A Test Case Generation Method and a Tool Implementation for Acceptance Testing of GUI Software

Deng-Jyi Chen, Ming-Jyh Tsai, Jen-Gaw Lee, Chien-Yuan Chen
Department of Computer Science and Information Engineering
National Chiao Tung University – Taiwan, R. O. C.
{djchen, mjtsai, jglee, cychen }@csie.nctu.edu.tw

Abstract

This paper proposes a simple and easy to follow black-box test case generation method for acceptance testing of GUI software. The idea of this method is from white-box path-based testing method. The zero-one optimal path selection method is used to find an optimal path set for a selected coverage criterion. Five reduction rules are used to greatly reduce computation time. A testing tool has been developed to facilitate the operation of the proposed method and an example is used to prove the applicability of the proposed method.

Key-words: Test case, Zero-one method, Reduce rules, Black box testing, white box testing

1: Introduction

GUI (Graphic User Interface) software has become more and more popular. When developing GUI software, it's not unusual to reuse existing software component.

The V-diagram, as shown in Fig. 1, illustrates different testing stages. This paper is focused on the acceptance testing of GUI software.

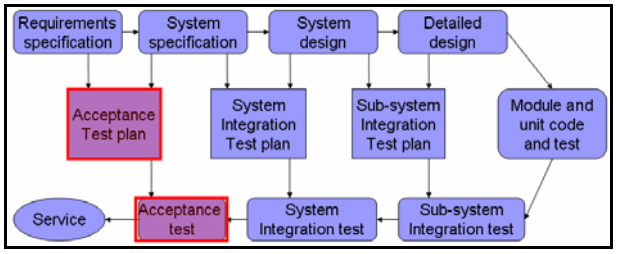

Fig. 1 the V diagram of testing

Software testing can be classified into back-box testing and white-box testing. In acceptance testing stage, black-box testing is mostly applied. Table 1 is a listing of existing black-box testing tools. Most of them provide record-and-replay function.

In summary, we need a software testing tool satisfying the following 2 conditions:

- Easy to generate test cases for the GUI software under test
- Easy to select an optimal path set

Table 1 Existing black-box testing tools

Tool	*Description*	*Company*
Win Runner	Writing TSL scripts to enable automation testing.	Mercury
Silk Test	Provides object-oriented programming language called 4Test.	Segue
Rational Robot	Generates test scripts in SQABasic, an integrated MDI scripting environment that allows you to view and edit your test script while you are recording.	IBM
QA Run	A capture-replay tool with an integrated database, The captured test sequence is encoded in a scripting language, so you can tweak the tests with a text editor.	Compuware

2: GUI test case generation method

Since the method proposed in this paper is based on the idea of path testing. We have to review path testing first. In discussion of path testing, the structure of the program under test is usually mapped to a program diagraph *G=(N,B)*, where N and B represent node set and branch set, respectively. Without loss of generality, it is assumed a sole source node and a sole terminal node both exist in the diagraph. A path, starting from the source node and ending with the terminal node, is a sequence of nodes each connected by branches. Using network methodologies such as node-reduction [1] or linearly independent circuit [2] the complete path set *P*, which is defined as the set containing all paths, of program diagraph G can then be constructed.

A program with loops may lead to an extremely large number of paths. To alleviate this problem, it is considered sufficient in practice to limit loop iterations up to a constant number.

Mapping to GUI program acceptance testing, a test case is a sequence of operations of the GUI program under test. For example, the GUI program under test, as shown in Fig. 2 (a), requires users to keyin ID, password, and then click ok button or cancel button. Intuitively, we can construct the control flow diagram as shown in Fig.

2 (b). From the control flow diagram, we can derive two paths.

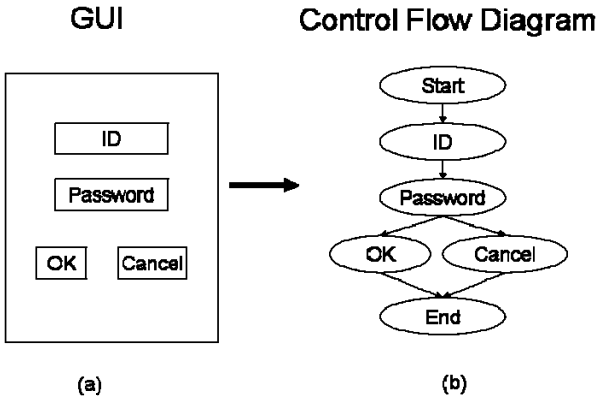

Fig. 2 GUI to Control Flow Diagram

A program is composed of three elements, namely, sequential, conditional branch, and loop. In Fig. 1, we can find sequential (from ID to Password) and conditional branch (from password to OK or cancel). Loop is not unusual in programs but hard to find in GUI. We will put loop into future works.

A GUI program may have two or more pages. Some pages are related. That is, page *i* may link to page *j*. Base on the inter-page relationship, we can construct an inter-page relationship diagram as shown in Fig. 3(a). For each page, we can construct the control flow diagram and then obtain all paths. Base on inter-page relationship and Cartesian product, we can derive all expanded paths. For example, base on the inter-page relationship and the number of paths of each page, as shown in Fig.3 (b), we can obtain 2x3x5 + 2x4x5 = 70 expanded paths. An expanded path maps to a test case in acceptance testing.

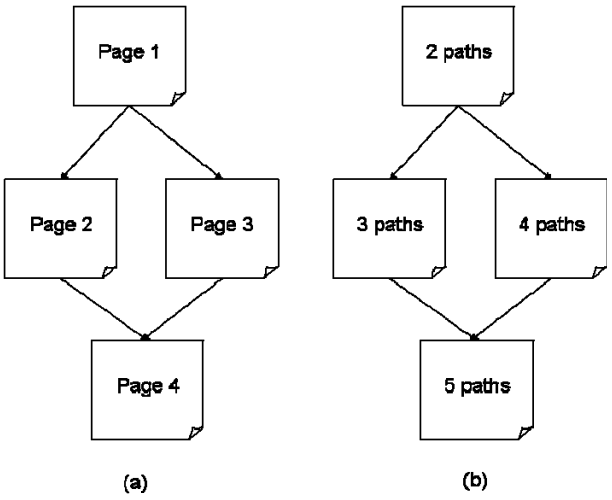

Fig. 3 Inter-page relationship and the number of paths of each page

If the total number of pages and the total number of paths of each page are both large, the total number of expanded paths will become un-testable. This problem is named path-explosion problem in path-based testing. The solution is to select a coverage criterion at a time. For example, you may select all-statements (or all-nodes) coverage criterion first and then all-branches.

Once the coverage criterion has been selected, the next problem is how to select a minimum path set satisfying the selected coverage criterion. We can use zero-one optimal path set selection method to solve this problem.

3: Zero-one optimal path set selection method

Given a complete path set P and a required coverage criterion C, a corresponding path-component coverage frequency matrix F can be generated, thus showing the coverage frequency relationship among the paths in P and the components C must cover. The rows in F represent the paths in P, while the columns represent the components C must cover, and F[i,j] represents the coverage frequency of the ith path over the jth component. For example, consider the control flow graph shown in Fig. 4(a), the complete path set P is shown in Fig. 4(b), for all-nodes coverage criterion, a path-node coverage frequency matrix is generated as shown in Table 2.

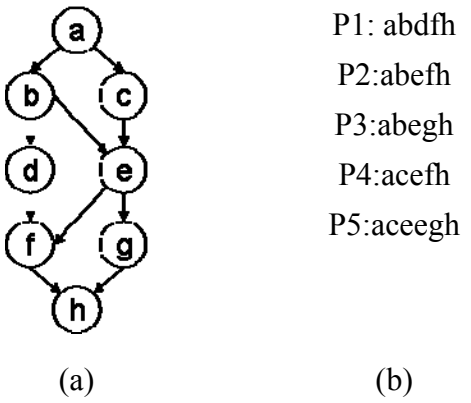

Fig. 4 Control Flow Graph and All Paths

Table 2 Path-node coverage frequency matrix

	a	b	c	d	e	f	g	H
P1	1	1	0	1	0	1	0	1
P2	1	1	0	0	1	1	0	1
P3	1	1	0	0	1	0	1	1
P4	1	0	1	0	1	1	0	1
P5	1	0	1	0	1	0	1	1

For the all-nodes coverage criterion, the optimal path set selection problem can be defined as follows. In the complete path set P, which paths should be selected to

guarantee that each node is covered at least once and the number of selected paths is minimal? This is a decision problem. In the above example, define the variable xi, i∈{1,2,3,4,5}, let xi be a decision variable corresponding to path pi, and xi=1, if pi is selected; 0, otherwise. Thus, the optimal path set selection problem can be formulated as:

$$\text{Min } z=\sum_{i=1}^{5} x_i$$

$$1\times x_1+1\times x_2+1\times x_3+1\times x_4+1\times x_5 \geq 1$$

$$1\times x_1+1\times x_2+1\times x_3 \geq 1$$

$$1\times x_4+1\times x_5 \geq 1$$

$$1\times x1 \geq 1$$

$$1\times x_2+1\times x_3+1\times x_4+1\times x_5 \geq 1$$

$$1\times x1+1\times x2+1\times x4 \geq 1$$

$$1\times x3+1\times x5 \geq 1$$

$$1\times x_1+1\times x_2+1\times x_3+1\times x_4+1\times x_5 \geq 1$$

The above constraint inequalities can also be represented in the following matrix form:

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Next, the requirement that the number of testing case must be minimized is considered. Since the value of objective function $z=x_1+x_2+x_3+x_4+x_5$ is the total number of testing case in the selected testing case set, z is the minimization target. Combining the objective function and the constraints, the optimal testing case is formulated as the following zero-one integer programming problem.

$$\text{Min } z=\sum_{i=1}^{5} x_i$$

s.t. (subject to)

$$\begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}^T \times \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} \geq \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

x_i=0 or 1, i∈{1,2,…,5}

The complexity of the zero-one integer programming method is proportional to ($2^{|paths|}\times|component|$ where |paths| represents the number of candidate paths while |component| represents the number of nodes to be covered. The computational time is exponentially proportional to |paths|, a feature which is the major drawback of the method.

The computation time required by the zero-one optimal path set selection method can be reduced if the size of the coverage frequency matrix (i.e., the number of candidate paths and the number of components to be covered) is reduced.

It's easy to observe that

- if a component is not required to be covered, the component can be ignored in the path selection problem,
- if a component is required to be covered and is covered by only one path, the path which covers the component must be selected,
- if any path covering a component, say c_i, also covers another component, say c_j, then the requirement that c_i and c_j must be covered at least once can be reduced to c_i must be covered at least once, and
- if a path, say p_i, covers all components covered by another path, say p_j, and some additional components, then p_j can be ignored in the path selection problem owing to the existence of p_i..

Based on these observations, the following five reduction rules have been proposed in [3-4].

- Rule 1. Surely Satisfied Constraint: If a component does not have to be covered at least once, its corresponding constraint is surely satisfied and can thus be ignored.
- Rule 2. Essential Paths: A path is essential if and only if it alone covers one or more components. After an essential path, say, p_k has been selected, the components covered by p_k can be ignored

during subsequent computation because they have been covered by the selected path p_k.

- Rule 3. Dominant node/ Dominant Column: Component c_i dominates component c_j if and only if every path covering c_i also covers c_j. In this situation, the requirement that c_j must be covered at least once can be ignored because c_i must be covered at least once.
- Rule 4. Dominant testing case/ Dominant Row: Path p_i dominates path p_j if and only if p_i covers all the components covered by p_j. In this situation, p_j can be ignored due to the existence of p_i.
- Rule 5. Zero testing case/Zero Row: Path p_i is a zero path if and only if it does not cover any component. This situation happens after Rule2 has been applied and the essential path is dominant over another path. In this case, the zero paths can be deleted without affecting the problem solution.

In the next section, we will use a real world example to show how these five reduction rules can greatly reduce matrix size.

4: Example and implementation

A testing tool has been developed to facilitate the operation of the proposed method. That is, this testing tool helps testers to 1) generate all paths, 2) for a selected coverage criterion, generate a corresponding coverage frequency matrix, and 3) apply 5 reduction rules and 4) apply zero-one optimal path set selection method to select an optimal path set. Currently, this testing tool supports both all-nodes and all-branches coverage criteria. To solve zero-one integer programming problem, an existing software tool named LINDO [5] is integrated into the testing tool we developed.

In this section, e-bay registration web-pages are used to illustrate the operations of the testing tool. It has four web-pages to finish the registration operation. In section 4.1, we just consider the 1st web page only. In section 4.2, we take the four web-pages into consideration.

4.1: Single web-page testing

Step 1: capture the webpage under test (the result is shown in Fig. 4)

Step 2: identify nodes (the result is shown in Fig. 5)

Step 3: identify inter-node relationship (the result is shown in Fig. 6)

Step 4: click generate all paths button (the result is shown in Fig. 7)

Step 5: select a coverage criterion and click find an optimal path set button.

In step 5, we select all-branches coverage criterion, the optimal path set found by the testing tool is shown in Fig. 8. This testing tool also provides statistics data, as shown in Fig. 9, which shows that the size of the original path-branch coverage matrix is 247x51, i.e., there are 247 paths and 51 branches, and after applying the 5 reduction rules, the size of the matrix is reduced to 25x15, i.e., 25 paths and 15 branches.

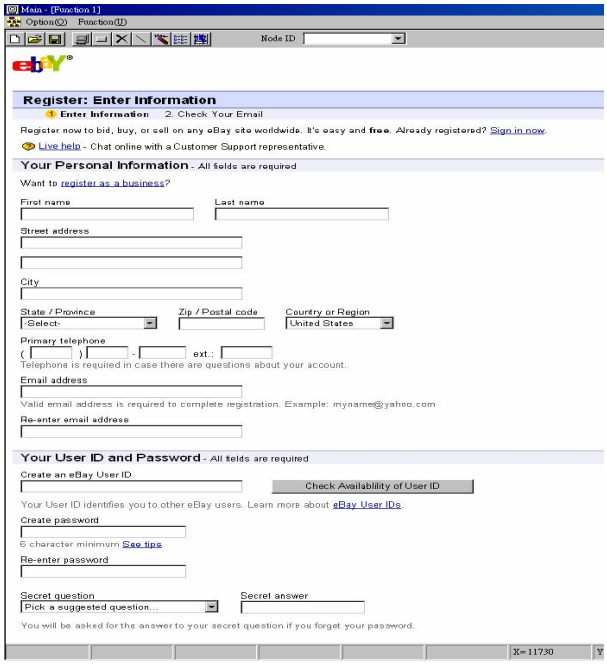

Fig. 4 Capture the UI under test

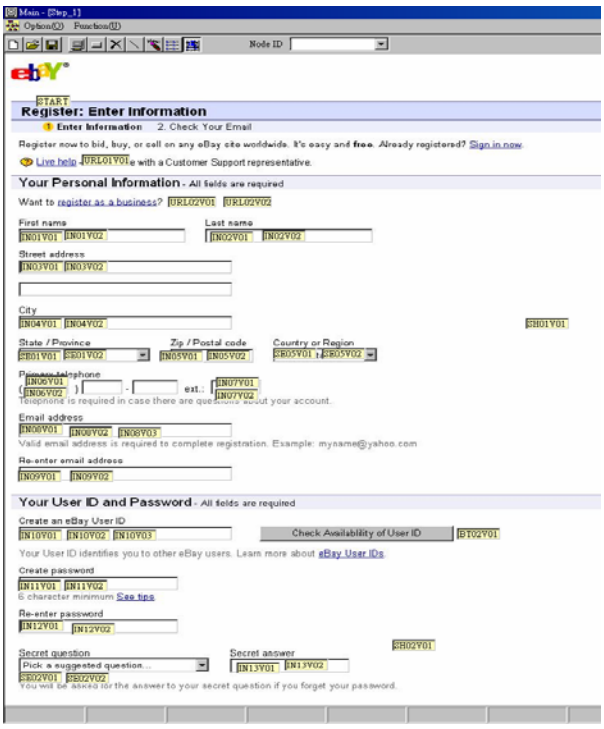

Fig. 5 Identify nodes

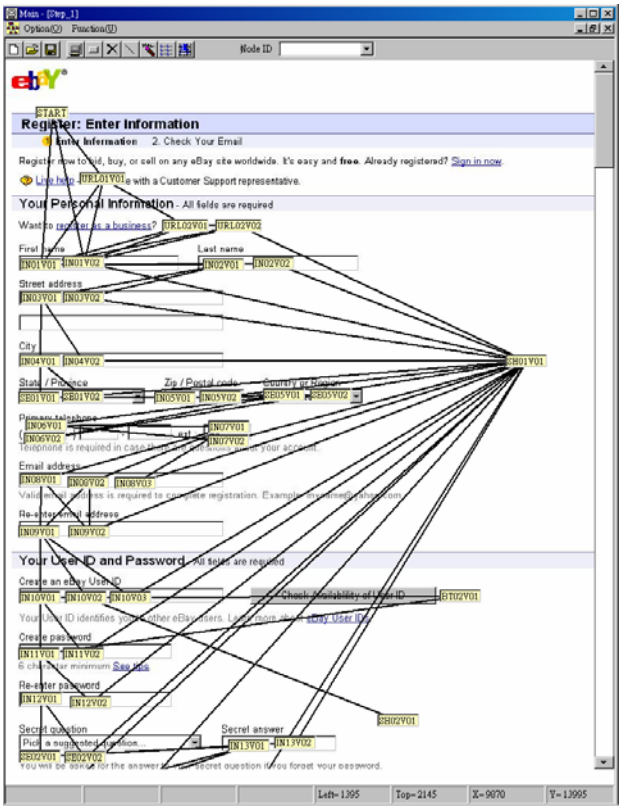

Fig. 6 Identify inter-node relationship

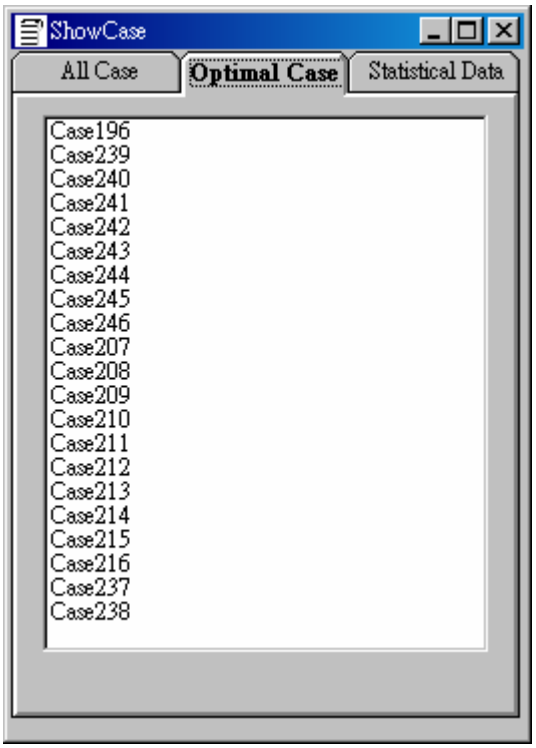

Fig. 8 An optimal path set for all-branches coverage criterion

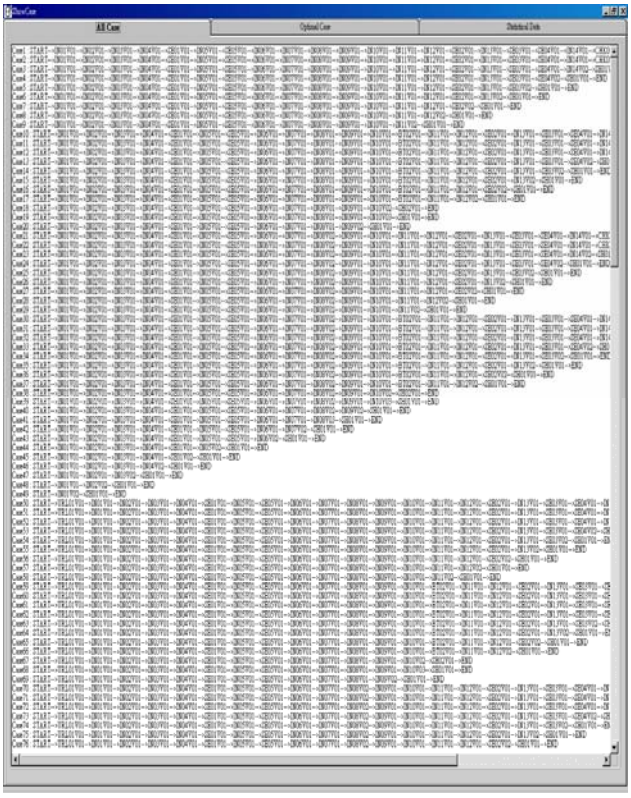

Fig. 7 listing of all paths

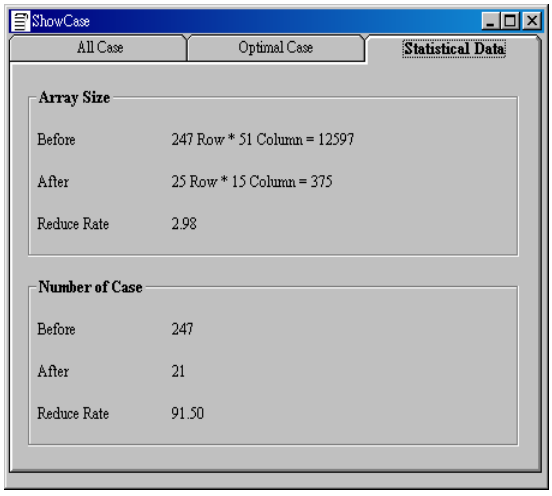

Fig. 9 Statistics data

4.2: Multiple webpage testing

Step 1: capture each webpage under test

Step 2: identify inter webpage relationship

Step 3: for each webpage, identify nodes, identify inter-node relationship, and click generate all paths button

Step 4: click generate all expanded paths button

Step 5: select a coverage criterion and click find an optimal path set button.

As mentioned ahead, the eBay registration process

has four web-pages. The inter-node relationship and inter-page relationship are shown in Fig. 10. In step 5, we select all-branches coverage criterion, the testing tool automatically generates a 332x142 matrix, i.e., there are 332 expanded paths and 142 nodes. After applying the five reduction rules, the matrix is reduced to 48x25, i.e., 48 paths and 25 nodes. For the reduced matrix, the testing tool takes only 4.562 ms to find an optimal path set.

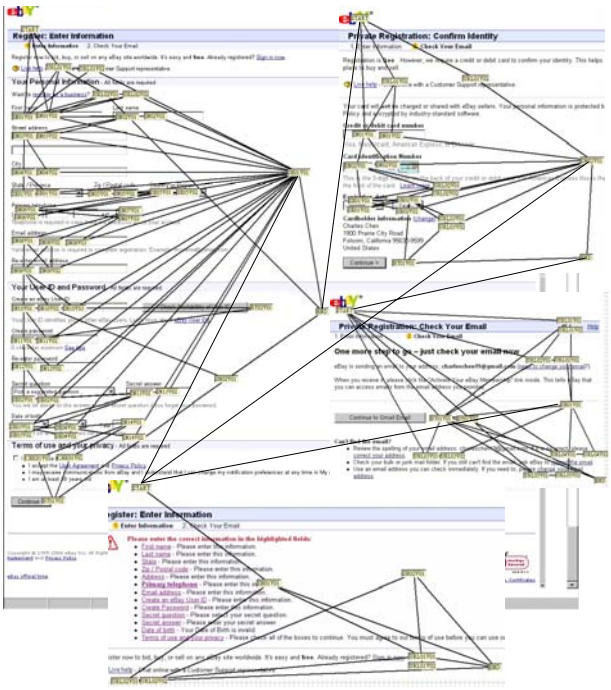

Fig. 10 Inter-page relationship diagram

5: Conclusions and future work

This paper proposes a simple and easy to follow black-box test case generation method for acceptance testing of GUI software. The idea of this method is from white-box path-based testing method. The zero-one optimal path selection method is used to find an optimal path set for a selected coverage criterion. Five reduction rules are used to greatly reduce computation time. A testing tool has been developed to facilitate the operation of the proposed method and an example is used to prove the applicability of the proposed method.

A control flow graph has three basic components, 1) sequential, 2) conditional branch and 3) loop. The examples in this paper does not include loop. In single web-page, it's hard to find loop. However, in multiple web-pages, loop appears in inter-page relationship. How to deal with loop is under study.

In multiple web-pages testing, Cartesian product is used to generate all expanded paths. However, not all expanded paths are feasible. For example, web-page 1 links to web-page 2, web-page 1 has 10 paths and web-page 2 has 8 paths. By Cartesian product, 10x8=80 paths will be generated. However, maybe only 5 paths of web-page 1 will go to web-page 2. This is a semantic issue, how to take this into consideration is under study.

REFERENCES

[1]. B. Beizer, Software testing techniques, Data System Analysis Inc., Pennsanken, New Jersey, 1984.

[2]. J. McCabe. A Complexity Measure, IEEE Trans. Software Eng. Vol. SE-2(4), pp. 308-320, Dec. 1976.

[3]. C. G. Chung and J. G. Lee. An enhanced zero-one optimal path set selection method, Journal of Systems and Software, 39(2):145-164, 1997.

[4]. J. G. Lee and C. G. Chung. An optimal representative set selection method, Information and Software Technology, 42(1):17-25, 2000.

[5]. LINDO, http://www.lindo.com/lindof.html

[6]. R. A. DeMillo, A. J. Offutt. Constraint-based automatic test data generation. IEEE Transactions on Software Engineering, 17(9):900-910, 1991.

[7]. K. F. Fischer. A test case selection method for the validation of software maintenance modifications. In proceedings of COMPSAC 77, Nov. 1977, pp.421-426.

[8]. W. T. Tsai, D. Volovik, and T. F. Keefe. Automated test case generation for programs specificed by relational algebra queries. IEEE Transactions on Software Engineering. 16(3):316-324, 1990.

[9]. M. M. Syslo, N. Deo, and J. S. Kowalik. Zero-one integer programming, Prentice-Hall, 1983, pp. 100-116.

[10]. R. D. Yang and C. G. Chung. Path analysis testing of concurrent programs. Information and Software Technology. 34(1):43-56, 1992.

Acknowledgment

This project is supported in part by the National Science Council of Taiwan, the CAISER of National Chiao-Tung University, and the Bestwise International Computing Co., Taiwan