# Path Analysis in Web Page Application

**Sonali Pradhan, Mitrabinda Ray**

***Abstract:*** ***The key to a web application is the information and a variety of facilities and features present according to the needs of the user's requirements. For the superior quality and maintenance of the website, new methodologies and tools are developing day by day. In this paper, we propose an algorithm using the concept of existing A* algorithm to get the shortest path from the home page to the target page. Before reaching the target page, the user visits other pages. Our motto here is to reduce the longest shortest path of a site, which is an indicator of better structure. For this, we propose a web model. A web model is an intermediate representation of a given web application, which is designed based on pages, frames and links of the application. The A* algorithm is not competent to get the Unreachable pages on the website. We create our sample HTML web page (a case study) to observe Unreachable pages (broken links), frame count and, the order of the frames. The web page is embedded in various websites such as www.learn-automation.com, www.seleniumhq.org, and www.html.com. Frame testing is conducted on the proposed web model using a well-known tool, Selenium web driver. The Selenium tool shows how the frames are moving from one frame to other (order of frame execution), frame count, confirm the broken links or unreachable pages and the ghost pages in our sample HTML web page. These methodologies are very useful for the implementation and maintenance of websites.***

***Index Terms: frame testing, Selenium web driver, static analysis, web application, web browser, web model***

## I. INTRODUCTION

The web application is a client-server interaction of software application on the web. A client-server environment is sharing information with multiple computers environment, like time-sharing computing environment, personal computing environment, distributed computing environment, client-server computing environment etc. The need for the web application is essential in day to day life. Without the internet, it is tough to fulfil customer demands. A web application uses a web browser as a client. The client uses some specific web browser like Google Chrome, Mozilla Firefox, and Internet Explorer etc. The web application uses the client-side script such as HTML, JavaScript, etc. and server-side script such as ASP.NET, PHP, J2EE, Perl/Plack etc.

**Revised Manuscript Received on 30 January 2019.**
* Correspondence Author

**Sonali Pradhan,** Computer Science and Engineering , SOA, University, Bhubaneswar, India.
**Mitrabinda Ray,** Associate Professor in SOA, University, Bhubaneswar, India

Maintenance of websites is becoming a crucial issue on the web application — some central issues the users find at the time of accessing the internet as decreasing speed of the internet, link failure, unreachable pages, page error etc.

For the maintenance of websites, various tools and techniques are increasingly developing, and more and more advanced technology with advanced features and appealing interfaces are coming to the market [1] So for the advancement of excellence software systems, the production of reliable and high-quality websites are desired with proper adopted methodology and techniques. In the early days, the web application is delivered to the client as a static document, where the user only gives the request to the server, and it returns user only viewing the documents. Gradually, the dynamic document is possible by giving user input through a web form and getting back the information from the server with refreshing the required part without refreshing the whole or entire page [2, 3]. To find out possible faults and anomalies, HTML pages of the website are examined. Here, we are centralising on the website for verifying the Unreachable pages/broken links and Ghost pages. Unreachable pages mean pages are available but cannot be reached along any path starting from the initial page. Ghost pages say the pending links or non-existing pages of the website. We also conduct frame testing to examine whether the frames of the web page is correctly working or not. A frame is a part of a web page or web browser, which displays its content independently. A page decomposed into frames can interact with each other. A web page is decomposed into frames and pages. To reach a target page, different representation of the structure has to be created. First, we propose an algorithm using the concept of existing *A* algorithm* to get the reachable pages in the website. Then we create a sample HTML web page to examine the Unreachable pages. We have different websites embedded in one web page using HTML iframes. An HTML iframe (Inline Frame) is an HTML page embedded inside another HTML page on a website [4]. We check the order of frame moving and total frame count in a given web page then we checked whether our application is behaving correctly or not. For this type of verifications and analysis, the structure of the web model [5] is essential. A web model is a general concept, addressing some certain issues in the design and development of web applications which describes the different paths in the model to reach the user's goal on a website. Here, we choose Selenium web driver tool of version (2.53.1) [6]. We conduct frame testing to verify how the frames are working in order, frame count etc. It also finds out the broken links in the corresponding web page.

The paper is distributed as follows: Section 2 describes the background study, Section 3 describes the related work, Section 4 describes the analysis that is observed in the structure of the website, Section 5 describes the frame testing with different web browsers and testing of broken links through Selenium tool, Section 6 elaborates comparison with the existing work and finally, the conclusion is drawn in Section 7 followed by the future work.

## II. Background Study

In this section, we discuss the state-of-the-art web-based testing techniques which are widely used in software testing literature.

### A. Web application

Web application [7] is a program, that process at the server side and the corresponding result is displayed at the client side or client browser. The browsers supported some combinations of languages such as HTML, XML, CSS, JSP, Servlet, JavaScript and DOM programming interface etc. to process the request in the server side and display the response of the server in the client browser. There are mainly two types of web applications such as static web application and dynamic web application.

***Static web application:***

According to the client request, the static web pages are generated through the server, and it's always displayed in the client browser. The static web pages are always fixed, and the context of static pages are the same for all clients on the web. The static web pages are available to the server through HTTP request with HTML format. However, every application with .html extension is not always applicable to static web application [8]. HTML frames divided the client browser into some frames and separate HTML document is defined for each frame. A group of frames is known as a frameset. Iframe is inline frame with HTML tag <iframe> [4].

***Dynamic web application:***

It is written in the scripting language. It is compiled on the server side and displayed on the client side. The dynamic web application is developed by languages like JSP, PHP, Servlet, RUBY etc. It is not so easy to develop the dynamic web application. Developers get some benefits in developing dynamic web applications such as enhancing the functionalities of the website; new contents can be added, easy to update the website and to collaborate different sites etc. [8, 9]. To collect some data from the site visitors, users use HTML forms. A form took input from the site user and processed it at the back-end application. After processing, it retrieved the data from the database and with the server response, send back the information to the client browser. There are various forms available such as radio buttons, text area, drop down menus, check boxes etc.

### B. *Testing Techniques*

There are some important testing techniques we find to perform the testing activities according to the requirement.

***Model-Based testing of the web application:***

A web model is created for a target application using reverse engineering and web crawler [10, 11, 12]. Software reverse engineering is done to retrieve the source code of a program. The web model is a graph with a root node, the starting web page, and all other nodes linking to the root node, are known as link pages. The link connecting from one page/node to other page/ node is known as an edge. A web model is a combination of nodes and edges, where each node represents a page, and each edge represents a link. In the web application, the links are some HTML links that are generated from submits, automatic redirections etc. In web page testing, a test case is composed of a sequence of pages and input values [10, 11, 12]. By traversing the navigational web model, the test cases are excerpted.

***Code-Coverage Testing of the web application:***

In Code-Coverage Testing [13], a control flow model is generated from the source code. The objective is to generate test cases to cover all statements as well as branches of the program based on a control flow model. The generated test suite determines the level of code coverage of the program. Some necessary tools used for the Code-Coverage Testing are Java Eclimma [14], Clover [15] etc. The disadvantages of Code-coverage testing are- time consuming and expensive, some lines of codes are also get missing at the time of Code-Coverage, and depth of knowledge about the programming language is very much essential in the cases of Code-Coverage testing [13].

***Mutation-Based Testing of the web application:***

The objective of previous testing approaches discussed above is to find the faults in the source code, whereas in the mutation-based testing, the objective is to measure the efficiency of the generated test cases. More traditional uses of mutation are imposing mutants to the source code and making the test suite exercised to find out the seeded mutants and to test the efficiency of the test cases [16]. In the context of web applications, Mutation-Based Testing is used to recover web model [16].

## III. Related Work

For the website development process, researchers suggest various analysis to develop the testing of web applications. In static analysis, the website structure and historical analysis are focused. History analysis indicates the modification that is not in the original design or the modifications that produce undesirable effects. Dynamic analysis tracks on the information like the DBMS tables, the frequency of exercising a particular link and the session and cookie data etc. [17]. In paper [18], objects are focused on their history and structure analysis. The purpose of focusing these items are carrying out the maintenance activity. Here several websites are chosen for analysis. To study the history, the sites are downloaded every day by ReWeb tool. A summary of their features like number of links, the line of codes, number of HTML pages is recorded. Results of the analysis are provided to the user by exploiting different visualisation technique. Using structural analysis gives very powerful navigational facilities.

When the web site immigrates maintenance and needs to improve its quality, a set of analysis is applied, that is discussed in [17]. Using with the traditional software system, some of the proposed analyses are derived. The analysis is divided into two categories. First one is the analysis of the structure and the second one is the analysis of the evolution. A graph represents the web model structure. To traverse the directed graph, they use flow analysis, traversal algorithms, and pattern matching. To represent website evolution, the website can be characterized by using colours, which is a time indicator. History of individual pages and links can be shown to the user in a compact and expensive form which degrades the original structure. The ReWeb tool is used to implement the above criteria. Using the tool, results provided to the user are exploiting different visualisation techniques. With great navigation facilities, structural and system views are improved while colours are employed with system view. The demerit is that the tool is not sufficient to capture the HTML web elements in the dynamic analysis. So, the tool needs modification to improve the quality as well as for the improved dynamic analysis of the web pages. After six years the same paper [17] is followed-up. A method is proposed in 2002 to reverse engineer a web application model. Page merging heuristics and dynamic analysis were used for extracting the model. The web model which is proposed in 2002, was very much adopted but features of future web applications challenge its future applicability.

In paper [19], they have used a formal model for verification of websites and applications using the model checker technique. To validate the proposed approach, they implement a prototype system. The analyser resolves the links and pages. Then it evaluates the dead links, the structure of the websites reachable & unreachable pages and somehow checking the dynamic pages too. In the paper [20] they give an exciting proposal for structural analysis. They propose white box testing for static web applications. There a navigational model focuses on HTML pages and navigational links of the web applications. The paper [21] describes the difference between web-based application and traditional testing, where they discuss a various type of structural analysis for web applications. In that paper, they have discussed various aspects of testing web application, such as test models, testing tool, and the scope of testing as well as testing strategies.

Our study complements and extends existing research on the following:

- Shortest path using a suggested algorithm, implemented to our proposed model
- Frame testing with different web browsers
- Quality of the web page

## IV. Proposed Web Model for Structural Analysis

A conceptual model is proposed to represent the structure analysis of the website. A website is identified, all information is retrieved from a given web server. Documents accessed through different servers, the website is organised as the structure into pages and linked between pages. To reach a document, it is impossible without navigating other pages in the website, which is called *dominators* of the page of interest and provides a site as *.html, to the server. The structure of the web model with frames is represented through a directed graph. The directed graph G is expressed as G = (N, E), where $n \in N$ represents a single HTML page, and N represents a set of pages or nodes. E represents a set of edges or links between $n_1$, $n_2$, where $e = (n_1, n_2) \in E$, which connects two nodes $n_1$, $n_2$ if there is an HTML link from the page $n_1$ to page $n_2$. Here, we consider pages containing frames. The primary website model can be represented as $N = N_1 \cup N_2 \cup \mathcal{F}$, where $f_1 \in \mathcal{F}$, $f_2 \in \mathcal{F}$ …and $\mathcal{F}$ is the set of all frames. Edges can be split into two subsets as $E = E_1 \cup E_2$. Here, $E_1$ is a subset of $N_1 \times (N_1 \cup N_2)$, and an edge $e = (n_1, n_2) \in E_1$ imitates the presence of a normal link from page $n_1$ to $n_2$ if $n_1 \in N_1$ and $n_2 \in N_1$. Decomposition of page $n_1$ into the frames in $n_2$ occurred, when $n_1 \in N_1$ and $n_2 \in N_2$. $E_2$ is the set of edges that loading the Initial page into a frame.

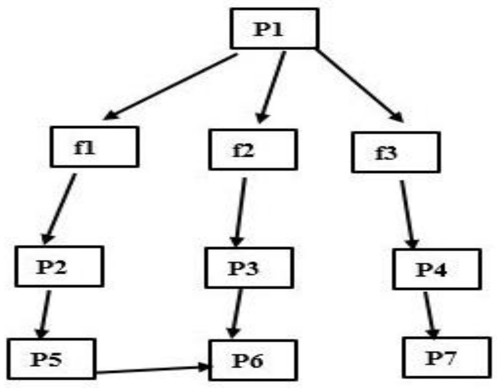


**Fig 1. Example of a website model including page decomposition into frames and reaching the required documents with the shortest path**

Fig 1 illustrates an example of a proposed web model. The link between $P_2$ and $P_5$, $P_5$ and $P_6$, and between link $P_3$ and $P_6$ are normal navigation of HTML pages as ($E_1$). $E_1$ has set of links between pages as $\{P_2 - P_5, P_5 - P_6, P_3 - P_6, P_4 - P_7\}$. The link between $P_1$ and {f1, f2, f3}, indicates that page $P_1$ decomposes to f1, f2, f3 where the pages $P_2$, $P_3$, $P_4$ are initially loaded to the frames $f_1$, $f_2$, $f_3$ respectively ($E_2$). $E_2$ has set as $\{P_1\text{-} (f_1, f_2, f_3), (f_1 - P_2, f_2 - P_2, f_3 - P_4)\}$. In the web model, $N_1$ has set of nodes $\{P_5, P_6, P_7\}$ with simple HTML pages without frames, and $N_2$ have a set of nodes $\{P_1, P_2, P_3, P_4\}$ pages with frames. Page $P_6$ is considered to be the target page in this example. In the flow analysis of the web structure, $P_1$ is the initial node or home page. Subsequently, nodes covered as $P_2$, $P_3$, $P_4$, $P_5$, $P_6$, and $P_7$. Observing Fig 1 of our web model, we have three *dominators*, to reach $P_6$ (target page).

The *dominators* in the dominator analysis are the root node $P_1$, the frames $\{f_1, f_2, f_3\}$ and other nodes ($P_2$, $P_3$, $P_5$) covering $P_3$. The shortest path from $P_1$ to $P_3$ has weight 0, having the sequence $P_1$, {f1, $f_2$, $f_3$}, {$f_2$, $P_3$ }. As pages $P_2$, $P_3$, $P_4$ are automatically loaded into the frames $\{f_1, f_2, f_3\}$, weight count is 0. The link selection or weight is 1 for $P_1$ to $P_6$, since the requirement for navigation is from $P_3$ to $P_6$ (target page).

Searching for a given document is very difficult when the path is long. So, the developer always tries to provide a well-designed website to reach the documents with the shortest path.

## A. *Reaching frames and frames to pages*

The computation of reaching frames and reaching frames to pages is essential to determine the set of frames in which the page appears. The set of frames are treated as an initial page. In our web model in Figure 1, the set of three frames are in the initial page $P_1$. Computation of reaching frames and the frames to pages on a website is discussed in the frame analysis framework [22]. The propagation of flow information is the basic idea behind the flow analysis.

The flow equations required for the computation of reaching frames will be:

$$FLOW_n = \{n\}\ IF\ n \in F$$
$$FLOW_{n=\{f\}\ IF\ \exists(m,n)} \in E_n$$
$$FLOW_{n} =\emptyset\ otherwise$$
$$FLOWKILL_n = F\ IF\ n \in F$$
$$FLOWKILL_n = \emptyset\ otherwise$$

$$FLOWIN_n = \bigcup_{p\, \in\, pred\,(n)} FLOWOUT_p$$
$$FLOWOUT_n = FLOW_n \cup \{FLOWIN_n \backslash FLOWKILL_n\}$$

A frame, $(n \in F)$, represents n is a node generates flow information, $(FLOWKILL_n = F)$ says, it overrides any previous frame. The flow information representing f is generated, but flow information is not killed. That shown in the notation ($FLOWKILL_n = \emptyset$). The notation (pred (n)), the set $FLOWIN_n$ of each node n collects information about all closing frames with $FLOWOUT$ sets of predecessors of n (pred (n)). Here, $FLOWOUT$ set is acquired by subtracting the $FLOWKILL$ set and adding with the $FLOW$ set. The site server provides the root of the graph associated with the first page. The graph has an empty $FLOWIN$ set. Until a fixed point is reached, the flow of information is repeatedly propagated to the graph. According to Figure 1, flow propagation is occurred along the edges $E_1, E_2,$ and $E_3$ and from each node n $=\{f_1, f_2, f_3\}$. The resulting analysis is found in the $FLOWOUT$ set of each node, and that is the collection of all frames into the page associated to the node.

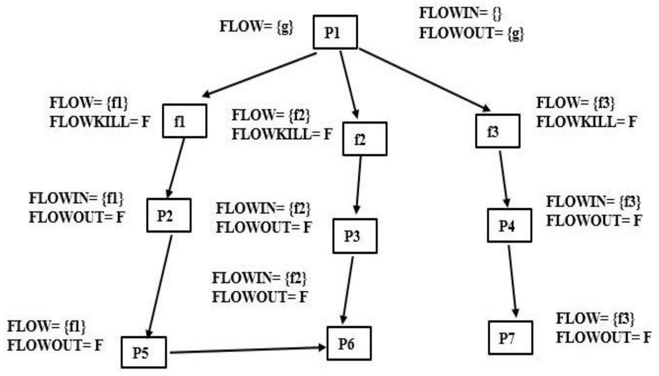


**Fig 2. Frame computation for reaching frames to pages**

The result of the reaching frames and frames to pages is shown in Fig 2. The fixed point is reached to get the proper node after the propagation flow information. This propagation flow information is determined using $FLOWIN$ and $FLOWOUT$ sets given in Figure 2. In that Figure 2, $P_1$ is the top level node. All other pages are displayed in $\{f_1, f_2, f_3\}$ frames. If an edge added from $P_1$ to $P_6$ then $FLOWIN = FLOWOUT = \{g, f_2\}$. From this, it is clear that $P_6$ is loaded at $f_2$ or at the top level. By this reaching frame and frames to page analysis, the presence of unwanted pages and frames can be traced out.

Fig 2 shows the frame computation of Figure 1. The initial page of this site $P_1$ has a label g in its FLOW set to denote the top frame. Frames $f_1, f_2, f_3$ generate themselves as flow information, overriding any incoming data. The $FLOWIN$ and $FLOWOUT$ sets are determined, after propagating the flow information until the fixed-point is reached.

## B. *Shortest path*

A website may require traversing several pages to reach a document. Useful information about a website is the minimum number of pages that must be visited before reaching a target document. In a graph, the root node taken as m, the shortest path from m to any other node n is the path from m to n with the minimum total weight associated with the edges if there is real user selection, which is a “mouse clicks”, then the edge is weighted as 1. Otherwise, weight is 0, as it is loaded automatically. We suggest an algorithm named Shortest_path_A*, to find the shortest path from the root node to any target node in a web model, using the concept of A* algorithm. A* algorithm is known as a heuristic featured algorithm or informed search algorithm, which calculates f(n) = g(n) + h(n), where f (n) is the heuristic function, g(n) is the distance/cost of a node n from the initial node, and h(n) is the distance/cost of a node n from goal node. For the implementation of the algorithm, we use two lists as OList (open list) and CList (closed list). The proposed Shortest_path_ A* algorithm is used to satisfy the covering nodes in the web model is as follows:

```
Shorest_path_ A* ALGORITHM
//INPUT1: weighted web model  //weights are assigned to the links, for the "mouse clicks", the edges are weighted as 1 otherwise weights 0, when it loads automatically.
//INPUT2: targeted page
Step 1: Place the initial node or start node on to an OList. //P1 placed in OList
Step 2: Remove from OList and process it and place onto the CList.  // P1 placed in CList
Step 3: If it is a goal node, return success and stop the process.
// Not a goal node
Otherwise
Step 4: Place the children node if any on to the OList.  // {P2, P3, P4} placed in OList
Step 5: Calculate heuristic value [f (n)] of each node in the OList.
```

Step 6: The node having minimum heuristic value will be removed from OList and placed into the CList. // P3 is placed to CList as f (n) value of P3 is 1, and that is the minimum heuristic value
Step 7: The process will be continued until the goal node is reached.
Step 8: End the process.

## C. *Explanation of the algorithm*

P1 is the initial node, and P6 is our goal node in the web model. Initially, the two lists OList and CList are empty. OList is used to place the Initial node and the associated children nodes. CList is used to place the heuristic value of the initial node taken and the corresponding minimum heuristic values among the children nodes in the web model. Here, f (n) values of all existing nodes are calculated in the web model. The initial node P1 is placed in the OList. Then f (n) value of P1 is calculated. We get the f (n) value of P1 will be 3 calculating f (n) = (0+3) = 3. If P1 is a goal node, then it returns success. Otherwise, it will Place the children nodes P2, P3, P4 in the OList. Calculating f (n) value of P2, P3, P4, we get f (n) value of P2, P3, P4 will be 2, 1, and 5 respectively. The node P3 has minimum heuristic value among all other nodes P2, P4. So P3 will be placed in CList. Now, we get P3 is not a goal node. Then the children nodes P2, P4, P6 are placed in the OList. Again calculating f (n) value of P2, P4, P6, we get 2, 5, and 1 respectively. The node P6 has minimum heuristic value will be placed in the CList, which is a goal node. We get the f (n) value of P5 and P7 will be 2, and 7, which are bigger than the f (n) value of P6. So P6 is our goal node, placed in CList. So, after getting the goal node, the process is stopped.

Static Verification

To detect possible faults and anomalies, static analysis can be done by scanning the HTML pages on the website. Static checks can be done with navigation paths provided to the user as well as data flows of the information gathered by the user. More discussion on static analysis is given in the paper [18]. In the static analysis the presence of unreachable pages, ghost pages can be detected. For the unreachable pages, pages are found in the server side, but it cannot reach to the client side given to the user starting from the initial page to any path covered in the same way ghost page associated with pending links, which are linking to non-existing pages. For the static verification testing, All-path has an essential factor to detect faults in the web. In our paper, we apply the same method to find unreachable pages or failure links as well as for frame testing in the subsection (Testing with selenium web driver). We have embedded three frames and checks for static analysis using some criteria as:

I. Page testing- In some test cases every page is visited at least once.
II. Hyperlink testing- In some test cases every hyperlink from every page in the website traversed at least once.
III. All-paths testing- Every given path should be traversed at least once.

## D. *Testing with Selenium web driver*

The utilisation of A* algorithm is very much helpful for reachable pages without covering unnecessary/pointless pages. However, to find out Unreachable pages or broken links, A* algorithm is not competent. So to get the Unreachable pages, we utilise different methodologies by implementing an HTML sample page. In this section, we have taken three nested frames. An HTML code is composed for the nested frames (f1, f2, f3) for our web model shown in Fig 1. In the nested frame, we find f1 frame has website http://www.learn-automation.com, f2 frame has website http://www.seleniumhq.org, and the f3 frame has web site http://www.html.com. We conduct testing of nested frames, and we test to get the Unreachable pages and ghost pages. The Selenium tool shows how the frames are moving from one frame to other (order of frame execution), frame count, and confirm the broken links or Unreachable pages.

HTML CODE:

```
<! DOCTYPE html>
<Html>
<Body>
<iframe src="http://www.learn-automation.com"
width="400" height="400" name="selenium">
<p>your browser does not support iframe. </p>
</iframe>
<iframe src="http://www.seleniumhq.org” width="400"
height="400" title="selenium_news">
<p>your browser does not support iframe. </p>
</iframe>
<iframe src="http://www.html.com" width="400"
height="400" id="html">
<p>your browser does not support iframe. </p>
</iframe>
<br>
<a href="http://www.learn-automation.com">
Click here for selenium tutorial</a>
</body>
</html>
```

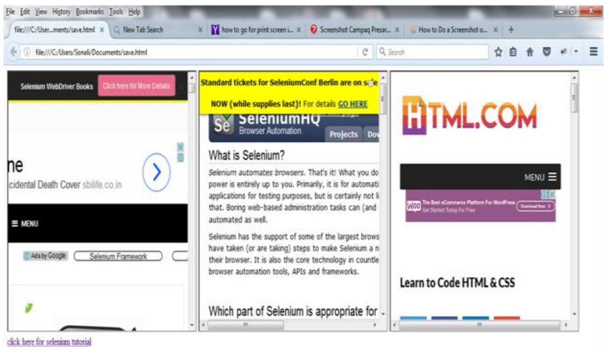
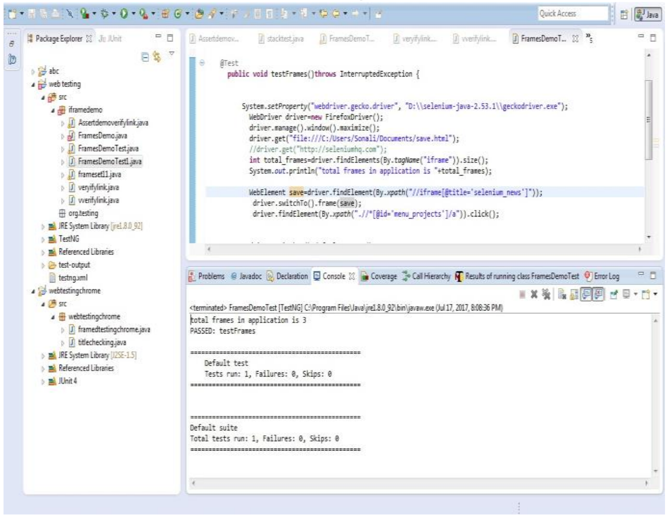

File Edit View History Bookmarks Tools Help
file:///C:/Users/Sonali/Documents/save.html
Selenium WebDriver Books
Click here for More Details
MENU
Selenium Framework
Standard tickets for SeleniumConf Berlin are on sale
NOW (while supplies last)! For details GO HERE
SeleniumHQ Browser Automation
What is Selenium?
Which part of Selenium is appropriate for
HTML.COM
MENU
Learn to Code HTML & CSS
click here for selenium tutorial

Fig 3. Embedded frames

Fig 3 shows an embedded frame using the above HTML code. In this figure, three websites are embedded into one web page.

The websites we have taken are www.learn-automation.com, www.seleniumhq.org, www.html.com. With the help of Selenium web driver, we found the frame count as well as the order of the frames in the web page.

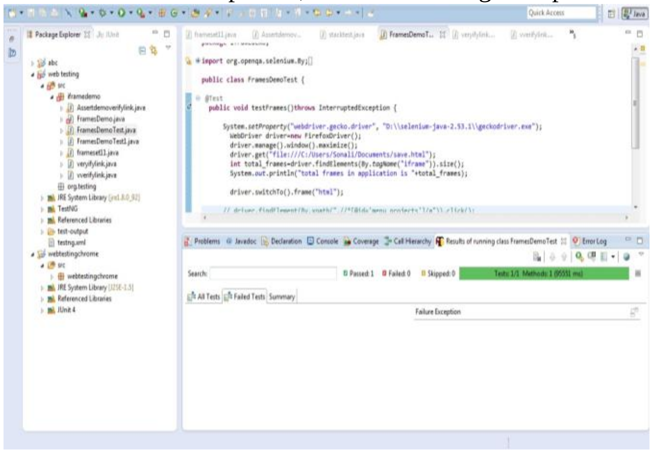


```
@Test
 public void testFrames()throws InterruptedException {


     System.setProperty("webdriver.gecko.driver", "D:\\selenium-java-2.53.1\\geckodriver.exe");
      WebDriver driver=new FirefoxDriver();
      driver.manage().window().maximize();
      driver.get("file:///C:/Users/Sonali/Documents/save.html");
      //driver.get("http://seleniumhq.com");
      int total_frames=driver.findElements(By.tagName("iframe")).size();
      System.out.println("total frames in application is "+total_frames);

      WebElement save=driver.findElement(By.xpath("//iframe[@title='selenium_news']"));
       driver.switchTo().frame(save);
       driver.findElement(By.xpath(".//*[@id='menu_projects']/a")).click();
```

```
<terminated> FramesDemoTest [TestNG] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (Jul 17, 2017, 8:08:36 PM)
total frames in application is 3
PASSED: testFrames

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

**Fig 4. Frame testing with Firefox web browser**

Fig 4 shows the frame testing which is done using the Firefox web browser (47.0.1) and makes our test frames passed. As shown in Fig 3, the frame count is 3 in the given web model (Fig 1). Using findElement () method, we count the total size of frames in the web model. Total size is 3 (shown in Fig 4) for the given web model. This type of testing is beneficial when some frames are embedded in one website. To switch from one frame to the other, firstly we have to come to the parent frame from any other frame. Here, we have taken our parent frame at www.learn-automation.com. This is the proper way to identify the frames when internally the number of frames are more and not possible to count the frames visually. With the iframe testing, the tester can switch to the first frame, do some operation and moves to the second frame, third frame etc. The console window shows that the testing makes our test frames passed, and it is working as expected.

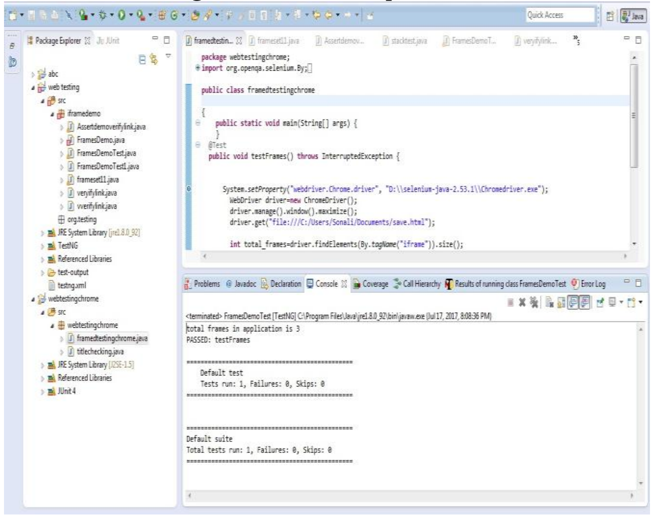


```
import org.openqa.selenium.By;

public class FramesDemoTest {

 @Test
  public void testFrames()throws InterruptedException {

     System.setProperty("webdriver.gecko.driver", "D:\\selenium-java-2.53.1\\geckodriver.exe");
      WebDriver driver=new FirefoxDriver();
      driver.manage().window().maximize();
      driver.get("file:///C:/Users/Sonali/Documents/save.html");
      int total_frames=driver.findElements(By.tagName("iframe")).size();
      System.out.println("total frames in application is "+total_frames);

      driver.switchTo().frame("html");
```

Search: Passed: 1 Failed: 0 Skipped: 0 Tests: 1/1 Methods: 1 (9551 ms)

**Fig 5. Result for frame testing using TestNG in Firefox web browser**

In the Selenium framework environment, it has two windows as a console window and TestNG window. The console window makes the testing pass or fail, and the TestNG window gives the summary of the testing used. In the same Selenium environment, TestNG is a testing framework inspired by JUnit and it supports a variety of tools and plug-ins such as Eclipse, IDE, and Maven etc. TestNG provides so many additional functionalities, using them tester can create a very robust framework itself. Reports are auto-generated by TestNG and for the Selenium web driver, TestNG is the default framework. Fig 5 shows the result for the running class we have used and the complete report of the testing. The report in the Fig 5 shows the testing of the frames as pass.

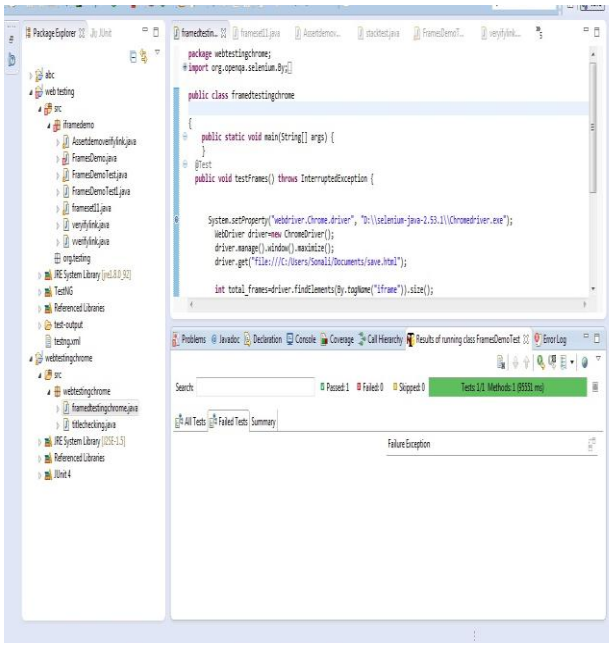


```
package webtestingchrome;
import org.openqa.selenium.By;

public class framedtestingchrome

{
   public static void main(String[] args) {
   }
 @Test
 public void testFrames() throws InterruptedException {


     System.setProperty("webdriver.Chrome.driver", "D:\\selenium-java-2.53.1\\Chromedriver.exe");
      WebDriver driver=new ChromeDriver();
      driver.manage().window().maximize();
      driver.get("file:///C:/Users/Sonali/Documents/save.html");

      int total_frames=driver.findElements(By.tagName("iframe")).size();
```

```
<terminated> FramesDemoTest [TestNG] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (Jul 17, 2017, 8:08:36 PM)
total frames in application is 3
PASSED: testFrames

===============================================
    Default test
    Tests run: 1, Failures: 0, Skips: 0
===============================================


===============================================
Default suite
Total tests run: 1, Failures: 0, Skips: 0
===============================================
```

**Fig 6. Frame testing in Google Chrome web browser**

Again the same frame testing is done with Google Chrome (59.0.3), which is shown in Fig 6. Firefox web browser is very compatible to Selenium web driver, but the processing speed is much faster in Google Chrome rather than Firefox web browser.



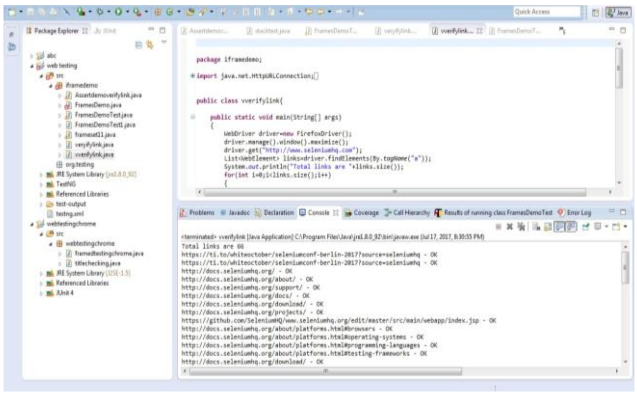
```
package webtestingchrome;
import org.openqa.selenium.By;

public class framedtestingchrome

{
   public static void main(String[] args) {
   }
 @Test
 public void testFrames() throws InterruptedException {


     System.setProperty("webdriver.Chrome.driver", "D:\\selenium-java-2.53.1\\Chromedriver.exe");
      WebDriver driver=new ChromeDriver();
      driver.manage().window().maximize();
      driver.get("file:///C:/Users/Sonali/Documents/save.html");

      int total_frames=driver.findElements(By.tagName("iframe")).size();
```

Search: Passed: 1 Failed: 0 Skipped: 0 Tests: 1/1 Methods: 1 (9551 ms)

**Fig 7. Result for frame testing using TestNG in the Google Chrome web browser**

Fig 7 shows the result of the class we have used for the Google Chrome web browser, showing in green.

In this section of TestNG, it shows the summary of our testing method.



```
package iframedemo;

import java.net.HttpURLConnection;

public class vverifylink{

    public static void main(String[] args)
    {
        WebDriver driver=new FirefoxDriver();
        driver.manage().window().maximize();
        driver.get("http://www.seleniumhq.com");
        List<WebElement> links=driver.findElements(By.tagName("a"));
        System.out.println("Total links are "+links.size());
        for(int i=0;i<links.size();i++)
        {
```

```
<terminated> vverifylink [Java Application] C:\Program Files\Java\jre1.8.0_92\bin\javaw.exe (Jul 17, 2017, 8:30:55 PM)
Total links are 66
https://ti.to/whiteoctober/seleniumconf-berlin-2017?source=seleniumhq - OK
https://ti.to/whiteoctober/seleniumconf-berlin-2017?source=seleniumhq - OK
http://docs.seleniumhq.org/ - OK
http://docs.seleniumhq.org/about/ - OK
http://docs.seleniumhq.org/support/ - OK
http://docs.seleniumhq.org/docs/ - OK
http://docs.seleniumhq.org/download/ - OK
http://docs.seleniumhq.org/projects/ - OK
https://github.com/SeleniumHQ/www.seleniumhq.org/edit/master/src/main/webapp/index.jsp - OK
http://docs.seleniumhq.org/about/platforms.html#browsers - OK
http://docs.seleniumhq.org/about/platforms.html#operating-systems - OK
http://docs.seleniumhq.org/about/platforms.html#programming-languages - OK
http://docs.seleniumhq.org/about/platforms.html#testing-frameworks - OK
http://docs.seleniumhq.org/download/ - OK
```

Fig 8. Shows the testing of unreachable pages of the website named as www.seleniumhq.org in the Firefox web browser. We verify all the links of the website and the result shows that there is no broken links and links are working correctly with giving "Ok" messages. If it is a "page not found" message according to the response code, which means a page is unreachable or page error occurred. With broken links testing, we can also be able to find broken images and the overall status of the website. It can also check for the Ghost pages in the website giving the messages as "page error", "connection timeout" etc. The model of the site created can analyse the existence of the Unreachable pages. Testing is done with checking all links of the page, and also it checked all anchor links. It verifies how many links are there with the returning list of web elements. Here, in seleniumhq.org website, we found total 66 links and all are working as expected without any error message. Similarly, other websites are taken in our application can be checked for broken links or unreachable pages.

### E. *Evaluation output report*

**Table 1. Frames with the associated websites**

| Frame 1 | Frame 2 | Frame 3 |
|---|---|---|
| www.learnautomation.com | www.seleniumhq.org | www.html.com |
| Total links 83 No broken links found | Total links 86 No broken links found | Total links 89 No broken links found |

**Table 2. Frame count and passed test frames**

| Frame count in Firefox web browser | The result o frame testing in Firefox web browser | Frame coun in Google Chrome web browser | Result of frame testing in Google Chrome web browser |
|---|---|---|---|
| 3 | Passed tes frame | 3 | Passed test frame |

Here Table1 and Table 2 present an assessment report of the frame testing. Table 1 demonstrates the yield report of the aggregate links and broken links of the sites, where Table 2 demonstrates the yield report of the frame count and test frames passed/ failed.

## V. Comparison with Existing Work

In paper [18], they claim that the longest shortest path of a site may be an indicator of better structure. In their approach before reaching the page enMapITC. htm, four other pages must be visited. They conclude that the longest shortest path of a site can be an indicator of good structure. However, the demerit they find that a high value of this measure can be associated with the possible structural problem. Our approach holds that unnecessarily visit other pages in not always ideal. We go for reaching the target page earlier with consuming less time. Before reaching the target page, the user visits minimum no of pages. They have used Dijkstra's Algorithm which finds the single source shortest path to all reachable destinations in a graph. It doesn't allow negative edges. We have used A* Search Algorithm, on the other hand, uses a heuristic function to guide the search. That is: f(n)=g(n)+h(n)f(n)=g(n)+h(n), where g(n) is the distance from source to node n and h(n) is a heuristic to estimate the distance from node n to destination node. The g (n) & h (n) are two forces in opposite directions. If search goes way too deep in a path where the heuristic function doesn't have much to promise, then g (n) pulls it back to relax more promising paths. The key difference between Dijkstra and A* search algorithm is that A* algorithm focuses on reaching the goal node from the current node, not to reach every other node. In paper [18], they have focused on the structure and history analysis. However, in our view, the Centre of attention is the maintenance issues of the website as well as the implementation of HTML code to confirm reachable and unreachable pages.

## VI. CONCLUSION

In this paper, we proposed a web model. We suggested an existing algorithm for shortest path calculation that uses the concept of A* algorithm and our proposed work is also evaluated with Selenium web driver tool to find out broken links or unreachable pages and ghost pages in the website. In this process of activities, we concluded that automatic support for verification and validation activities could be very beneficial. We found that all paths in the website are thoroughly examined before delivering the web pages. In this testing, we observed a high level of automation in generating the test cases using Selenium web driver. It is essential to find out the shortest path to reach the target page for the user when the path is long for the websites having set of web pages. In that condition, our algorithm satisfactorily accomplished in finding the shortest path to reach the targeted document. Our experimental applications for Selenium testing guaranteed the quality of websites having set of web pages. Future work suggests that testing of dynamically created pages with extending the conceptual model is required to analyse and display websites for that dynamic testing also advisable for thorough testing of websites.

## REFERENCES

1. Montgomery, D. C. *Design and analysis of experiments*. John Wiley & Sons, 2017.
2. Marchetto, A., Tonella, P., & Ricca, F. Testing techniques applied to Ajax web applications. In *Proceedings of the Workshop on Web Quality, Verification and* Validation, WQVV'07, 2007.

3. Costa, M., Gomes, D., & Silva, M. J. The evolution of web archiving. International Journal on Digital Libraries, 18(3), 2017, pp. 191-205.
4. Felke-Morris, T. Basics of web design: HTML5 & CSS3. *Pearson*, 2014.
5. Conallen, J. Building Web applications with UML. *Addison-Wesley Longman Publishing Co., Inc*, 2002.
6. Gojare, S., Joshi, R., & Gaigaware, D. Analysis and Design of Selenium WebDriver Automation Testing Framework. *Procedia Computer Science*, *50*, 2015, pp. 341-346.
7. Panthi, V., & Mohapatra, D. P. An approach for dynamic web application testing using MBT. International Journal of System Assurance Engineering and Management, 8(2), 2017, pp. 1704-1716.
8. Hall, M., Brown, L., & Chaikin, Y. Core Servlets and JavaServer Pages: Advanced Technologies. Pearson Education, 2, 2007.
9. Brown, D., Pandya, A., Mulgrew, Z., Smith, J., Miller, A., & Kusuma, A. Dynamic loading of routes in a single-page application. *U.S. Patent No. 9,967,309*. Washington, DC: U.S. Patent and Trademark Office, 2018.
10. Andrews, A. A., Offutt, J., & Alexander, R. T. Testing web applications by modelling with FSMs. *Software and Systems Modeling*, *4*(3), 2005, pp. 326-345.
11. Bellettini, C., Marchetto, A., & Trentini, A. TestUml: user-metrics driven web applications testing. In *Proceedings of the 2005 ACM symposium on applied computing*, 2005, pp. 1694-1698.
12. Ricca, F., & Tonella, P. Building a tool for the analysis and testing of web applications: Problems and solutions. Tools and Algorithms for the Construction and Analysis of Systems, 2001, pp. 373-388.
13. Tonella, P., & Ricca, F. A 2-layer model for the white-box testing of web applications. In Web Site Evolution, Sixth IEEE International Workshop on (WSE'04), 2004, pp. 11-19.
14. Hoffmann, M. R., Brock, J., & Mandrikov, E. Eclemma-java code coverage for eclipse, 2009.
15. Kessis, M., Ledru, Y., & Vandome, G. Experiences in coverage testing of a Java middleware. In Proceedings of the 5th international workshop on Software engineering and middleware, ACM, 2005, pp. 39-45.
16. Bellettini, C., Marchetto, A., & Trentini, A. Dynamical extraction of web applications models via mutation analysis. INFORMATION-YAMAGUCHI-, 8(5), 673, 2005.
17. Tonella, P., & Ricca, F. Dynamic model extraction and statistical analysis of web applications. In Web Site Evolution, 2002. Proceedings. Fourth International Workshop on IEEE, 2002, pp. 43-52.
18. Ricca, F., & Tonella, P. Web Site Analysis: Structure and Evolution. In icsm, 76, 2000.
19. Di Sciascio, E., Donini, F. M., Mongiello, M., & Piscitelli, G. An Web: a system for automatic support to web application verification. In Proceedings of the 14th international conference on Software engineering and knowledge engineering, ACM, 2002, pp. 609-616.
20. Ricca, F., & Tonella, P. Understanding and restructuring Web sites with ReWeb. *IEEE MultiMedia*, *8*(2), 2001, pp. 40-51.
21. Di Lucca, G. A., & Fasolino, A. R. Testing Web-based applications: The state of the art and future trends. Information and Software Technology, 48(12), 2006, pp. 1172-1186.
22. Aho, A. V., Sethi, R., & Ullman, J. D. Compilers, Principles, Techniques. Addison Wesley, 7(8), 9, 1986.

## AUTHORS PROFILE

**Sonali Pradhan,** has completed her B.Tech and M.Tech in Computer Science and Engineering from BPUT, Odisha. Now she is continuing her research (PhD.) in SOA, University, Bhubaneswar. She has attended many conferences like IEEE, Springer, Elsevier etc. Her area of interest is Software testing.

**Mitrabinda Ray** has completed her PhD. degree from NIT, Rourkela. Now she is working as an associate professor in SOA, University, Bhubaneswar, in the department of Computer Science and Engineering. Her area of interest is software testing, software reliability & estimation. She has published a number of papers in different journals and conferences.

.