Baltic J. Modern Computing, Vol. 3 (2015), No. 3, 193-204

HTML Document Content Comparison Algorithm

Armantas OSTREIKA, Andrius LAURAITIS

Kaunas University of Technology, Department of Multimedia Engineering,
Studentų St. 50, LT-51368, Kaunas, Lithuania

armantas.ostreika@ktu.lt, andrius.lauraitis@ktu.lt

Abstract. This paper describes still encountered problems of documents visual content comparison in contemporary computerized workplaces. There are many ways for creating HTML documents and plenty of invisible to user data that carries no information in terms of content. Such circumstances make the automation and visualization process of HTML document comparison rather complex. Introduced algorithm compares versions of HTML documents and displays changes in a result document. The comparison is carried out in such a way that all style and metadata of the document is preserved. Furthermore, the design phases and implementation aspects of the algorithm are investigated in order to share achieved results, to create an effectively working tool and draw guidelines for future work.

Keywords: document content comparison, style data preservation, changes visual tracking, HTML document.

1. Introduction

HTML files that are the fundamental elements of web-based systems get modified periodically. Many cases occur where HTML documents are edited by creating, deleting or updating some of the existing text and evolving to a new version of the document resulting to a different file. Thus the original file has a revision and both of them can be compared to analyze and track the changes by the 'end user'. Typically, but not necessarily, file comparison can be used for generalization purposes, e.g. reporting distinguishing alterations of system modules status between particular date range, etc. The mentioned example just introduces capabilities of applying such a comparison procedure in practice. In real world's scenarios it could also be beneficial in system file monitoring and version control, tracking changes of various project work or thesis report, providing assistance for software and web developers, quality managers.

However the comparison of document content is not so simple and trivial task as it might seem at the first glance. Visible text in document files is encoded and formatted in a variety of tags and accompanied by metadata, which are dependent on the document creator. The problems of detecting HTML content changes and providing a good quality comparison results' document still persist today and are not resolved completely. Aim and intention of this work is to find a possible solution for this problem. One of the most appropriate approaches to apply a document comparison procedure in practice is to utilize it as a tool. This way it could be a tangible product for users to benefit from. To clarify the work that has been done in this area a carried out analysis as a research method is presented in the next chapter of the paper.

2. Existing document comparison tools analysis

There is a huge variety of file comparison tools in the market. Some of them are free (bundled with GPL1 usually) other have proprietary software licenses and cost money. Moreover, the functionality of such tools is very different: from showing in-line and character changes, moved lines, defining structured comparison, supporting Unicode to merging changes, generating reports, version control, directory comparison etc. The selected tools were chosen for the analysis due to number of reasons:

1. To show the elementary, easy to use comparison tools that meet the basic expectations when working with text data;
2. To show the contrast between text based and graphical tools, emphasizing quality of visual representation;
3. To distinguish those particularly dealing with HTML files as they fit closest to the tasks under investigation.

Next sections of this chapter explore selected comparison tools, their working principles and summarize common features.

2.1 Built-in Windows and Linux file comparison tools

Some of the most trivial tools in Windows OS for comparing files are the COMP and FC commands. In Linux OS, a classical file comparison diff utility can't be ignored. It is based on solving the longest common subsequence problem (Balcan, 2011) and has many built-in features. To sum up, briefly analyzed file comparison tools does not support graphical result representation. It can sometimes be inconvenient because users must have at least minimum skills how to use the commands and read output at file source level. Furthermore the format of a HTML file would be distorted and it could not be loaded into a browser directly.

2.2 More feature-rich file comparison tools

In this chapter more sophisticated file comparison tools are analyzed. These selected tools are designed to display visual aspects of tracked content changes and are especially adapted for HTML documents comparison. The key aspects of proprietary visual HTML document comparison tools are high content change detection quality and document style preservation. So the next sections of this chapter show the basic functionality of these tools. In order to check the quality of a compared result document better, each of the tools are tested with a set of prepared HTML files that have comment information, nested tags, Unicode characters and inline styles.

[1] The GNU General Public License (GPL) is the most widely used free software license.

Daisydiff (Code Google project base, 2007). It provides features like: comparing badly formed HTML, detecting text fragments content changes, diffing source code coherently. In addition, a Daisydiff compared result document has a navigation and modification report system which helps finding a particular change faster. Although being a solid comparison tool, Daisydiff doesn't preserve style related data and maintain Unicode characters properly.

HTML-diff. This tool (Charles University, 2010) compares HTML files not only at source level but shown text as well (Charras and Lecroq, 2004). Html-diff does a pretty good job with HTML file comparing. However the result document is not well formatted. Moreover some HTML tags are treated as content changes and it does not show any visual changes with complex large HTML files.

HTML Match. A powerful commercial GUI tool (Salty Brine, 2005) for Windows OS. Features include: choosing a comparison mode (visual aspects, source code), defining output format, choosing difference detail level (character, word, line and document), ignoring whitespace, navigating between found content changes, and associating text extraction engine with MS Word. HTML Match is the highest quality comparison tool of all selected ones. It preserves all style data, has comparison modes. However when loading large complex HTML files the visual aspect mode does not show any results (although the source code and text mode works).

2.3 Summary of the selected tools

After analyzing selected comparison tools, a list of features are made to show the key aspects to consider when implementing a high quality HTML document comparison utility. Results are displayed in Table 1.

Table 1. List of selected document comparison tools and their features

Title	User interface	Change tracking level	Visualization of changes	Formatting and style preservation	Large file handling
Diff	Textual	Line, Document	No*	No	Yes
FC, COMP	Textual	Character, Line	No	No	No
Daisydiff	Textual‡	Word, Line, Document	Yes	Partial°	Yes
HTML-diff	Textual	Word, Line, Document	Yes	Partialº	Yes
HTML Match	Graphical	Word, Line, Document	Yes	Partial˜	Yes

* Classical version doesn't have this feature, however there are some modifications.

‡ Basic utility is console based, but there is a plugin for a graphical user interface system DaisyCMS.

° Structure of the result document was malformed as some extra unnecessary HTML tags were generated.

º Some output text symbols were not recognized as valid ones in browser. In addition, content changes were not detected in HTML table elements.

˜ Visual comparison mode of the tool did not provide any results when multilevel nested HTML tags with inline comments were tested.

In conclusion, all the reviewed tools have their advantages and practical application areas. Unfortunately tool features that represent HTML document content changes visually have their downsides and need some improvement.

3. Determining a design solution for a HTML document comparison algorithm

In order to implement a more efficient HTML document comparison algorithm different design strategies for such task should be considered. Firstly, HTML files are structured documents with embedded mark-up used for defining the semantics of various elements according to a schema. Additionally, HTML files can be represented as the Document Object Model (DOM) where every document node is organized in a tree structure (Deng, C. et al., 2003). Two different techniques are considered in this chapter: tree structure navigation and single line preprocessing.

3.1 Considerations about applying a tree structure

Here, two special cases are emphasized in the analyzed task: 1) Comparison of data that forms the content (text, images etc.); 2) Analysis of document structure and identification of formatting elements.

In the first case analysis is not dependent on the document structure i.e. text fragment comparison must be performed at character level, interpreted and merged to word or entire paragraph level afterwards. Performance of such analysis is not related to document structure whereas identification of formatting elements could be done in different ways: analyzing document line by line or considering the format and structure of a file. However hierarchical tree structure of a document might be very complex since its height and width is not fixed. Each node of such tree must be analyzed because the node itself or data inside the node could result in a content change (Chen et al., 2003). Except the HEAD section all other HTML elements which are located in BODY part could be repeated many times or nested inside each other, e.g. a paragraph has subparagraph that has even more subparagraphs etc. (Fig. 1).

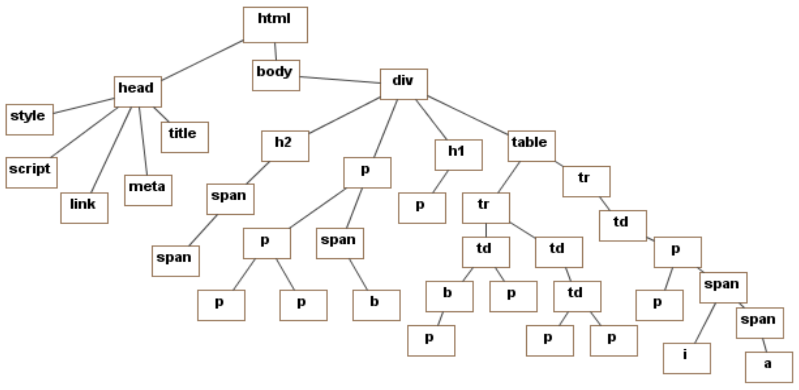


Fig. 1. HTML tree structure fragment. Node name correspond to HTML tag name.

The key point is that an algorithm should search for content changes rather than changes in file structure. If a document consists mainly of text paragraphs then a tree will be very low but wide and the analysis of such tree will not be different from one that applies a cyclic text line processing approach. Moreover, from a tree structure viewpoint, if the font size of particular words is changed in a line then these changes will appear in a separate level or additional sub-branch of a hierarchical tree structure (Artail and Fawaz, 2008). These shifts are not meaningful because in such cases no content changes are made. So, if a tree structure is applied then additional analysis must be performed to check if new document nodes are not partial fragments of nodes which were found previously. Consider the following HTML code fragment (illustrated as A in Fig. 2).

```
<p style="margin-top: 0pt; margin-bottom: 0pt;" align="left">
Additional material for laboratory work</p>
<p style="margin-left: 10px;">
<a href="http://www.personalas.ktu.lt/MMedia/MMediaH.htm">
     For 3rd course students (T120B186)</a></p>
```

However if words become bold or font is changed tree structure changes (B in Fig. 2).

```
<p style="margin-top: 0pt; margin-bottom: 0pt;" align="left">
<font color="#4e0000" face="Arial Black" size="4">
Additional material for laboratory work</font></p>
<hr><p>andnbsp</p><p style="margin-left: 10px;">
<a href="http://www.personalas.ktu.lt/MMedia/MMediaH.htm">
    For <b>3rd course</b> students (<i>T120B186</i>)</a></p>
```

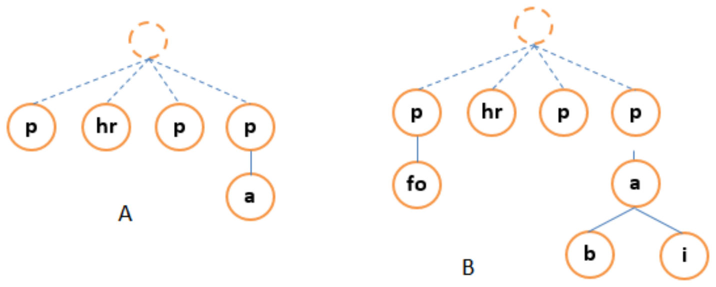


Fig. 2. Changing tree structure (not content) of a HTML document.

In conclusion, Fig. 2 illustrates an important concept: despite of the fact the structure of the tree changed content should be the same. Node “a” should have been treated as identical though the tree structure implies differently. Due to these reasons a single file line (not a HTML tag) processing approach is selected.

3.2 Proposed algorithm

Algorithm analyzes two HTML documents and detects content changes preserving style data. After processing original and revised HTML documents a comparison result document is formed. Tracked content changes are visualized.

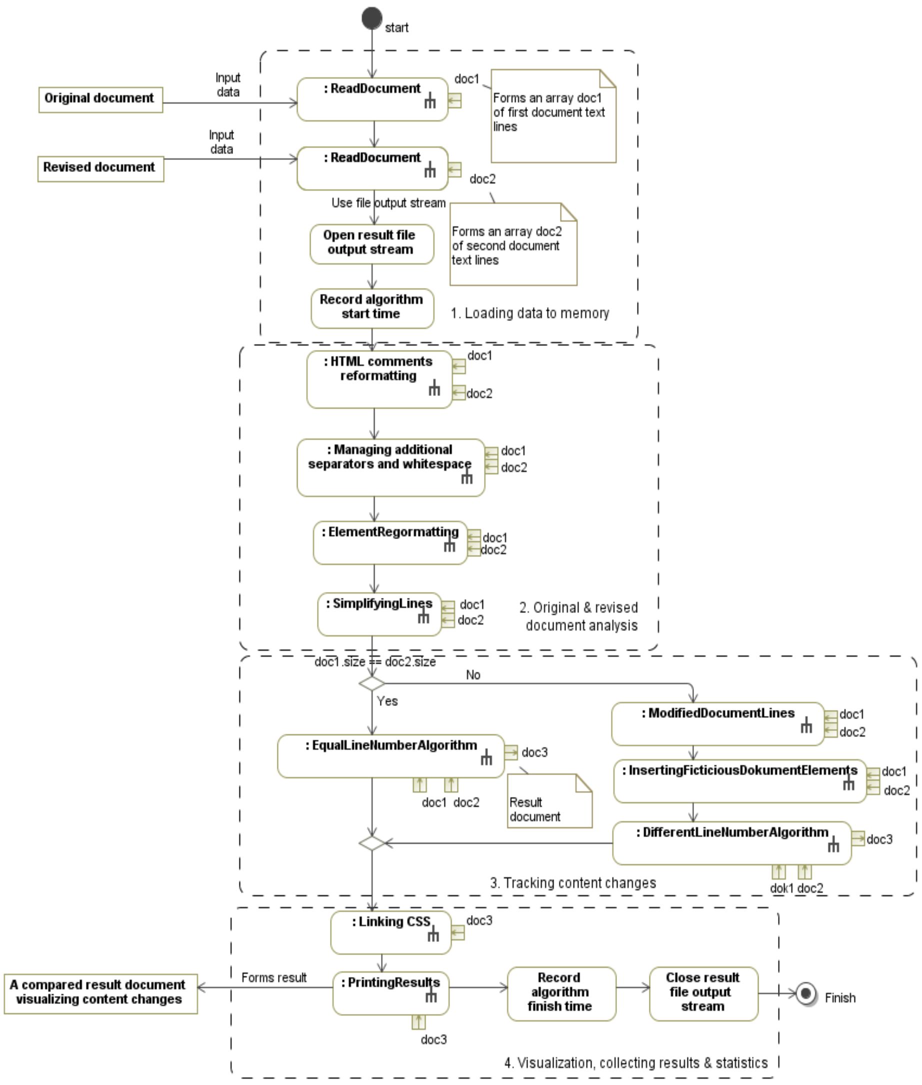

start
Input data
Original document
: ReadDocument
doc1
Forms an array doc1 of first document text lines
Input data
Revised document
: ReadDocument
doc2
Use file output stream
Forms an array doc2 of second document text lines
Open result file output stream
Record algorithm start time
1. Loading data to memory
: HTML comments reformatting
: Managing additional separators and whitespace
: ElementRegormatting
: SimplifyingLines
2. Original & revised document analysis
doc1.size == doc2.size
No
Yes
: ModifiedDocumentLines
: EqualLineNumberAlgorithm
doc3
: InsertingFicticiousDokumentElements
Result document
: DifferentLineNumberAlgorithm
3. Tracking content changes
dok1 doc2
: Linking CSS
Forms result
A compared result document visualizing content changes
: PrintingResults
Record algorithm finish time
Close result file output stream
Finish
4. Visualization, collecting results & statistics

Fig. 3. Proposed algorithm workflow schema

Algorithm processing phases and working principle. Four essential processing phases (Fig. 3) are defined: 1) Loading data to memory; 2) Original and revised document analysis; 3) Tracking content changes; 4) Collecting results and statistics.

As the detailed workflow schema implies, algorithm is divided into two branches: one with equal line numbers in both documents, other with different ones. Each action in particular branch corresponds to specific functionality. In the first stage all initial document data is loaded into computers memory. Second stage includes document line analysis i.e. reformatting HTML tags and counting total number of lines. In third stage the document levels where content changes were found are detected. Depending on the determined level (document, line, word, symbol) appropriate further actions to find changed elements are executed. During fourth algorithm processing phase final results, including statistics, are gathered.

Loading data to memory. In this stage, the number of lines in each document are calculated. This determines the size of initial document data. In the implementation function, file is read until the end of it and text lines are stored in an array structure after that. Procedure is applied with different arguments to the original and revised documents. Loaded data is used in the next phase where documents are analyzed.

Original and revised document analysis. This stage is a mandatory step in performing a comparison routine. HTML documents are arranged in the way that detection of content changes would be as easy as possible. Features include:

- HTML comment handling and avoiding situations when the end of one comment in the document coincides with the start of other comment tag (Altinel and Franklin, 2000);
- Ignoring whitespace and other separators at the beginning and end of a line. This gives more capabilities to prepare initial documents by copying and pasting text;
- Reformatting[2] standard tags (<p>, <table>, <td>, <tr>, <h1>, <h2>, <!--, etc.). In order to gain better comparison results, this kind of analysis is made when text elements of one tag are lied out in several lines (Lim and Ng, 2001);
- Simplifying document text lines by making a copy of the original element. Simplified lines only have tag name and text content (all additional information is excluded).

Tracking content changes. Phase working principle is based on two scenarios: 1) With equal line numbers in both documents; 2) With different line numbers in both documents. In the first scenario content changes are tracked at character, word and line level. In the second scenario size of documents are unified first by detecting changes at document level and referring to first scenario afterwards. Moreover, second scenario is more time consuming because many data insertion operations are done in the entire document. Required procedures in the first scenario:

- Document line decomposition. Each distinguished text fragment (Flesca and Masciari, 2003) is saved into an array. Procedure is applied to both documents.
- Insertion of fictitious line elements. Analyzes text fragment array for both documents and determines which elements should be inserted, deleted or left unchanged;
- Capture of line changed text fragments (MacKenzie et al., 1993). This procedure relies on the insertion of fictitious line elements routine and generates a new text fragment array which stores information about detected changed text tokens[3];

---

[2] Changing the format of existing HTML tags is done in such that if the same element wraps in multiple lines then it is treated as single line element preserving all style and content data.

[3] Include deletion or insertion indication symbols for changed characters or words.

- Formation of a result line. Data from a newly formed array in the previous procedure is appended into one line. A set of such lines forms the final comparison result HTML document (Yang and Shang, 2001).

The second scenario requires even more procedures because it refers to all functions from the first scenario at a particular phase. However there are some additional routines that must be performed:

- Search for modified document lines (Lecroq, 2007). Both initial documents are analyzed and specific indicators are formed to mark modified document lines. Procedure is important in order for algorithm to find content changes not only at entire document level but also at line, word and character level.
- Insertion of fictitious document elements. Unification of both documents sizes is performed during this procedure which deals with not found document lines. Moreover, modified lines can't be treated as fictitious elements (Cobena et al., 2002).
- Detecting changes in the whole document level. According to previously formed indicators, lines are inserted or deleted (Mikhaiel et al., 2005). If only modified lines are found then functions from first scenario are executed.

Regardless of initial document sizes and the scenario that algorithm tracks content changes, a result document forming process is always conducted.

Visualizing content changes and gathering statistics. The control of how the final result (Štěpánek and Šimková, 2012) is shown is made in an external CSS file. A particular procedure is applied to establish a link between the result HTML comparison document and a CSS file. Style sheet file define classes that are used to various HTML multimedia objects to track inserted (green color) and deleted (red color) content. Relevant display rules from CSS are applied according to different detected element type e.g. changing text font in a paragraph etc.

4. Achieved experimental results

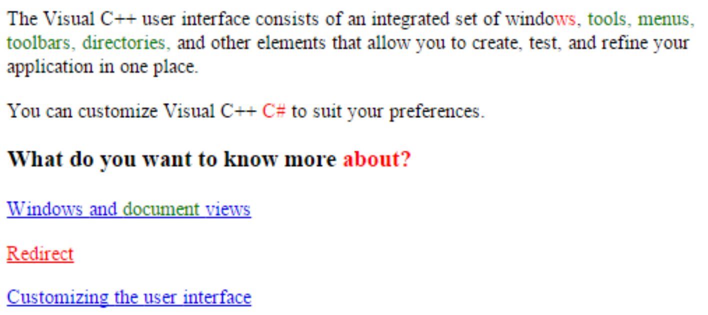
This chapter specifies results that were achieved with the implemented HTML document comparison algorithm. Selected files for testing had associated comment and blank lines, nested HTML tags with inline styles, lines with no text content, HTML tables, paragraphs, headers, and various multimedia elements (images, videos, audio, embedded objects etc.). Moreover, documents prep ared in Microsoft Word and saved as regular or filtered HTML web pages are supported and can be compared to view content changes. Further chapter segments are laid out in such a way that different content detection levels (word, line and document) are emphasized.

4.1 Detecting changes at word, line and document level

In the entire content change detection process the document level has the highest priority because it equalizes[4] the size of original and revised documents and allows tracking content changes at line and word level for remaining unchanged

[4] This is managed by inserting fictitious units to newly added or deleted lines in corresponding document location.

lines in further processing stages. This is illustrated in another, slightly more complex example (Fig. 4) with different document line numbers where content changes at all three levels in HTML paragraphs, headers and hyperlinks are tracked (Julian, 2006). Inline HTML element styles could also be provided. Rendered comparison result in a web browser:


The Visual C++ user interface consists of an integrated set of windows, tools, menus, toolbars, directories, and other elements that allow you to create, test, and refine your application in one place.

You can customize Visual C++ C# to suit your preferences.

What do you want to know more about?

Windows and document views

Redirect

Customizing the user interface

Fig. 4. Detecting content changes at word, line and document level

4.2 Detecting HTML multimedia objects

So far, previous sections of this chapter described how text based content changes are detected and displayed in document. However HTML files may include many other interactive content forms like images, video, audio etc. Designed HTML document comparison algorithm is compatible with tracking changes in such multimedia objects. Currently HTML 5 media tags (video, audio, Youtube and other plugins) are supported along with formats (embed, object, iframe tags) in older HTML versions. Changed multimedia object is detected only if it was added or deleted as entre unit in the revised document i.e. tracking partly modified graphical elements is considered to be a separate task. Fig. 5 illustrates detected multimedia objects in green or red border.

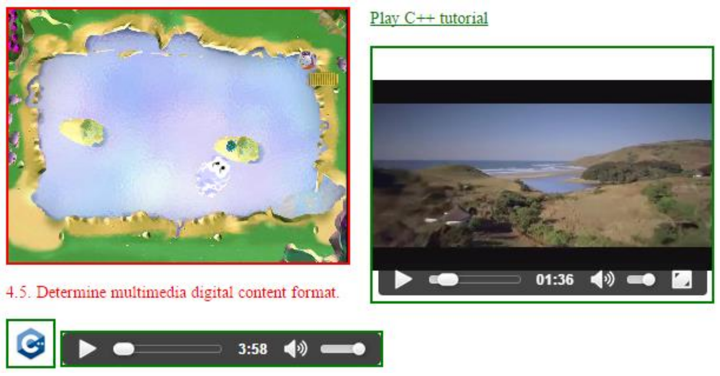


Fig. 5. Detecting changed multimedia objects in a document

Thus provided algorithm test scenarios emphasize the quality aspects of a comparison result. However algorithm speed performance also matter as files can get very large.

4.3 Algorithm speed performance

Several HTML document creation approaches were chosen in algorithm testing process. Furthermore different structure HTML files were considered which were created either with a HTML editor or generated by Microsoft Word. In order to get proper experimental results computers were disconnected from internet, leaving only active system processes. Three computers (numbered ascendingly by CPU frequency) were picked randomly and test sets were run separately in each computer to compare speed results. Selected test computers can be further analyzed and investigated in context of overall CPU Benchmark rating system (CPU Benchmark, 1998). PC1: Intel® CPU T2060 1.60 GHz, 0.99 GB RAM, 32-bit; PC2: Intel® Core™ i5 – 2430M, 2.4 GHz, 8 GB RAM, 64-bit; PC3: Intel® Core™ i5 – 3470 CPU 3.2 GHz, 4 GB RAM, 64-bit.

Table 2. HTML document comparison algorithm speed performance results

Document Type	Equal Line Number	Document size (lines / symbols)	Inline styles	Computer Type	Algorithm Speed (s)
”Clean“* HTML document	Yes	796 / 95930	No	PC3	49,37
				PC2	71,35
				PC1	147,04
	No	798 / 95989 with 813 / 96227	No	PC3	54,59
				PC2	67,17
				PC1	147,82
HTML table‡ document	Yes	3327 / 124593	No	PC3	108,35
				PC2	136,20
				PC1	475,48
	No	2232 / 162647 with 2021 / 144287	Yes	PC3	80,33
				PC2	106,18
				PC1	280,42
MS Word generated document	No	947 / 52032 with 2700 / 142568	Yes	PC3	30,78
				PC1	40,23
				PC2	46,56
		9819 / 618263 with 9590 / 602921	Yes	PC3	524,36
				PC2	653,90
				PC1	1086,37

* Prepared without any additional tags like <!--, <script>, <meta> etc. Does not require reformatting.
‡ The basis of document content is formed using HTML table elements when corresponding data fields are edited. Tag reformatting operation is required.

To sum up, the better CPU tactical frequency the faster a result is formed. Moreover computers with a higher rank in CPU Benchmark system process the HTML document comparison algorithm quicker. Algorithm performance is slower if these factors increase: number of lines in document, reformatting operations, size difference between both documents (more content changes needs to be detected) and inline styles.

5. Conclusions and further work

The developed HTML document comparison algorithm tracks content changes at word, line, document level, preserves style data and can process large files. Mentioned features improve the quality of presenting results visually. Compared HTML document can be viewed in a browser directly. Algorithm was implemented and tested in MS Windows platform with C++ programming language.
Implementing a tree structure for HTML file comparison routine would be meaningful if the tree is applied not only for documents but for complex hierarchical structure websites. An intermediate algorithm combining single line processing approach and using a tree structure (to a certain hierarchical level) should be considered as well.
However ability to process very complicated HTML code results in relatively large processing time. Proposed and implemented HTML document comparison algorithm is still in a prototype version therefore further work should be carried out in: optimizing the algorithm and improving its speed by exploiting different data structures (e.g. linked lists), switching from high level programming language like C++ to middle level C. In addition investigating parallel computing on CUDA-enabled GPU should be considered where each text line or node processing is executed in a separate core.

References

Altinel, M., Franklin, M.J. (2000). Efficient Filtering of XML Documents for Selective Dissemination of Information. USA, University of Maryland and University of California at Berkeley. Proceedings of the 26th VLDB Conference, Cairo, Egypt http://www.cs.berkeley.edu/~franklin/Papers/XFilterVLDB00.pdf

Artail, H., Fawaz K. (2008). A fast HTML web page change detection approach based on hashing and reducing the number of similarity computations. Data and Knowledge Engineering, Volume 66, Issue 2, pp 326-337.

Balcan, Maria-Florina. (2011) Longest common subsequence problem. http://www.cc.gatech.edu/~ninamf/Algos11/lectures/lect0311.pdf. Accessed 12 Dec 2014.

Charles University (2010), Faculty of Mathematics and Physics. HTML-diff Project. https://code.google.com/p/html-diff/. Accessed 10 Oct 2014.

Charras, C., Lecroq, T. (2004). Handbook of exact string matching algorithms. http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf. Accessed 05 Sep 2014.

Chen, Y., Wei-Ying M., Hong-Jiang Z. (2003). Detecting Web Page Structure for Adaptive Viewing on Small Form Factor Devices. Accessed Apr 2015.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.331.8878&rep=rep1&type=pdf.

Cobena, G., Abiteboul, S., Marian, A. (2002). Detecting Changes in XML Documents. INRIA, Rocquencourt France, Columbia University/ N.Y.USA. Data Engineering, Proceedings. 17th International Conference. Accessed 10 Apr 2015.
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=994696.

Code Google project base (2007). Daisydiff. https://code.google.com/p/daisydiff/.
Accessed 10 Oct 2014.

CPU Benchmark. PassMark (1998). http://www.cpubenchmark.net/. Accessed Apr 2015.

Deng, C., Shipeng Y., Ji-Rong W., Wei-Ying M. (2003). VIPS: a Vision-based Page Segmentation Algorithm. Microsoft Research Asia. Technical Report MSR-TR-2003-79 http://research.microsoft.com/pubs/70027/tr-2003-79.pdf. Accessed Apr 2015.

Flesca, S., Masciari E. (2003).Efficient and effective Web change detection. Data and Knowledge Engineering 46 203–224. Rende, Italy Accessed Feb, 2015. http://www.sciencedirect.com/science/article/pii/S0169023X02002100.

Julian (2006). Comparing Strings: An Analysis of Diff Algorithms. http://www.somethinkodd.com/oddthinking/2006/01/16/comparing-strings-an-analysis-of-diff-algorithms. Accessed 10 Oct 2014.

Lecroq, T. (2007). Fast exact string matching algorithms. Information Processing Letters, vol. 102, issue 6, pp. 229-235.

Lim, S-J., Ng, Y-K. (2001). An Automated Change-Detection Algorithm for HTML Documents Based on Semantic Hierarchies. USA, Brigham Young University. Data Engineering, Proceedings. 17th International Conference http://ieeexplore.ieee.org/stamp/st amp.jsp?tp=&arnumber=914842. Accessed 20 Feb 2015.

MacKenzie, D., Eggert, P., Stallman, R. (1993). Comparing and merging files http://www.chemie.fu-berlin.de/chemnet/use/info/diff/diff_3.html. Accessed 10 Jan 2015.

Mikhaiel, R., Stroulia, E. (2005). Accurate and Efficient HTML differencing. University of Alberta, Edmonton, Canada. Software Technology and Engineering Practice. 13th IEEE International Workshop http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1691644. Accessed Nov 2014.

Pehlivan, Z., Ben-Saad, M., Gancarski, S. (2010). Vi-DIFF: Understanding Web Pages Changes. University P. and M. Curie Paris, France. Lecture Notes in Computer Science Volume 6261, pp 1-15

Salty Brine (2005). HTML Match http://www.htmlmatch.com. Accessed Apr 2015.

Sanka, A., Chamakura, S., Chakravarthy S. (2006). A dataflow approach to efficient change detection of HTML/XML documents in WebVigiL. USA, University of Texas at Arlington. Computer Networks, Volume 50, Issue 10, Pages 1547–1563.

Štěpánek, J., Šimková, M. (2012). Comparing web pages in terms of inner structure. Czech Republic, University of Hradec Králové. 2nd World Conference on Educational Technology Researches – WCETR2012. Procedia - Social and Behavioral Sciences 83, 458 – 462.

Yang, Y., Zhang, H. (2001). HTML Page Analysis Based on Visual Cues. Microsoft Research China. Proceedings Sixth International Conference on Document Analysis and Recognition http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=953909. Accessed Feb 2015.

Received July 31, 2015, accepted August 29, 2015