A Combinatorial Approach to Building Navigation Graphs for Dynamic Web Applications

W. Wang\textsuperscript{1}, Y. Lei\textsuperscript{1}, S. Sampath\textsuperscript{2}, R. Kacker\textsuperscript{3}, R. Kuhn\textsuperscript{3}, J. Lawrence\textsuperscript{4}

\textsuperscript{1}University of Texas at Arlington
\textsuperscript{2}University of Maryland, Baltimore County
\textsuperscript{3}National Institute of Standards and Technology
\textsuperscript{4}George Mason University

9/24/2009
Outline

• Introduction
  • Basic concepts, Challenges
• Our approach
  • Abstract URL, Pairwise strategy, Algorithm design, Tool
• Experiments
  • Design, Subject applications, Empirical results
• Related Work
• Conclusion
Navigation graph

• A navigation graph represents the navigation structure of a web application.
  • A node represents a web page.
  • An edge represents one transition between two nodes.

• Usage: regression testing, impact analysis
  • Has an expected navigation path been implemented?
  • Has an unexpected navigation path been introduced?
  • What pages will be affected if one page is changed?
Challenges

• Page explosion problem
  • An astronomical number of dynamic web pages, possibly infinite web pages
  • Example: a web application may dynamically generate greeting pages for different users.

• Navigation structure capture problem
  • Some dynamic web pages may not be reached unless appropriate requests are supplied.
  • Example: searching flights in the studentuniverse web site.
Challenges

- Form parameters: departure city, arrival city, departure date, return date.
- City name: Dallas, Denver, Detroit, Edmonton.
- Date: Sep. 29, Sep. 30.

- home page->error page, captured by special combinations between two parameters.
  - the departure city is the same to arrival city.
  - the return date is before departure date.
- home page->searchResults page, captured by other ordinary combinations.
Outline

• Introduction
  • Basic concepts, Challenges

• Our approach
  • Abstract URL, Pairwise strategy, Algorithm design, Tool

• Experiments
  • Design, Subject applications, Empirical results

• Related Work

• Conclusion
Abstract URL

- One abstract URL represents a group of concrete URLs.
  - These concrete URLs have the same base component and the same parameters in the query component.
- Example:
  - $u1 = \text{"http://test.com/foo.jsp?x=1&y=2"}$
  - $u2 = \text{"http://test.com/foo.jsp?x=0&y=3"}$
  - $U = \text{"http://test.com/foo.jsp?x&y"}$
Pairwise strategy

- Given any two out of the $k$ parameters, we ensure that every value combination between any two parameters is covered in at least once.
- Our approach generates pairwise input combinations for forms to capture navigation structures behind forms.

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 2: Combinations from the exhaustive testing

<table>
<thead>
<tr>
<th>p1</th>
<th>p2</th>
<th>p3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 3: Combinations from pairwise testing
Algorithm design

Figure 4: Algorithm flow graph
Figure 5: Tansuo’s architecture
Tansuo’s architecture

• Builder:
  • drives the entire exploration process.
• Fetcher:
  • fetches a page from the web server.
• Parser:
  • extracts static links and forms from a page.
• Form Handler:
  • Obtains values for form parameters.
  • fills forms with combinations.
  • Obtains URLs from form submissions.
• Fireeye:
  • generates pairwise input combinations.
• State Manager:
  • Resets the database.
  • Re-exercises the path from starting page to the current page.
• Viewer:
  • displays the current page our approach is working on
Exploration demo

Figure 6: Exploration demo of Tansuo
Features of Tansuo

- Define exploration scope.
  - Define keywords for exploration scope.
  - Example:
    - Navigation structures for ordinary user.
    - Navigation structures for administrators.
- Semi-automated/automated exploration
  - GUI interface interaction.
  - Predefined files.
- Extract option values
  - Values of select menus
  - Values of check boxes
  - Values of radio buttons
  - Default values of text fields
Outline

• Introduction
  • Basic concepts, Challenges

• Our approach
  • Abstract URL, Pairwise strategy, Algorithm design, Tool

• Experiments
  • Design, Subject applications, Empirical results

• Related Work

• Conclusion
Experiment design

- Environment:
  - Hardware:
    - CPU: 1.66GHz, RAM: 2G, Hard disk: 80G.
  - Software:
    - Windows XP SP2, Resin 2.1.8 web server, Apache 2.0.48, MySQL Server 4.1.

- Subject applications:
  - www.gotocode.com
  - Use five jsp web applications because of using the Clover tool.
    - Get source code statistics of subject applications with Clover.
    - Clover processes only JSP web applications.
## Application statistics

### Table 1: Source code statistics of subject applications

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NLOC</td>
</tr>
<tr>
<td>Bookstore</td>
<td>18385</td>
</tr>
<tr>
<td>BugTrack</td>
<td>8094</td>
</tr>
<tr>
<td>Classifieds</td>
<td>11599</td>
</tr>
<tr>
<td>Links</td>
<td>8849</td>
</tr>
<tr>
<td>Portal</td>
<td>17621</td>
</tr>
</tbody>
</table>

### Table 2: Form statistics of subject applications

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Forms</td>
</tr>
<tr>
<td>Bookstore</td>
<td>18</td>
</tr>
<tr>
<td>BugTrack</td>
<td>8</td>
</tr>
<tr>
<td>Classifieds</td>
<td>11</td>
</tr>
<tr>
<td>Links</td>
<td>11</td>
</tr>
<tr>
<td>Portal</td>
<td>19</td>
</tr>
</tbody>
</table>

09/24/2009

Table 1: Source code statistics of subject applications

Table 2: Form statistics of subject applications
Results: navigation graph size

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
</tr>
<tr>
<td>Bookstore</td>
<td>93</td>
</tr>
<tr>
<td>Bug Track</td>
<td>43</td>
</tr>
<tr>
<td>Classifieds</td>
<td>50</td>
</tr>
<tr>
<td>Links</td>
<td>52</td>
</tr>
<tr>
<td>Portal</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 3: Size-statistics of generated navigation graphs

Notes:
Conn.(Connectivity): the average incoming and outgoing edges per node.
## Results: performance & cost

### Table 4: Time and memory usage

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>Total Time (hours)</th>
<th>State Restoration Time (hours)</th>
<th>Memory Usage (M Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bookstore</td>
<td>33.4415</td>
<td>27.7654</td>
<td>42.6328</td>
</tr>
<tr>
<td>BugTrack</td>
<td>0.1321</td>
<td>0.0641</td>
<td>19.5625</td>
</tr>
<tr>
<td>Classifieds</td>
<td>0.2999</td>
<td>0.2123</td>
<td>39.0078</td>
</tr>
<tr>
<td>Links</td>
<td>0.1275</td>
<td>0.0581</td>
<td>19.4570</td>
</tr>
<tr>
<td>Portal</td>
<td>1.2218</td>
<td>0.9519</td>
<td>80.3554</td>
</tr>
</tbody>
</table>

Notes:
Bookstore that contains a large number of images, which increased exploration time dramatically.
For example, a search result page for Bookstore contained 20 images, whereas a search result page for Portal contained no images.
Results: completeness

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>Manual</th>
<th>Tansuo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Edges</td>
</tr>
<tr>
<td>Bookstore</td>
<td>97</td>
<td>596</td>
</tr>
<tr>
<td>Portal</td>
<td>91</td>
<td>836</td>
</tr>
</tbody>
</table>

Table 5: Completeness result statistics

Notes:
Some nodes and edges are missed because of missing some complicated scenarios.
For example, the page-flipping is missed because our approach, for efficiency, just place one order in the ShoppingCartRecord page.
Table 6: Comparison results

<table>
<thead>
<tr>
<th>Subject Application</th>
<th>WebSphinx</th>
<th></th>
<th>LCP</th>
<th></th>
<th>Tansuo</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nodes</td>
<td>Edges</td>
<td>Nodes</td>
<td>Edges</td>
<td>Nodes</td>
<td>Edges</td>
</tr>
<tr>
<td>Bookstore</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>11</td>
<td>93</td>
<td>484</td>
</tr>
<tr>
<td>BugTrack</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>43</td>
<td>175</td>
</tr>
<tr>
<td>Classifieds</td>
<td>15</td>
<td>16</td>
<td>9</td>
<td>9</td>
<td>50</td>
<td>313</td>
</tr>
<tr>
<td>Links</td>
<td>11</td>
<td>12</td>
<td>11</td>
<td>11</td>
<td>52</td>
<td>259</td>
</tr>
<tr>
<td>Portal</td>
<td>17</td>
<td>22</td>
<td>17</td>
<td>22</td>
<td>80</td>
<td>652</td>
</tr>
</tbody>
</table>

Nodes:
LPC: Link Checker Pro.
VeriWeb is not public accessible.
Outline

• Introduction
  • Basic concepts, Challenges
• Our approach
  • Abstract URL, Pairwise strategy, Algorithm design, Tool
• Experiments
  • Design, Subject applications, Empirical results
• Related Work
• Conclusion
VeriWeb [WWW 02]

- Page explosion problem
  - Solution: sets length limits on navigation paths.
  - Results:
    - Can not address the page explosion problem indeed.
    - May cause losing navigation structures.

- Navigation structure capture problem
  - Does not consider input combinations for forms.
    - May miss navigation structures behind forms.
WebSphinx [WWW 98]

• Page explosion problem
  • Does not consider the page explosion.
    • Uses concrete URLs as nodes directly.

• Navigation structure capture problem
  • Can not handle forms.
    • Misses navigation structures behind forms.
Google’s deep-web crawl
[VLDB 08]

- Page explosion problem
  - Solution: uses content discovery strategy to pick pages with most information.
    - Example: “Login” page will be discarded because it contains little information.
  - Results: loses navigation structures

- Navigation structure capture problem
  - Solution: uses bottom-up fashion to generate input combinations for forms.
    - In fact, this solution works like exhaustive testing, which may produce a huge number of test cases.
  - Results: causes low efficiency.
Conclusion

- Our approach is effective for generating practical navigation graphs.
  - Abstracting URLs controls navigation graph size effectively.
  - Pairwise input combinations of forms help capture most navigation structures.

- Future work:
  - Constraint support.
  - Improve the efficiency of state restoration.
  - Improve user interface.
References


Thanks!