# An Efficient Algorithm for Web Page Change Detection

*Thesis submitted in partial fulfillment of the requirements for the award of degree of*

**Master of Engineering**
in
**Software Engineering**

*Submitted By*
**Srishti Goel**
**(801031028)**

Under the supervision of:
**Dr. Rinkle Rani**
Assistant Professor
CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

**June 2012**

# CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, "*An Efficient Algorithm for Web Page Change Detection* ", in partial fulfilment of the requirements for the award of degree of Master of Engineering in *Software Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Dr. Rinkle Rani and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Srishti Goel
(Srishti Goel)
801031028

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Rinkle 07/06/12
(Dr.Rinkle Rani)
Assistant Professor,
Computer Science and Engineering Department.

**Countersigned by**

**(Dr. Maninder Singh)**
Head,
Computer Science and Engineering Department,
Thapar University,
Patiala.

**(Dr. S. K. Mohapatra)**
Dean (Academic Affairs),
Thapar University,
Patiala.

# Acknowledgement

I would like to express my deepest appreciation to Dr. Rinkle Rani, my mentor and thesis supervisor for her constant support and motivation. She had been instrumental in guiding me throughout the thesis with her valuable insights, constructive criticisms and interminable encouragement.

I am also thankful to Dr. Maninder Singh, Head, computer Science and Engineering Department and Mr. Karun Verma, P.G. Coordinator for their constant support and encouragement.

I express my thanks to my family for their support and affection and for believing in me always.

In the end, I would like to thank all the faculty members and staff of the department and my friends who directly or indirectly helped me in the completion of this thesis.

Srishti Goel

Srishti Goel

(801031028)

# Abstract

Internet is being used for the exchange of information. People are using internet actively for exchange of information across the world resulting in uploading of information on web pages and updating new web pages very frequently. The contents of web page changes continuously & rapidly. Hence it becomes very difficult to observe the changes made in web pages and retrieve the original web pages. For efficient retrieval and monitoring the changes made in web pages and compare the difference between refreshed page and old page efficiently that too in minimum browsing time, an effective monitoring system for the web page change detection based on user profile is needed. The web page change detection system can be implemented by using various Tools or Algorithms. Various services are also available which can be used to detect the changes in a web page.

In this thesis, a new algorithm for the structural as well as content change detection has been proposed and described. For better results tree has been designed for the corresponding web pages. The proposed change detection algorithm is based on assigning hash value to each leaf node and tag value to the non leaf nodes. Bottom up approach has been used for assignment. The level of each node has been used to find hash values and modification in a node. It has been shown with the help of suitable examples that the proposed algorithm extracts the changes very efficiently from the various web pages.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1 Introduction

## 1.1 Web

The World Wide Web or Web is a way of accessing information over the medium of the Internet. [27]It is an information-sharing model that is built on top of the Internet. The Web uses the HTTP protocol, only one of the languages spoken over the Internet to transmit data. Web services which use HTTP to allow applications to communicate in order to exchange business logic use, the Web to share information. The Web also utilizes browsers, such as Internet Explorer or Firefox, to access Web documents called Web pages that are linked to each other via hyperlinks. Web documents also contain graphics, sounds, text and video. The Web is just one of the ways that information can be disseminated over the Internet. The Internet, not the Web, is also used for e-mail, which relies on SMTP, Usenet news groups, instant messaging and FTP. So the Web is just a portion of the Internet, but the two terms are not synonymous and should not be confused.

## 1.2 Web Pages

A web page [30] [31] is a document or information resource that can be accessed through a web browser and displayed on a monitor or mobile devices. This information is usually in HTML or XML form, which allows for the information to be easily structured and then quickly read by the client's web browser. They provide navigation to other web pages via hyperlinks. The information in all those web pages is located on remote web servers in the form of text, image, or script files. Information on a web page is displayed online with the help of a web browser they are further connected with the servers where the website's contents are hosted through the Hypertext Transfer Protocol (HTTP). For instance, if you look at the URL of the web page, it could notice the prefix 'http://', which tells the browser what protocol to use to execute the particular URL request.

## 1.3 Web Crawler

A Web crawler is a computer program that browses the World Wide Web in an orderly fashion. Web crawlers [29] are also known as ants, automatic indexers, bots, Web spiders, Web robots, etc. The process of browsing the world wide web is called Web

crawling or spidering. Search engines, use spidering as a means of providing up-to-date data. Web crawlers are mainly used to create a copy of all the visited pages for later processing by a search engine that will index the downloaded pages to provide fast searches. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses (usually for sending spam).

It starts with a list of URLs to visit, called the seeds. As the crawler visits these URLs, it identifies all the hyperlinks in the page and adds them to the list of URLs to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies.

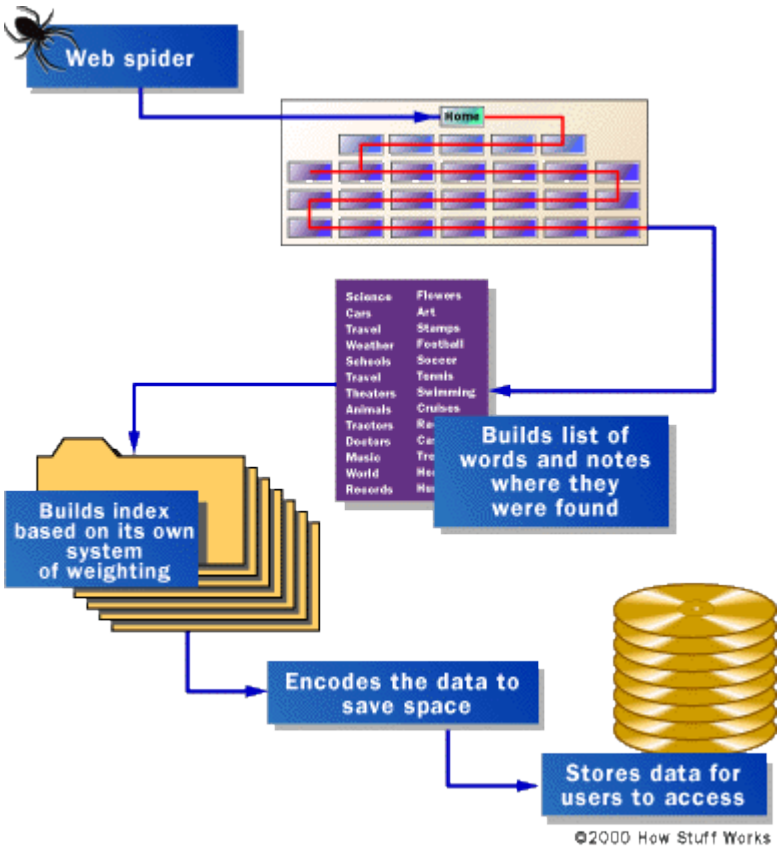



**Figure 1: Working of a Web Crawler[29]**

## 1.4 Web Page Change Detection System

Internet has become an effective way for the exchange of information across the world. It delivers the information mainly in the form of the web pages. New pages are uploaded frequently to provide new and more information to the user. Now the users are not only interested in gathering the new information but also the changes that have taken place in the web page. If a user is observing a web page that is refreshed very frequently then the user might not be interested in downloading or viewing the entire web page each and every time. He will be interested in observing the changes that have taken place in the web page since last visit. In order to find these changes user will select a zone in which he is interested to observe the changes. The web page detection system will find the zones selected by the user in old and new web pages. The system then compares the two web pages to detect the changes which have taken place.

User may be interested to know the changes in the web pages every time he visits the page. For example a stock trader may be interested in knowing the current status of the market or he may be interested in knowing the new price of the stocks. Similarly a user may be interested in knowing the ranking of the cricket teams, statistics of players playing in IPL which changes frequently with each match.

Web can be considered as a large file system which contains millions and millions of documents which may be referenced by other documents. These documents are being used by a large number of people, these documents are accessed for various purposes and also perform multiple operations such as adding, removing, indexing, rearranging, modifying and copying the available resources and files.

The rate at which the changes will occur varies from site to site. It is impossible to predict the frequency at which the changes will occur in a web page as it follows the poison process. Therefore, managing the local collection afresh as much as possible becomes a challenging task. For instance, generally government sites change once in a week or so and daily update of pages of these sites will definitely lead to wastage of precious bandwidth. On the other hand, a website such as news website needs to be updated recursively at short intervals of time.

Therefore, any attempt to design a system that would help the user in his or her daily interactions with Web pages is considered helpful. Suppose the user wants to monitor a zone in a (static) HTML Web page for changes, say after every five hours. After five hours, the same Web page is refreshed and entirely downloaded by the system. The main objective is to find, in the entire new Web page, the zone most similar to that selected by the user. After finding this similar zone, a percentage similarity and a mapping that links every element of the old page to its most similar counterpart in the selected zone should be outputted. This comparison process is performed by the Comparator.

XML has become a standard on the web and is still gaining popularity. It is used in more and more applications as syntax for document publishing, data-exchange, enterprise integration and many more purposes. For detecting these changes, Unix , Linux and other Unix based systems provide us with the diff command . Diff is able to compare and output the differences between two files. Like similar programs, diff is designed to find the differences in text files and handles those files as a series of lines. Although it can find the differences in XML documents, it doesn't handle it like a tree structured document. When XML documents are handled like tree-structured documents, more information can be gathered about differences.

To find these changes XML document are represented as a tree structure, the changes are detected in these trees. Algorithm that are used to detect the changes or compute the difference can be divided into two categories depending on whether they deal with ordered tree or an unordered tree.

1. **Ordered Tree**

Ordered tree is one in which both the ancestor (parent child) relationship and the left-to-right ordering among siblings are significant.

2. **Unordered Tree**

An unordered tree is one in which only ancestor relationships are significant, while the left-to-right order among siblings is not significant.

### 1.4.1 Change detection

To detect the changes in a web page detection system we convert our HTML pages into the tree structure. Then compare the tree structure of old web page and the modified web page. The two trees can be compared and the equal nodes and equal subtree can be matched. The nodes and subtree that don't match are said to be different. The web page change detection system helps the user to detect such changes efficiently and in minimum browsing time.

## 1.5 Classification of Changes in a Web Page

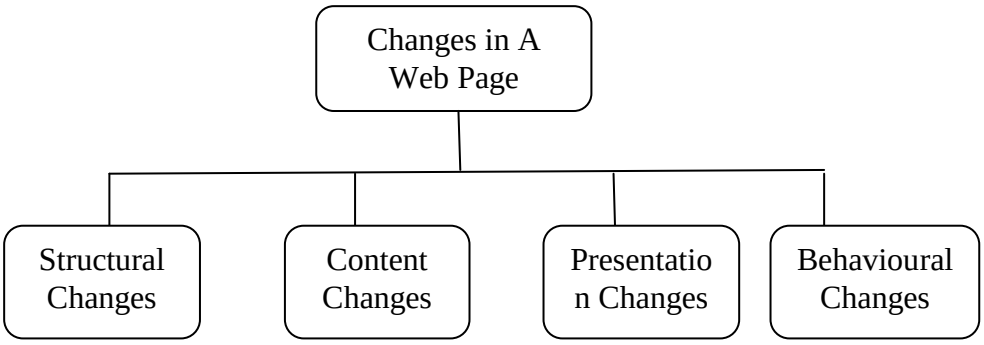



**Figure 2: Classification of Changes in a web page**

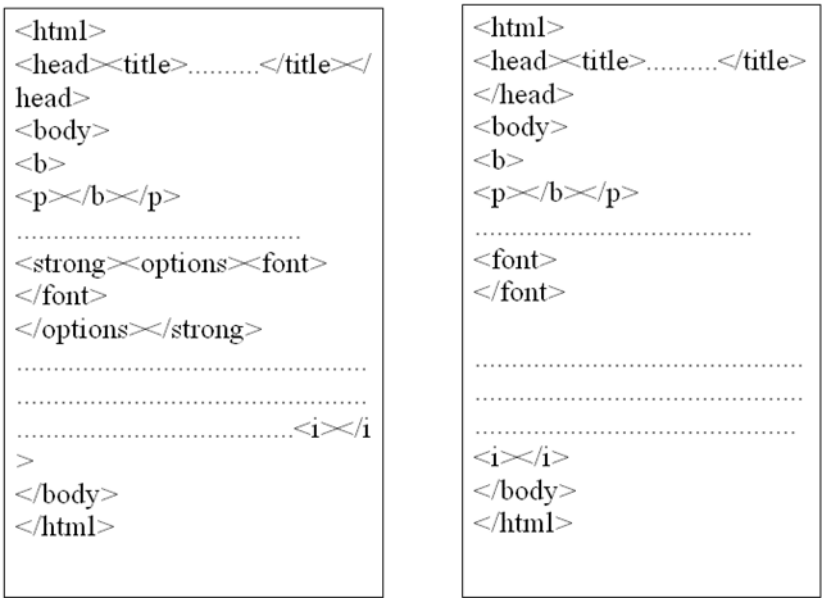
Web page changes can be classified as follows:

- Structural Changes
- Content Or Semantic Changes
- Presentation Or Cosmetic changes
- Behavioural Changes

#### 1.5.1 Structural Changes

These changes occur when ever a tag is added or deleted in a web page i.e. addition or deletion of a tag causes structural change in a web page. Sometimes the

addition/deletion/modification of a link also causes a structural change. These types of changes are important to find as they are not visually perceptible.




**(a)** **(b)**

**Figure 3: a) Initial Version b) Modified Version**

### 1.5.2 Content or Semantic Changes

These changes occur whenever the content of a web page changes according to the user point of view. A stock trader may be interested to know the changed status of the market or the current price of the share. He is interested in the current or the changed status of the market and not in the old price or the old market status. Another example could be the web pages displaying the score of a match. The user viewing the page is interested in the current score and the content of the page changes every time whenever the score changes. Web pages containing the records or schedule of a tournament changes accordingly with the change in a tournament.

```
<html>
<head>
 <title>PORTFOLIO</title>
<link href="style.css" rel="stylesheet" type="text/css" /></head>
<body>
<p> Welcome to My World.</p>
  <p>HIiiiiiii to all.....<br />
    I am a first year B Tech student studying in the LNMIIT </p>
  <p>( LNM Institute of Information Technology, Jaipur).<br />
    Pursuing Electronics and Communication Engineering course.<br
/>
    The 'My Bio' page describes myself, my thoughts, interests<br />
    and links to connect with me socially. </p></body>
</html>
```

**(a)**

```
<html>
<head>
 <title>PORTFOLIO</title>
<link href="style.css" rel="stylesheet" type="text/css" /></head>
<body>
<p> Welcome to cricket World.</p>
  <p>Today's score<br /></p>
  <p>INDIA V/S ENGLAND<br />
   India played very well and beat England by 6 wickets<br />
    England made 253 runs in 46 overs and India made 254 runs nad
beat England by 6 wickets. </p></body>
</html>
```

**(b)**

**Figure 4: (a) Initial Version (b) Modified Version**

### 1.5.3 Presentation or Cosmetic Change

This Type of changes Occurs whenever the appearance of a web page changes but the contents of the web page remains the same[39]. For example with the changes in tags the appearance of a page may change without change in the content of a page.

```
<html>
<head><title>.....</title>
</head><body>.....</body>
<p>my first web page</p>
<p>Page change detection system
</p>
</body>
</html>
```

**(a)**

```
<html>
<head><title>.....</title>
</head><body>.....</body>
<p style =”backgroundcolor = red”><u>my first web page</u></p>
<p>Page change detection system
</p>
</body>
</html>
```

**(b)**

**Figure 5: a) Initial Version b) Modified Version**

### 1.5.4 Behavioural Changes

Behavioural changes refer to changes in the active components which are present in a document. For example web pages may contain scripts, applets etc. as active components. When such hidden components change, the behaviour of the document gets changed. However, it is difficult to catch such changes especially when the codes of these active components are hidden in other files.

# Chapter 2 Literature Survey

Various algorithms have been proposed by researchers for the design of efficient algorithms for detecting changes in Web pages. In [9], [37] and [13], different algorithms are described for detecting changes in XML documents. The algorithm in [13] is based on finding and then extracting the matching nodes from the two trees that are being compared. From the non matching nodes, the change operations are next detected. Matching of nodes is based on comparing signatures (functions of node content and children) and order of occurrence in common order subsequences of nodes. The research work in [9] and [37] transform the pages to trees according to the XML structure and use edit scripting to compare them. The strength of these algorithms lies in their low time-complexity, which is in the order of
O (nlogn).

## 2.1 CHANGE DETECTION ALGORITHMS

### 2.1.1 Novel Approach

It is assumed that only text nodes can be changed elements and attributes nodes are not considered [15]. This algorithm finds the subtrees and deletes those trees that do not show changes. Firstly similarities between the two trees are find, whether the two trees are identical or not. If not, then the changes in two versions of the tree by comparing each and every element in the old version and new version. This algorithm follow a top down approach it start with the root node by comparing the signature values of each node to its corresponding node in the two trees. In order to reduce the number of comparisons node signature has been used. Algorithm assigns the hash value to all child nodes in the trees the child nodes are basically the text nodes. Signature is mainly the function of the hash value calculated from the contents of the node. Signature of child node is basically the summation of its entire child node signature except the leaf node.

### 2.1.2 Level Order Traversal Algorithm

This algorithm finds the changes in structure as well as in the content. [38]To find the changes in the structure i.e. to detect whether the new node has been added or deleted levels in the tree structure has been used instead of traversing down the tree. In order to find the

content changes RMS values of the ASCII values of the character are extracted from the web document. Changes in different versions of a web document can be effectively and efficiently extracted with the help of an algorithm. The change extractor performs the three things:

1. Document tree construction
2. Level order child enumeration
3. Finding the RMS values of the node.

To evaluate the changes in the old version and the new version of a web document. Specialized set of arrays has been used to show the relationship between the parent and the child node.

### 2.1.3 La Diff

La Diff is used for hierarchical structural information. [4][5]It uses certain criteria two compare the two node of different version of a document in order to detect the changes between the two. Time complexity of this algorithm is $O(n*e+e^2)$ where e is the weighted edit distance between the two trees and n is the number of leaf nodes. Its complexity can be either linear or quadratic it can be linear in terms of size of the document and can be quadratic in terms of the number of changes between the two version.

### 2.1.4 X Key Match

Algorithm for matching the two version of documents using XML keys. [2]This algorithm is used before diff algorithm so as to compare the two documents. The architecture of the system is shown in the figure.

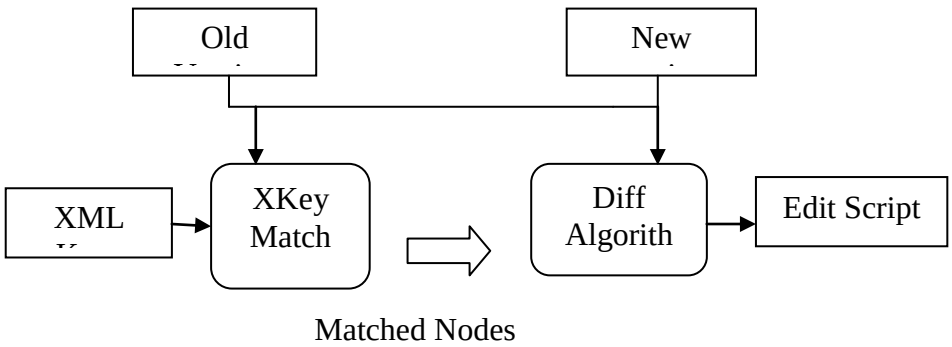



**Figure 6: X key match[2]**

This algorithm is based on the construction of DFA from the set of XML keys. Matching is based on these keys only with a preorder traversal. Then the candidates are selected for both the trees. These candidates can further be used for matching the two versions of trees. Function which is named matched is used to find the candidate having the same key values.

### 2.1.5 X Diff

This algorithm can be used to detect the changes in two unordered tree. Edit script is used to convert one tree into another. Edit script is basically a sequence of edit operations. X diff algorithm [26] uses three basic edit operations that are Insert, Delete and Update. X diff is used to find the minimum cost matching and also generates the minimum cost edit script. Various steps used in x diff are as follows:

1. *Parsing and Hashing*

It parses the two XML document into the corresponding tree structure and assign a hash value to each and every node in the tree.

2. *Matching*

This algorithm starts with the root node and compares the hash values of the node in the two trees. If the hash value is same then the two trees are considered to be identical and if these values are different then minimum cost of matching is calculated for the two trees.

3. *Generating minimum cost edit script*

Then it generates minimum cost of edit script based on the minimum cost of matching which is being generated in the previous step.

### 2.1.6 MH Diff

It is a heuristic algorithm[26] which is used to detect the changes in unordered structure documents. It uses edit script to transfer the tree into other. Except the basic edit operations it uses two other operation move and copy, which result in higher quality of edit script. The Goal of this algorithm is to find the minimal edge cover, which corresponds to the minimum cost edit script. This algorithm can be used only for small document and its worst case is $O(n^3)$, where n is the number of nodes.

### 2.1.7 CX Diff ordered

This algorithm is used to detect the customized changes in the XML document. To implement this algorithm[13] first we extract the object in which the user is interested and then we apply the algorithm. In order to do so we detect the matching nodes between the two

trees and from the non-matching we will be able to detect the changed operations. Phases of this algorithm are as follows:

1. *Object Extraction and Signature Computation*

2. *Filtering Unique Insert/Delete*

3. *Finding the common order subsequence*

### 2.1.8 XML Tree Diff

XML Tree Diff is the set of java beans that are used to efficiently differentiate and update DOM trees. [9] It works same as Diff and Patch algorithm. To implement this algorithm no file format is needed it can be directly implemented on the DOM structure. It uses operations such as change node, delete node and insert node. It is well used for version management in two document and tree structured data.

### 2.1.9 MM Diff and XM Diff

Presents an external memory algorithm XM Diff (based on main memory version MM Diff) for ordered trees in the spirit of Selkow's variant. Intuitively, the algorithm [6][7] constructs a matrix in the spirit of the "string edit problem", but some edges are removed to enforce that deleting (or inserting) a node will delete (or insert) the subtree rooted at this node. More precisely,

1. Diagonal edges exists if and only if corresponding nodes have the same depth in the tree

2. Edges from (x, y) to (x+1,y) exists unless the depth of node with prefix label x+1 in D1 is lower than the depth of node y+1 in D2. For MM Diff, the CPU and memory costs are quadratic O(|D1| * |D2| ).With XM Diff, memory usage is reduced but IO costs become quadratic.

### 2.1.10 IBM XML Diff and Merge Tool

Created by IBM it is a java program which is used to detect the changes in the two versions of a XML document. This tool [33] indicates each difference which is made in the document by symbols and colour coding.

### 2.1.11 3DM's Matching Tools

3DM's matching tool is used for performing 3 way merge and difference of XML files.3DM tool[18] is not only limited to insert, delete and update but also include move and copy operations.

### 2.1.12 XY Diff

XY diff [26] is a fast algorithm which support move operation as well as other XML features. It matches the large subtree found in both the XML documents. Matching of nodes is done according to the key attributes of the node. It first starts with matching the large subtree and then the smaller subtree if the matching fails. If matching is succeeded then the parent node and the descendant of that particular node is compared. It supports insert, delete, update and move operations. It is not able to find the minimum edit script. It uses greedy approach because of which it does not provide optimal solution.

### 2.1.13 VM Tool

VM Tool is a collection of java XML oriented tool and is available as an open source tool. It [25] contains the diff and patch tools which are used to match the two trees and automatically find the difference between them. VM Tool mainly focuses on the size of the XML file.

### 2.1.14 Diff XML

Diff XML is a java based tool which is used to find the difference between the XML documents. It [11] operates on the hierarchical tree structure. The result is generated in a separate file. This file can be a HTML file. The output report will contain the sections that

were not matched. The tool matches each and every node, their content, attributes and marks the difference with a colour.

### 2.1.15 XMiddle

It is a data sharing middleware [15]. Mainly used in mobile computing so that data can be share between the applications. Data is mainly encoded in XML. It allows other host on the network to access the data even when they are disconnected from the network. It also helps to find the changes if there any. The use of XML as a underlying data structure enable XMiddle to find the difference and define reconcile policies for a particular document.

### 2.1.16 KF- Diff+

In this section we propose KF-Diff+, a highly efficient algorithm which need not to compute the hash signatures of all the nodes in the documents. [36]As mentioned before, the key property of KF-Diff+ is that the algorithm transforms the traditional tree-to tree correction into the comparing of the key trees. Since any two different nodes in the key tree should not have the same key path, the comparing between two trees will be more efficient.
There are two steps in KF-Diff+ as follows:
1. *Parsing*
KF-Diff+ parses DOC1 and DOC2 into trees T1 and T2.
2. *Matching*
The goal of this step is to find a minimum-cost matching between T1 and T2. Due to the space limitation, the generating of the edit scripts is omitted here.

### 2.1.17 XML Diff and Patch

[34]Microsoft created a tool [35] that is able to differentiate and patch two XML documents regardless if they are mapped into ordered or unordered tress. Moreover, it offers an option of ignoring whitespaces, XML comments and processing instructions, and offers some interesting options about the namespaces. It represents the differences with a XML Diff Language (XDL) that is called XML diffgram. However, it is not an open-source tool, it is not accompanied with any kind of documentation and it uses the Document Object Model . Furthermore, it fails to compare files that differ dramatically and the online version of the tool is unable to handle files greater than 100KB.

### 2.1.18 Delta XML

Delta XML [10] is a commercial tool created by Monsell EDM Ltd, capable of comparing, merging and synchronizing XML documents. It is able to merge both ordered and unordered trees fast, where the size is up to 50MB. However, the state of the trees either ordered or orders less has to be explicitly stated. The XML documents have to be well-formed and with the same root element, whilst knowledge of the DTD or the Schema that it they follow is not required. One of its interesting features is that it supports both 2-way and 3-way merging. Contrary to other tools, it only supports the insert, delete and update edit operations. The move edit operation is not implemented.

### 2.1.19 Tree Patch

Tree Patch is an open-source XML Diff and Patch Tool, designed. [40]It extended the Diff XML algorithm. Tree Patch describes changes using Extended Delta Update Language (EDUL). The EDUL output format describes the update, insert, delete and move operation.

### 2.2.20 Tree Match

Tree Match [40] is a fast tree pattern-matching algorithm for XML Query. The goal of Tree Match is to directly find all distinct matching of a query tree pattern in XML data sources. The algorithm is not applicable when detecting changes in XML documents because it is designed as a tree pattern matching algorithm and not as a tree-matching algorithm.

### 2.2.21 Bio Diff

Based on X-Diff, BioDiff is an algorithm specially designed for detecting changes in genomic and proteomic data specified as XML[24]. It reduces the size of the set of biological data and focuses on the special semantics of the XML elements.

### 2.2.22 VI Diff

[22]VI Diff detects changes in a visual representation of a web page. It detects content as well as structural changes. Content changes include modification of the text, hyperlink or image. Whereas structural change include change in the visual appearance of the page. It can

be implemented in various applications such as crawl optimization, archive maintenance, web changes browsing, etc.

Vi-DIFF consists of three steps:

1. *Segmentation*

Web pages are visually segmented into semantics blocks by extending the VIPS algorithm.

2. *Change detection*

Detects both content as well as structural changes.

3. *Generation of delta file*

Changes detected are saved in Vi – delta files.

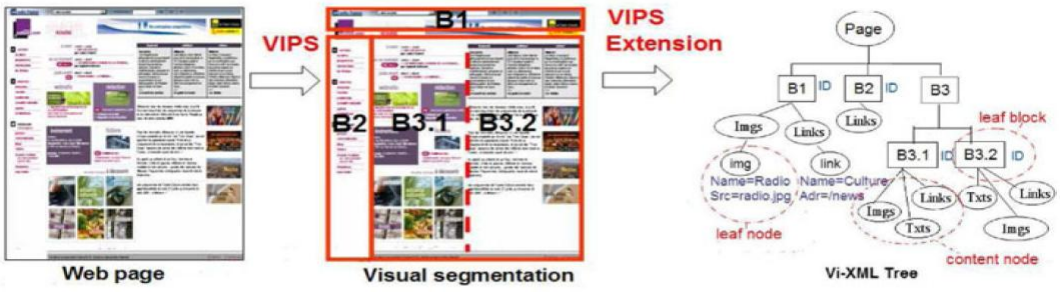



**Figure 7: Steps of VI Diff Algorithm**

In addition to basic operations, it supports a move operation, if there is no structural change. However, it does not support yet the move on content changes when there are structural changes.

### 2.2.23 Optimized Hungarian Algorithm

This algorithm transforms an HTML page into the XML document and then converts it into the tree like structure. Hungarian algorithm [14] has been used in the comparator module so as to compare two trees. Most similar subtree in the new page and old version of the page were found. Three optimizations have been used over the Hungarian algorithm so as to compare the more efficiently.

Table 1: Comparison Of Various Change Detection Algorithm

| | **Refrences** | **Time complexity** | | **Memory** | **Ordered/Unordered** | **Section** |
|---|---|---|---|---|---|---|
| NOVEL APPROACH | 6 | Linear | ? | ? | Orderd | 4.1 |
| LEVEL ORDER TRAVERSAL ALGORITHM | 2 | Linear | ? | ? | Ordered | 4.2 |
| LA DIFF | 14,15 | Linear | $O(n*e+e^2)$ | Linear | Ordered | 4.3 |
| X KEY MATCH | 13 | ? | ? | ? | Both | 4.4 |
| X DIFF | 24 | Quadratic | $O(n^2)$ | Quadratic | Unordered | 4.5 |
| MH DIFF | 24 | Quadratic | $O(n^2 \log n)$ | ? | Unordered | 4.6 |
| CX DIFF | 9 | ? | ? | ? | ? | 4.7 |
| XML TREE DIFF | 4 | Quadratic | $O(n^2)$ | Quadratic | Ordered | 4.8 |
| MM DIFF | 16,17 | Quadratic | $O(n^2)$ | Quadratic | Ordered | 4.9 |
| XM DIFF | 16,17 | Quadratic | $O(n^2)$ | Linear | Ordered | 4.9 |
| IBM XML DIFF AND MERGE TOOL | 23 | ? | ? | ? | ? | 4.10 |
| 3DM'S MATCHING TOOL | 19 | Linear | O(n) | ? | Ordered | 4.11 |
| XY DIFF | 24 | Linear | O(n log n) | Linear | Ordered | 4.12 |
| VM TOOL | 22 | ? | | ? | Unordered | 4.13 |
| DIFF XML | 21 | Linear | $O(ne+e^2)$ | Linear | Ordered | 4.14 |
| XMIDDLE | 1 | Linear | | | | 4.15 |
| KF-DIFF+ | 7 | Linear | O(n) | ? | Both | 4.16 |
| XML DIFF AND PATCH TOOL | 11,12 | ? | ? | ? | Both | 4.17 |
| DELTA XML | 3 | Linear | O(\|x\| *D) | Linear | Both | 4.18 |
| TREE PATCH | 10 | Linear | $O(ne+e^2)$ | Linear | ? | 4.19 |
| TREE MATCH | 10 | ? | ? | ? | ? | 4.20 |
| BIO DIFF | 18 | Quadratic | $O(n^2)$ | Quadratic | Unordered | 4.21 |
| VI DIFF | 25 | ? | ? | ? | ? | 4.22 |
| OPTIMIZED HUNGARIAN ALGORITHM | 7 | Linear | $O(n^2)$ | ? | Ordered | 4.23 |

## 2.2 Steps for Change Detection

### 2.2.1 Root Mean Square Based Content Level Change Detection Method

This method computes Root Mean Square (RMS) value as checksum for the entire web page contents as well as for its various paragraphs appearing in the page. [39]While updating the page in the repository, comparison between the checksums of both versions of web pages is performed and in case of an anomaly, it is concluded that the web page at web server end has been modified as compared to the local copy maintained in the repository at search engine end. Thus the document needs to be updated. Paragraph checksums help to detect changes at micro level and help in identifying the locality of changes within the document.

The following formula has been developed to calculate the RMS value:

$$R.M.S = \left[ \frac{(a1)^2 + (a2)^2 + \ldots\ldots\ldots\ldots + (a'n)^2}{n} \right]^{1/2}$$

Where a1, a2, ----------, *an* are the ASCII code of symbols and 'n*'* is the number of distinct symbols/characters excluding the white space present in the web page.

Following inferences are drawn about the RMS method for content level change detection:

- It produces unique checksum (RMS value) for entire web page as well as for paragraph level contents of a web document.
- The formula developed is capable to detect minor changes even addition/deletion of few words accurately.
- Paragraph checksum helps to identify changes at smaller level i.e. at paragraph level.
- It may be noted that ASCII values have been used in the formula as each symbol has a unique representation in ASCII Table and thus leading to no ambiguity.

Though the above method performs efficiently and guarantees to detect content level changes among different versions of web pages but it assigns uniform weightage to the entire contents of the web document whereas in real life changes in some contents carry more weightage than other. For example changes in main headings of a web document carry more weightage

than any other content change. So keeping in mind these points the modified checksum based content level change detection method is being proposed.

### 2.2.2 Checksum Based Content Level Change Detection Method

Similar to the previous method, [39] checksum based content level change detection also produces a single checksum for entire web page as well as for each paragraph present in the web document.

Paragraphs level checksum help to know the changes at micro level i.e. paragraph level. By Comparing the checksums of different versions of a web page, it is known whether the web page content has been changed or not.

The following formula has been developed to calculate the checksum:

Checksum = Σ I_ parameter * ASCII code * *K-factor*

Where: *I*_parameter = importance parameter,

*ASCII code* = ASCII code of each symbol (excluding white space)

*K-factor* = scaling factor

## 2.3 WebVigil Architecture

WebVigiL is a change detection and notification system, which can monitor and detect changes to unstructured documents in general. The current work addresses HTML/XML documents that are part of a web repository. WebVigiL [23] aims at investigating the specification, management, and propagation of changes as requested by the user in a timely manner while meeting the quality of service requirements. Figure 8 summarizes the high level architecture of WebVigiL. Users specify their interest in the form of a Sentinel that is used for change detection and presentation. Information from the sentinel is extracted and stored in a data/knowledge base (currently Oracle) and is used by the other modules in the System. The functionality of each module in the architecture shown in Figure is described briefly in the following sections. WebVigiL architecture shown in the Figure has five modules and many components within them.

### a) Sentinel

WebVigiL provides an expressive language with well-defined semantics for specifying the monitoring requirements pertaining to the Web. Each monitoring request is termed a Sentinel.

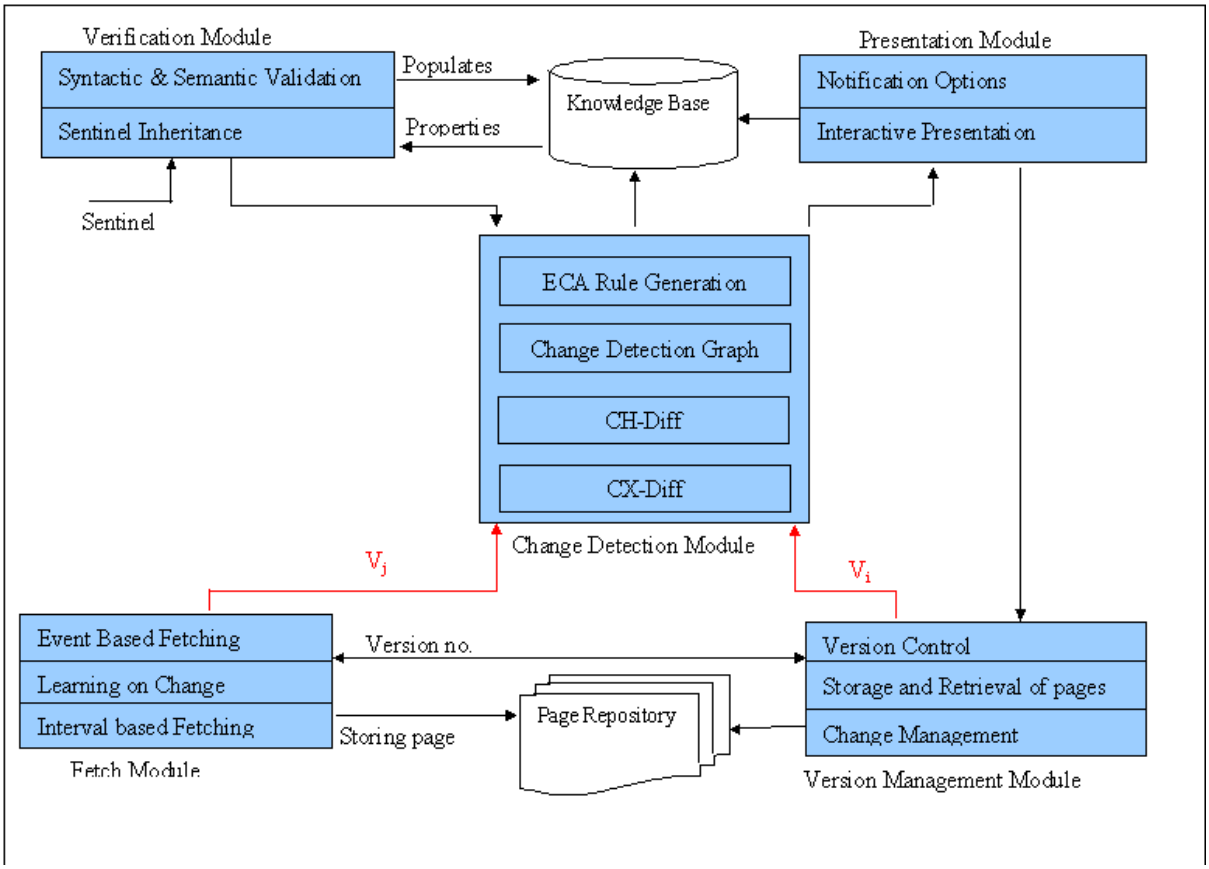



**Figure 8 : WebVigil Architecture**

### b) Verification Module

Verification module provides the required communication interface between the system and the user for specification of sentinels. User requests (sentinels) are processed for syntactic and semantic correctness. Valid sentinels are populated in Knowledge base (Oracle is used currently) and a notification of the valid sentinels is sent to the change detection module.

**c) Knowledge Base**

Knowledge Base is a persistent repository containing meta-data about each user, number and names of sentinels set by each user, and details of the contents of the sentinel (frequency of notification, change type etc.). The details of a sentinel need to be stored (in a persistent and recoverable manner) as several modules use this information at run time.

**d) Change Detection Module**

Every valid user request arriving at WebVigiL, initiates a series of operations that occur at different points in time. Some of these operations are: creation of a sentinel (based on start time), monitoring the requested page, detecting changes of interest, notifying the user(s) of the change, and deactivation of sentinel. In WebVigiL, for every sentinel, the ECA rule generation module generates ECA rules [8] to perform some of these operations.

**e) Fetch Module**

The Fetch Module [9] of WebVigiL is responsible for retrieving the pages registered with it and thus serves as a local wrapper for the task of fetching pages depending upon the user set fetching policy i.e., fetching a page after a specified interval (set by the user) or fetching the page on change (the system determines the frequency of fetching based on actual change frequency of the pages). The Fetch module informs the version controller of every version it fetches, stores it in the page repository and notifies the change detection graph (or CDG) of a successful fetch. The wrapper fetches the page only when there is change in the properties of the pages. By properties, we mean the size of the page and the last modified time stamp. When there is a change in time stamp of the page with an increase or decrease in page size, the wrapper fetches and caches the page. In cases where time stamp is modified, but the page size remains the same, the wrapper fetches and calculates the checksum of the page. This version of the page is cached only if the calculated checksum differs from the checksum of the cached (previous) version of this page.

**f) Version Management**

An important feature of WebVigiL architecture is its server-based repository service (Version controller) that archives and manages versions of pages. WebVigiL retrieves and

stores only those pages needed by a sentinel. The primary purpose of the repository service is to reduce the number of network connections to the remote web server, thereby reducing network traffic. When a remote page-fetch is initiated, the repository service checks for the existence of the remote page in its cache and if present, the latest version of the page in the cache is returned. In cases of cache miss, the repository service requests that the page be fetched from the appropriate remote server. Subsequent requests for the web page can access the page from the cache instead of repeatedly invoking a fetch procedure.

**g) Presentation Module**

The principal functionality of this module is to clearly present the detected differences between two web pages to the user. Therefore, computing and displaying the detected differences is very important.

## 2.4 Tools for Change Detection

Several tools are also available to assist users to track when the web pages of interest have changed. Most of the tools offer service from a centralized server or a client's machine. Client-based tools focus mainly *when* to fetch the pages of interest rather than *how* the pages have changed. This is because of the complexity involved in keeping track of changes to the content for numerous versions. If specific changes to a page have to be detected, a differencing tool has to run on the client machine. In spite of having such a tool, when the user wants to track composite changes (for example, links AND images change on a page) additional information has to be maintained. And as the requirements grow, the complexity at the client end also increases. This gave way to the server-based systems. Server based tools track pages that are previously registered or submitted by users and notifies them via email or over the web upon request. The following are some of the server-based tools developed for change monitoring to web pages:

**a) AIDE (AT&T Internet Difference Engine)**

AIDE [12] is both client and server based. It is a collection of tools. The tools consist of:w3newer, which detects changes to pages; snapshot, which permits a user to store a copy

of the page and to compare any subsequent versions of the page; HtmlDiff, the differencing tool used to compute the changes between two pages. W3newer runs on the client machine and when observes a change on a page, it informs the snapshot to save a copy. Snapshot is an external service that archives the versions of the page and whenever a new version of a page arrives to the system through w3newer, it invokes the HtmlDiff. When the user requests the changes, the snapshot is contacted. The user could obtain the difference between any versions of the page. Because of its architecture the necessity of grouping users who monitor the same page does not arise. The drawbacks of this system are that the user cannot specify customized changes (links, images, keywords) or composite change (links AND images) on apage. Changes to XML pages are not supported.

**a) WebGUIDE**

WebGUIDE is an extension to AIDE. It consists of the following tools: AIDE and Ciao [3].Ciao is a graphical navigator that allows users to query and browse structural connections embedded in a document repository. The same drawbacks described for AIDE apply to this system as well.

**b) NetMind**

NetMind [20] formerly known as URL-minder provides keyword or text-based change detection and notification service over web pages. NetMind detects changes to links, images, keywords and phrases in an HTML page. The medium of notification to users about the change is via e-mail or mobile phone. The user might be interested in a change to a page but not when there are changes to articles or some set of words the user is not interested. Such change detection request cannot be specified. There is no support for composite changes (when both links AND images change) on a page. There is no provision for the user to come back later and view the last changes that have been detected. The frequency of when to poll the page is predefined. The user cannot explicitly specify when to poll the page for change detection and on what versions of page the change should be computed. Since the implementation is hidden behind a CGI interface, how changes are detected is not known. Change detection to XML pages is not supported.

**c) WebMon**

WebMon [16] is proposed for tracking information change over the Internet. The user can specify the web page to be monitored, select the monitoring function and state the monitoring frequency. The monitoring function can be any customized change such as change in the time stamp, links, images or phrases. The change detection is based on the structure of the page. The assumption is that, HTML pages have stable structure. Issues such as grouping users having overlapping monitoring requests are not considered. Change detection to XML pages is not supported. A combination of changes on a page is not supported.

**d) WebCQ**

WebCQ [17] is a prototype system for large-scale web information monitoring and delivery, which makes use of the structure present in hypertext and the concept of continuous queries. WebCQ is designed to discover and detect changes to the web pages and to provide a personalized notification of the changes to the users. Users monitoring requests are modelled as continuous queries on the web. WebCQ change detection robot is responsible for discovering and detecting changes to web pages. The authors specify that composite changes can be detected, but currently the system does not seem to support them. WebCQ lacks a fine grouping strategy. For example, the change is computed more than once for two users having the same set of keywords. The grouping is based on a single keyword rather than a set of keywords. Only change detection to HTML documents is supported. The user has to set the polling frequency explicitly, the system does not tune to the change frequency of the page i.e., the system does not learn from the polling patterns. The input specification language is limited. The user cannot specify the monitoring request to be dependent on the status of other monitoring requests. Change is always computed between the successive versions of the page. The user cannot specify *what* versions (window concept) of the page should participate in change computation.

**e) Xyleme**

In [37], the authors present a Dynamic Warehouse for Web -- Xyleme, which monitors the flow of incoming documents. The flow of documents consists of XML pages and HTML pages. The authors present a subscription language for specifying the pages to be monitored. Depending on type of information requested by the user the pages are monitored using either monitoring or continuous query. For query of type monitoring, changes to a page are

discovered when the system reads the page and for continuous queries changes are discovered by regularly asking the same query. Composite changes are also supported. The user cannot specify the monitoring request to be dependent on the status of other monitoring requests. The users also cannot specify what versions of the page should participate in change computation.
The following are some of the other distinct characteristics of WebVigiL:

1. Properties of monitoring requests can be inherited: The user has the option of specifying the monitoring request to be dependent on the status of other monitoring requests. One can specify the start/end of a request to be the start/end of another request.

2. Flexible specification of versions: All the above systems compute changes between two successive pages. In WebVigiL the user can explicitly specify the pages that can participate in change detection.

**f) Copernic Tracker**

Is a software aimed at monitoring websites. It can track changes in the text and images and monitor for the presence of specific text. The system however does not allow for specifying how much emphasis to place on monitoring different aspects of the web page and does not provide a utility for restricting the detection to a specific zone. Furthermore, it does not reveal performance data that discuss speed or accuracy.

**g) Website Watcher**

It includes the ability to monitor pages behind logins [32]. The system offers limited freedom for selecting a zone to monitor and lacks a proper user interface to show the changes. This system also does not provide objective performance data other than subjective user reviews. Full version of this software is available on a 30 day free trial. It track an unlimited number of pages and user can choose to ignore HTML tags, images/banners, number of dates. User can enter user names and passwords for password protected pages that user wish to track. Pages can be checked once a day, once a week, or on a specified day or days of the week. User can even specify the checking frequency during a day either in hours or minutes.

Prices are 29.95 Euros for the basic edition, 49.95 for the personal edition, 99 Euros for a single user business licence, 1990 Euros for a site license and 9990 Euros for an enterprise licence.

## 2.5 Services to Track Changes

Many services are also available that track web page content for changes. Some are free whilst other priced service may offer limited facilities as a free taster.
This review looks at two types of services : web based services and software programs for PCs and intranets.

### 1. Change Detection

This is a free service allowing us to ttrack an unlimited number of pages. The frequency of the alerts can be daily, weekly or monthly and user can choose to only have alerts for "sizeable changes", when content has been added or removed, or for specific keywords. The email merely alerts us to the fact that there have been changes; user have to click on a link in the email to view them in the change detection web site. RSS as well as email alerts available.

### 2. Change Detect

The home page gives the impression that this is a totally free service. It isn't. The free trial version allow user to track maximum of 5 number of pages and last only for two weeks .User receive web page change notification via email, ICG or text message. The subscription service allow user to track the pages and password protected pages. Change Detect personal costs USD 1.95 a month and track 10 web pages. Change Detect plus costs USD 14.95 a month for 100 web pages and allows the user to set up keyword and phrase notification triggers and colour coded notifications. Change Detect professional costs USD 39.95 a month for 500 web pages with content checked twice daily.

### 3. Femtoo

This service allow the user to have upto10 "trackers" and a maximum of 5000 characters trcked for changes free of change. User can select the contents to be tracked and how it is to

be treated (text or numerical). Alerts can be delivered by email, SMS and a personal tracker RSS Feed. Content is checked every 30 minutes.

**4. Infominder**

The free trial tracks up to 10 pages free of charges for 30 days and allow the user to associate categories and description with a "minder". Within the advanced options user can specify how often, in days, the page is to be checked. The most frequent check allowed for free is every I day. User can also specify the minimum number of changes that must occur before the users are notified and any keywords or phrases that must appear in the changes.

Infominder professional costs USD 30/yr to track 100 pages, USD 60/yr for 250 pages, USD 120/yr for 500 pages and USD 250/yr for 1000 pages.

**5. Watch That Page**

It is a free service run by consulting, a Norwegian company that specializes in software development. User can track an unlimited number of pages, which can be grouped into folders and tracked an a daily or weekly basis. There is a keyword matching option that filters the change that are relevant to the user, for example if user is only looking for news where a certain term or phrase occur such as a company or a product name. Channels enable the user to divide the pages into groups based on importance or content type. Each channel can have different properties: some can have keywords matching and daily reports while others can be checked less frequently and report all changes. Email alerts can include the text that has changed on used page or just list the URLs of pages that have changed. If user are a professional or heavy user, user are requested to pay a free. Watch that page will notify user by email if user fall into the heavy user category.

**6. Page2RSS**

Page2RSS track web pages for changes and notifies the user of those changes by RSS. Simply type in the URL of the page which the user wish to track and then add the feed URL to user favourite feed reader. Excellent tool for pages that do not offer their own RSS feeds**.**

### 7. Websnitcher

Websnitcher is a free service that checks user pages every 3 hours and gives user a detailed list of changed text areas. Email notifications are sent once a day and it also generates RSS feed from the collected data. It alerts user only to what it considers are relevant changes; the intelligent filter tries to ignore changes in date and what it considers to be irrelevant text areas such as how many users are online.

# Chapter 3 Problem Statement

World Wide Web is the most important media for sharing information resources and continues to grow at an alarming rate. Users surfing the web may either be searching for specific information or simply browsing the web. Different users may be interested in knowing changes to specific web pages and want to know when those changes take place. Users are mainly interested in knowing the changes that have taken place in a web page instead of looking for the whole page. This helps the users to reduce the browsing time and also to detect the changes efficiently. While detecting the changes in the new and old web pages three main problems are encountered.

These problems and their proposed solutions are:

1. How can changes in web pages be detected effectively?

*Solution:* First the problem is classified into two broad categories- the structural changes and the content based changes. The changes are detected using a hypertext document tree encoding of the document. Tag value is assigned to each non leaf node. This tag value is used to compared and thus detect the changes.

**2.** How can changes between different versions of web documents be presented/extracted effectively?

*Solution:* An efficient and effective algorithm has been proposed for comparing different versions of web documents. The change extractor based on this algorithm includes two phases: document tree construction and comparison module which will compare the two trees and detect the changes. The crawler is used to save the HTML code of the web page whenever the URL is provided by the user. This HTML code is used by tree building module to design the tree. Then these two trees will be compared to detect the changes.

**3)** How should a change both in structure or content be detected and evaluated?

*Solution:* A specialized set of  has been used for HTML pages. The list shows the relationship among the tree nodes, i.e. a child or a parent. Is analysed to determine the differences between the older and the newer version of the web page.

# Chapter 4 Proposed Algorithms

## 4.1 Architecture

The general architecture is shown below in the figure 4. This architecture contains the comparison module where various algorithms can be applied to compare the two web pages. This architecture follow a path from the start state to the end state these states are denoted by the ellipse. The user will input the web pages in which the changes is to be detected. The user will input the old web page and the new web page. The input module then asks the crawler to fetch the pages. The old web page will be fetched from archive and the new web page will be fetched from the site. These pages will be saved in page report archive. These pages will be sending to the tree builder module via manager. Tree builder module will design the trees corresponding to the old and the new web page. These trees will be passing on to the comparator module. In this module comparison algorithm is applied to find the difference between the two web pages. Then the result will be given to the presentation module. Presentation module prepares a report from these results and save it in page report archive. Notification centre will compare all the reports related to a page compile them and send it to the user. There is an inverse tree builder module which will generate the page from the tree and give the pages to the browser to show the result.

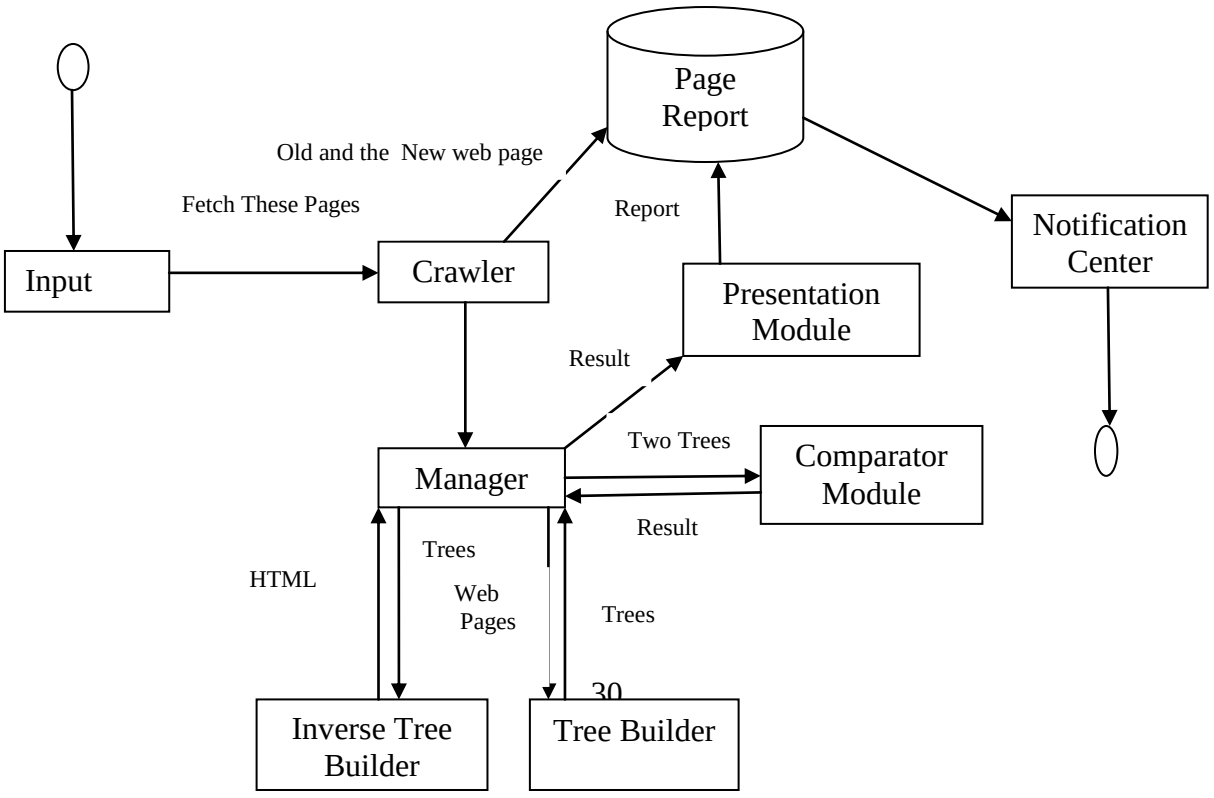


**Figure 9: Architecture of a web Page Change Detection System**

## 4.2 PROPOSED ALGORITHM

An algorithm has been proposed to find structural as well as the content changes in a web document. Before finding the changes URL of a particular web page is to be searched. A crawler has been designed to save the HTML code of that particular page. Proposed algorithm is Subdivided into two phases. The proposed algorithm has been implemented in C# and uses the inbuilt classes and methods of C#.

**Phases of Proposed Algorithm**

The proposed algorithm is divided into two phases:

1. Tree Construction Algorithm
2. Change Detection Algorithm

### 4.2.1 Phase 1: Tree Generation Algorithm

This algorithm describes how to develop a tree corresponding to a particular web page. Whenever a user provides a URL for a web page web crawler will search the particular web page. Store the web pages in the form of a HTML code. Then this HTML code is used by the tree building module to extract the HTML tag. After extracting the tags it assign index number to the tag and then the node number. After assigning the node number to each node it finds the number of child of each node. With the help of node number and the child of each node the tree building module build the tree corresponding to the web page. For extracting the tags one will look for the index number of the tag. If the node exists then assign index number and nodename to each tag and then add the node to the tree. If the tag does not exist then it will return -1. For assigning node number we will check for the closing brackets and the opening brackets. If it is a closing bracket then we don't assign any node number and if it is a opening bracket then we will assign node number to that tag. Same is for to assign the child to a node if it is a closing bracket then it won't have any child and if it is a opening braces then we will check for opening bracket – closing bracket, if it is equal to 1 then it is a child node oh that node.

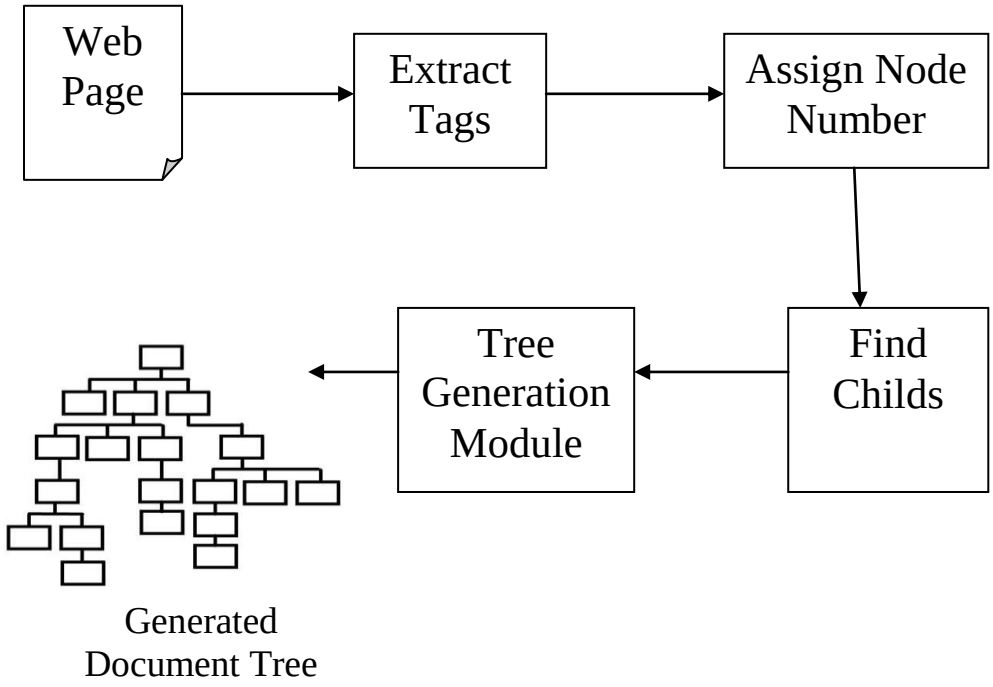



**Figure 10 : Tree Genertaion Module**

**Step 1: Extract Tags**

**Input:** Web page searched by the crawler

**Output:** Tags are extracted from the web page

1. Find index No. of each tag
2. If tag does not exist
   then
   return -1.
3. Assign index No. and nodename to each node
4. Add the node to the tree

node.index = index;

node.nodename = "html";

tree.Add(node);

**Step 2: Assign Node Number To each node**

**Input:** Extracted tags are given

**Output:** Node Number is assigned to the tag

1. Check for each tag t
2. If (It is a closing tag)
   Then
   do not assign any Node number.
3. Else
   Count opening tag and closing tag until we get the closing tag of t
   if (countopen - countclose == 1)
   tree[j].nodeno = k; k++;

**Step: 3 Find Childs**

**Input:** Nodes will be given as a input

**Output :** Find the child of each node

1. Check for each tag t
2. If ( It is a closing tag)
   then
   don't assign any child.
3. Else
   if (countopen - countclose == 1 && !subs.Equals("/"))
   tree[i].childs = tree[i].childs + tree[j].nodeno + ",";

### 4.2.2 Phase 2: Change Detection Algorithm

This algorithm is used to detect the structural as well as content changes. This algorithm is subdivided into four steps. First step calculates the hash value which is assigned

to each leaf node. Second step is used to calculate the tag value. Finally the last step is used to detect the changes in a web page. To detect these changes hash value is calculated for each leaf node. Further this hash value is used to calculate tag value which will be assigned to each non leaf node in the tree. In order to assign hash value to each leaf node we assign a number to all the tags that commonly appear in a web page. Also the level of each node is calculated with the help of the tree. For every node child of each node is find the node which is having no child is considered as a leaf node. Sum of nodename and level number is the hash value for each leaf node. After assigning the hash function to leaf node tag value is assigned to every non leaf node. Tag value is calculated in the bottom up manner. Tree is traversed from bottom to top. Tag value is equal to the sum of tag value assigned to all its child. To find the changes, nodes at each level are compared in the old level as well as in the new tree. If the nodes at a particular nodes is less or more than the old tree then the tag value of the parent level are compared. If the tag value of new tree is greater than the tag value of the old tree it means that the new node is added and if tag value is less it means that the node is deleted.

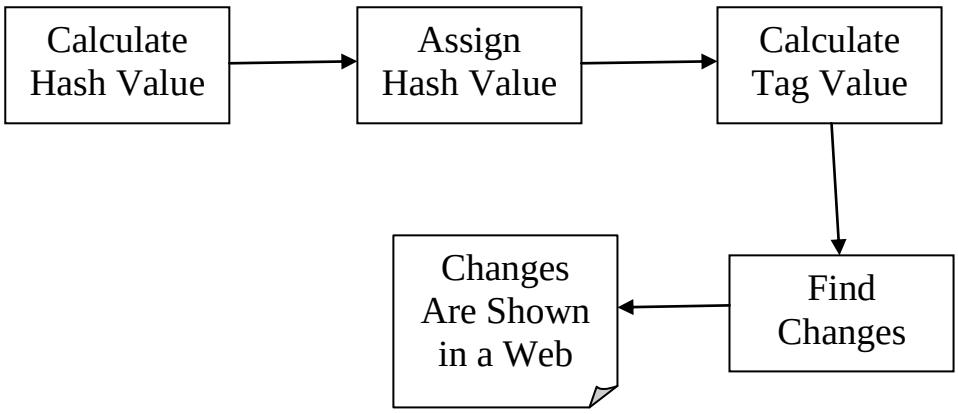



**Figure 11: Steps of Change Detection Algorithm**

**Step 1: To Calculate Hash Value**

**Input:** Nodename is name of leaf node to which hash value has to assign, level is depth at which that node exists.

**Output:** Hash value of node.

1. Provide some number to all tags

   i.e. nodename which can exist in webpage or can take enumeration of tags.

2. Foreach node n in tree

   if (n.childs.Equals(""))

   n.tag = GetHash(n.nodename, n.level);

**Step 2: Assign Hash Function**

**Input:** All nodes of a tree.

**Output:** Hash value of leaf nodes.

1. For each node of a tree.
2. If (node.childs.equals(null)).

   Call to calculate hash value function.

   Hash function = nodename + Levelno.

**Step 3: Calculate Tag**

**Input:** All nodes of a tree.

**Output:** Tree with calculated Tag value of all nodes.

1. Start from bottom of a tree for each node n.
2. We will assign tag values to non leaf nodes.

   If (!node.childs.equals(null))

   n.tag = sum of tag values of it's all childs

**Step 4: Find changes**

**Input:** Old tree and new tree to find changes in them.

**Output:** Nodes added or deleted in old tree.

1. Find nodes at each level of tree in old tree as well as new tree.

   if (!n1.childs.Equals(""))

   nextchildnodes = nextchildnodes + n1.childs + ",";

2. Compare nodes at each level

If number of nodes at any level of new tree are less than or equal to node at same level of old tree

3. Compare the tags of parents of that level
4. If tag of any new parent(n) > old parent(o)

   if (oldparentChild.Count() > newparentChild.Count())

   int d = n1.tag - n3.tag;

   5. Then

   addednode=n.tag – o. Tag

   6. Else

   int d = n3.tag - n1.tag;

   7. Deletednode = o.tag – n.tag.

# Chapter 5 Experimental Results

## 5.1 Introduction

This section summarizes the results of the research work done using the proposed algorithm. The proposed algorithm detects the changes that are made in the web pages. The proposed algorithm is divided into two sub algorithms. The first algorithm constructs the tree corresponding to the web pages and the second algorithm detects the structural as well as content changes. Proposed algorithm is mainly based on the tag value which is assigned to the non leaf nodes in order to detect the changes and the hash function which is assigned to each leaf node. While designing the tree it is traversed in the depth first search manner.

### 5.2 Illustrative Examples

This section shows the working of the proposed algorithm with the help of few examples. Original tree and the corresponding modified tree are shown and the changes are presented with the help of screenshots.

**Example 1:** Consider the initial version of the tree in figure 12. The given tree is modified to detect the structural and content changes as shown in figure 13 and 15. The results are shown with the help of snapshots in figure 14 and 16.

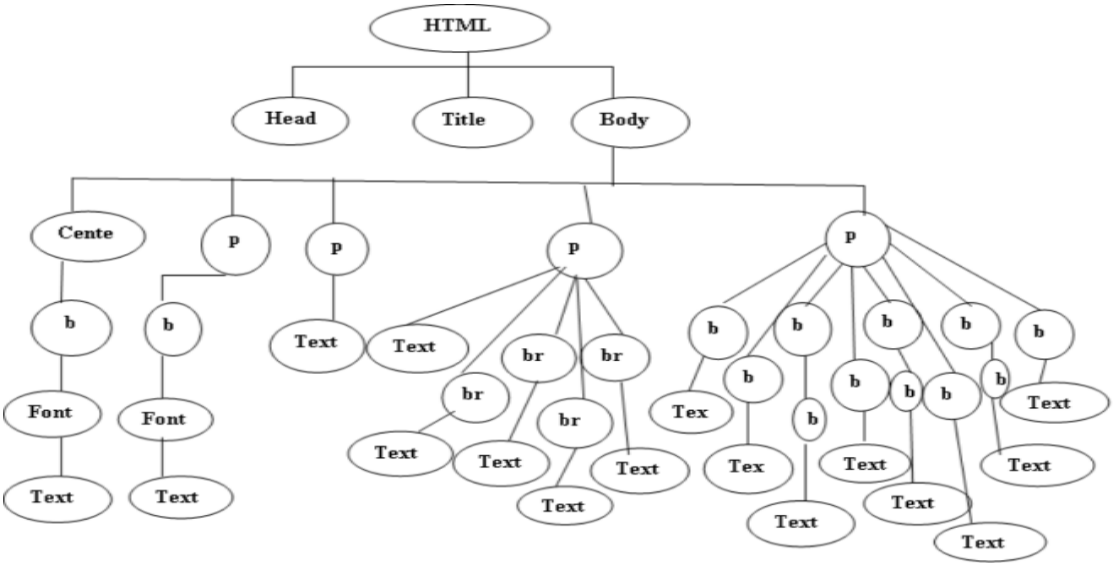



**Figure 12: Initial Tree**

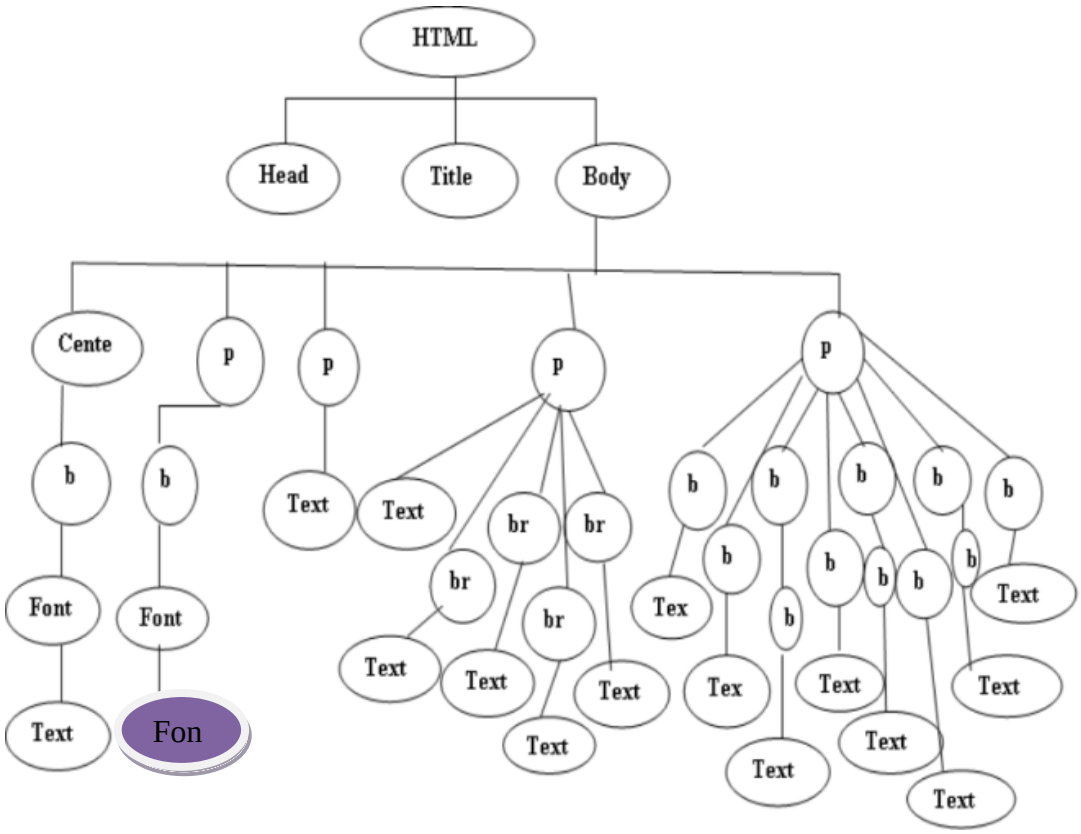



**Figure 13: Modified Tree**

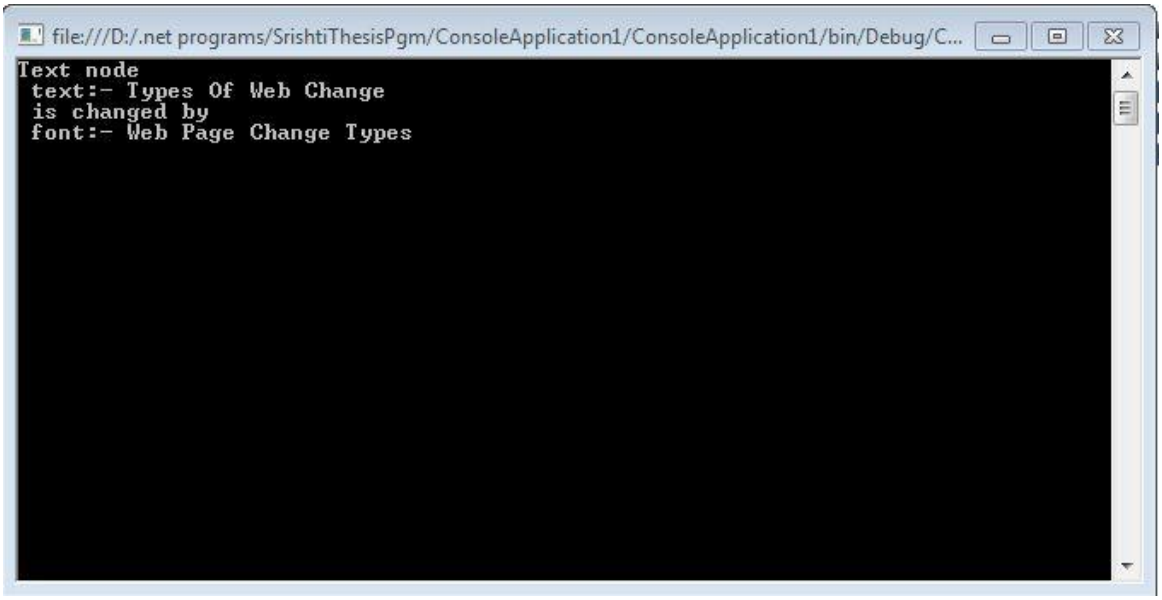



**Figure 14: Snapshot Showing the Content Change**

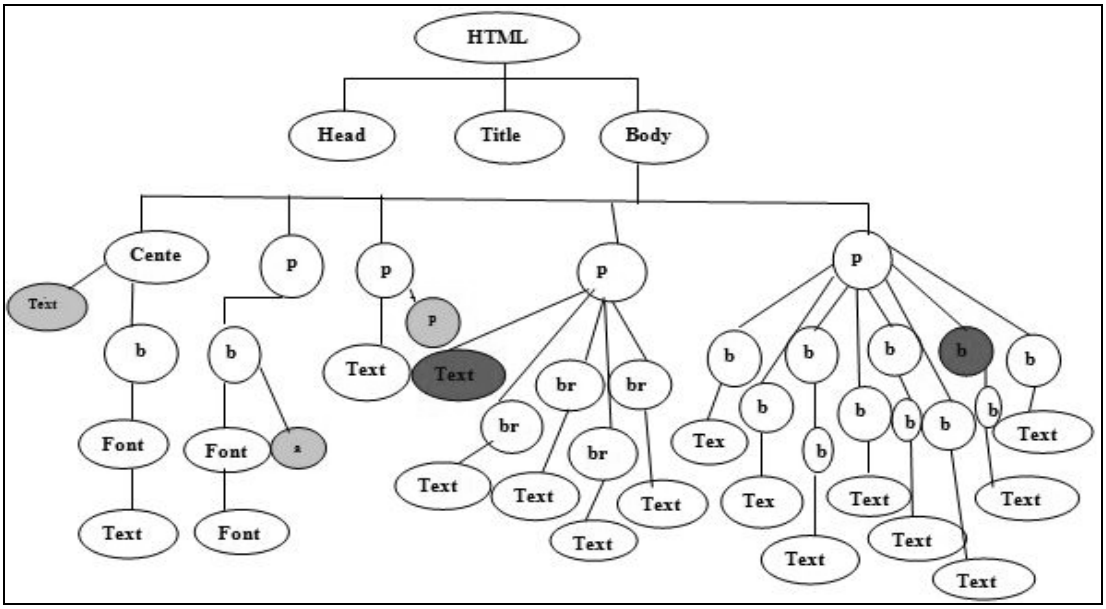



**Figure 15: Modified Tree**

New nodes that are added in the tree

Nodes that are deleted from the tree

In the initial tree various nodes are added and deleted to detect the structural change. The result is shown below with the help of a snapshot.

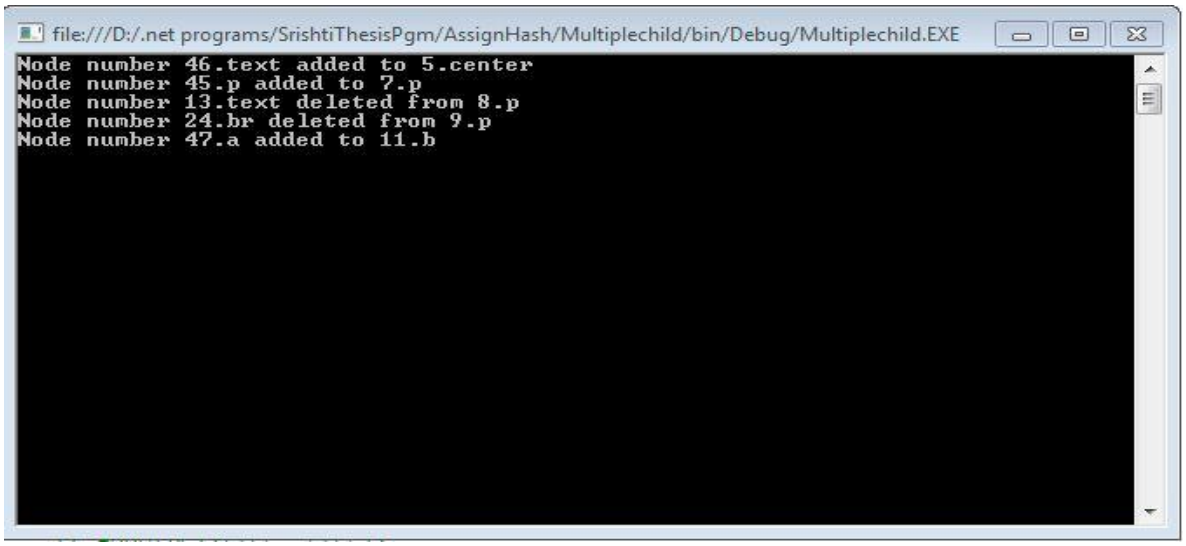



**Figure 16: Snapshot Showing the Structural change**

**Example 2:** Consider the initial version of the tree in figure 17. The given tree is modified to detect the structural and content changes as shown in figure 19. The results are shown with the help of snapshots in figure 18,20 and 21 .

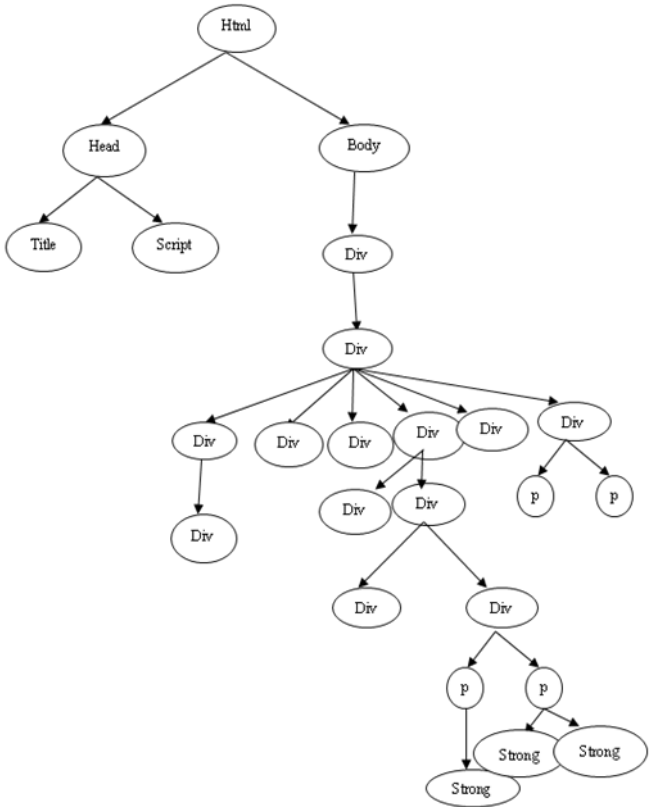



**Figure 17: Initial Version of the Tree**

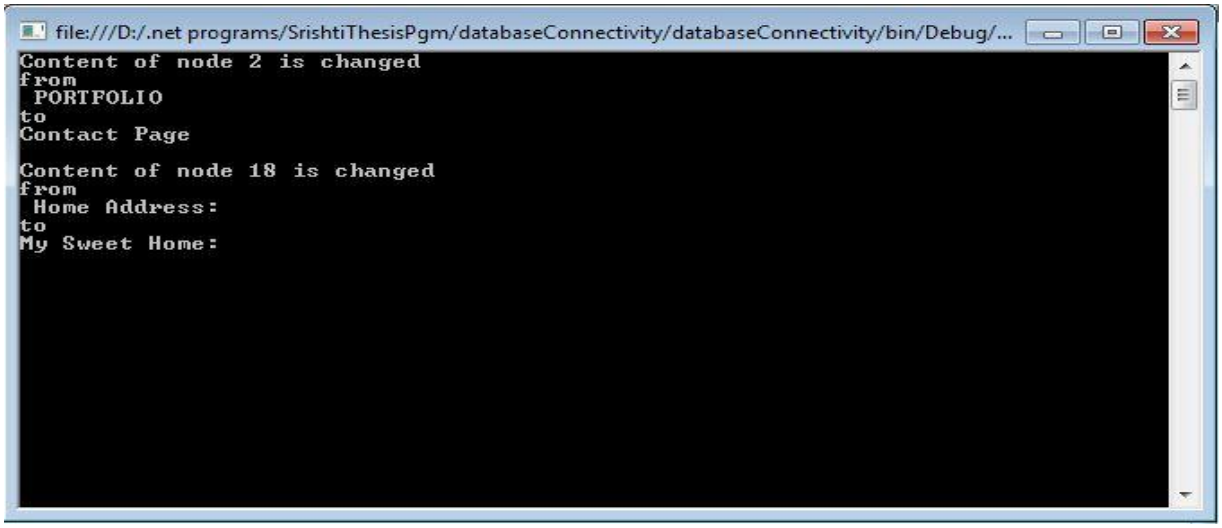


**Figure 18: Snapshot Showing Content Change**

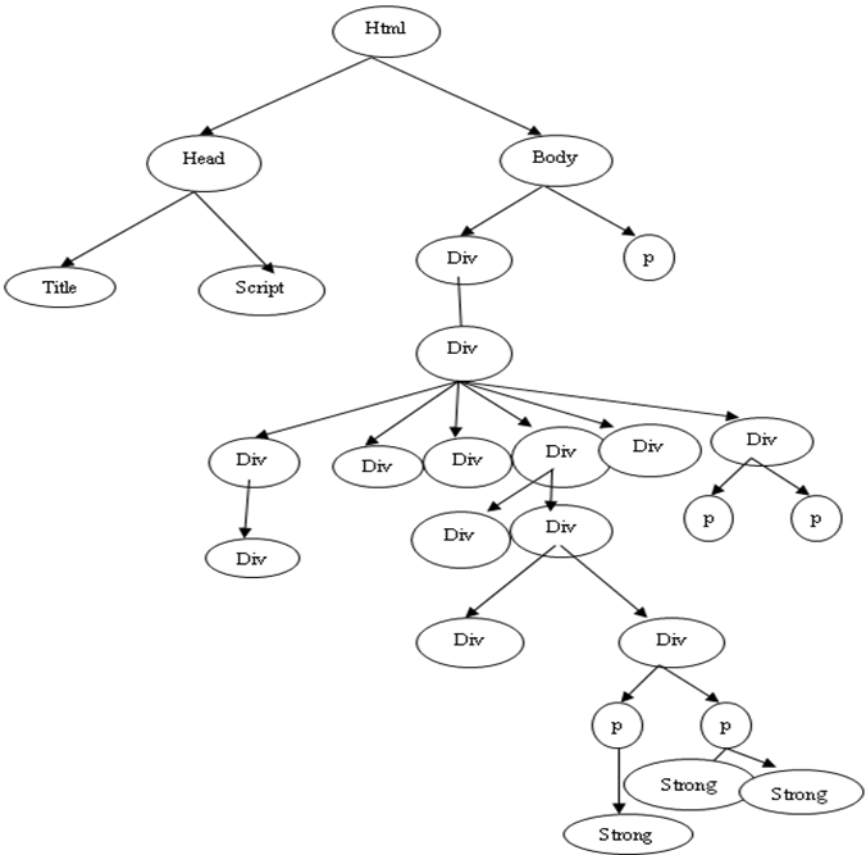



**Figure 19: Modified Version of the Tree**

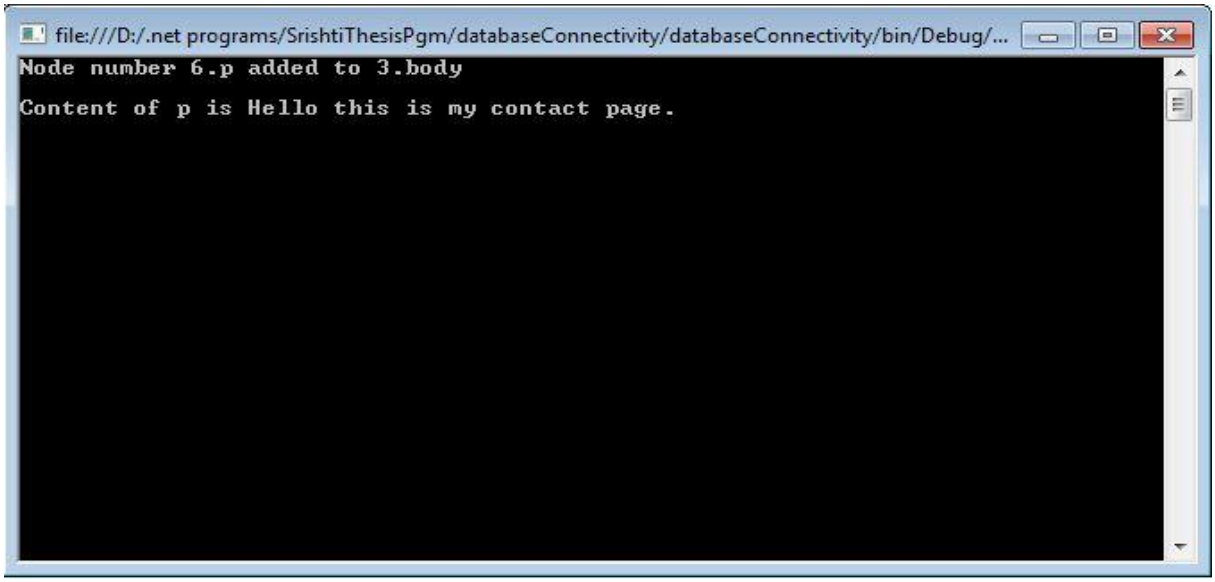



**Figure 20: Snapshot showing Addition of a Node**

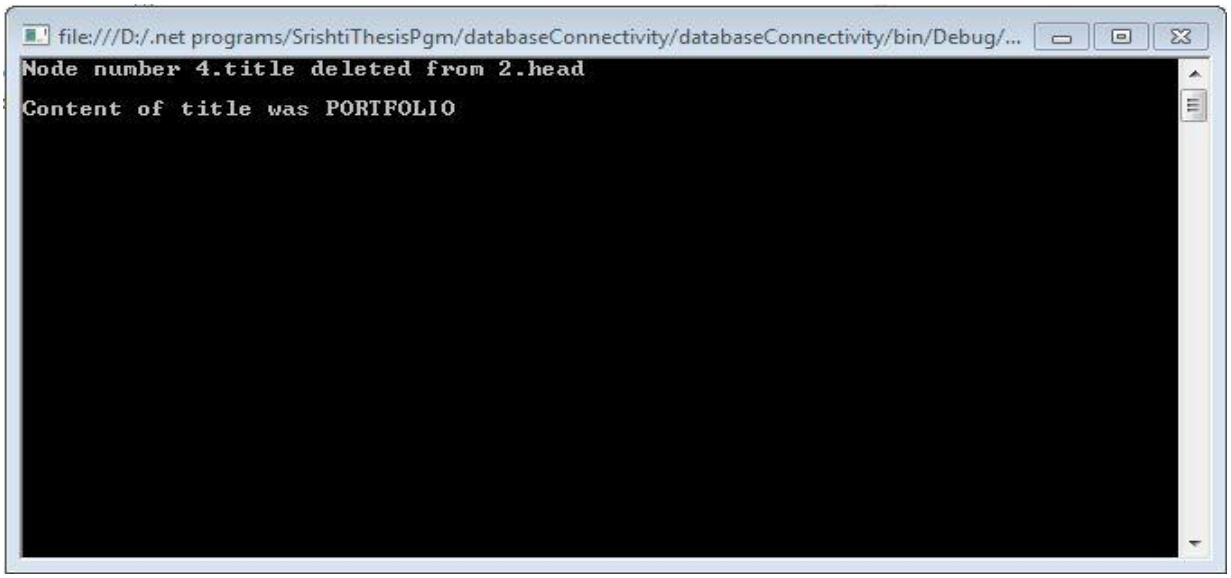



**Figure 21: Snapshot showing  Deletion of a Node**

# Chapter 6 Conclusion

## 6.1 Conclusion

This study is to develop an efficient web page change detection system which will detect the changes in a web page. The proposed algorithm extracts changes between different versions of web pages. The algorithm is able to make out structural as well as content based changes by developing an efficient application in C# which scores on other techniques on the basis of simplicity and understandability. This algorithm detects the changes, based on the change in the tag value. Tag value is assign to every node. To find the changes algorithm traverses each node in two version of the tree.

Tree traversing is based on the depth first search. Two different algorithms are used first one is used to construct the tree corresponding to the web pages and the second one is used to detect the changes which are made in the web page. While detecting the changes in the new and old web page three main problems are encountered. Solution for these problems has been provided through these algorithms. This system is useful for saving the browsing time and also gives the user time to time update regarding the changes which will occur in a web page.

## 6.2 Future Scope

For further study we can consider the pages having frames are not handled. A page with multiple frames has a base page, in which the reference to the pages in frames is given. The algorithm can be parallelized through multithreading, which can increase performance significantly. Another suggested future work involves the application of this algorithm to the Deep (Hidden) web, mostly concerning dynamic web pages that are generated in response to submitted queries.

# References


[1] Artail H. and Fawaz K., "A Fast HTML Web Page Change Detection Approach Based on Hashing and Reducing the Number of Similarity Computations", *journal of Data & Knowledge Engineering* , vol. 66, no. 2, pp 326–337, 2008.

[2] Buneman P., Davidson S., Fan W., Hara C., and Tan W. , "Keys for XML" in *Proceedings of International Conference on Computer Networks,* vol.39, no.5, pp 473–487, Aug 2002.

[3] Chen Y. F., Koutsofios E., WebCiao*: "*A Website Visualization and Tracking System. in WebNet", 1997.

[4] Chawathe S., Rajaraman A., Molina H. G. and Widom J., "Change Detection in Hierarchically Structured Information", *in proceedings of ACM SIGMOD International Conference on Management of Data,* vol. 25, no. 2, pp 493-504, June 1996.

[5] Chawathe S. and Molina H. G., "Meaningful Change Detection in Structured Data", *in the proceedings of ACM SIGMOD International conference*, vol. 26, no.2, pp 26-37, May1997.

[6] Chawathe S. , "Comparing Hierarchical Data in External Memory", *in the proceeding of 25$^{th}$ international conference Very Large Database,* pp 90- 101, 1999.

[7] Chawathe S., Abiteboul S. And Widom J., "Managing Historical Semistructured Data", *Theory and Practice of Object Systems,* vol. 5, no. 3, Pp. 143–162,1999.

[8] Chakravarthy S. and Mishra D., *"*Snoop: An Expressive Event Specification Language for Active Databases*" , Journal of Data and Knowledge Engineering,* vol. 14 no.1, pp.1-26, 1994.

[9] Cobena G., SAbiteboul S., and Marian A., "Detecting Changes in XML Documents," *in Proceeding of 18th International Conference on Data Engineering*, pp. 41-52, 2002.

[10] DeltaXML by Mosell EDM Ltd. Available at http://www.deltaxml.com. Accessed May 2003.

[11] Diffxml Available at: https://sites.google.com/site/diffxml/

[12] Douglis F., Ball T., Chen Y.F. and Koutsofios E. , "The AT&T Internet Difference Engine: Tracking and Viewing Changes on the Web*", in World Wide Web, vol. 1, no. 1,* pp. 27-44,1998.

[13] Jyoti J., Sachde A., and Chakravarthy S., "CX-DIFF: A Change Detection Algorithm for XML Content and Change Visualization for WebVigiL, " *Data and Knowledge Engineering*, vol. 52, no. 2, pp. 209-230, 2005.

[14] Khoury I., Student Member, IEEE, El-Mawas R.M., Student Member, IEEE, El-Rawas O., Mounayar E. F., and Artail H., Member, IEEE ,"An Efficient Web Page Change Detection System Based on an Optimized Hungarian Algorithm", *in IEEE Transactions on knowledge and Data Engineering*, vol. 19, no. 5, pp 599-612, May 2007.

[15] Khandagale H. P. and Halkarnikar P.P., "A Novel Approach for Web Page Change Detection System" *International Journal of Computer Theory and Engineering*, vol. 2, no. 3, pp. 364-367, June 2010.

[16] Lu, B., Hui S. C., and Zhang Y.," Personalized Information Monitoring over the Web" *in Proceeding of First International Conference on Information Technology and Applications(ICITA)*, 2002. Australia.

[17] Liu L., Pu C., and Tang W., "*WebCQ:* Detecting and Delivering Information Changes on the *Web*" *in Proceedings of International Conference on Information and Knowledge Management (CIKM), pp. 512- 519,* 2000. Washington D.C.

[18] Lindholm T., "A 3-way merging algorithm for synchronizing ordered trees – the 3DM merging and differencing tool for XML", *Master's thesis, Helsinki University of Technology, Dept. of Computer Science,* Sept. 2001.

[19] Mascolo C., Capra L., Zachariadis S. and Emmerich W., "xmiddle: A Data-Sharing Middleware for Mobile Computing," *journal Wireless Personal Communication: An International Journal*, vol. 21,no. 1, pp. 77- 103, 2002.

[20] Mind-it, http://www.netmind.com.

[21] Pandrangi N., "WebVigiL: Adaptive fetching and user-profile based change detection of HTML page*", The University of Texas at Arlington*, 2003.

[22] Pehliwan Z., Saad M., Gan S. And Carski " Vi-DIFF: Understanding Web Pages Changes" *in a Proceeding of 21st International Conference on Database and Expert System Application*, pp. 1-15,2010.

[23] Sanka A. , "A Dataflow Approach to Efficient Change Detection of HTML/XML Documents in Webvigil", *M.S Thesis, University of Texas at Arlington*, 2003.

[24] Song, Bhowmick. "BioDIFF An Effective Fast Change Detection Algorithm for Genomic and Proteomic Data" *in Proceedings of the Thirteenth ACM conference on Information and knowledge management*, pp. 146-147, Nov. 2004.

[25] VM Tools by VM Systems, last release Feb. 2002 Available at: http://www.vmsystems.net/vmtools/, accessed May 2005.

[26] Wang Y. , DeWitt D., and Cai J., "X-Diff: An Effective Change Detection Algorithm for XML Documents," *in Proceedings of 19th International Confrence Data Engineering*, pp. 519-30, 2003.

[27] World wide web. http://en.wikipedia.org/wiki/World_Wide_Web.

[28] WebCQ product. http://www.cc.gatech.edu/projects/disl/WebCQ.

[29] Web crawler: http://en.wikipedia.org/wiki/Web_crawler.

[30] Webpages: http://en.wikipedia.org/wiki/Web_page

[31] Webpage: http://www.webopedia.com/TERM/W/web_page.html

[32] WebSite-Watcher product. <http://www.aignes.com>.

[33] XML Diff and Merge Tool by IBM alphaWorks, last update Mar. 2001,Available at:

http://www.alphaworks.ibm.com/tech/xmldiffmerge, accessed May 2005.

[34] Xml diff and patch Komvoteas K., 2003, XML Diff and Patch Tool, Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.58.4650.

[35] Xml tree and patch Microsoft Corporation. Available at http://apps.gotdotnet.com/xmltools/xmldiff/*accessed June 2003*.

[36] Xu H., Wu Q., Wang H., Yang G., and Jia Y., "KF-Diff+: Highly Efficient Change Detection Algorithm for XML Documents", *in Proceedings of. CoopIS/DOA/ODBASE*, pp.1273-1286, 2002.

[37] Xyleme, http://www.xyleme.com.

[38] Yadav D., Sharma A.K., Gupta J.P. "Change Detection In Web page," *in a proceeding of 10th international conference on information technology*, pp. 265-270, 2007.

[39] yadav D., "Design Of A Novel Incremental Parallel web crawler", *Phd thesis, Jaypee Institute Of Information Technology University*, 2009.

[40] Yao J. T., Zhang M. "A Fast Tree Pattern Matching Algorithm for XML Query" *IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 235-241. Sept. 2004.

# Paper Published

1. Goel S. and Aggarwal R. R, "Comparative Analysis of Web Page Change Detection System", in Proceedings of *International Conference on Competitiveness and Innovativeness in Engineering, Management and Information Technology,* vol. 2, no. 2, 2012.

2. Goel S. and Aggarwal R. R, "A Tree Based Algorithm for Web Page ChangeDetection System, in Proceedings of *International Conference on Advancements in Computing and Communications,* 2012.

3. Goel S. and Aggarwal R. R, "Comparative Analysis ofWeb Page Change Detection System", in Proceedings of *International Conference on Advancements in Computing and Communications*,vol.2 IISN: 2249-3905, 2012.

4. Goel S. and Aggarwal R. R, "A Tree Based Algorithm for Web Page ChangeDetection System", *International Journal of Computing and Business Research, special issue on computing and communication.* **[Accepted]**

5. Goel S. and Aggarwal R. R, "An Efficient Algorithm for Web Page Change Detection", *International Journal of Computer Applications.* **[Communicated]**