client page cp_i represented by vertex v_i to client page cp_j represented by vertex v_j triggered by user's input x .

Let us consider the cheap airfare search engine of the online system YahooTravel.com (<http://travel.yahoo.com>) as an example web application. Here, we only focus on the major functionalities of this search engine. In home page cp_0 , the user can issue a search by providing search criteria such as the date, departure city, destination city, and so on. If no such flight is available, a client page with information "Our providers do not have any flights for your search criteria" will be returned, and we use cp_1 to denote this client page. Otherwise, a client page with the detailed information about all the available flights will be displayed in cp_2 . When the information for the search criteria is incomplete (e.g., no destination city is entered) or infeasible (e.g., the return date is before the departure date for a round trip), an error page cp_3 will be returned. In client page cp_2 , the user can select any available flight for reservation by clicking the corresponding hyperlink. Then the personal information is required from the user in a new client page cp_4 . Again, when the keyed-in personal information is incomplete or erroneous (e.g., no last name is entered or telephone number contains alphabet letter), client page cp_5 will be prompted to user for correction. Client page cp_6 is generated with all the personal and flight information for the user to review. In cp_6 , the user can choose "Continue", or "back", which will result in client page cp_7 (for request confirmation), and client page cp_4 , respectively. The "Contact us" page (denoted by cp_8) and the home page can be reached by special image hyperlinks from all the client pages. cp_0 and cp_3 are generated by the server page represented by v_0 , cp_1 and cp_2 are generated by server page represented by v_1 , cp_4 and cp_5 are generated by

server page represented by v_2 , cp_6 is generated by server page represented by v_3 , cp_7 is generated by server page represented by v_4 and cp_8 is generated by server page represented by v_5 . Figure 3.1 shows the web pages of the online system.

The screenshot shows the Yahoo! Travel website. At the top, there is a navigation bar with links for Home, Research, TRAVEL GUIDES, Book FARECHASE, Deals, and My Travel. Below this is a search bar labeled "SEARCH TRAVEL:". Underneath the search bar, there is a section titled "Find It Fast" with the text "Search here for cheap airline tickets, hotels, cars, cruises, and vacations". There are two radio buttons: "Classic search" and "Search multiple sites with FareChase". Below this is a search form with tabs for Flights, Hotels, Cars, Vacations, and Cruises. The "Flights" tab is selected. The form has fields for "FROM", "TO", "DEPART" (02/09/09), and "RETURN" (02/10/09). There are also dropdown menus for "ADULTS" (set to 1) and "CONNECTIONS" (set to Any). A "SEARCH" button is visible on the right side of the form.

(a) cp_0

The screenshot shows a message box with a warning icon. The text inside the box reads: "Our providers do not have any flights for your search criteria." Below this text is a link that says "Please try a new search".

(b) cp_1

Flights Hotels Cars

Windsor, ON to Toronto, ON
 Depart: Fri 02/13/09 Return: Fri 02/27/09 Round-trip, 1 Adult

Modify Search

Refine results

Flight Times from

OUTBOUND

before 9am \$961

9am - 3pm \$961

3pm - 9pm \$961

RETURN

before 9am \$961

9am - 3pm \$961

3pm - 9pm \$961

after 9pm \$1,283

Airlines from

check all | uncheck all

Air Canada \$961

Stops from

Non-stop \$961

Sort by: [Low price](#) | [Outbound times](#) | [Return times](#)

\$961 Round-trip **Air Canada** flight search

Flight: 6:00 am - 7:07 am Non-stop YQG-YYZ

Flight: 7:20 am - 8:29 am Non-stop YYZ-YQG

[air.ca.com](#) | [Book Now](#) | [Learn More](#)

[5 other return times](#) available on this page

\$961 Round-trip **Air Canada** flight search

Flight: 6:50 am - 9:57 am Non-stop YQG-YYZ

Flight: 7:20 am - 8:29 am Non-stop YYZ-YQG

[air.ca.com](#) | [Book Now](#) | [Learn More](#)

[4 other return times](#) available on this page

Sites Searched

(c) cp₂

Flights Hotels Cars Messages

Please fix the following error(s):

⚠ Please enter arrival city

Round-Trip One-Way

From - Find an airport To - Find an airport

Check nearby airports

Depart: 02/09/09 Anytime Return: 02/10/09 Anytime

Adult: 1 Child: 0 Senior: 0 Infant: 0

[More Search Options](#)
(Filter, Price range)

(d) cp₃

3 Who's traveling?

Each traveler's name must match the name on his/her government-issued photo ID. Airlines do not allow passengers to transfer tickets or to change names on tickets.

Traveler

	*First/given name	M.I.	*Last name/surname	Suffix
Adult	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

I am a resident of the European Union (EU). [Why not?](#)

We guard your privacy


Orbitz pledges to keep your personal information private and protected. [View our privacy policy.](#)

(e) cp₄

3 Who's traveling?

Each traveler's name must match the name on his/her government-issued photo ID. Airlines do not allow passengers to transfer tickets or to change names on tickets.

Traveler

 Please specify the last name of this traveler. (Message 1579)

	*First/given name	M.I.	*Last name/surname	Suffix
Adult	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>

I am a resident of the European Union (EU). [Why not?](#)

We guard your privacy

Orbitz pledges to keep your personal information private and protected. [View our privacy policy.](#)

(f) cp₅


Review trip cost

Total Trip Duration: 15 days / 14 nights

1 traveler, round-trip

Total Airfare	\$815.00
Taxes and Fees	\$146.40

Total trip cost \$961.40 USD

 **Price Assurance** Get an automatic cash refund if another customer books the same flights at a lower price.

Continue booking your flight only.

[Continue](#)

[Back](#)

(G) cp₆

Review fare rules, terms, and conditions

- This ticket is non-refundable.
- Changes to this ticket will incur [change fees](#).
- This is an international trip requiring special [travel documentation](#) for each traveler.
- See an overview of all the [rules and restrictions](#) applicable for this fare.
- View the complete Orbitz site [terms and conditions](#) updated January 27, 2009 which incorporate the Orbitz [privacy policy](#).
- Read the airline's [ticket terms and conditions](#).
- Read the [terms and conditions](#) for Ticket Protector Plus.

Please note: At least one airline in this trip charges an additional fee for checked baggage. This fee is not included in your total trip cost. [See details](#)

I agree and accept the Orbitz site [terms and conditions](#) updated January 27, 2009 which incorporate the Orbitz [privacy policy](#) and all other rules, terms and conditions referenced above.

[Agree and continue](#)

[Back](#)

(H) *cp₇*

Overview Investors Partnerships Careers Protect Planet Earth Press

Contact Us

Orbitz Worldwide, Inc.
500 W. Madison
Chicago, Illinois 60661
U.S.A.
<http://cs.re.orbitz.com>

Customer Service

The following email addresses and links to phone numbers will connect you with customer service information for Orbitz.com, CheapTickets.com, ebookers.com, RatestoGo.com and HotelClub.com.

- **Orbitz:** For more information on customer support, visit [Orbitz customer support](#)
- **CheapTickets:** For more information on customer support, visit [CheapTickets customer support](#)
- **ebookers:** For more information on customer support, visit [ebookers customer support](#)
- **HotelClub:** For more information on customer support, visit [HotelClub customer support](#)
- **RatestoGo:** For more information on customer support, visit [RatestoGo customer support](#)

Exchanging Paper Tickets (for Orbitz.com, CheapTickets.com only)

Orbitz/CheapTickets Ticketing Department
1961 Premier Drive
Suite 320
Mankato MN 56001

(I) *cp₈*

Figure 3. 1 some client pages of the website of YahooTravel

This part of the web application is specified as digraph G shown in Figure 3.2 and Table 3.1. Here, for $0 < i < 8$, vertex v_i represents cp_i . Since hyperlinks in different client pages or even within one client page can refer to the same client page, we use x_j^i to denote the input symbol that can produce y_i , where j is some integer. y_i denotes the characteristics of cp_i .

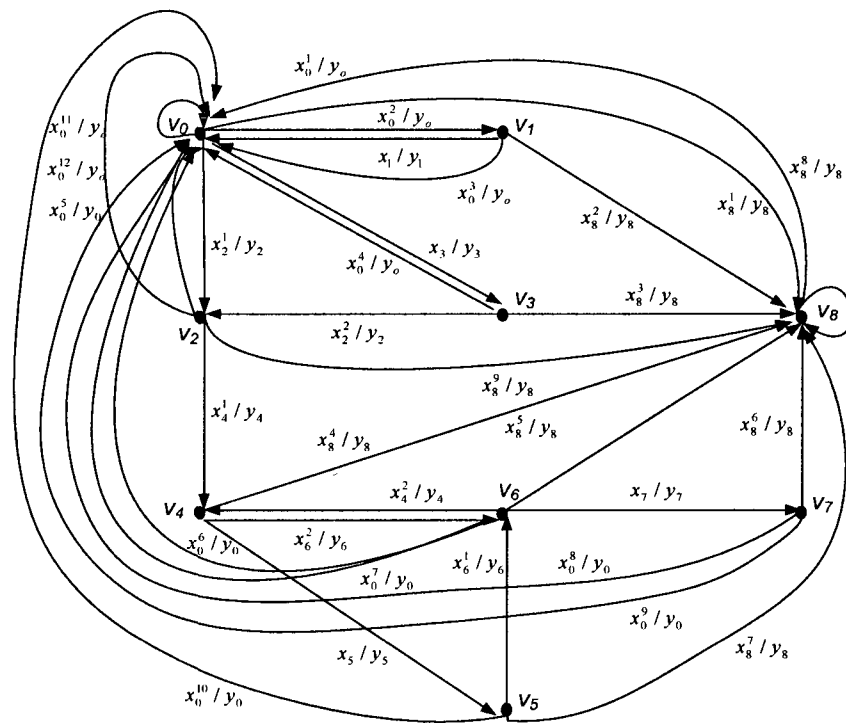


Figure 3.2 an example graph for the cheap airfare search engine

Symbol	Meaning	Symbol	Meaning
y_0	The characteristic of client page cp_0	y_1	The characteristic of client page cp_1
y_2	The characteristic of client page cp_2	y_3	The characteristic of client page cp_3
y_4	The characteristic of client page cp_4	y_5	The characteristic of client page cp_5
y_6	The characteristic of client page cp_6	y_7	The characteristic of client page cp_7
y_8	The characteristic of client page cp_8	v_0	Client page cp_0
v_1	Client page cp_1	v_2	Client page cp_2
v_3	Client page cp_3	v_4	Client page cp_4
v_5	Client page cp_5	v_6	Client page cp_6
v_7	Client page cp_7	v_8	Client page cp_8
x_0^1	Clicking the hyperlink to v_0 with <i>null</i> (search parameters) in cp_8	x_1	Clicking the hyperlink to v_1 with the no-flight-return search parameters
x_0^2	Clicking one hyperlink to v_0 with <i>null</i> (search parameters) in cp_1	x_2^1	Clicking the hyperlink to v_1 with correct search parameters
x_0^3	Clicking the other hyperlink to v_0 with <i>null</i> (search	x_2^2	Clicking the hyperlink to v_1 with correct search

	parameters) in cp_1		parameters
x_0^4	Clicking the hyperlink to v_0 with null (search parameters) in cp_3	x_3	Clicking the hyperlink to v_0 with incomplete search parameters
x_0^5	Clicking the hyperlink to v_0 with null (search parameters) in cp_2	x_4^1	Clicking one hyperlink to v_2 with flight information (search parameters)
x_0^6	Clicking one hyperlink to v_0 with null (search parameters) in cp_6	x_4^2	Clicking another hyperlink to v_2 with flight information (search parameters)
x_0^7	Clicking the other hyperlink to v_0 with null (search parameters) in cp_6	x_5	Clicking the hyperlink to v_2 with flight and customer's information (search parameters)
x_0^8	Clicking one hyperlink to v_0 with null (search parameters) in cp_7	x_6^1	Clicking the hyperlink to v_3 with flight and customer's information (search parameters)
x_0^9	Clicking the other hyperlink to v_0 with null (search parameters) in cp_8	x_6^2	Clicking the hyperlink to v_3 with flight and customer's information (search parameters)

x_0^{10}	Clicking the hyperlink to v_0 with null (search parameters) in cp_5	x_7	Clicking the hyperlink to v_4 with flight and customer's information (search parameters)
x_0^{11}	Clicking the hyperlink to v_0 with null (search parameters) in cp_0	x_8^1	Clicking the hyperlink to v_5 with flight and customer's information (search parameters)
x_0^{12}	Clicking the hyperlink to v_0 with null (search parameters) in cp_2	x_8^3	Clicking the hyperlink to v_5 with flight and customer's information (search parameters)
x_8^2	Clicking the hyperlink to v_5 with flight and customer information (search parameters)	x_8^5	Clicking the hyperlink to v_5 with flight and customer's information (search parameters)
x_8^4	Clicking the hyperlink to v_5 with flight and customer information (search parameters)	x_8^7	Clicking the hyperlink to v_5 with flight and customer's information (search parameters)
x_8^6	Clicking the hyperlink to v_5 with flight and customer information (search parameters)	x_8^8	Clicking the hyperlink to v_5 with flight and customer's information (search parameters)

	parameters)		parameters)
--	-------------	--	-------------

Table 3. 1 explanations of the symbols

Given a digraph describing the expected behavior of a *web application under test*, we would like to see, for each transition t , if we click on a hyperlink (possibly with some input data) corresponding to an input of the page represented by the starting vertex of t , then the resulting client page is correctly generated in the sense that both the displayed text content and hyperlinks are as characterized in the output of t . Furthermore, we would like to generate an input/output sequence (called a *test sequence*) from a sequence of transitions such that each transition in the graph is traversed at least once.

In the above example, suppose that the current navigation history is $cp_0 cp_2 cp_4 cp_6 cp_8$. Now we want to visit client page cp_7 . Without the *back button*, the shortest path to reach cp_7 is to visit the following client pages in sequence: $cp_0 cp_2 cp_4 cp_6 cp_7$. By making use of the *back button*, cp_7 can be reached by only two clicks: first click the *back button* to return to cp_6 and then click the hyperlink associated with cp_7 in cp_6 , and Figure 3.3 shows the shortest path (bold lines).

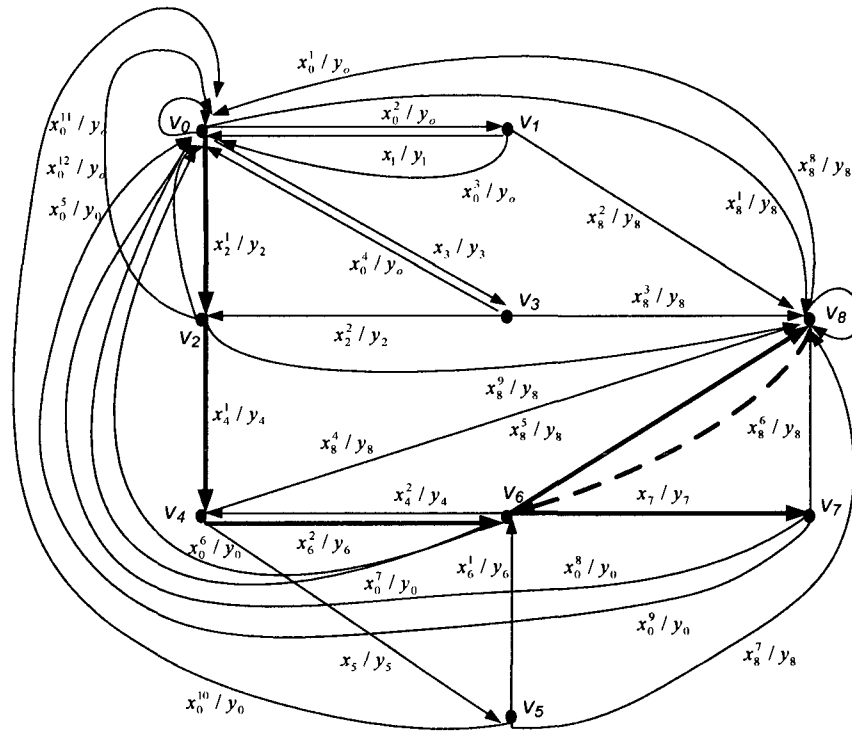


Figure 3.3 an example to show how to use the back button

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

A complete specification of *the cheap Airfares Search Engine* application in the above example consists of 25 vertexes and 118 transitions. With this specification, the test

sequence generated by our method is of length 157 while the one generated by CPP is of 236. The saving is 33%.

CHAPTER IV

IMPLEMENTATION

In this section, we give a brief introduction to CPP algorithm which can be applied to our context without using back button. Then we explain our proposed algorithm by applying Genetic Algorithm, considering the use of back button.

4.1 THE IMPLEMENTATION OF THE CPP ALGORITHM

To well understand the algorithm to solve the CPP, we introduce some graph-theoretic terms first. The number of edges going into a vertex v is the in-degree denoted by $\text{deg}_{in}(v)$, and the number of edges pointing out of a vertex v is the out-degree denoted by $\text{deg}_{out}(v)$. Let δ be the difference between the in and out degrees: $\delta(v) = \text{deg}_{out}(v) - \text{deg}_{in}(v)$. If $\delta(v) = 0$, we say the vertex v is balanced. Otherwise, let $D^+ = \{v \mid \delta(v) > 0\}$ be the set of unbalanced vertices with an excess of out-going edges, and $D^- = \{v \mid \delta(v) < 0\}$ the set of unbalanced vertices with an excess of in-coming edges. Let path p_{ij} be a path from vertex v_i to vertex v_j where $v_i \in D^-$ and $v_j \in D^+$. In general, a CPP may take some of the p_{ij} paths more than once. Let f_{ij} be the number of times the path p_{ij} must be taken, specifically, how many times the path must be added to the graph as an edge to make it Eulerian.

In [13], an algorithm of CPP can be sketched as follows.

Step 1: determine δ of each vertex v in graph G . If $\delta(v) = 0$ for all v , go to Step 5;

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

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

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

Step 5: Construct an Eulerian circuit by Fleury's algorithm [31] based on the least cost paths $i \rightarrow j$ and each path repeated $f_{ij} \geq 0$ times.

The complexity of this algorithm is $O(n^2 m^3 (\log n))$.

Here we use Figure 4.1 to show the algorithm to solve the CPP. In this figure, the weight of each edge is '1'. In Step 1, $\delta(0) = 1$, $\delta(1) = 1$, $\delta(2) = -1$ and $\delta(3) = -1$ are computed. From the result of Step 1, we can determine that $D^+ = \{0, 1\}$ and $D^- = \{2, 3\}$ in Step 2. In Step 3, we determine p_{ij} and 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. Let us say that we choose the paths $2 \rightarrow 1$ and $3 \rightarrow 0$. In Step 4, an Eulerian circuit is found: $0, 1, 3, 0, 1, 2, 3, 0, 2, 3, 0$.

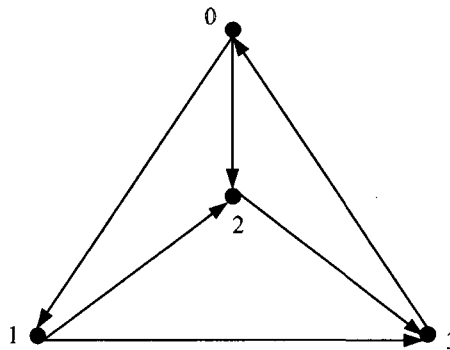


Figure 4. 1 an example of the CPP

4.2 THE IMPLEMENTATION OF THE GENETIC ALGORITHM

The present work focuses on the Genetic Algorithms (GA), a learning algorithm, and its application to improving the efficiency of the traverse of web pages. GA is a population-based, robust global optimization method. That is, it is able to find a global optimal solution without being trapped in local minima. As a result, it has been successfully employed in a variety of real-life optimization problems.

4.2.1 DESCRIPTION OF GA

GA is a part of evolutionary computing, which is a rapidly growing area of artificial intelligence. Genetic algorithms are inspired by Darwin's theory about evolution. In simple words, solution to a problem solved by genetic algorithms is evolved. Idea of evolutionary computing was introduced in the 1960s by I. Rechenberg. He introduced “evolution strategies” (evolutionsstrategie in original German). His idea was then

developed by other researchers. GA was invented by John Holland and developed by him and his students and colleagues [15].

The GA is started with a set of solutions (represented by chromosomes) called population. Solutions from one population are taken and used to form a new population. This is motivated by a hope, that the new population will be better than the old one. New solutions (offspring) are selected according to their fitness (the more suitable they are the more chances they have to reproduce). This evolution is repeated until some condition (for example the number of populations or improvement of the solution) is satisfied. In general, there are three genetic operators (processes): selection, reproduction (crossover), and mutation, which make the transition from one population generation to the next. The process is shown in Figure 4.2. The microbiological and model terminology are explained in Table 4.1.

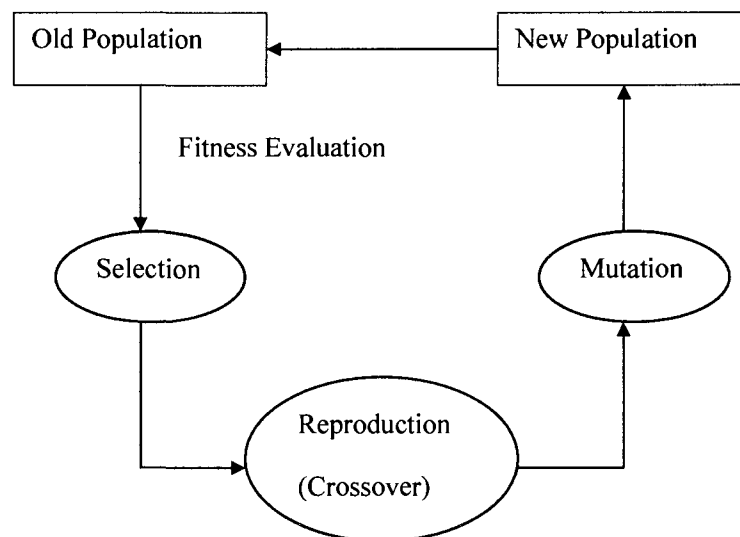


Figure 4.2 block diagram of basic GA cycle

Genetic Term	Microbiological Definition	Model Implication
Chromosomes	Threadlike strand found in the nucleus made up of a series of genes; carries genetic information, DNA	Population of solutions
genome	Genetic makeup of an organism	Solution set
Crossover	Mating of two organisms where genetic information is exchanged	Strata from different solutions will be exchanged to produce new ones
Mutation	A change which, when transmitted to offspring, gives rise to heritable variations	Genes within solutions are randomly changed
Offspring	Results of a cross; "new population" generated by reproduction	New population of solutions (children)

Table 4.1 GA microbiological and model terminology

From above we can see, the basic GA is very general. There are many things that can be implemented differently in various problems. The first question is how to create chromosomes, and what type of encoding we choose. Crossover and mutation are two basic operators of GA for this. The next question is how to select parents for crossover.

This can be done in many ways, but the main idea is to select better parents with the hope that the better parents will produce better offspring. Note that making new population only with new offspring can cause loss of the best chromosome from the previous population.

4.2.2 CHROMOSOME AND GENOME

All living organisms consist of cells. In each cell there is a same set of chromosomes. Chromosomes are strings of DNA and serve as a model for the whole organism. Here a chromosome corresponds to an edge-cover, i.e. a path that contains each edge in G at least once. Each chromosome can be randomly generated to make up an initial population of solutions.

The complete set of genetic material (all chromosomes) is called genome. In the experiment, we defined 50 chromosomes as the genome. Each chromosome represents an edge-cover.

4.2.3 ENCODING

The chromosome should in some way contain information about the solution it represents. The most common way of encoding is a binary string. The chromosome may look like this:

Chromosome 1	1101100100110110
Chromosome2	1101111000011110

Each chromosome has one binary string. Each bit in this string can represent some characteristic of the solution. There are many other ways of encoding. This depends mainly on the problem being considered. For example, one can encode directly integer or real numbers and sometimes it is useful to encode some permutations.

Binary encoding gives many possible chromosomes even with a small number of alleles. On the other hand, this encoding is often not natural for many problems and sometimes corrections must be made after crossover and/or mutation. Here we use permutation encoding. It can be easily used in ordering problems, such as CPP, RPP, travelling salesman problem, and task ordering problem. With permutation encoding, every chromosome is a string of numbers, which represents a number in a sequence. Take CPP as an example. Each chromosome can express a sequence of transitions, where each number represents an edge.

Chromosome 1	153264798
Chromosome 2	856723149

4.2.4 CROSSOVER

The crossover operator produces two new offspring from two parent strings, by copying selected bits from each parent. The bit at position i in each offspring is copied from the bit at position i in one of the two parents. Typically, crossover has three types: single-point crossover, two-point crossover and uniform crossover. Here we apply single-point crossover.

Single-point crossover has one crossover point to be selected. Till this point, the permutation is copied from the first parent. Then the second parent is scanned and if a number is not yet in the offspring it is added. Note that there are many ways to produce the rest after the crossover point. For example, $(1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9) + (4\ 5\ 3\ 6\ 8\ 9\ 7\ 2\ 1) = (1\ 2\ 3\ 4\ 5\ 6\ 8\ 9\ 7)$

In our context, we have used a rule for crossover as following. Here $\text{start}(\sigma)$ denotes the starting vertex of the first transition of σ , and $\text{end}(\sigma)$ denotes the last vertex of the last transition of σ .

- Let $G = \langle V, E \rangle$ be a digraph.
- Let C denote a set of edge-covers of G , and $c_1, c_2 \in C$.
- Let $c_1 = \rho \sigma_1 \zeta$, where ρ, σ_1 and ζ are paths.
- Let $\text{path}_{\sigma_2} \in c_2$.
- If $\text{start}(\sigma_1) = \text{start}(\sigma_2)$, $\text{end}(\sigma_1) = \text{end}(\sigma_2)$.
- For any edge $e \in \text{edges}(\sigma_2) - \text{edges}(\sigma_1)$, $e \in \rho$ or $e \in \zeta$, $|\sigma_2| < |\sigma_1|$.
- Then $c' = \rho \sigma_2 \zeta \in \text{next_gen}(C)$.

Here $\text{next_gen}(C)$ denotes the set of the next population of C . And back denotes the input of clicking the back button. Note that function $\text{edge}()$ does not contain any back edge.

This rule is illustrated in Figure 4.3.

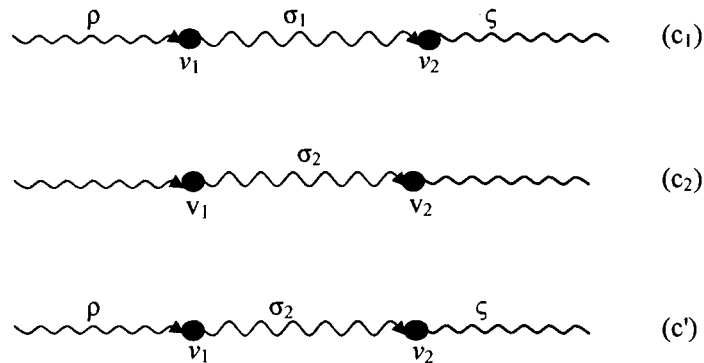


Figure 4.3 crossover rule

4.2.5 MUTATION

In addition to crossover operators that produce an offspring by combining parts of two parents, a second type of operator produces offspring from a single parent. In particular, the mutation operator produces small random changes to the bit string by choosing a single bit randomly, and changing its value. Mutation is often performed after crossover.

Crossover and mutation are two basic operators of GA. Performance of GA heavily depends on them. In order to reduce the test sequence length more significantly, we make a mutation rule as following:

- Let $G = \langle V, E \rangle$ be a digraph.
- Let C denote a set of edge-covers of G .
- Suppose that $c = \rho a_1 a_2, \dots, a_k \zeta \in C$, where ρ and ζ are paths, $a_1 a_2, \dots, a_k \in E$.

- If $\text{start}(a_1) = \text{end}(a_k)$, $\text{cookie}(a_1) = \text{false}$, and for all a_i ($2 \leq i \leq k$) $a_i \in \rho$ or $a_i \in \zeta$.
- Then $c' = \rho a_1 \text{ back } \zeta \in \text{next_gen}(C)$.

This rule is illustrated in Figure 4.4:

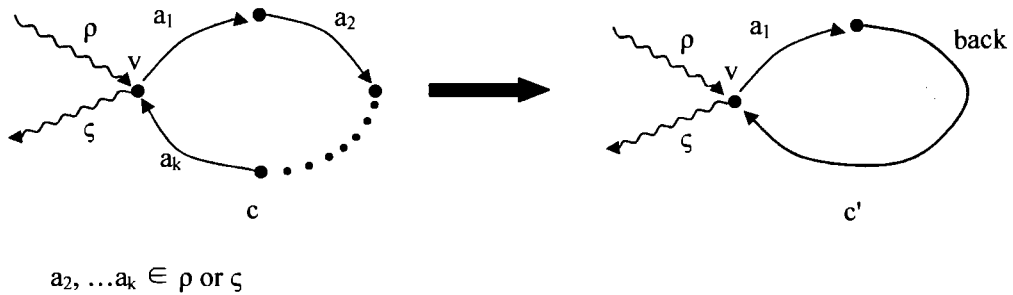


Figure 4.4 mutation rule

The above crossover and mutation rules lead to two phenomena, which are the stagnation (pre-mature) and convergence. The stagnation is due to the random choice of results in a sub-optimal way. Both the *Mutation Rule* and the *Crossover Rule* allow for various ways to derive the next generation, and as a consequence, may lead to stagnation. The convergence is due to the use of the back operation in mutation rule and requiring that the substituted subpath be shorter than the original one in crossover rule. For example, given an edge-cover $c \in C$, there may exist various possible choices to derive c' of the next generation according to the *Mutation Rule*. Some of these choices can be applied one after another without conflict while some choices will prevent other derivations.

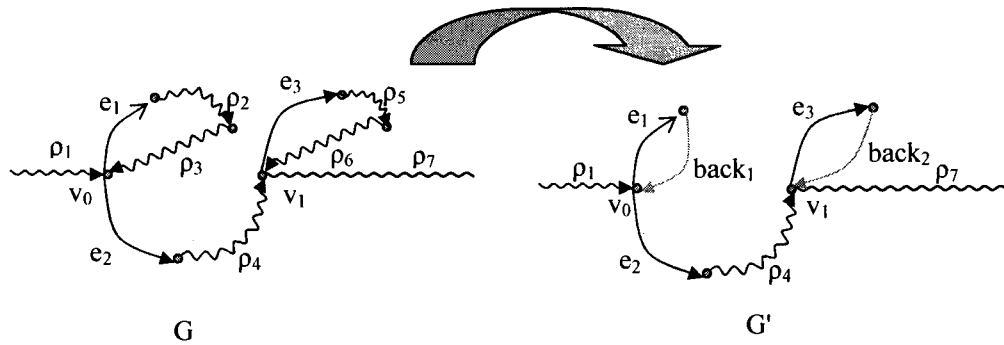


Figure 4.5 (a) no mutually exclusion

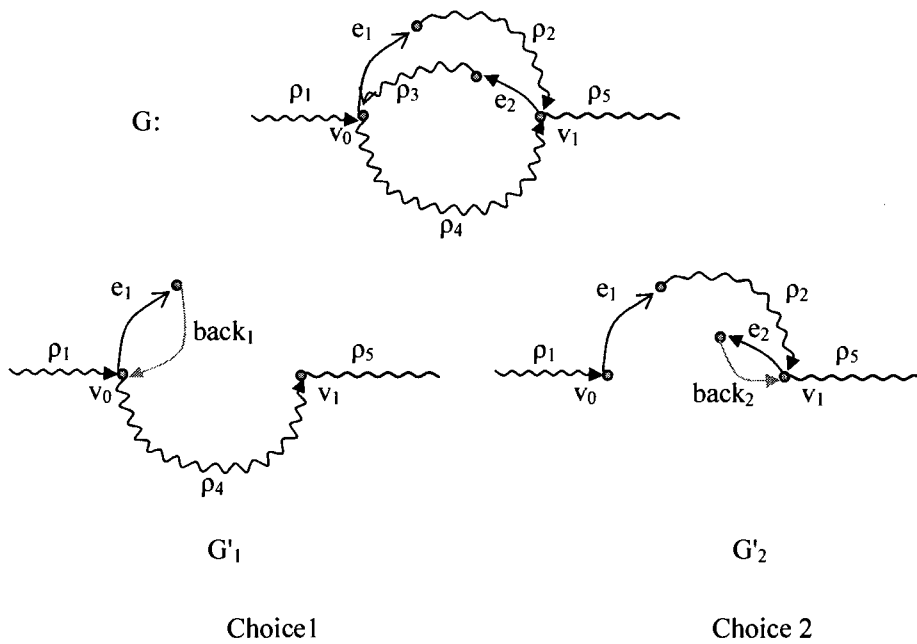


Figure 4.5 (b) mutually exclusion

Figure 4.5 shows the pre-mature and convergence phenomena. In figure 4.5 (a), $G = \langle V, E \rangle$ is a digraph, $\rho_1, \rho_2, \rho_3, \rho_4, \rho_5, \rho_6,$ and ρ_7 are paths. $e_1, e_2,$ and $e_3 \in E$. ρ_2, ρ_3, ρ_5 and ρ_6 have appeared before, so we can use the *mutation rule* to add back edge in G , and then

the graph G changes to G' . In graph G' , the two back edge $back_1$ and $back_2$ do not have the cross point, so we say $back_1$ and $back_2$ are not mutually exclusive. However, in figure 4.5 (b), $G = \langle V, E \rangle$ is a digraph, $\rho_1, \rho_2, \rho_3, \rho_4$ and ρ_5 are paths. e_1 and $e_2 \in E$; ρ_2, ρ_3, ρ_4 and e_2 have appeared before, so we can use the *mutation rule* to add back edge in G , and then the graph G can be changed into two different graphs. When choice 2 was selected choice 1 reduces the length of the test sequence in a better way. Thus the next generation does not provide the best solution for reducing the length of the test sequence.

4.2.6 FITNESS FUNCTION

The fitness function defines the criterion for ranking potential hypotheses and for probabilistically selecting them for inclusion in the next generation population. A fitness function is a particular type of objective function that quantifies the optimality of a solution (that is, a chromosome) in a GA.

Given the initial genome $G = \{c_1, \dots, c_n\}$, we want to find a feasible subset $c_{min} \subseteq c$ of minimal cardinality in our performance. So the fitness function is $f_i = \sum_{j=1}^N c_{ij}$, where c_{ij} is the value of the j -th generation corresponding to the i -th individual.

4.2.7 THE ALGORITHM

This Genetic Algorithm can be summarized as follows:

1). The Main GA Algorithm

1: **Input:** $G = (V, E)$, AcceptableFactor, MaxGeneration;

2: **Output:** a set of test sequences and *tempGen*

3: Algorithm:

4: *currentGen* := \varnothing

5: *tempGen* := \varnothing

6: generation := 0;

7: *currentGen* := InitializePopulation(G);

8: bestfit := minLength(*currentGen*);

9: **While** (generation < MaxGeneration and bestfit > AcceptableFactor * |E|)

10: generation := generation + 1;

11: *tempGen* := crossover(mutation(*currentGen*));

12: *currentGen* := Select(*tempGen*, *currentGen*);

13: bestfit := minLength(*currentGen*);

14: **End while**

15: **Output** bestfit;

2). Population Initialization Algorithm

We use the population initialization algorithm to generate the first generation of chromosomes. Each chromosome is an edge-cover of the graph representing the application.

1: **Input:** $G = \langle V, E \rangle$, n.

2: **Output:** $InitGen = \{c_1, \dots, c_n\}$

3: Algorithm:

4: initially $InitGen = \varnothing$;

```

5: for each  $k$  ( $1 \leq k \leq n$ ) do
6:    $c_k = \varnothing$ ;
7:   mark all edges in  $E$  unvisited;
8:   Let current be a randomly selected node from  $V$ ;
9:   while (there exists an unvisited edge  $e$ )
10:    apply breadth-first-search to find a shortest path  $\rho$  from current to  $e = (n_i, n_j)$ ;
11:    append( $c_k, \rho$ ); // since  $\rho$  has an unvisited edge  $e$ , we append it to  $c_k$ 
12:    mark  $e$  as visited; // after appending  $e$  to  $c_k$  we mark  $e$  as visited.
13:    current :=  $n_j$ ;
14:   end while
15:   add(InitGen,  $c_k$ );
16: end for
17: Output InitGen;

```

Since BFS takes $O(|E| + |V|)$ and there are $|E|$ edges to traverse, Population Initialization Algorithm takes $O(|E|(|E| + |V|))$.

3). Crossover Algorithm

We use crossover algorithm to combine two chromosomes (parents) to produce a new chromosome (offspring).

1: **Input:** *currentGen* = $\{c_1, \dots, c_n\}$

2: **Output:** *nextGen* = $\{c'_1, \dots, c'_n\}$

3: Algorithm:


```

23:    add(nextGen,c'j);
24:    mark ci and cj as selected;
25: end while
26: Output nextGen

```

Suppose the diameter of G is d . Since the length of each c_i is less than $d|E|$, Crossover Algorithm takes $O(d|E|)$.

4). Mutation Algorithm

In the mutation algorithm we make use of the *back* button of the browser to reduce the length of the chromosomes.

```

1: Input: currentGen = {c1, ..., cn}
2: Output: nextGen
3: Algorithm:
4: for each chromosome ci ∈ currentGen do
5:   if (ci has a subpath  $p = a_1 a_2, \dots, a_k$  ( $k \geq 2$ ), s.t  $c_i = \rho p \zeta$ , that
start ( $a_1$ ) = end ( $a_k$ ) & for all  $a_i$  ( $2 \leq i \leq k$ ) cookie ( $a_i$ ) = false &  $a_i \in \rho$  or  $a_i \in \zeta$ )
6:   then  $c' = \rho a_1$  back  $\zeta$ ;
7:     add(nextGen,c');
8:   else add(nextGen, ci);
9:   end if
10: end for

```

11: **Output** *nextGen*;

Suppose the diameter of G is d . Since the length of each c_i is less than $d|E|$, Mutation Algorithm takes $O(d|E|)$ in the worst case where $p = c_i$.

From the algorithm, the time complexity of the presented GA is $O(2|E|(d+|p|+|q|))$.

CHAPTER V

METHOD EVALUATION

We have conducted experiments to evaluate the performance of our proposed method in terms of the savings we gain on the test sequence length. This is in comparison to the *direct application* of CPP on test sequence generation. All experiments are performed on a PC with 1.66 GHz CPU. Both methods are implemented in Java and run under Java Runtime Environment 1.6.0 with 256 Megabytes maximum memory assigned for Java Virtual Machine.

Let the generation (denoted by g) of our method be 50. We compare the performance of the two methods according to the following factors of digraphs G :

- the number of back edges (denoted by b).
- the number of edges in G (denoted by q), also called the *size* of G .
- the *diameter* of G (denoted by d), i.e., the greatest value in the set $\{distance(v_i, v_j) \mid v_i, v_j \in V\}$, where $distance(v_i, v_j)$ is the length of a shortest path from v_i to v_j .
- the ratio of cookie-affecting transitions in G , denoted by r . A transition is cooking-affecting if it has cookie code 1. We have $r = (\text{No. of cookie-affecting transitions})/size$.

Note that the stack variable *history* in our algorithm is different from the history stack of the browser. Here, it only records the vertices that have been visited by traversing a

sequence of transitions with cookie code 0. That is, the execution of these transitions will not change the cookie values of the web application. If a transition with cookie code 1 is traversed, the stack variable *history* will be emptied before pushing a new vertex into it. In this way, the proper use of the *back* button is guaranteed and a desired web page can be visited again by clicking the *back* button.

The saving is calculated as $\text{diff}/\text{CPPlength} * 100\%$, where *diff* is the difference between the length of the test sequence derived by using CPP method and that of ours, and *CPPlength* is the length of the test sequence derived by using CPP method.

5.1 THE NUMBER OF |BACK|

In this subsection, we analyze how the change of the number of back edges affects the lengths of the generated test sequences in our methods. Let the number of vertices in G be 100 and let $q = 160$, $d = 25$, $r = 5$ and $g = 50$. We change the number of back edges b . For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 5.1.

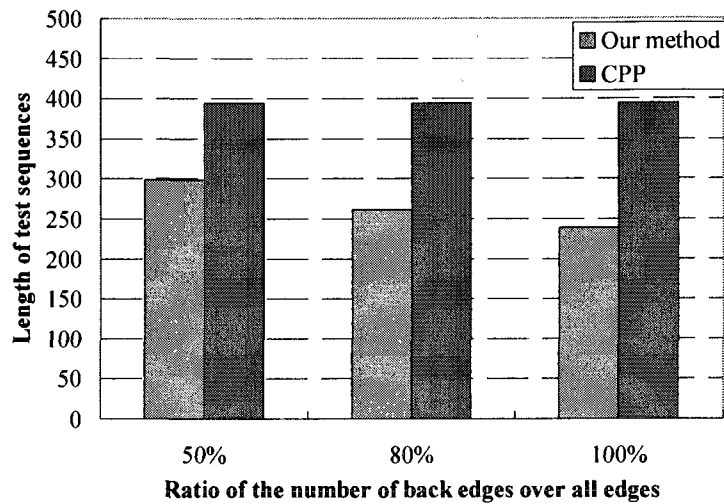


Figure 5.1 an illustration of the changes of the lengths of the test sequences according to the number of back edges of G

According to this figure, we have the following observations:

O.1.1 Our method considerably reduces the lengths of the test sequences compared to CPP, leading to 39%-54% of savings;

5.2 SIZE OF G

Now we analyze how the change of the size of G affects the lengths of the generated test sequences for both methods. Let the number of vertices in G be 100 and let $d = 25$, $r = 5$ and $g = 50$. We increase the value of size q . For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 5.2.

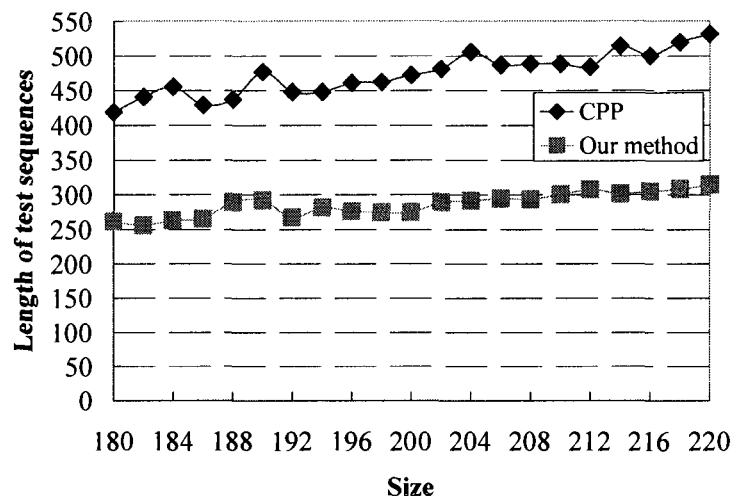


Figure 5.2 (a) the changes of the lengths of the test sequences according to the increase of the size of G

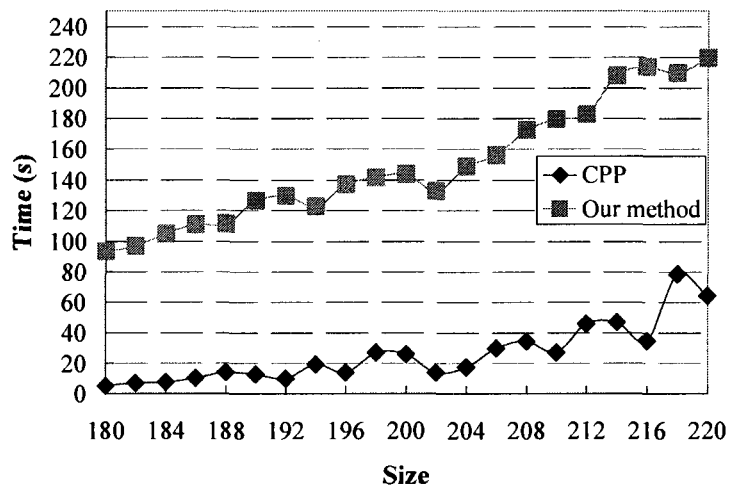


Figure 5.2 (b) the time changes with the lengths of the test sequences according to the increase of the size of G

According to this figure, we have the following observations:

O.2.1 Our method considerably reduces the lengths of the test sequences compared to CPP, leading to 34%-42% of savings;

O.2.2 With the increase of the *size* of G the increasing rate of the lengths of the generated test sequences according to our method is similar to that of CPP.

O.2.3 Our method needs more computational time than CPP.

With the increase of the *size* of G the number of edges increases. As a consequence, the corresponding path of a desired test sequence needs to traverse more edges, yielding longer test sequences in general for both methods.

The generations of the base population of our methods is 50, while the CPP is just run once. Accordingly, the corresponding time by our method is greater than that of CPP.

5.3 DIAMETER OF G

In this subsection, we analyze how the increase of the diameter of G affects the length of the generated test sequence for both methods. Let the number of vertices in G be 100, and let $q = 160$, $r = 5$, and $g = 50$. We increase the value of diameter d . Again, for each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The experimental results are shown in Figure 5.3.

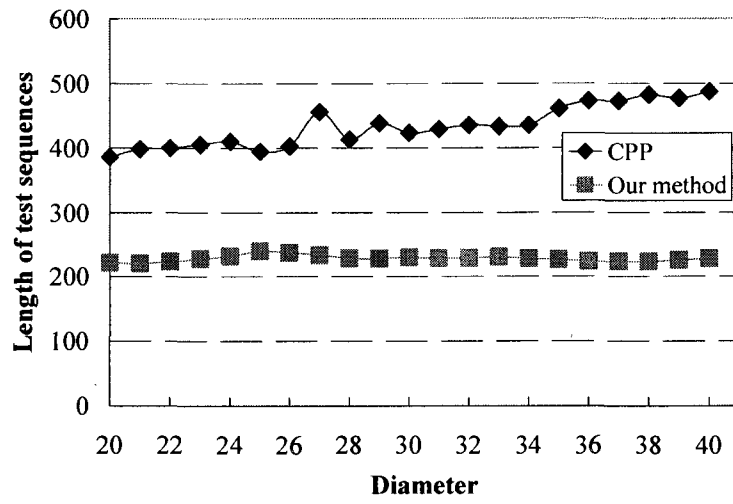


Figure 5.3 (a) the changes of the lengths of the test sequences according to the increase of the diameter of G

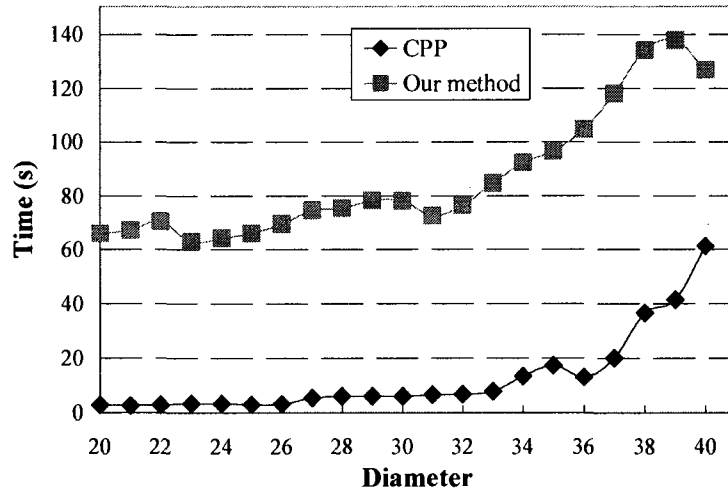


Figure 5.3 (b) the time changes of the lengths with the test sequences according to the increase of the diameter of G

According to this figure, we have the following observations:

O.3.1 Our method considerably reduces the lengths of the test sequences compared to CPP, leading to 39% - 54% of savings;

O.3.2 With the increase of the diameter of G , the increasing rate of the lengths of test sequences generated by CPP is higher than that of ours.

O.3.3 Our method needs more computational time than CPP.

5.4 RATIO OF COOKIE-AFFECTING TRANSITIONS IN G

Here we present our experimental results on how the change of the ratio of cookie-affecting transitions in G affects the lengths of the generated test sequences. Let the number of states in G be 100, $q = 160$, $d = 25$, and $g = 50$. We increase the value of the ratio of privilege transitions r . For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 5.4.

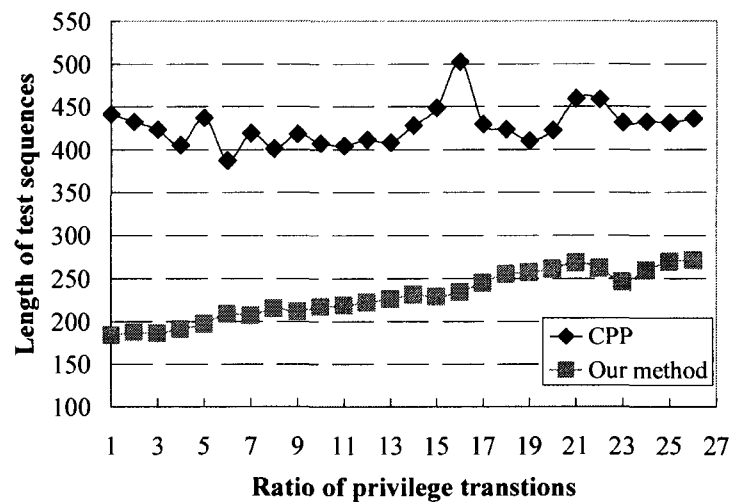


Figure 5. 4 (a) the change of length of the test sequences according to the ratio of the privilege transitions

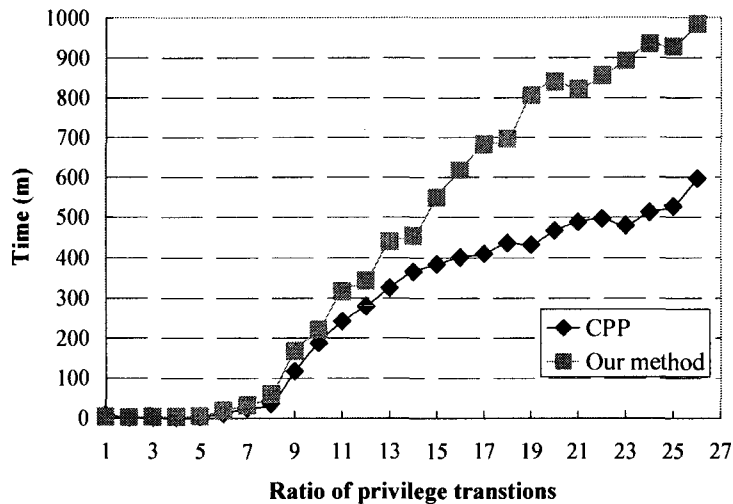


Figure 5. 4 (b) the time changes according to the ratio of the privilege transitions

According to this figure, we have the following observation.

O.4.1 Our method reduces the lengths of the test sequences, leading to 34%-42% of savings compared to the CPP.

O.4.2 With the increase of the ratio of the cookie-affecting transitions, the increasing rate of the lengths of the generated test sequences according to our method is similar to that of CPP.

With the increase of the ratio of the cookie-affecting transitions, there are more cookie-affecting transitions, executing which the cookie values are changed. Our test

sequence generation algorithm empties the stack variable *history* to guarantee the proper use of the *back* button whenever a cookie-affecting transition is about to be traversed. It follows that there are fewer vertexes in our stack variable *history* to be used. As a result, the length of the generated test sequence by our method is increased. However, the CPP method is not affected by the content of the history stack, and thus the length of test sequences generated by the CPP method does not change too much in most cases. This conforms to our observation O.4.2.

5.5 SPECIAL CASE

In this subsection, we consider some special cases. Firstly, we analyze how the increase in the size of G affects the length of the generated test sequence for both methods when the ratio of cookie-affecting transitions in G is 0. Let the number of vertices in G be 100, and let $d = 25$, $r = 0$, and $g = 50$. We increase the value of size q . For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 5.5.1.

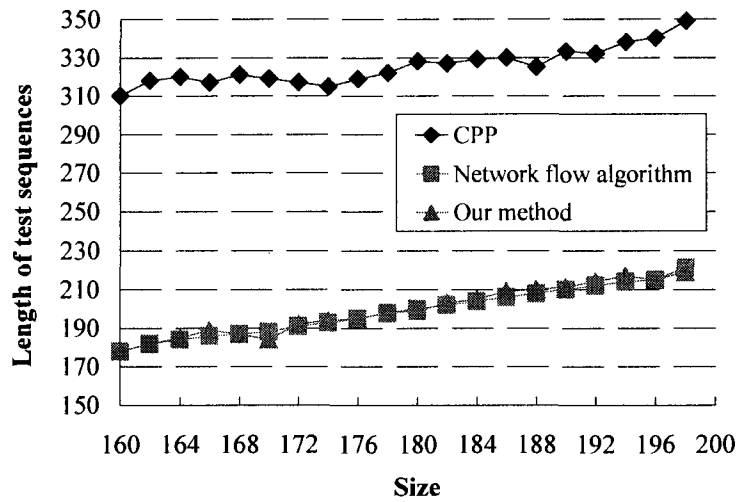


Figure 5.5.1 a special case of the changes of the lengths of the test sequences according to the increase of the size of G

O.5.1 Our method considerably reduces the lengths of the test sequences leading to 36%-46% of savings compared to CPP, and 0%-2% savings compared to the network flow algorithm.

Then we analyze how the increase of the diameter of G affects the length of the generated test sequence for both methods when the ratio of cookie-affecting transitions in G is 0. Let the number of vertices in G be 100, and let $q = 160$, $r = 0$, and $g = 50$. We increase the value of diameter q . For each combination of these values, we have randomly selected 100 instances and calculated the average length of the generated test sequences. The result is shown in Figure 5.5.2.

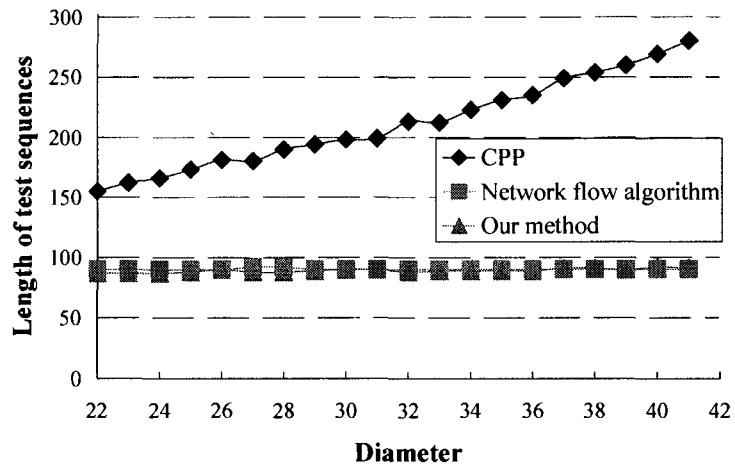


Figure 5.5.2 a special case of the changes of the lengths of the test sequences according to the increase of the diameter of G

O.5.2 Our method considerably reduces the lengths of the test sequences leading to 44% - 68% of savings compared to CPP, and 0%-4% savings compared to the network flow algorithm.

CHAPTER VI

CONCLUSIONS AND RECOMMENDATIONS

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

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

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. 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 directed graph. 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. How to deal with this problem remains an interesting issue.

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

APPENDICES

APPENDIX A

APPENDIX B

REFERENCES

- [1] A. V. Aho, A. Dahbura, D. Lee, and M. Uyar. An optimization technique for protocol conformance test generation based on UIO sequences and Rural Chinese Postman Tours. *IEEE Trans Comm.*, 39(11): 1604-1615, Nov. 1991.
- [2] A. Andrews, J. Offutt, and R. Alexander. Testing web applications by modeling with FSMs. *Software systems and Modeling*, 4(3): 326-345, 2005.
- [3] H. V. Beek and S. Mauw, "Automatic conformance testing of internet applications", In *Proc. of the 3rd International Workshop on Formal Approaches to Testing of Software, Lecture Notes in Computer Science, vol. 2931*, pages 205-222, 2004.
- [4] W. K. Chang, S. K. Hon, and C. Chu. A systematic framework for evaluating hyperlink validity in web environments. In *Proc. of 3rd International Conference on Quality Software 2003*, pages 178-185, 2003.
- [5] J. Chen and S. Subramaniam. Specification-based testing for GUI-based applications. *Software Quality Journal*, 10(3): 205-224, 2002.
- [6] J. Chen and X. Zhao. Formal models for web navigations with session control and browser cache. In *Proc. of International Conference on Formal Engineering Methods. LNCS 3235*, pages 46-60. Springer-Verlag, 2004.
- [7] T. S. Chow. Testing software design modeled by finite-state machines. *IEEE Trans. Software Eng.*, SE-4(3): 178-187, May 1987.
- [8] L. Duan, Y. Wang and J. Chen. Enhanced traverse of web pages. In *Proc. of the 10th IEEE High Assurance Systems Engineering Symposium (HASE 2007)*, 2007.
- [9] A. Gibbons, *Algorithmic Graph Theory*, Cambridge University Press, 1985.

- [10] A. Gill, *Introduction to the theory of Finite-State Machines*, Mc Graw-Hill Book Company, Inc, 1962.
- [11] G. Gonenc. A method for the design of fault detection experiments. *IEEE Trans. Computers*, 19(6):551-558, June 1970.
- [12] P. Graunke, R. B. Findler, S. Krishnamurthi, and M. Felleisen. "Modeling Web Interactions". In *European Symposium on Programming*, 2003.
- [13] Harold W. Thimbleby. The directed Chinese Postman Problem. *Journal of Software Practice and Experience*. 33(11) pages: 1081-1096. September 2003.
- [14] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proc. of 5th Ann. Symp. Switching Circuit Theory and Logical Design*, pages 95-110, 1964.
- [15] J. Holland. *Adaption in Natural and Artificial Systems*. MIT Press. 1975.
- [16] 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.
- [17] S. C. Lee and J. Offutt. Generating test cases for XML-based web component interactions using mutation analysis. In *Proc. of the 12th IEEE International Symposium on Software Reliability Engineering (ISSRE'01)*, Pages 200-209, 2001.
- [18] C. Liu, D. Kung, P. Hsia, and C. Hsu. Structural testing of web applications. In *Proc. 11th IEEE International Symposium on Software Reliability*. Pages 84-96, 2000.
- [19] G. D. Lucca, A. Fasolino, f. Faralli, and U. de Carlini. Testing web application. In *Proc. of IEEE international Conference On software Maintenance (ICSM' 02)*, pages 310-319, 2002.

- [20] 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.
- [21] A. Memon, M. Pollack, and M. Soffa. Hierarchical GUI test case generation using automated planning. *IEEE Trans. Software Eng.*, 27(2): 144-155, 2001.
- [22] R. E. Miller and S. Paul. On the generation of minimal length conformance tests for communications protocols. *IEEE/ACM Transactions on Networking*, 1(1): 116-129, 1993.
- [23] S. Nation and M. Tsunoyama. Fault detection for sequential machines by transition tours. In *Proc. of 11th. IEEE Fult Tolerant Computing Symposium*, pages 238-243, 1981.
- [24] T. Ostrand, A. Anodide, H Foster, and T. Goradia. A visual test development environment for GUI systems. In *Proc. of ACM SIGSOFT Internation Symposium on Softeare Testing and Analysis*. Volume 23, pages 82-92, 1998.
- [25] Y. Qi, D. Kung, and W. E. Wong. An agent-based data-flow testing approach for web applications. *Journal of Information and Software Technology*, 48(12):1159-1171, 2006.
- [26] F. Ricca and P. Tonella, "Building a Tool for the Analysis and Testing of Web Applications: Problems and Solutions", In *Proc. of Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2001)*, *Lecture Notes in Computer Science*, vol. 2031, pages 373-388, 2001.
- [27] F. Ricca and P. Tonella, "Testing Processes of Web Applications", *Annals of Software Engineering*, vol. 14, pages 93-114, 2002.
- [28] K. K. Sabnani and A. Dahbura. A protocol test generation procedure. *Computer Networks and ISDN Systems*, 4(15):285-297, 1988.

- [29] E. D. Sciascio, F. M. Donini, M. Mongiello, G. Piscitelli, "AnWeb: a System for Automatic Support to Web Application Verification", In *Proc. of the 14th International Conference on Software Engineering and Knowledge Engineering*, 2002.
- [30] E. D. Sciascio, F. M. Donini, M. Mongiello, G. Piscitelli, "Web Applications Design and Maintenance Using Symbolic Model Checking", In *Proc. of IEEE the 7th European Conference on Software Maintenance and Reengineering (CSMR'03)*, 2003.
- [31] S. Skiena. *The Algorithm Design Manual*. Springer Verlag. 1998.
- [32] R.E. Tarjan. *Data Structures and network algorithm*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1983.
- [33] H. Ural, X. Wu, and F. Zhang. On minimizing the lengths of checking sequences. *IEEE Transactions on Computers*, 46(1):93-99, 1997.
- [34] L. White and H. Almezen. Generating test cases for GUI responsibilities using complete interaction sequences. In *Proc. of 11th International Symposium on Software Reliability Engineering*, pages 110-121, 2000.
- [35] W. Xu, J. Offutt, and J. Luo. Testing web services by xml perturbation. In *Proc. of IEEE International Symposium on Software Reliability Engineering*. pages 257-266, 2005.
- [36] H. Zhu, P. A. V. Hall, and J.H.R. May. *Software unit test coverage and adequacy*. ACM Computing Surveys, 29:366-427, 1997.

VITA AUCTORIS

Name: Li, Man

Place of Birth: Shangqiu, China

Year of Birth: 1983

Education: University of Windsor, Windsor, Ontario, Canada
2006-2009 M.Sc. in Computer Science
Zhengzhou University of Light Industry, Zhengzhou, China
2001-2005 B.A.Sc. in Computer Science and Engineering

Working Experience: Software Developer and Quality Assurance, Wuxi Huayang
Software Co., Ltd. Wuxi, 2005-2006.