A fast HTML Web page change detection approach based on hashing and reducing the number of similarity computations

Hassan Artail[1] and Kassem Fawaz

Department of Electrical and Computer Engineering
American University of Beirut
P.O.Box: 11-0236, Riad El-Solh, Beirut 1107 2020, Lebanon
Phone: +961 1 350000, ext. 3520, Fax: +961 1 744462
E-mails: {hartail, kmf04}@aub.edu.lb

ABSTRACT

This paper describes a fast HTML Web page detection approach that saves computation time by limiting the similarity computations between two versions of a Web page to nodes having the same HTML tag type, and by hashing the web page in order to provide direct access to node information. This efficient approach is suitable as a client application and for implementing server applications that could serve the needs of users in monitoring modifications to HTML Web pages made over time, and that allow for reporting and visualizing changes and trends in order to gain insight about the significance and types of such changes. The detection of changes across two versions of a page is accomplished by performing similarity computations after transforming the Web page into an XML-like structure in which a node corresponds to an open-closed HTML tag. Performance and detection reliability results were obtained, and showed speed improvements when compared to the results of a previous approach.

Keywords: Web page change detection, change monitoring, similarity computation, HTML, tree similarity.

[1] Corresponding author

1. Introduction

Changes occurring in web pages are best classified as content changes (e.g., deletions, and additions of text), layout change (e.g., changes in the position of elements in the page), and attributes change (e.g., changes in fonts and colors) [10]. Hence, in addition to tracking content changes, any useful detection system should also be able to track changes in layout.

Most change detection approaches are computationally complex and require non-polynomial running time [4, 5]. Some of the well-known systems that fit the above characteristics are HTMLDiff [11], NetMind [22], WebCQ [20], WebVigiL [3], and CMW [10]. These basically work by estimating the rate of change that occurred between the reference web page and its updated version, and eventually locating the differences between them.

In this work, we propose an efficient method for detecting web pages changes. It generates subtrees corresponding to elements that are directly connected to the `BODY` HTML tag. The tags found are used to *mark* the nodes in the subtrees belonging to the two pages being compared and are employed to limit the similarity computations to nodes having the same mark. Subtrees with the highest average similarity coefficients are considered to be the most similar. Using this information, changes in the updated version of the Web page are identified and located. Additionally, a scheme was employed to speed up the algorithm through hashing the web page in order to provide direct access to subtree nodes during the comparison process.

2. Related work

One challenge that has not been addressed sufficiently in the literature is the large time it takes to compare HTML web pages, a task that is necessary to detect and locate differences between them. This is because in order to infer changes between two HTML web pages, all the different HTML nodes (corresponding to content and attributes of tags) have to be compared,

typically leading to an NP-hard problem [4, 5]. In this regard, the approach in [10] uses the $O(N^3)$ Hungarian algorithm to compute the maximum weighted matching on a weighted bipartite graph and has a running time in $O(N_2 \times N_1^3)$, where N_1 and N_2 are respectively the number of nodes in the old page and in the new (changed) page. This running time becomes significantly large as the value of N_1 is increased, i.e., as the selected region of interest in the old page (to be monitored for changes) is increased from a small portion to the whole page.

An existing web change detection product is *Copernic Tracker* [24], which 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 discusses speed or accuracy. A second product is *WebSite-Watcher* [29], which includes the ability to monitor pages behind logins. 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. A third system is *WebCQ* [28], which offers personalized delivery of change notifications and summarization plus prioritization of changes. Notifications can be sent via email to the user reporting content changes only. These describe modifications to text, hyper-links, image references, and keywords, in addition to reporting modification date and page size changes. The authors however promise to implement into WebCQ a structure-aware change detection and difference algorithm in the future [19].

In terms of published research work, several papers were found that tackle the design of efficient algorithms for detecting changes in Web pages. In [7], [27] and [14], various *diff* algorithms are described for detecting changes in XML documents. The algorithm in [14] 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 works in [7] and [27] use *edit scripting* to compare two documents and transform the pages to trees according to the XML structure. The strength of these algorithms lies in their low time-complexity, which is in the order of $O(n\log n)$. However, this high performance cannot be achieved when comparing HTML documents as it relies on certain XML features. Edit scripting alone is not sufficient for achieving $O(n\log n)$ or polynomial running time, especially if move operations (parts of the document are moved around) are to be considered. In fact, it has been shown that edit scripting with move operations is NP-hard [15, 5]. Basically, an edit script on a tree $\mathbf{T}_1$ is a sequence of operations (insertions, deletions, and updates) that generates another tree $\mathbf{T}_2$. The discussed *diff* algorithms consider that if a node in $\mathbf{T}_1$ is matched to a node in $\mathbf{T}_2$, then their parents are also matched. Under this convention, two trees that have unmatched roots can never include matches: if M is a matching from $\mathbf{T}_1$ to $\mathbf{T}_2$ then $M=\phi$ if and only if (Root ($\mathbf{T}1$), Root ($\mathbf{T}2$)) $\notin$ M, according to [27]. Under this hypothesis, high performance can be achieved since a matching between two nodes can now be propagated bottom-upwards in the tree. Unlike HTML, XML enforces structure in that there are no unclosed and out-of-order tags, and more importantly, in a well-formed and typical XML document, children of some node tend to have a parent of a *unique tag type*. In the example below for instance, the tag type `<book>` is a unique child of the tag type `<library>`:

```
<library><book>...</book><book>...</book></library>
```

In other words, it is unlikely to find `<book>` as a child of another tag type. Typically, well formed XML documents are used for structuring data, and in that context, the hierarchy tends to

be quite well defined. However, in HTML markup, this relationship between tag types and their ancestors does not hold, as we obviously can have the same tag type present in several subtrees that are rooted at different nodes. For such reasons, the parsing and matching of HTML is much more difficult [18].

The work in [10] transforms an HTML document into a tree structure and categorizes node information into content, structure, and attributes. Three similarity measures are used to detect changes of these three categories: intersect (percentage of similar words), typedist (measure of the position of the elements in the tree), and attdist (measure of the relative weight of similar attributes), respectively. In searching for the most similar subtree between two pages, the system supposedly uses the Hungarian algorithm [16], but no details on its use are given. The experimental results only show the effects of the emphasis measures on detection accuracy and do not discuss speed performance. It should be mentioned that this approach is able to detect node type changes (e.g., a `UL` node changing to an `OL` node, or vise versa, in the document tree) whereas our proposed approach applies the node comparison between same types, and thus will not detect such changes. An earlier change detection system was described in [8]. This system, which was called AIDE, provides personalized views of how pages on the Internet change. It uses the so-called *HtmlDiff* algorithm, which is built on top of the UNIX *diff* utility [13]. It tries to find a common (not necessarily contiguous) subsequence of two sequences of words that has the longest length. Although the system was designed as a server application, no speed performance data was discussed.

3 Improved detection framework

The operation of the proposed approach is generally depicted in Figure 1. The entire process is started by the crawler, which is launched by a daemon process that runs periodically. Every

time this process runs, it checks a schedule to determine if there are Web pages that need to be downloaded and subsequently compared to their corresponding stored versions. The schedule in turn is populated through a user interface that allows the user to specify the web page to monitor in addition to supplying information that controls the monitoring process. Moreover, when a user adds a URL to the list of pages to be monitored, he or she can specify a zone within the page that limits the change detection to this zone. At the conclusion of each change detection occurrence, the application writes to the disk data that describe the changes and their locations within the page. This allows the user or another program to query the application to view the actual changes on the page itself (highlighted), to generate reports that describe the type and significance of changes, or to plot the changes history.

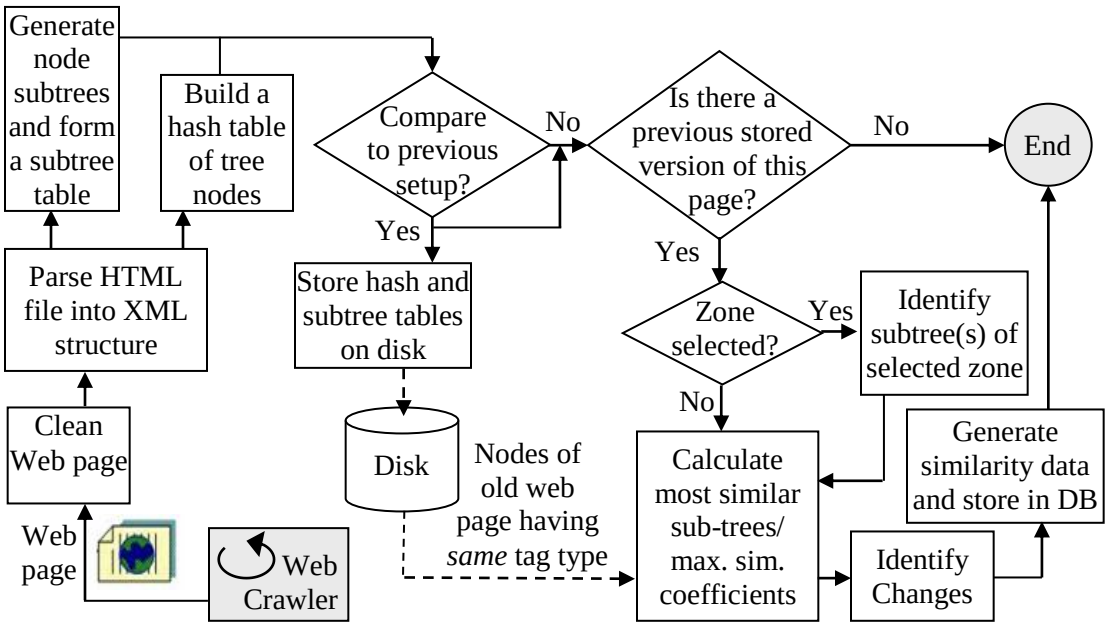

Figure 1. General diagram of proposed approach

Change detection is accomplished by comparing the newly-downloaded Web page to a previously-downloaded version stored on disk. Detection is based on calculating the similarity

among the different parts of two HTML documents and on deducing the ones that are most similar. The Web page parts that are not 100% similar are considered to have been changed.

3.1 General design

To speed up the process of web change detection to the greatest extent, the design of the detection system implemented several hashing-based techniques for direct lookup of subtree node information during comparisons, and eliminated irrelevant node comparisons by limiting them to nodes of the same type (i.e., same HTML tag). We build on the system in [10], which describes a complete framework for detecting changed parts in Web pages, and as such we will refer to it throughout the paper as the *original approach* and to our system as the *enhanced approach*. More specifically, we improve on the performance of the original approach, which has a running time in $O(N_2 \times {N_1}^3)$, through reducing the number of nodes in the edit mapping between the updated page (with N_2 nodes) and its previous version (having N_1 nodes) by restricting the similarity computations to nodes (corresponding to HTML tags) having the same tag type. In contrast, and in order to reduce the number of valid edit mappings, the original approach only considered the edges that have a similarity weight greater than a predefined threshold in an attempt to remove the redundant nodes that do not have the same type as the compared one. Moreover, it has to perform the similarity computation in order to evaluate the similarity weight, while the enhanced approach avoids it altogether by simply comparing two tag types. A second remark could be made concerning finding the most similar subtree. The original approach scans the nodes of the updated page and divides the latter into variable subtrees provided that their included number of nodes is less than four times the number of nodes in the subtree of the selected zone in the original page. The factor of four was deduced from experimental results. After the division, the algorithm proceeds to comparing all the nodes,

computes nodes similarity, and then subtree similarity for every variable subtree until it finds the one with the highest subtree similarity coefficient (i.e., the most similar one). In our case, two subtrees are compared based on the similarities of their respective nodes having identical tag types, regardless of their relative sizes. It should be mentioned though that this property introduces one limitation associated with the enhanced approach concerning the inability *sometimes* to detect changes when the root tag of a subtree is changed. We elaborate on this limitation in Section 6 and explain that it does not pose a serious issue.

Another technique that we integrated into our enhanced approach is the use of hash tables to significantly speed up the access to subtree nodes during the subtree comparison process. This technique along with limiting node comparison to those with the same tag types have allowed for achieving HTML web page change detection times that are in the order of seconds for very complex web pages (reaching 1000 or more nodes). In fact, our system achieves speeds that match those of $O(n \log n)$ approaches, which were designed specifically to work with XML documents, such as the X-Diff algorithm described in [27].

3.2 Extracted Information and Representation

The tree representation of the web page was implemented using XSLT and XPath. The extraction of essential information from the XML file and transformation into a tree representation was done with the help of the Oracle XSLT parser, where an XSL file was written to meet the criteria of the desired output XML file. The designed XSL file divides the HTML file into nodes where each one represents an HTML tag and such that for every node, the information below is extracted (a sample is illustrated in Figure 2):

- Node (HTML tag) name that becomes the value of the attribute “element”, which is also referred to as *node*.

- Path from the root element (<HTML> tag) to the concerned element. HTML elements within a path are separated by semicolons and are enclosed by the keyword *type*.
- Set of words associated with the tag or any of its child tags (referred to as *weight*).
- Set of attributes associated with the concerned HTML tag, and is repeated if there are more than one attribute. The tag's value is the HTML attribute value while its name constitutes the attribute of the *attribute* element.

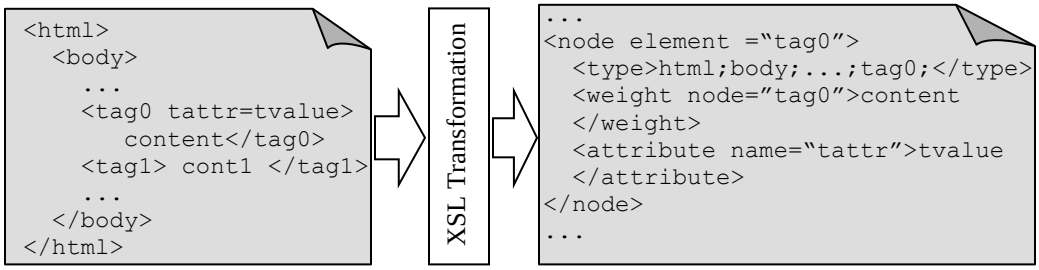

Figure 2. Example Tree-node output after XSL parsing

Next, we follow the same terminology as in [10] to represent the extracted information:

- $\boldsymbol{T}_i$ is a subtree in the document tree **T**, and encloses all nodes between the open and closed tags of r_i. Since we will be dealing with two trees $\mathbf{T}^{\mathbf{1}}$ and $\mathbf{T}^{\mathbf{2}}$ representing two Web pages, their respective subtrees are denoted by $\boldsymbol{T}_i^1$, i=1,…,I and $\boldsymbol{T}_j^2$, j=1,…, J. Hence, I and J are the number of subtrees in $\mathbf{T}^{\mathbf{1}}$ and $\mathbf{T}^{\mathbf{2}}$ respectively.
- *Nlvl*(r_i) is called *Node Level* and is the number of ancestors that element r_i has from the root node <HTML> to itself (inclusive).
- *Slvl* is called *Subtree Level* and is the level according to which nodes are grouped together. It signifies the common node level *Nlvl* among all nodes in the same subtree. For example, *Slvl* =3 divides the HTML page to sub-parts directly below the <BODY> tag as shown in Figure 3.

The subtree level *Slvl* is set to 3 as to allow for grouping the HTML nodes as subtrees directly below the BODY tag. By defining the subtrees at level 3, it is ensured that all nodes in

the page are accounted for. This however does not imply that the granularity of subtrees is limited to that of the subtrees strictly at level 3, as it also concerns subtrees at lower levels. We note that an *Slvl* greater than 3 will neglect relevant nodes, which may lead to undetected changes, while an *Slvl* smaller than 3 will model the whole page as one big subtree that starts at the BODY tag. Moreover, the tags that come before the BODY tag usually refer to HTML meta-tags or scripting code (e.g., JavaScript). In this regard, it is worth mentioning that the original approach apparently included the above described meta-tags in the comparison process.

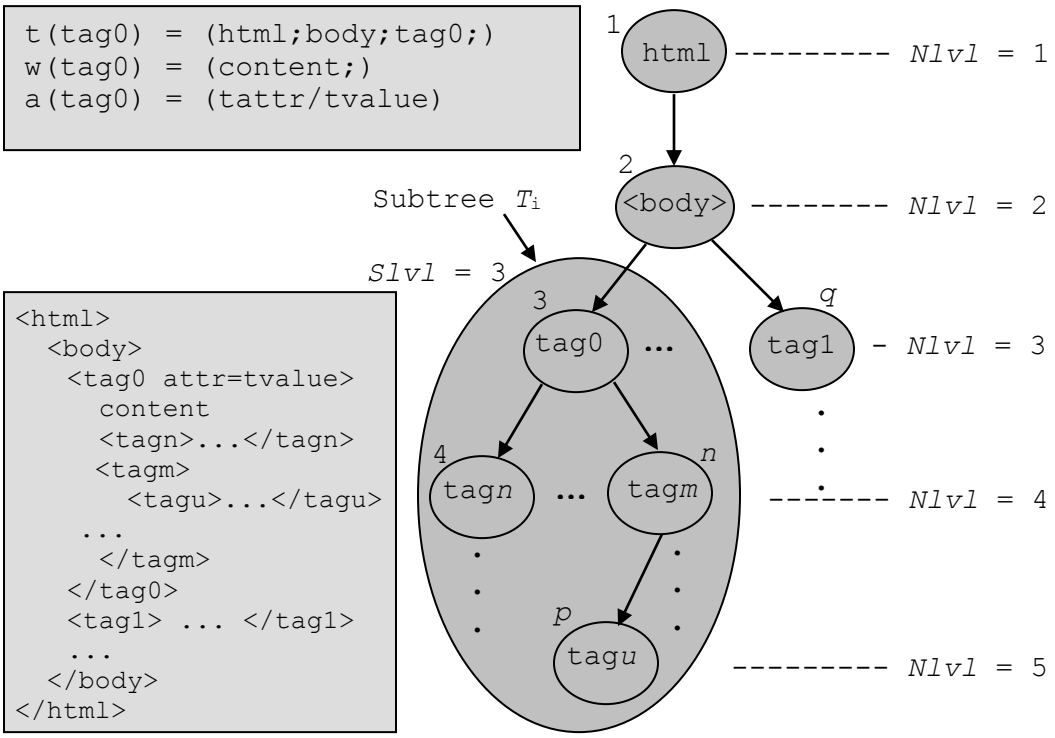

Figure 3. Illustration of nodes and subtrees: numbers next to nodes are node IDs

3.3 Framework Steps

The basic task of the system is to detect changes that occurred to a Web page relating to *content*, *layout* (*structure*), and *attributes*. Given two pages P_1 and P_2 (updated version of P_1), the detection process comprises several steps that involve P_2, while assuming that P_1 was previously parsed and stored. P_2 is first cleaned from possible HTML anomalies and then

transformed into a tree, in which each node represents a tag of the Web page and holds information and properties about it. In order to use P_2 for future comparisons, the created XML file by the XSL transformation is stored on disk and then parsed into a hash table when needed (described in subsection 3.3.2). The subtrees within the tree of P_2 are next scanned for node types (referred to as *marks*) that will form the basis of the performance improvement made to the web change detection problem. The similarity coefficients of the compatible nodes are then calculated for each subtree in P_1. Out of the individual node similarities, a similarity for each subtree of P_1 is computed relative to all the subtrees of P_2. With this data, the most similar subtree in P_2 is identified for each subtree in P_1. A change would then have occurred if the similarity coefficients of the most similar subtrees are different than 1.

In the following, each step is described in more details in a separate subsection.

3.3.1 Web page cleaning

Web page cleaning is implemented using HTML *Tidy* [12], which has many features, including detecting and correcting missing or mismatched end tags plus correcting end tags that are out of order. At the end of this phase, the processed HTML file is saved as an XML structure. It should be noted that although the output of Tidy is an XML structure, the markup remains HTML, and thus the same issues that were discussed in Section 2 still hold.

3.3.2 Page hashing and subtree generation

To improve performance of the detection process, the nodes of the entire web page to be compared are hashed into a table. This table is extensively used during the $O(N^2)$ subtrees comparison process in which the nodes of each subtree in the updated page are compared to nodes in all the subtrees of the reference page that have similar marks. The hash table for the reference web page is saved on disk after its creation along with the subtree table that is

described below. It is later read into memory when a new version of the web page is downloaded for the purpose of detecting changes relative to the reference page. The hash table is created by reading into a tree-like list (using the Java `NodeList` class) the XML file obtained by XSL transformation. The hash table is populated by fetching nodes from the list and examining them for the following attributes that become data members of the hash table: `node_id`, `node` (node's HTML tag name), `type` (path from root to the node), `att_name`, `att_value`, and `weight` (defined above). The `node_id` is the key that maps to a position in the hash table using one to one mapping. Table 1 presents an example of a part of a hash table.

row	node_id	node	type	att_name	att_value	weight
24	25	Div	html;body;div;	Id	logo	NULL
25	26	A	html;body;div;a;	accesskey	1	NULL
26	27	Img	html;body;div;a;img;	Alt	NULL	NULL
27	28	Div	html;body;div;	Id	Date	Jan 1, 2006

Table 1. A part of an example hash table

The generation of the subtrees was done while hashing the page by checking the depth of each node: if it is 3 then a new subtree begins with the current node being its starting node, and the preceding node being the last node of the previous subtree (the last node of the page is the last node of the last subtree). Simultaneously, each subtree is given its corresponding mark, which is the mark of the first node of the subtree (level 3). The generated subtrees along with their marks are stored in a specifically designed table that comprises the columns `Mark`, `Start_id`, and `End_id`. As an example, node 25 in Table 1 (`div` node) has level 3 and encloses a subtree of all elements beneath it until the next node with level 3 is encountered, which happens to be node 71 (`p` node, as shown in Table 2).

Mark	Start_id	End_id
Div	25	70
P	71	71
div	72	120

div	121	186
P	187	187
P	188	223
P	224	224

Table 2. Example of a subtree table

3.3.3 Subtree Comparison and Mapping

To further improve performance, a look up table (LUT), which contains references to the subtrees of the second page, was employed. It is made of an array of lists (illustrated in the example below of Figure 4), whereby each array slot corresponds to a mark (`p`, `img`, `div`, `a`, `ol`, `ul`, …), while each list contains references to the subtrees in the second page having the same mark. Actually, these references are row positions in the subtree table. For instance and as shown in Figure 4, the `div` tag corresponds to rows 0, 2, and 3 in Table 2. This arrangement minimizes comparison time, since searching for the subtrees of a specific mark can be done in $O(1)$ instead of $O(n)$ sequential search through the whole subtree table.

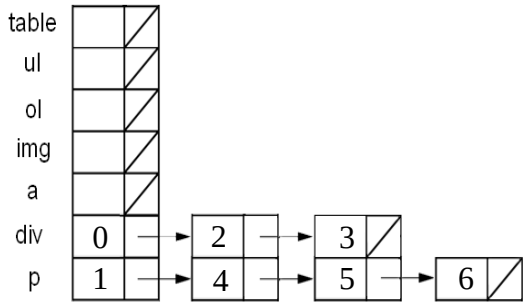

Figure 4. Look up Table (LUT) example

To store comparison results for use later in the node mapping operation, a temporary table is created that includes seven columns (`node_id1`, `node_id2`, `attribute`, `type`, `weight_position`, `weight_total`, and `CS`). The columns `weight_position` and `weight_total` correspond respectively to the numerator and denominator used in (1) to figure out the content similarity between two subtrees, while `attribute` corresponds to the

attdist computed in (2), and `type` is *typedist* calculated in (3) . The algorithm iterates over the subtree table of the first (reference) page, and compares the current subtree with all the subtrees having the same mark in the second page (with the aid of the LUT of the second page). Actually, this involves comparing the nodes in the current subtree with those in the subtrees with the same mark, by making use of the hash table to directly access the data associated with the desired nodes. It should be noted that memory is allocated only for comparing one subtree, which involves computing the number of rows in the temporary table, as follows. Having the mark of the subtree of the first page, the LUT gives the references to all the matching rows in the subtree table (i.e., pointers to the subtrees of the same mark in the second page). For each one of those subtrees, the number of nodes is calculated (end2-start2+1: retrieved from the subtree table) and next multiplied by the number of nodes in the considered subtree of the first page. The sum of all the multiplications (i.e., for all matching subtrees of the second page) is then the total number of rows for which memory is allocated.

Node mapping is then applied to find the most similar subtree from the second page to the considered subtree of the first page. This continues for the consequent subtrees, where memory is reclaimed and then allocated.

3.3.4 Subtree similarity Computations

To describe the computations, we first define variables that represent elements of the subtree:

- **m** denotes the set of element types (marks) that are detected in all subtrees. For HTML, usually this set mostly includes `<TABLE>`, `<IMG>`, `<LIST>`, `<A>`, and `<P>`. We consider that **m** consists of *K* possible elements (i.e., tags that are found in **T**). Moreover, we define $\boldsymbol{m}_i$ as the set of subtree marks that denotes the group of element (HTML tag) types contained in subtree $\boldsymbol{T}_i$ (i.e., $\boldsymbol{m}_i \subseteq \mathbf{m}$). For example, if subtree $\boldsymbol{T}_i$ contains an image and an unordered list,

then $\boldsymbol{m}_i$ ={UL, IMG}. Also, $m_{i_k} \in \boldsymbol{m}_i$ denotes a mark found in subtree $\boldsymbol{T}_i$, with $k \in \{1, 2, \ldots, K\}$ being the kth index in the set **m**.

- If $\{r_1, r_2, \ldots, r_i\}$ is the path from the root node r_1 to the node r_i then, $\boldsymbol{type}(r_i)$ is the concatenation of corresponding HTML elements starting from the root of the tree and ending at r_i. Here, a node refers to an open tag plus its corresponding closed tag.
- $r^1_{i_p,m_k}$ is the pth element of subtree $\boldsymbol{T}^1_i$ having mark $m_k \in \mathbf{m}$, and $r^2_{j_q,m_k}$ is the qth element of subtree $\boldsymbol{T}^2_j$ having the same mark m_k.
- $\boldsymbol{w}(r_i)$ is the set of words in the text associated with the leaves of subtree rooted at node r_i. In $\boldsymbol{w}(r^1_{i,m_k})$, the superscript denotes the page index (1 for stored page and 2 for new page), while m_k denotes a subtree mark corresponding to the kth HTML tag in the set of possible tags.
- $\boldsymbol{a}(r_i)$ is the set of attributes associated with r_i. Hence, in $\boldsymbol{a}(r^1_{i,m_k})$, the superscript and subscripts have the same meanings as those in $\boldsymbol{w}$.

The word *content* similarity between two nodes is expressed using the definition of the function *Intersect*, which returns the percentage of words appearing in both $\boldsymbol{w}(r^1_{i,m_k})$ and $\boldsymbol{w}(r^2_{j,m_k})$:

$$Intersect(\boldsymbol{w}(r^1_{i,m_k}), \boldsymbol{w}(r^2_{j,m_k})) = \frac{|\boldsymbol{w}(r^1_{i,m_k}) \cap \boldsymbol{w}(r^2_{j,m_k})|}{|\boldsymbol{w}(r^1_{i,m_k}) \cup \boldsymbol{w}(r^2_{j,m_k})|} \quad (1)$$

For computing the similarity between node *attributes*, the function *Attdist* is used:

$$Attdist(\boldsymbol{a}(r^1_{i,m_k}), \boldsymbol{a}(r^2_{j,m_k})) = \frac{\sum_{a_l \in \{\boldsymbol{a}(r^1_{i,m_k}) \cap \boldsymbol{a}(r^2_{j,m_k})\}} Weight(a_l)}{\sum_{a_l \in \{\boldsymbol{a}(r^1_{i,m_k}) \cup \boldsymbol{a}(r^2_{j,m_k})\}} Weight(a_l)} \quad (2)$$

The above function gives a measure of the relative weight of the attributes that have the same value in r^1_{i,m_k} and r^2_{j,m_k} with respect to all their attributes. As indicated in Equation 2, specifically

the function *Weight*(a_l), the attributes are weighted differently according to their relevance of use. For example, `HREF` is considered more relevant than the formatting attribute `rules`. Attribute weights vary between 0, least important, and 100, most important. Some of the attributes weights were assigned according to their frequency of use, as specified in [30], and stored in an XML file, which is read by the application when it is first started (independent of the web comparison process) into a look up table that holds the attributes as "strings" in one column and weights as "integers" in the other column.

To compute the similarity between the *paths* from the root nodes of $\boldsymbol{T}_1$ and $\boldsymbol{T}_2$ to the considered nodes r^1_{i,m_k} and r^2_{j,m_k} respectively, the function "*Typedist*" is used:

$$Typedist(\boldsymbol{type}(r^1_{i,m_k}), \boldsymbol{type}(r^2_{j,m_k})) = \frac{\prod_{i=0}^{suf}(2^{\max - i})}{\prod_{i=0}^{\max}(2^i)} \tag{3}$$

where *suf* and *max* represent, respectively, the length of the common suffix (number of common nodes from root to the two concerned nodes, i.e., number of common HTML tags) and the maximum cardinality (number of HTML tags of the longest path) between $\boldsymbol{type}(r^1_{i,m_k})$ and $\boldsymbol{type}(r^2_{j,m_k})$.

Now, having the characteristics for the two nodes r^1_{i,m_k} and r^2_{j,m_k}, the similarity between them, given they have the same subtree mark m_k, is computed as follows:

$$\begin{aligned} CS(r^1_{i,m_k}, r^2_{j,m_k}) = -1+2\times(&\alpha\times Typedist(\boldsymbol{type}(r^1_{i,m_k}), \boldsymbol{type}(r^2_{j,m_k}))+ \\ &\beta\times Attdist(\boldsymbol{a}(r^1_{i,m_k}),\boldsymbol{a}(r^2_{j,m_k}))+\gamma\times Intersect(\boldsymbol{w}(r^1_{i,m_k}),\boldsymbol{w}(r^2_{j,m_k}))) \end{aligned} \tag{4}$$

where α, β, and γ are weights such that $\alpha + \beta + \gamma = 1$. Moreover, the values of α, β and γ are selected on the basis of emphasizing certain types of changes and for normalizing the output result of *CS*. It is obvious from (4) that the returned similarity varies between (-1, 1], where -1 corresponds to maximum difference and 1 to maximum similarity.

3.3.5 Determining Subtree similarities

After computing the node *CS* values, the next step is to calculate similarity coefficients between subtrees. Given two subtrees $\boldsymbol{T}_i^1$ and $\boldsymbol{T}_j^2$ belonging to trees $\mathbf{T}^1$ and $\mathbf{T}^2$, a mapping $M(\boldsymbol{T}_i^1, \boldsymbol{T}_j^2)$ from $\boldsymbol{T}_i^1$ to $\boldsymbol{T}_j^2$ with $\boldsymbol{S}_i^1$ and $\boldsymbol{S}_j^2$ as their respective sets of nodes, and r_{i_p,m_k}^1 and r_{j_q,m_k}^2 being respectively the *p*th and *q*th nodes belonging to $\boldsymbol{T}_i^1$ and $\boldsymbol{T}_j^2$ and having the same mark m_k, the *Subtree Source Node Similarity* is given by:

$$Sim_{M(\boldsymbol{T}_i^1,\boldsymbol{T}_j^2)}(r_{i_p,m_k}^1) = max(CS(r_{i_p,m_k}^1, r_{j_q,m_k}^2)) \ \forall \ r_{j_q,m_k}^2 \in \boldsymbol{T}_j^2 \tag{5}$$

That is, for each node in subtree $\boldsymbol{T}_i^1$ the above finds the most similar node in $\boldsymbol{T}_j^2$.

Next, the similarity of two subtrees $\boldsymbol{T}_i^1$ and $\boldsymbol{T}_j^2$ having nodes of the same mark, $r_{i,m_k}^1 \in \boldsymbol{T}_i^1$ and $r_{j,m_k}^2 \in \boldsymbol{T}_j^2$, is defined as follows:

$$Sim_M(\boldsymbol{T}_i^1, \boldsymbol{T}_j^2) = \frac{\sum_{p=1}^{P_i} Sim_{M(\boldsymbol{T}_i^1,\boldsymbol{T}_j^2)}(r_{i_p,m_k}^1)}{P_i} \tag{6}$$

The above states that the similarity of each pair of subtrees belonging to $\mathbf{T}^1$ and $\mathbf{T}^2$ is the average similarity taken across all nodes of $\boldsymbol{T}_i^1$ (P_i is the number of nodes in $\boldsymbol{T}_i^1$).

Finally, the *Document Subtree Similarity* is the maximal subtree similarity between $\boldsymbol{T}_i^1$ and $\forall \ \boldsymbol{T}_j^2 \in \mathbf{T^2}$, as defined in (7) below. The value of $Sim(\boldsymbol{T}_i^1, \boldsymbol{T}_j^2)$ indicates whether a change took place inside the subtree or not: a value of 1 implies no change has occurred in this subtree, while a value less than 1, means a change has taken place. Usually, one is not directly interested in the node where the change took place, but rather in the subtree where the changed node resides.

$$Sim(\boldsymbol{T}_i^1, \boldsymbol{T}_j^2) = max\,(Sim_M(\boldsymbol{T}_i^1, \boldsymbol{T}_j^2)) \ \forall \ \boldsymbol{T}_j^2 \in \mathbf{T_2} \tag{7}$$

4. Performance Results

To produce relative performance results, both the enhanced and original approach were implemented. Tests were performed on a 1.8 GHz Pentium M NEC laptop, model i-Select M5410, running Windows XP SP2, with 1 GB RAM, and 1MB cache. For implementation, Java was used to program the functionality of both approaches, Tidy [12] for cleaning HTML web pages and tags, and Oracle Parser for parsing and building the XML trees.

4.1 Results Validation

Before proceeding with illustrating the performance results, we describe a procedure we used to validate the results. We focused on a highly dynamic page (home page of CNN.com) and downloaded from the way-back machine a total of 520 pages covering the whole year of 2006, and therefore had an average of one to two pages per day. For this experiment, we conducted three trials, each of which involved choosing four pages randomly from the pool of 520 pages and comparing the next 20 pages that follow each one of those four pages to it using a balanced configuration (4,3,3). The obtained results showed major differences in the similarity coefficients at the start, but became nearly constant at the end. The results are presented in Figure 5, where the y-axis represents the fraction of the page that has been changed, which was computed by differentiating the similarity coefficients (note that it uses a log-scale). The graph reveals a linear trend, which indicates an exponential decay and corresponds to a Poisson distribution that resembles the output reported in [6]. In this regard, it should be mentioned that the web change frequency has been studied in the literature and modeled as a Poisson distribution [21] [2] [6].

The reason for choosing 20 pages for this experiment is because they correspond to 10 days of monitoring, after which the web page contains content that is mostly new, as evidenced by the fact that the similarity coefficients tended to an asymptote toward the end of the comparison.

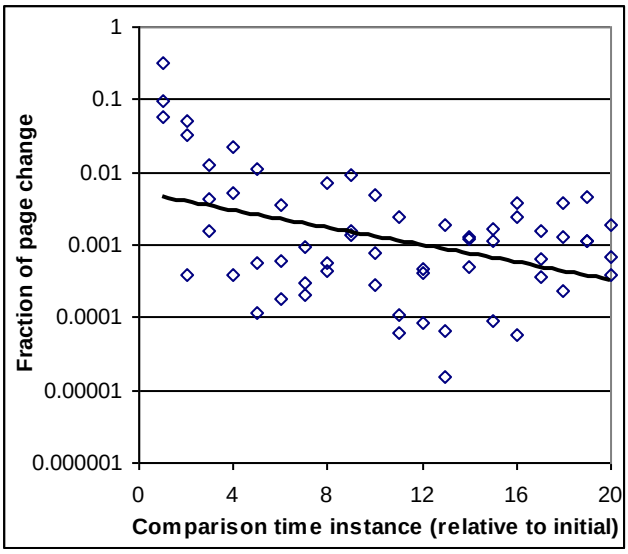

Figure 5. Web page change fraction versus time

The resemblance that exists between the results shown in Figure 5 and those reported in [6] (which is in turn referenced in many other works) may be used to make inferences about the reliability of the output that our approach produces.

4.2 Presentation of Results

For the performance tests, over 250 web pages were downloaded from the Internet in order to run tests for assessing the relative performance of the proposed approach. Table 3 illustrates the types of these pages along with statistics that describe their content and size. From this collection, 26 representative Web pages were chosen.

Type	# of pages	Content range (number of items)						Size range (# of nodes)
		Links	Figures	Tables	Lists	Text	Scripts	
News sites	56	15 - 40	15 - 40	5 - 10	5 - 10	60 - 1000	15 - 40	60 – 950
Personal sites	12	5 - 10	3 - 5	5 – 10	15 - 40	60 – 1000	0 - 3	10 – 670
Academic sites	42	10 - 15	5 - 10	15 - 40	15 - 40	40 – 60	2 - 15	30 – 440
Commercial sites	104	10 - 15	5 - 10	15 - 40	15 - 40	40 – 60	10 - 15	20 - 1060
Online documentation	20	10 - 15	15 - 40	15 - 40	10 - 15	60 - 1000	0 - 10	240 - 2920
Wikipedia sites	20	> 600	10 – 50	10 - 30	40 - 200	50 - 300	5 - 15	1630 - 3010

Table 3 Types and distribution of Web pages that were used in the study

The experiments focused on the performance of the approach in terms of the number of node similarity computations and the time consumed to completely produce and store the similarity

coefficients. For every Web page tested, we noted the number of node comparisons for both the enhanced and original algorithms. Figure 6 shows the number of saved similarity computations plotted on a logarithmic scale to illustrate the improvements for small and large Web pages. One can notice the savings depicted as percentages as well, which translate to appreciable time savings, especially for large web pages. The data in the figure reveals that our approach is faster by at least 30% for pages having more than 500 nodes, while for less than 500 nodes, we still observe savings that can add up to significant times when many pages are processed.

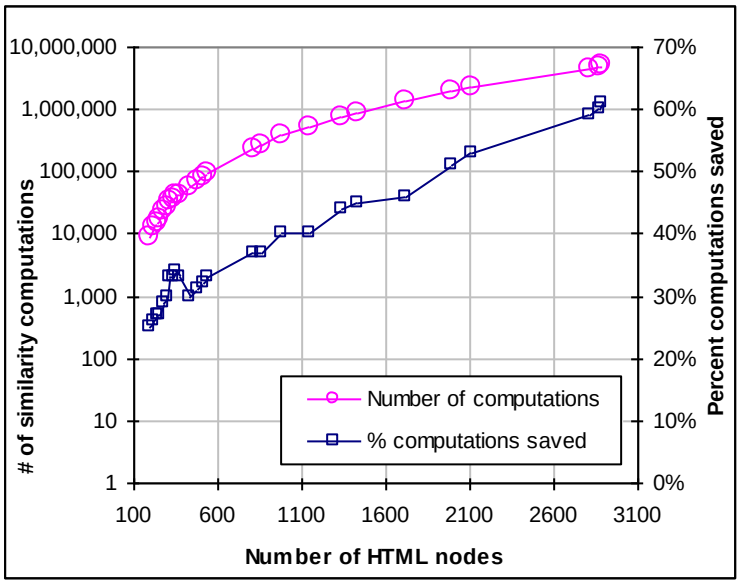

Figure 6. Saved node similarity computations

Figure 7 shows the average change detection time measured after running the enhanced algorithm 10 times on each pair (modified and original) of the 26 Web pages. This time includes reading the updated web page, building the hash and subtree tables, doing the comparisons and computing the similarity coefficients, and inferring the updated subtree mapping. The results in Figure 7 illustrate that pages with less than 500 nodes (40KB on average) can be processed and compared in less than 2 seconds while those that have about 1000 nodes can be compared in about 8 seconds. The figure also shows that the detection process is mostly consumed by subtree

node comparisons. In fact, the total time it takes to clean and parse the HTML document into an XML file, plus building the hash table and identifying the subtrees along with building the subtree table, is close to one tenth of the time consumed by comparing the nodes and computing the similarity coefficients. This demonstrates the key role of the hashing mechanisms that was integrated for providing direct in-memory access to node information during the comparisons.

Related to processing time, the work in [17] describes an analysis of over 21,000 web pages that were surveyed using three methods of seed generation via search engines. The first was the Yahoo random page CGI, which redirects to a random URL, the second used the Open Directory Project (ODP) database [23] to randomly select seed documents, while the third utilized two random English words as a query to the Google search engine and then used the top ten documents as seeds for the crawl. The average size of HTML web page was found to be 281 tags (nodes). For pages of this size, our proposed HTML web change detection approach can complete the detection process in less than one second (0.87 second on average).

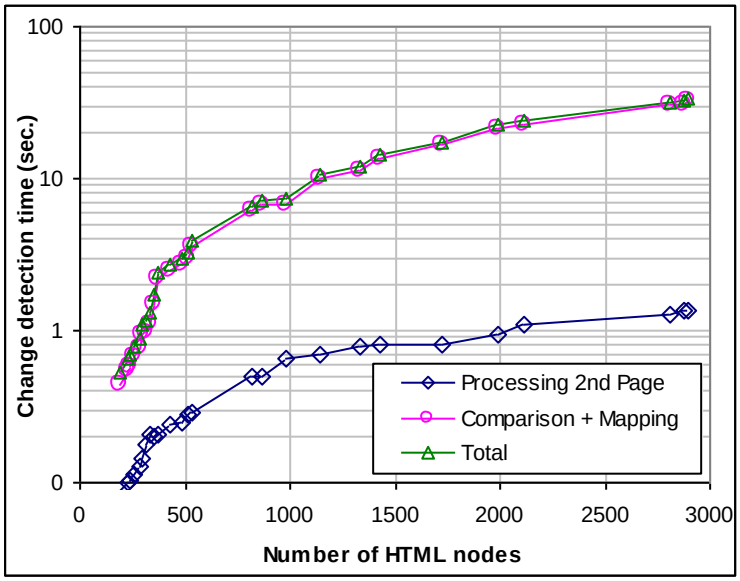

Figure 7. Change detection times for original and enhanced approaches

Next, the detection level is studied by applying different values of α, β, and γ while processing the web pages used for testing. For this, different versions of those pages were downloaded from the *way back* machine, and then each parameter, e.g., γ, was varied between 0 and 1 in increments of 0.2, while each of the other two parameters, e.g., α and β, was set to half of the remainder, e.g., $\alpha = \beta = (1-\gamma)/2$. For each combination of α, β, and γ, the different versions of each web page were processed, and the measured similarity coefficients for each page when compared to the previous version were stored on disk, as was mentioned earlier. Each coefficient's value that is not equal to one was considered a change. Figure 8 plots the detection reliability, which we define as the ratio of the number of detected changes to that of the actual ones. The figure illustrates the computed reliability when considering each type separately (i.e., content, attribute, or style), and when considering the overall changes across all types. The former case was meant to study the effect of the α, β, and γ emphasis parameters on the changes they relate to (i.e., style, attribute, and content, respectively). The "Overall" curve corresponds to the weighted average, which is why it drops dramatically when any of the parameters' values is one (implying that the other two parameters were set to zero), or is zero (in which case the values of the other two parameters sum up to one). For example, the left-most part of the graph illustrates the case where *one* of the parameters (α, β, or γ) is set to zero, whereas each of the other two is set to 0.5. In this situation, the system is able to detect changes associated with the two non-zero valued parameters, thus giving a detection reliability close to two thirds. The right-most part, on the other hand, depicts the case where *two* of the parameters are set to zero, whereas the remaining one is set to 1. Here, the system is able to detect changes associated with the one non-zero valued parameter, thus giving a detection reliability very close to one third.

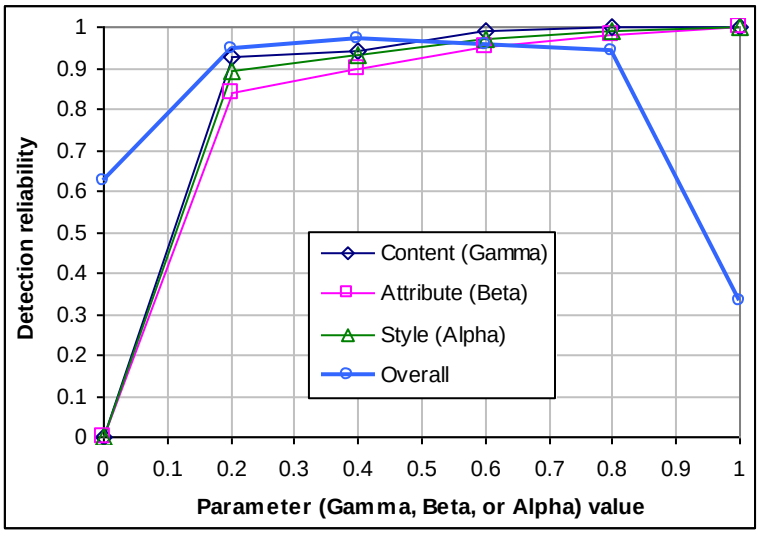

Figure 8. Detection level relative to maximum number of detections

Several remarks can be made about the graph in Figure 8. The update detection is generally independent from the emphasis parameters when their values are not equal or close to zero. This comes from the fact that in our implementation there is no differentiation on the basis of how significant the change is. That is, if the subtree similarity coefficient value is different than 1, even by a small amount, a change is declared. A value near zero for one of the three parameters will eliminate the corresponding coefficient's participation in the overall similarity calculation and will therefore weaken the detection of a change if it is of the corresponding type. The above suggests that the parameters α, β, and γ can take on Boolean values without affecting the functionality of the system. However, the coefficient of similarity (CS) expression in Equation 4 requires numeric values for these parameters and that their sum is equal to 1. Obviously, these parameters could be of more use if changes are to be classified according to their severity, in which case the estimated severity of the change could be directly related to the distance of the similarity coefficient from 1. Then one can visualize the magnitude of the changes right on the updated page using, for example, color coding or some other mechanism.

4.3 RSS Feeds Change Detection

The proposed change detection algorithm has also been applied to RSS feeds that may be present in certain web pages. RSS (Really Simple Syndication) is a format for delivering regularly changing web content, such as news. The most common standard for this format is RSS 2.0, which is used by more than 80% of the web sites [26] that publish dynamic content, including Microsoft, CNN, Google News, Yahoo News, and AFP. RSS content is defined in an XML file, consisting of one channel per feed that includes many items which represent stories or events, and each item comprises attributes that include title, description, publication date, and so on [25]. Many websites, however, ignore other attributes that are optional, although important (like the `guid`), and only supply `title` and `description`. As a consequence, we only rely on the latter two attributes when applying the detection algorithm, which makes the mapping process between feeds reliable, yet more complex.

To include RSS feeds in the comparison process of two web pages, we need to define criteria for detecting changes. In [1], two criteria to monitor news are defined, and since RSS mainly represent news and events, we will adopt the same criteria (termed event detection and event tracking) and apply our algorithm so that each RSS feed becomes a subtree represented by an RSS tag (e.g., RSS news or RSS sports) so as to avoid comparing non-compatible RSS feeds to each other. Each RSS tag encompasses a subtree consisting of the items mentioned above. The description of the item is analogous to the *weight* definition mentioned before while the title is analogous to the *attribute*. We note that *type*, which represents the Path from the root element, has no relevance in comparing RSS feeds. Finally and similar to alpha and beta, we empirically assigned 30% of the comparison weight to the title and 70% to the description.

Now concerning the detection process, when an RSS tag is encountered when a page is being retrieved, the XML file is processed and then hashed in a manner similar to the web page. All

items are then compared to each other using the similarity functions employed for the main page (i.e., (1) and (2)), and the similarity coefficients are stored in a special array. Then a mapping is applied to define the nearest item in the second file to each item in the first file, which is done by finding the item of the next page which corresponds to the highest similarity coefficient.

To classify the type of change, the similarity coefficient of the mapping between two items will be used as indication that the item was modified if it is positive (less than 1), and that the item of the first page was removed if the coefficient is negative. As for added items, every mapped item of the second page is traversed to see if it is not mapped to any item in the first page (does not exist), or if it is only mapped to item(s) in the first page with negative similarity coefficient(s). In such cases, it will be concluded that the item was added.

Table 4 illustrates an example of a comparison between two RSS feeds retrieved from yahoo.com on Sep 3, 2007 at 11:30 GMT and then at 15:46 GMT**.** From the perspective of the first page (P1), we observe that items 8, 13, 15, 16, and 18 were not modified, items 1, 2, 3, 14, and 20 were modified, while items 4, 5, 6, 7, 9, 10, 11, 12, 17, and 19 were removed. Now, from the perspective of the updated page (P2), we deduce that items 1, 5, 7, 9, 10, 11, 18, and 19 were added. We should stress while examining the coefficient values that Table 4 only shows the titles that only make up 30% of the weight, and does not reveal the descriptions that account for 70%.

Item P1	Item P2	Similarity Coefficient	Item Title 1	Item Title 2
1	2	0.227523	Felix becomes Category 5 hurricane	Felix becomes Category 5 hurricane
2	3	0.143421	Power outage from California heat wave	Calif. heat leaves 14,000 without power
3	4	0.52088	Britain pulls out from Basra base	Britain pulls out of downtown Basra base
4	10	-0.11353	Lines at United States borders longer	Bush makes war assessment in Iraq
5	5	-0.61324	Girl, 13, found dead in Ariz. mine shaft	Iranian-American scholar leaves Iran
6	6	-0.47722	Lebanese army hunts for fugitives	Rocket lands by Israeli day care center
7	12	-0.53326	Scientists test new bipolar remedies	Women may need different heart treatment
8	8	1	Report: U.S. workers are most productive	Report: U.S. workers are most productive
9	5	-0.6372	Movie studios bask in blockbuster summer	Director says Owen Wilson's doing better
10	10	-0.49718	Serena, Henin to square off at U.S. Open	Federer, Roddick highlight Open today
11	4	-0.46826	British troops quit Iraqi city of Basra	Bush holds “war council” with top aides in Iraq
12	12	-0.08292	Felix becomes rare top-ranked storm	Hurricane Felix threatens Central America
13	13	1	N.Korea says U.S. to remove it from terrorism list	N.Korea says U.S. to remove it from terrorism list
14	14	0.785882	APEC set for world trade, climate change talks	APEC set for world trade, climate change talks
15	15	1	Afghan Taliban vow to kidnap, kill more foreigners	Afghan Taliban vow to kidnap, kill more foreigners
16	16	1	Russian strategic bombers run Arctic exercise	Russian strategic bombers run Arctic exercise
17	15	-0.58235	China vows to clean up toxins amid food scares	Edwards gets fresh union backing for White House bid
18	17	1	Study finds smokers have higher risk of dementia	Study finds smokers have higher risk of dementia
19	5	-0.60234	Former Bangladesh PM Zia arrested	UN chief in Sudan to push for Darfur peace
20	20	0.43125	Maximum strength Hurricane Felix aims for Central America	Maximum strength Hurricane Felix aims for Central America

Table 4. An example of mappings between news feeds (based on title and description)

6. Discussion and Conclusion

This paper described an improved Web change detection approach based on restricting the similarity computations to subtree nodes having the same HTML tag, and hashing the web page for direct in-memory access to node information. A practical server application was developed to allow for scheduling web page monitoring jobs and producing reports and graphs against stored similarity data that describe processed comparisons between a page and its previous version(s). Performance measurements using a group of web pages selected from a pool of over 200 pages showed that the enhanced algorithm can perform change detection in the order of seconds for small and medium-sized web pages, and in the order of few tens of seconds for large web pages.

Concerning the limitation that was mentioned in Subsection 3.1, there are three situations in which changes using our approach will not be detected:

1. A new tag is *added* directly below the <BODY> tag (at level 3) in the new page, given that there is no tag of the same type at the same level in the old page.
2. A tag at level 3 in the new page is *changed* to a tag of type *k*, given that there is no tag of type *k* at level 3 in the old page.
3. A tag of type *j* at level 3 in the new page is *deleted*, given that it is the only tag of that type at that level in the old page, and that no other tag of type *j* now exists in the new page at level 3.

Other than the above situations, the algorithm will always detect the changes. The important question, however, is how probable are the above situations? To answer this question, we refer to the study that was reported in [9], which analyzed the changes made to web pages by crawling over 150 million HTML pages once a week, over a span of 11 weeks. Of concern to our work are the types of changes made to the HTML markup. Out of the almost 1.5 million changes in the markup that were detected (1% of the total sample size), 61% were for attribute changes, 32% for adding or deleting attributes, and 6% for adding or deleting tags. The tags that were affected (out of the 6%) were the <A> tag (48%), <IMG> tag (10%), comments (23%), and the rest were distributed among non-popular tags (<META>, <PARAM>, <INPUT>, etc.). The study did not provide information concerning the position of the changed tags within the web pages, but according to our relatively small set of web pages that we used for performance testing, we noticed that the probability of the <A> tag falling directly below the <BODY> tag is less than 5% while that of the <IMG> tag is less than 15%. From the above data, it is very easy to see that the probability of the three situations occurring in practice is very low (about 38 in one million).

For future works, the algorithm can be parallelized through multithreading, which can increase performance significantly. In particular, since each subtree of the original page (left subtree) is compared to all similar subtrees of the updated page (right subtrees), one thread could compare one or more left subtrees to all right subtrees and identify the most right similar subtree. Given that the comparisons (for computing the similarity coefficients) are by far the most time-consuming tasks of the algorithm, multithreading could potentially divide the running time of the algorithm by *M*, where *M* is the number of threads. 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. Moreover, and since in this work we have applied our algorithm to RSS feed changes, it should also be straightforward to handle online blogs in a similar fashion knowing they are defined in XML files just like RSS contents are.

References

[1] J. Allan, R. Papka, and V. Lavrenko, On-line new event detection and tracking. In *Proceedings of the 21st international ACM SIGIR conference on research and development in information retrieval*, August 1998, pp. 37–45.

[2] B. Brewington and G. Cybenko, How dynamic is the web? In *Proceedings of WWW2000*, March 2000.

[3] S. Chakravarthy, J. Jacob, N. Pandrangi, and A. Sanka, Webvigil: An approach to just-in-time information propagation in large network-centric environments, *2nd International Workshop on Web Dynamics*, Honolulu, Hawaii, 2002.

[4] S. Chawathe, J. Widom, A. Rajaraman, and H. Garcia-Molina, Change Detection in Hierarchically Structured Information**,** *ACM SIGMOD international conference on Management of data*, Montreal, Canada, 1996, pp. 493-504.

[5] S. Chawathe and H. Garcia-Molina, Meaningful Change Detection in Structured Data, *ACM SIGMOD international conference on Management of data*, v. 26, n. 2, 1997, pp. 26-37.

[6] J. Cho and H. Garcia-Molina, Synchronizing a database to improve freshness, *SIGMOD Record*, vol. 29, no. 2, pp. 117-128, June 2000.

[7] G. Cobena, S. Abiteboul, and A. Marian, Detecting changes in XML documents, *18th International Conference on Data Engineering*, San Jose, CA, 2002, pp. 41-52.

[8] F. Douglis, T. Ball, Y. Chen, and E. Koutsofios, The AT&T Internet difference engine: tracking on the Web, *World Wide Web*, v. 1, 1998, pp. 27–44.

[9] D. Fetterly, M. Manasse, M. Najork, A large-scale study of the evolution of Web pages, *Journal of Software - Practice and Experience*, v 34, n 2, 2004, pp. 213-237.
[10] S. Flesca and E. Masciari, Efficient and Effective Web Change Detection, *Data and Knowledge Engineering*, v. 46, n. 2, 2003, pp. 203-224.
[11] HTMLDiff, available from http://www.componentsoftware.com/Products/HTMLDiff/index.htm.
[12] HTML Tidy, available from http://www.w3.org/People/Raggett/tidy/tidy.html.
[13] J. Hunt and M. Mcllroy, An Algorithm for Differential File Comparison, *Technical Report*, TR #41, Bell Laboratories, Murray Hill, NJ, 1975.
[14] J. Jacob, A. Sache, and S. Chakravarthy, CX-DIFF: a change detection algorithm for XML content and change visualization for WebVigiL, *Data and Knowledge Engineering*, v. 52, n. 2, 2005, pp. 209-230.
[15] Z. Kaizhong, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems, *6th Annual Symposium on Combinatorial Pattern Matching*, Espoo, Finland, 1995, pp. 395–407.
[16] H. Kuhn, The Hungarian method for the assignment problem, *Naval Research Logistics*, v. 2, n. 1, 2005, pp. 7–21.
[17] R. Levering and M. Cutler, The portrait of a common HTML web page, In *Proceedings of the 2006 ACM symposium on Document engineering*, 2006, pp. 198 – 204.
[18] S-J. Lim, and Y-K. Ng, An automated change-detection algorithm for HTML documents based on semantic hierarchies, *17th International Conference on Data Engineering*, Heidelberg, Germany, 2001, pp. 303-312.
[19] L. Ling, T. Wei, D. Buttler, and C. Pu, Information monitoring on the Web: a scalable solution, *World Wide Web*, v 5, n 4, 2002, pp. 263-304.
[20] L. Liu, C. Pu, and W. Tang, WebCQ - detecting and delivering information changes on the Web, *9th International Conference on Information and Knowledge Management*, Atlanta, Georgia, 2000, pp.512-519.
[21] N. Matloff, Estimation of internet file access modification rates from indirect data, *ACM Transactions on Modeling and Computer Simulation*, vol. 15, pp. 233–253, 2005.
[22] Netmind, available from http://www.changedetect.com/cd-netmind.asp.
[23] Open Directory Project database. http://rdf.dmoz.org/.
[24] Opernic Technologies, Copernic Tracker product, available online at http://www.copernic.com/en/products/tracker/tracker-features.html.
[25] RSS 2.0 Specification, available online at http://cyber.law.harvard.edu/rss/rss.html
[26] RSS Feed statistics, available online at www.Syndic8.com/stats.php?section=feeds
[27] Y. Wang, D. DeWitt, and J. Cai, "X-Diff: An Effective Change Detection Algorithm for XML Documents" *International Conference on Data Engineering*, Bangalore, India, 2003, pp. 519-530.
[28] WebCQ product. Available: http://www.cc.gatech.edu/projects/disl/WebCQ
[29] WebSite-Watcher product. Available: http://www.aignes.com.
[30] A. Woodruff, P. Aoki, E. Brewer, P. Gauthier, and L. Rowe, An Investigation of Documents from the World Wide Web, *Computer Networks and ISDN Systems*, v. 28, n. 7, 1996, pp. 963-980.