Vi-DIFF: Understanding Web Pages Changes

Zeynep Pehlivan, Myriam Ben-Saad, and Stéphane Gançarski

LIP6, University P. and M. Curie Paris, France
{zeynep.pehlivan,myriam.ben-saad,stephane.gancarski}@lip6.fr

Abstract. Nowadays, many applications are interested in detecting and discovering changes on the web to help users to understand page updates and more generally, the web dynamics. Web archiving is one of these fields where detecting changes on web pages is important. Archiving institutes are collecting and preserving different web site versions for future generation. A major problem encountered by archiving systems is to understand what happened between two versions of web pages. In this paper, we address this requirement by proposing a new change detection approach that computes the semantic differences between two versions of HTML web pages. Our approach, called Vi-DIFF, detects changes on the visual representation of web pages. It detects two types of changes: content and structural changes. Content changes include modifications on text, hyperlinks and images. In contrast, structural changes alter the visual appearance of the page and the structure of its blocks. Our Vi-DIFF solution can serve for various applications such as crawl optimization, archive maintenance, web changes browsing, etc. Experiments on Vi-DIFF were conducted and the results are promising.

Keywords: Web dynamics, Web archiving, Change detection, Web page versions, Visual page segmentation.

1 Introduction

The World Wide Web (or web for short) is constantly evolving over time. Web contents like texts, images, etc. are updated frequently. As those updates reflect the evolution of sites, many applications, nowadays, aim at discovering and detecting changes on the web. For instance, there are many services [4] that track web pages to identify what and how a page of interests has been changed and notify concerned users. One of the recent work is proposed by Kukulenz et al. [14]. They propose two strategies: tracking where a number of segments is traced over time and pruning where less relevant segments are removed automatically. A browser plug-in [20], using the page cache, has been also proposed to explicitly highlight changes on the current page since the last visit. The goal of such applications is to help users to understand the meaning of page changes and more generally, the web dynamics. Detecting changes between page versions is also important for web archiving. As web contents provide useful knowledge, archiving the web has become crucial to prevent pages content from disappearing. Thus, many national archiving institutes [2,5,7,11] around the world are

P. García Bringas et al. (Eds.): DEXA 2010, Part I, LNCS 6261, pp. 1–15, 2010.

collecting and preserving different web sites versions. Most of the web archiving initiatives are described at [1]. Most often, web archiving is automatically performed using web crawlers. A crawler revisits periodically web pages and updates the archive with a fresh snapshot (or version). However, it is impossible to maintain a complete archive of the web, or even a part of it, containing all the versions of the pages because of web sites politeness constraints and limited allocated resources (bandwidth, space storage, etc.). Thus, archiving systems must identify accurately how the page has been updated at least for three reasons: (1) avoid archiving several times the same version, (2) model the dynamics of the web site in order to archive "the most important versions" by crawling the site at a "right moment", (3) ease temporal querying the archive.

Thus, our research problem can be stated as follows: *How to know and to understand what happened (and thus changed) between two versions of web page ?*

To address this issue, we propose, a change detection approach to compute the semantic differences between two versions of web pages. As we know, most of the pages on the web are HTML documents. As HTML is semantically poor, comparing two versions of HTML pages, based on the DOM tree does not give a relevant information to understand the changes. Thus, our idea is to detect changes on the visual representation of web pages. Indeed, the visual aspect gives a good idea of the semantic structure used in the document and the relationship between them (*e.g.* the most important information is in the center of the page). Preserving the visual aspect of web pages while detecting changes gives relevant information to understand the modifications. It simulates how a user understands the changes based on his visual perception. The concept of analyzing the visual aspect of web pages is not new. However, as far as we know, it had never been exploited to detect changes on web pages.

Our proposed approach, called Vi-DIFF, compares two web pages in three steps: (i) segmentation, (ii) change detection and (iii) delta file generation. The segmentation step consists in partitioning web pages into visual semantic blocks. Then, in the change detection step, the two restructured versions of web pages are compared and, finally, a delta file describing the visual changes is produced.

The main contributions of this paper are the following:

- A novel change detection approach (Vi-DIFF) that describes the semantic difference between "visually restructured" web pages.
- An extension of an existing visual segmentation model to build the whole visual structure of the web page.
- A change detection algorithm to compare the two restructured versions of web pages that takes into account the page structure with a reasonable complexity in time.
- An implementation of Vi-DIFF approach and some experiments to demonstrate its feasibility.

The remainder of the paper is structured as follows. Section 2 discusses some related works and presents different contexts in which our proposed approach can be exploited. Section 3 presents the VI-DIFF and the different change operations

we consider. Section 4 explains, in detail, the second step of Vi-DIFF that computes the semantic difference between two restructured versions of web pages. Section 5, presents the implementation of VI-DIFF and discusses experimental results. Section 6 concludes.

2 Context and Related Works

The first step of our approach is the segmentation of the web page. Several methods have been proposed to construct the visual representation of web pages. Most approaches discover the logical structure of a page by analyzing the rendered document or analyzing the document code. Gu et al. [12] propose a top-down algorithm which detects the web content structure based on the layout information. Kovacevic et al. [10] define heuristics to recognize common page areas (header, footer, center of the page, etc.) based on visual information. Cai et al. [6] propose the algorithm VIPS which segments the web page into multiple semantic blocks based on visual information retrieved from browser's rendering. Cosulshi et al. [9] propose an approach that calculates the block correspondence between web pages by using positional information of DOM tree's elements. Kukulenz et al.'s automatic page segmentation [14] is based on the estimation of the gramatical structure of pages automatically. Among these visual analysis methods, VIPS algorithm [6] seems to be the most appropriate for our approach because it allows an adequate granularity of the page partitioning. It builds a hierarchy of semantic blocks of the page that simulates well how a user understands the web layout structure based on his visual perception. We extended VIPS algorithm to complete the semantic structure of the page by extracting links, images and text for each block. At the end of the segmentation step, an XML document, called Vi-XML, describing the complete visual structure of the web pages, is produced.

In the change detection step, two Vi-XML files corresponding to two versions of a web page are compared. Many existing algorithms have been specially designed to detect changes between two semi structured documents. They find the minimum set of changes (delete, insert, update), described in a delta file, that transform one data tree to another. Cobéna et al. [8] propose the XyDiff algorithm to improve time and memory management. XyDiff supports a move in addition to basic operations. It achieves a time complexity of $O(n*log(n))$. Despite its high performance, it does not always guarantee an optimal result (*i.e.* minimal edit script). Wang et al. [21] propose X-Diff which detects the optimal differences between two unordered XML trees in quadratic time $O(n^2)$. DeltaXML [15] can compare, merge and synchronize XML documents for ordered and unordered trees by supporting basic operations. Both X-Diff and DeltaXML do not handle a move operation. There are several other algorithms like Fast XML DIFF [17], DTD-Diff [16], etc. After studying these algorithms, we decided to not use existing methods for our approach because they are generic-purpose. As we have various specific requirements related to the visual layout structure of web pages,

we prefer proposing our *ad'hoc* diff algorithm. As it is designed for one given specific type of document, it allows for a better trade-off between complexity and completeness of the detected operations set. Our proposed change detection step in Vi-DIFF algorithm detects structural and content changes. *Structural changes* alter the visual appearance of the page and the structure of its blocks. In contrast, *content changes* modify text, hyperlinks and images inside blocks of the page. Then, it constructs a delta file describing all these changes.

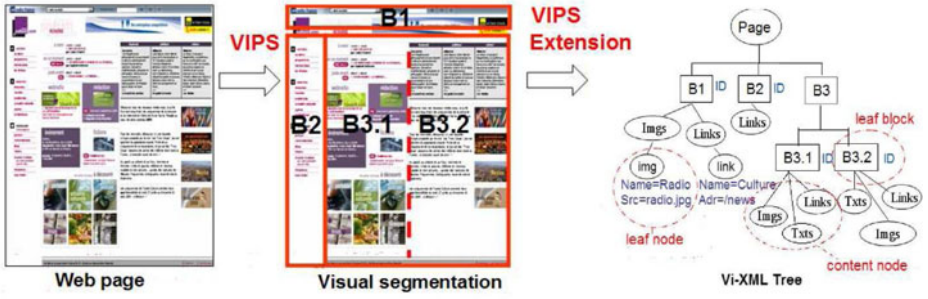
Our solution for detecting changes between two web pages versions can serve for various applications:

- **Web crawling.** The detected changes between web pages versions can be exploited to optimize web crawling by downloading the most "important" versions [3]. An important version is the version that has important changes since the last archived one. Based on (i) the relative importance of changes operations (insert, delete, etc.) detected in each block and (ii) the weight importance of those blocks [19], the importance of changes between two versions of web pages can be evaluated.
- **Indexing and storage.** Web archive systems can avoid wasting time and space for indexing and/or storing some versions with unimportant changes. An appropriate change importance threshold can be fixed through a supervised machine learning to decide when should pages indexed or stored.
- **Temporal querying.** The proposed change detection can greatly help to provide temporal query capabilities using structural and content changes history. Users can temporally query the archive about the structural changes that affects the visual appearance of web pages. For instance, what type of changes occur in a specific block during the last month...?
- **Visualization and browsing.** Our changes detection algorithm can be used to visualize the changes of web pages over time and navigate between the different archived versions. Several applications [13,18,20] have been designed to visualize changes in the browser by highlighting the content changes on pages such as a text update, delete, etc. However, to the best of our knowledge, no one explicitly view both the structural and content changes occurred on page such as a block insertion, move, update, etc. Our change detection approach can also be exploited to extend web browsers [20] that use the page cache to help users in understanding the changes on the current page since the last visit.
- **Archive maintenance.** Vi-DIFF can also be used after some maintenance operations to verify them in the web archive. The migration from the Internet Archive's file format ARC to WARC [1] is an example of maintenance operation. After the migration from ARC to WARC, our change detection Vi-DIFF can be exploited to ensure that the page versions are kept "as they are".

[1] The WARC format is a revision of ARC file format that has traditionally been used to store Web crawls url. The WARC better support the harvesting, access, and exchange needs of archiving organizations.

3 Vi-DIFF

In this section, we explain in detail the steps of the Vi-DIFF algorithm which detects structural and content changes between two web page versions. It has three main steps:




```
<xml>
  <Page url="www.radiofrance.fr" version="V_01-10-09">
    <Block Ref="B1" ID="001A" Pos="H:10-W:20">
      <Links ID="012C" IDList="042C">
        <link ID="1L23" Name="Culture" Adr="/news">
      </Links>
      <Imgs ID="g15K" IDList="g25K">
        <img ID="25jL" Name="Radio" Src="/radio.jpg"/>
      </Imgs>
    </Block>
    <Block Ref="B2" ID="150K" Pos="H:53-W:10">
      <Links ID="1k2M" IDList="4k2M"> ... </Links>
    </Block>
    <Block Ref="B3" ...>
      <Block Ref="B3.1" ...> ... </Block>
      <Block Ref="B3.2" ...> ... </Block>
    </Block>
  </Page>
</xml>
```

Fig. 1. VIPS Extension

- *Segmentation*: Web pages are visually segmented into semantic blocks by extending the VIPS algorithm.
- *Change Detection*: The structural and content changes between visually segmented web pages are detected.
- *Delta file*: The changes detected are saved in Vi-Delta files. Those three steps are described in detail in the next sections.

3.1 Segmentation

Based on the DOM tree and visual information of the page, VIPS detects horizontal and vertical separators of the page. Then, it constructs the "visual tree"of the page partitioned into multiple blocks. The root is the whole page. Each block

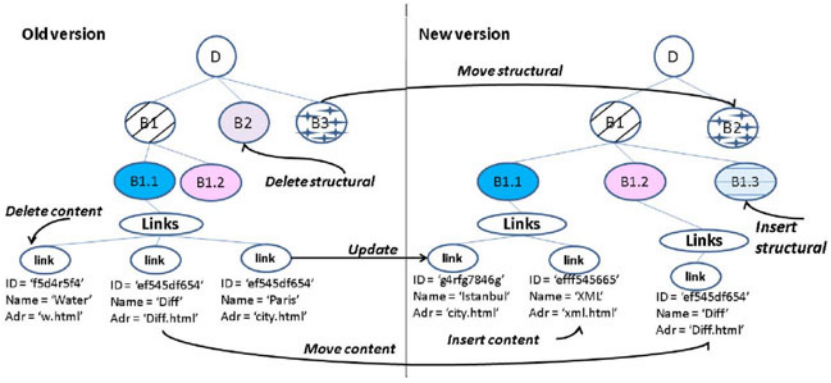


Fig. 2. Operations

is represented as a node in the tree. To complete the visual structure of the page, we extended VIPS to extract links, images and texts for each block as shown in Figure 1.

We define, three types of nodes: a *leaf block* is a block which has zero child block, a *content node* is a child of a *leaf block* like Links, Imgs, etc., and a *leaf node* is the node without a child such as img, link. These three types of nodes are uniquely identified by an ID attribute. This ID is a hash value computed using the node's content and it is children's node content. Links and Imgs (content nodes) have also an attribute called IDList which has list of all IDs of its leaf nodes. Each block of the page is referenced by the attribute Ref which contains the block's *Dewey Identifier*. Leaf nodes have other attributes such as the name and the address for links. The complete hierarchical structure of the web page is described in an XML document called Vi-XML. An example of Vi-XML document is shown in Figure 1.

3.2 Change Detection

In this section, we give the description of operations which are detected by Vi-DIFF. As mentioned in section 2, we consider two types of changes: content changes and structural changes. Content changes modify the content nodes of blocks. They are detected at leaf nodes. Structural changes are detected at the leaf blocks.

• Content change operations

As we mentioned earlier, content changes are realized in the content of blocks for links, images and texts in Vi-XML files through the following operations:

– *Insert(x(name, value),y):* Inserts a leaf node x, with node name and node value, as a child node of content node y. As shown in Figure 2, in the new version, a new leaf node (Name = 'XML') is added to content node Links of leaf block B1.1.

- *Delete(x):* Deletes a leaf node *x*. In Figure 2, in the old version, a leaf node (Name attribute = 'Water') is deleted from content node Links of leaf block B1.1.
- *Update(x,attrname,attrvalue):* Updates the value of *attrname* with *attrvalue* for a given leaf node *x*. In Figure 2, a leaf node's Name attribute 'Paris'is updated to 'Istanbul'in the new version.
- *Move(x,y,z):* Moves a leaf node *x* from content node *y* to content node *z*. As shown in Figure 2, a leaf node (Name = 'Diff') is moved from content node of the leaf block B1.1 to content node of the leaf block B1.2 in the new version. Most of the existing algorithms treat the move operation as a sequence of delete and insert operations. The use of move operation can have a great impact on the size of the delta script and on its readability. It reduces the file size but increments the complexity of the diff algorithm.

Delete, Update and Insert operations are considered as basic operations and existing algorithms support those operations. Move is only supported by few approaches.

• Structural changes operations

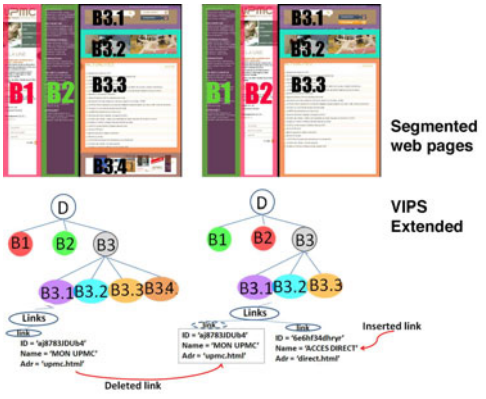
The structural changes are realized on leaf block nodes in Vi-XML files. Detecting structural changes decreases the size of the delta file and makes it easier to query, to store and to manage. Most of all, it provides a delta much more relevant with respect to what visually happened on the web page.

- *Insert(Bx,y,Bz):* Inserts a leaf block subtree Bx, which is rooted at *x*, to (leaf) block node *y*, after (leaf) block node Bz. For example, in Figure 2, a (leaf) block node B1.3 is inserted to block node B1, after leaf block node B1.2.
- *Delete(Bx):* Deletes a leaf block rooted by *x*. In Figure 2, B2 is deleted from the old version .
- *Move(Bx,By):* Moves sub-tree rooted by *x* (Bx) to the node *y*. After insert and/or delete operations, the nodes are moved (shifted). As shown in Figure 2, following the structural delete of B2 in old version, B3 is moved/shifted to B2 in new version.

3.3 Delta Files

Detected changes are stored in a delta file. We use XML to represent delta files because we need to exchange, store and query deltas by using existing XML tools. The delta files contain all operations (delete, insert, update, move, etc.) that transform one version to another one. The final result after applying delta file to the old version should be exactly the new version. As far as we know there is no standard for delta files.

The content changes are represented by operation tags (delete, insert, update, move) as children of their parent block's tag. The structural change operations such as delete and insert are added directly to the root of Vi-Delta. For move




```
<xml>
<Delta FROM="V1" TO="V2">
  <Delete>
      <Block Ref="B3.4" />
  </Delete>
  <Block Ref="B2" newRef="B1" />
  <Block Ref="B1" newRef="B2" />
  <Block Ref="B3.1">
      <Delete>
          <link Name="MON UPMC"
          Adr="upmc.html"/>
      </Delete>
      <Insert>
          <link Name="ACCES DIRECT"
          Adr="direct.html"/>
      </Insert>
  </Block>
</Delta>
</xml>
```

Fig. 3. Vi-Delta

operation in structural changes, a newRef attribute is added to the related block, so that we can keep track of a block even if it moves inside the structure.

Figure 3 shows an example of Vi-Delta. It contains three move and one insert as structural changes operations, one delete and one update as content changes operations. We can see, in this example, how detecting structural changes makes delta file easier to query, to manage and to store. For instance, in our example, instead of deleting and inserting all the content of moved blocks (B1, B2, B3), the structural move operation is used.

4 Change Detection

In this section, we give the details of Vi-DIFF's second step. The algorithm contains a main part and two sub-algorithms: one for detecting structural changes and other one for detecting content changes.

4.1 Main Algorithm

The main algorithm is composed of two steps as shown in Figure 4:

Input: Vi-XML Tree1, Vi-XML Tree2
Output Vi-Delta
Traverse Tree1, get list of blocks B1
Traverse Tree2, get list of blocks B2
Create Delta Tree DT
If the structure is changed **then**
    Call *DetectStructuralChanges*
else
    Call *DetectContentChanges*
endif
Save DT

Name: Name of link/image
ID: Hash value of link/image
Src: Address or source of link/image
BlockRef: Reference attribute of block who has link/image

Fig. 4. Main Vi-DIFF - Object Structure

- Step1: Is the structure changed?
  We compare the two trees to check if the structure is changed between the two versions. If they have a different number of block nodes, the structure is considered as changed. If they have the same number of blocks we start to compare their Ref in one iteration. As soon as we have a different Ref, the structure is considered as changed.
- Step2: Call change detection algorithms
  If the structure is the same, we directly call the content changes detection algorithm. If the structure is changed, the structural changes detection algorithm is called. It calls the content changes algorithm if there is also a content change. Those algorithms are explained in details in the next two sections.

4.2 Detecting Content Changes

The content changes are at the level of leaf node, for the children nodes of Links, Imgs and Txts. As Links and Imgs have nearly the same structure, they are treated in the same way. For Txts, we need to find text similarities, thus they are treated separately.

∘ *Links and images*

1. Create two arrays (one for each version), create a delta XML file
2. Traverse two trees (XML files) in the same iteration. For each leaf block not matching, check content nodes. If they do not match, create an object (see Figure 4) with each leaf node and add them in its version's array.
3. Sort both arrays by Name attribute (lexical order)
4. Define two counter variables (one for each array)
5. Advance both counters in an endless loop
   At each iteration:
   (a) Every time an operation is detected by comparing ID, Name, Src and Ref (see Figure 5 for details), add the operation to delta, remove the corresponding objects from arrays.
   (b) If objects (referenced by counters) in both arrays are different - advance the counter of the first array if it has the smallest Name and its next element has also the smallest Name, else advance the counter of the second array
   (c) Detect operations by comparing objects
   (d) if end of any array: set its counter to 0 which is necessary to compare the rest of the array
   (e) Break when both arrays' sizes are equal to 0

∘ *Texts*

We need to find the distance between two texts to decide if it is an update operation or a delete+insert operations. For this, we compute the proportion of distinct words between two versions. If this value is more than 0.5, they are considered as different texts and we have two operations delete then insert. Otherwise, the text is considered as updated. Each text in each block is compared separately.

```
Procedure DetectContentChanges(ASource,AVersion: Array,DT:Delta Tree)
Sort both arrays with merge sort(Name attribute)
while(true)
    if size of ASource and AVersion == 0 break
      if  ASource.ID == AVersion.ID  then
          if ASource.BlockRef != AVersion.BlockRef then
                MOVE detected
      else if (ASource.Name,ASource.BlockRef) == (AVersion.Name,AVersion.BlockRef)
               && (ASource.Adr/Src != AVersion.Adr/Src)  then
         UPDATE detected
      else if (ASource.Adr/Src,ASource.BlockRef) == (AVersion.Adr/Src,AVersion.BlockRef)
               && (ASource.Name != AVersion.Name)  then
         UPDATE detected
      else
        If end of any array
            If end of ASource && VSource.size !=0 then
               INSERT detected
            If end of VSource && ASource.size !=0 then
               DELETE detected
        else Increment counter
```

Fig. 5. Detect content change algorithm

4.3 Detecting Structural Changes

For detecting structural changes, we use two arrays respectively containing the leaf block nodes of the two compared versions. Two blocks (*e.g.* B1 in version 1 and B2.1 in version 2) are equal if their ID attributes are equal. If IDs are different, we compute the distance between them. To do this, we compare the IDList attribute for Links and Imgs nodes and IDs for Txts.

The distance measure of two blocks is defined on the basis of the distance between their content nodes. The following functions are defined to measure distance. Given two leaf block nodes B1 and B2, we define:

$$Distance(B1, B2) = \frac{Dist('Links', B1, B2) + Dist('Imgs', B1, B2) + DistText(B1, B2)}{3}$$

$$Dist(x, B1, B2) = \frac{\sum x \text{ changed between } B1 \text{ and } B2}{\sum x \text{ in } B1}, \text{ where } x \in \{'Links', 'Imgs'\}$$

$$DistText(B1, B2) = \begin{cases} 0 & if\ Text\ ID\ in\ B1\ =\ Text\ ID\ in\ B2 \\ 1 & otherwise \end{cases}$$

If Distance(B1,B2) is greater than a fixed value γ (*e.g.* $\gamma = 0.2$), the two blocks are considered as distinct, otherwise, they are considered as similar. If two blocks are similar, we go down in the tree by calling the detecting content change algorithm with the similar block nodes as arguments, here (B1, B2.1). If the two blocks are not similar, the counter of the longest array is incremented if the end of the both arrays is not reached.

Our algorithm has a quadratic complexity. But we should not forget that the asymptotic complexity is only appropriate with large inputs. The web pages are usually rather small and the cases where a page completely changes from one version to another one are very rare. Thus the asymptotic complexity is not relevant here. We decided to measure the actual complexity through experiments as explained in next section.

```
DetectStructuralChanges(Version1, Version2: XML, DT: Delta Tree)
Find and Add leaf blocks(b1,b2) in Version1 and Version2 to arrays A1 and A2
while(true)
    if size of A1 and A2 == 0 break
  If (b1.ID,b1.BlockRef) ==  (b2.ID,b2.BlockRef) then
      IDEM detected
  else if b1.ID == b2.ID  && b1.BlockRef!= b2.BlockRef
      MOVE detected
  else
      if b1 is similar to b2 then
           MOVE detected
           call DetectContentChanges (b1, b2, DT)
      else if end of one of arrays
           if( end of A1) then b2 is inserted
           if( end of A2) then b1 is deleted
      else Increment counter
```

Fig. 6. Detect structural changes algorithm

5 Experiments

Experiments are conducted to analyze the performance (execution time) and the quality (correctness) of the Vi-DIFF algorithm.

5.1 Segmentation

Visual segmentation experiments have been conducted over HTML web pages by using the extended VIPS method. We measured the time spent by the extended VIPS to segment the page and to generate the Vi-XML document. We present here results obtained over various HTML documents sized from about 20 KB up to 600 KB. These represent only sizes of container objects (CO) of web pages that do not include external objects like images, video, etc.

We measured the performance of the segmentation in terms of execution time and output size. Experiments were conducted on a PC running Microsoft Windows Server 2003 over a 3.19 GHz Intel Pentium 4 processor with 6.0 GB of

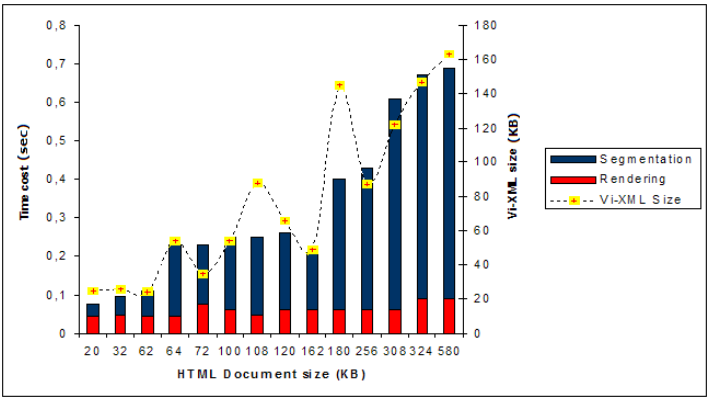


Fig. 7. Segmentation Time

RAM. The execution time for the browser rendering and the visual segmentation is shown in Figure 7. The horizontal axis represents the size of HTML documents in KBytes (KB). The left vertical axis shows the execution time in seconds for each document. The time for rendering is almost constant, about 0.1 seconds. The execution time of the visual segmentation increases according to the size of HTML documents. The time of the segmentation is about 0.2 seconds for documents sized about 150KB which represents the average size of CO in the web. This execution time seems to be a little bit costly but is counterbalanced by the expressiveness of the Vi-XML file that really simulates the visual aspect of web pages. Nevertheless, this time cost must be optimized. The main idea for that purpose is to avoid rebuilding the blocks structure for a page version if no structural change has occurred since the former version. We are currently trying to find a method that detects directly changes inside blocks based on the visual structure of previous versions of the document.

The right vertical axis in Figure 7 shows the the size of the output Vi-XML file with respect to the size of the original HTML document. From this experiment, we can observe that the Vi-XML document size is usually about 30 to 50 percent less than the size of the original HTML document (for those sized more than 100 KB). This is interesting for the comparison of two Vi-XML documents since it can help to reduce the time cost of changes detection algorithm.

5.2 Change Detection

We have two types of test in this section: one without structural changes and another with structural changes. For this section, tests are realized on PC running Linux over a 3.33GHz Intel(R) Core(TM)2 Duo CPU E8600 with 8 GB of RAM.

Without structural changes. To analyze the performance and the quality of the second step of Vi-DIFF algorithm, we built a simulator that generates synthesized changes on any Vi-XML document. The simulator takes a Vi-XML file and it generates a new one according to the change parameters given as input. It also generates a delta file, called Vi-Sim-Delta. The two Vi-XML files are compared with the Vi-DIFF algorithm and a Vi-Delta is generated. To check the correctness of change detection step, we compared all Vi-Delta et Vi-Sim-Delta files with DeltaXML trial version [15]. Vi-Delta and Vi-Sim-Delta files are always identical which shows the correctness of this step of Vi-DIFF. For this test, different versions are obtained from a crawled web page (http://www.france24.com/fr/monde) by using our simulator. For each type of operation (delete, insert, update, move) the change rate is increased by 10% until reaching 100%. It means that the last version with 100% change rate is completely different from the original one. As links and images are treated in the same way, we observe that the execution time is the same on average. To simplify the figure's readability, we only give the results for links. As the change detection part is measured in milliseconds, processor events may cause noise on the test results. Thus, all the process is executed for 10000 times and the results obtained are normalized (average). The results are given in Figure 8.

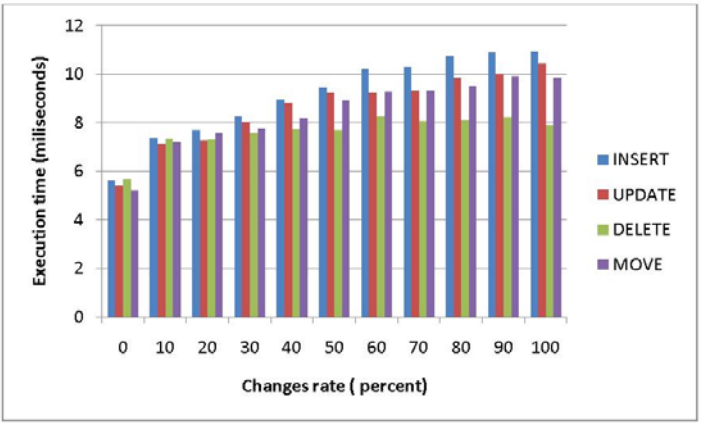


Fig. 8. Change Detection Execution Time

As we can see, move and update operations have nearly the same execution time because they do not change the size of Vi-XML files, but insert and delete operations change the size. The execution time increases along with the insert operation rate (more objects to compare), and decreases along with the delete operation rate (less objects to compare). Also a profiler analyzer is used to see the details of execution time. 70% of the time is used to parse Vi-XMLs and to save Vi-Delta, 10% is used to detect if there is a structural changes, 20% is used to detect changes.

With structural changes. In order to test the structural changes, the versions with the structural changes are obtained by modifying the segmentation rate (degree of coherence) in the VIPS algorithm. It can be seen as "merge and split"of blocks which change the Vi-XML's structure. It also allows us to test blocks' similarity by using hourly crawled web pages. Average file size of those pages is 80 Kb. To give an idea about initial environment, we first compare two identical Vi-XML files. The execution time is 7,3 ms. Then, two versions hourly crawled (http://www.cnn.com) are segmented with different segmentation rate into two Vi-XML files. These files are compared. The execution time is 9.5 ms and the generated Vi-Delta size is 1.8 Kb. Vi-DIFF algorithm is modified to not consider structural changes and the same Vi-XML files are compared. The execution time is 48 ms and the generated Vi-Delta size is 40 Kb. Because with structural change detection, there is no need to go down until leaf nodes for all blocks except similar blocks so the change detection is faster. Also, a smaller delta file is easier to store, to read and to query. As the simulator is not able to make structural changes for the moment, the correctness of delta file for this section is checked manually.

According to our experiments, Vi-DIFF is able to detect basic operations and move (without structural changes) correctly between two web page versions. The total execution is between 0,1 seconds and 0.8 seconds depending on the page

versions' size. The execution time for change detection step is satisfying as it allows to process about one hundred pages per second and per processor. Thus, we are working on reducing the segmentation time.

6 Conclusion and Future Works

We propose Vi-DIFF, a change detection method for web pages versions. Vi-DIFF helps to understand what happened and changed between two page versions. It detects semantic differences between two web pages based on their visual representation. Preserving the visual structure of web pages while detecting changes gives relevant information to understand modifications which is useful for many applications dealing with web pages. Vi-DIFF consists of three steps: (i) segmentation, (ii) change detection and (iii) generation of delta file. It detects two types of changes; structural and content changes. Structural changes (insert, delete, move) alter the block structure of the page and its visual appearance. Content changes (insert, delete, update) modify links, images, texts. In addition to basic operations, it supports a move operation, if there is no structural changes. However, it does not support yet the move on content changes when there are structural changes. This should be handled by future versions.

The proposed Vi-DIFF algorithm generates, as output, a Vi-Delta file which describes change operations occurred between the two versions. The structure of the produced Vi-Delta helps to visually analyze the changes between versions. It can serve for various applications: web crawl optimization, archive maintenance, historical changes browsing, etc. A simulator was developed to test the performance of the algorithm. Experiments in terms of execution time were conducted for each step. The execution time for change detection step is very promising, but the execution time cost for segmentation step must be optimized.

Another on-going future work is to detect splitting and merging of blocks as structural changes. The simulator also needs new features like generating new versions with structural changes. Also we want to exploit the produced Vi-Delta to analyze/evaluate the importance of changes between versions. Further work will be done to efficiently query archived Vi-XML versions and the visual/structural changes occurred between them as described in the Vi-Delta.

References

1. The Web archive bibliography, `http://www.ifs.tuwien.ac.at/~aola/links/WebArchiving.html`
2. Abiteboul, S., Cobena, G., Masanes, J., Sedrati, G.: A First Experience in Archiving the French Web. In: Agosti, M., Thanos, C. (eds.) ECDL 2002. LNCS, vol. 2458, p. 1. Springer, Heidelberg (2002)
3. Ben-Saad, M., Gançarski, S., Pehlivan, Z.: A Novel Web Archiving Approach based on Visual Pages Analysis. In: 9th International Web Archiving Workshop (IWAW'09), Corfu, Greece (2009)
4. Blakeman, K.: Tracking changes to web page content, `http://www.rba.co.uk/sources/monitor.htm`

5. Lampos, D.J.C., Eirinaki, M., Vazirgiannis, M.: Archiving the greek web. In: 4th International Web Archiving Workshop (IWAW'04), Bath, UK (2004)
6. Cai, D., Yu, S., Wen, J.-R., Ma, W.-Y.: VIPS: a Vision-based Page Segmentation Algorithm. Technical report, Microsoft Research (2003)
7. Cathro, W.: Development of a digital services architecture at the national library of Australia. EduCause (2003)
8. Cobena, G., Abiteboul, S., Marian, A.: Detecting changes in XML documents. In: ICDE '02: Proceedings of 18th International Conference on Data Engineering (2002)
9. Cosulschi, M., Constantinescu, N., Gabroveanu, M.: Classification and comparison of information structures from a web page. In: The Annals of the University of Craiova (2004)
10. Evi, M.K., Diligenti, M., Gori, M., Maggini, M., Milutinovi, V.: Recognition of Common Areas in a Web Page Using Visual Information: a possible application in a page classification. In: The Proceedings of 2002 IEEE International Conference on Data Mining ICDM'02 (2002)
11. Gomes, D., Santos, A.L., Silva, M.J.: Managing duplicates in a web archive. In: SAC '06: Proceedings of the 2006 ACM Symposium on Applied Computing (2006)
12. Gu, X.-D., Chen, J., Ma, W.-Y., Chen, G.-L.: Visual Based Content Understanding towards Web Adaptation. In: De Bra, P., Brusilovsky, P., Conejo, R. (eds.) AH 2002. LNCS, vol. 2347, p. 164. Springer, Heidelberg (2002)
13. Jatowt, A., Kawai, Y., Nakamura, S., Kidawara, Y., Tanaka, K.: A browser for browsing the past web. In: Proceedings of the 15th International Conference on World Wide Web, WWW '06, pp. 877–878. ACM, New York (2006)
14. Kukulenz, D., Reinke, C., Hoeller, N.: Web contents tracking by learning of page grammars. In: ICIW '08: Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services, Washington, DC, USA, pp. 416–425. IEEE Computer Society, Los Alamitos (2008)
15. La-Fontaine, R.: A Delta Format for XML: Identifying Changes in XML Files and Representing the Changes in XML. In: XML Europe (2001)
16. Leonardi, E., Hoai, T.T., Bhowmick, S.S., Madria, S.: DTD-Diff: A change detection algorithm for DTDs. Data Knowl. Eng. 61(2) (2007)
17. Lindholm, T., Kangasharju, J., Tarkoma, S.: Fast and simple XML tree differencing by sequence alignment. In: DocEng '06: Proceedings of the 2006 ACM Symposium on Document Engineering (2006)
18. Liu, L., Pu, C., Tang, W.: Webcq - detecting and delivering information changes on the web. In: Proc. Int. Conf. on Information and Knowledge Management (CIKM), pp. 512–519. ACM Press, New York (2000)
19. Song, R., Liu, H., Wen, J.-R., Ma, W.-Y.: Learning block importance models for web pages. In: WWW '04: Proceedings of the 13th International Conference on World Wide Web (2004)
20. Teevan, J., Dumais, S.T., Liebling, D.J., Hughes, R.L.: Changing how people view changes on the web. In: UIST '09: Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology, pp. 237–246. ACM, New York (2009)
21. Wang, Y., DeWitt, D., Cai, J.-Y.: X-Diff: an effective change detection algorithm for XML documents. In: ICDE '03: Proceedings of 19th International Conference on Data Engineering (March 2003)