Advanced Information Retrieval from Web Pages

A. Vedeshin
Tallinn University of Technology
Ehitajate tee 5,
19086 Tallinn, Estonia
anton.vedeshin@ttu.ee

Abstract

A lightweight, web based with near to real-time speed algorithm is proposed in this work. It is able to retrieve main parts (menu, main text, header and footer) of a randomly selected web page entirely using CSS, JavaScript, frames, layers, images, etc. for retrieval. Moreover shortcomings of well-known modern algorithms for content retrieval from web pages are discussed in this proposal. The algorithm is useful for the improvement of existing: searching, content matching, summaries making, web graph calculation, and etc. engines. Moreover it is practical as a data provider for classification and data mining. The experimental results of a PHP realization of the algorithm showed near to real-time speed, 20-25% error rate for the multipurpose mode and less than 1% error rate for the specific mode.

Keywords: web information retrieval, information extraction from web

1 Introduction

The approach proposed in this work consists of very simple (lightweight) algorithm that could be used for a near real time processing of web pages (and even could be implemented on web scripting languages) and retrieve common web page parts like navigational menu, main text, header and footer.

While using complicated algorithms, it is vital to analyze not only pure HTML, but also other web technologies like CSS, JavaScript, frames and images. Although many other works omit analysis of these technologies – they often base decision making on information that is mainly carried by these technologies (for example boldness, size and style of the text) in order to distinguish relevant information from other.

As a result a lot of works do not reflect on other design structures of web pages, thus it could be a matter of wrong recognition of some web page types.

We will discuss several aspects of both well-known modern branches of algorithms for content retrieval from HTML page (in section 2) and propose an algorithm that is able to resolve various shortcomings of the previous approaches (in section 3). Experimental results are given in sections 4, followed by the conclusion.

2 Related work

There are at least two main modern branches of algorithms for content retrieval from HTML pages: methods based on DOM tree analysis [7, 12, 10], and methods based on HTML page visual representation [5, 10, 3, 11].

Each of the methods mentioned has its advantages and disadvantages to others. Nevertheless a lot of works have one weakness in common – they consider only pure HTML. In this section we provide a short overview and discussion about the algorithms from both of those branches.

Deng Cai et al. [5] proposed a Vision-based Page Segmentation Algorithm. VIPS uses visual representation of a web page with the DOM backend. Intuitively, it is clearly a good solution as people also see a web page not as a tree of HTML tags but in visual representation. On the other hand - using DOM tree at the backend still can cause several problems. First of all, considering some of HTML tags (`<table>`, `<tr>`, `<td>`, `<p>`) and images, other page elements made using JavaScript, CSS, frames, `<div>` tag could be left unrecognized. Moreover, even visual representation might vary from one browser to another. There are a lot of vivid graphics used on some web sites, so sometimes even a person can not recognize the segments of a web page at the first glance. The VIPS algorithm is dependent on web browser and seems to have one more weak side: two blocks of information could not be recognized if there is no big or clear separator between them.

Another work by Shian-Hua Lin and Jan-Ming Ho [12] provides the algorithm for informative content blocks discovery using entropy. They use only `<table>` tag for segmentation, which indicates the ability to recognize only web sites of one exact type.

The information retrieval algorithm based on DOM tree is proposed by Suhit Gupta et al.[7], which also cleans the content from unnecessary information like advertisement and images without any serious change in web site HTML layout. This algorithm absorbed the benefits of earlier works on information retrieval. In the beginning of the work it is claimed that conventional cleaning methods eliminate JavaScript, images, etc., and it is awful from the author's point of view. However the approach for getting over both JavaScript and images is not mentioned in the work itself and in the algorithm. The algorithm provides methods for removing links, empty tables and styles. The advertisement detection engine uses a list of common advertisement servers in order to remove the ads. That clearly indicates to the lack of universality of the algorithm.

The algorithm by Bernhard Krüpl et al. [11] tries to extract information from web pages not only through DOM tree analysis, but also using "visual cues". The algorithm is based on Mozilla boxes rendering ability, which is the slowest part of the algorithm as authors claim. Therefore, that the technique strongly depends on Mozilla engine. In addition, for "visual cues" detection, some OCR algorithm is used. As soon as the whole web document bitmap is to be processed, it tends to be very slow and complicated task.

Mikhail Ageev et al. [1] propose an algorithm, which is to some extent similar to the one described in the present work (although we also work with tokens of a web page, but they are analyzed in a different way). The idea is based on tokenizing the HTML code of a page. It conducts a search for similar files (similarity is based on navigational part of a page) and creates a cluster. The next step is to eliminate part of a page which is common for all the files in the cluster. A weak side of the method is inability of processing more complicated technologies than HTML. The results could be achieved if only many files (a cluster) are analysed.

In the algorithm suggested by Milo Kovačević [10], each HTML object's exact position on a screen is considered in order to recognize most common parts of a web page (such as header, footer, center, left and right menus). The author claims that "Web pages are designed for humans!", implying that analyzing only DOM structure of a web page could not be enough. However, the proposed algorithm has many limitations as size of the captured screen, ability to map structure information with visual one, skipping `<script>` tags, and also not considering layers, frames and style sheets. The rules used are based on a very common, but still not the only way of page layout (menu can be either on the left or right side of a web page). Nowadays web design industry creates incredibly impressive and some-times even shocking web site designs, where people at the first glance could not catch the idea about location of the menu, main text, etc.

The Algorithm by Radek Burget [3] has the same shortcomings as VIPS [5] with web pages rich of vivid graphics and visually complicated design. Visual representation of text is seen as being affected by HTML tags only.

3 The Proposed Methodology

The algorithm proposed in this work is able to process and accurately extract the main parts (navigational menu, main text, header and footer) from a modern web page or from a collection of web pages (web site) in order to save them in proper way (each link or paragraph/block separately) in a database for further use.

Main requirements for the algorithm are the following: 1) it should be quite fast, near to real time, because it supposed to work in web based information systems 2) simple and lightweight 3) universal in terms of ability to extract data from any web page suggested 4) effective 5) possibility to be implemented on web scripting languages (in our case PHP).

Fig. 1. The Proposed Algorithm Concept.

The algorithm is mainly based on knowledge of extremely common and, in a way, atomic design and information markup rules. Not only pure HTML, but also JavaScript, pictures and CSS are processed. If we look at HTML processing further, in addition to layout made with table tags, a lot of attention is paid to layers and frames.

The algorithm analyses within a scope of a single web page with maximum effort to distinguish main parts of a page. For the extraction of tags, parameters and texts rules based on Regular Expressions are used. These rules are divided into several groups (for menu, header, footer and main text); it is very easy to generate and add new patterns and groups; the algorithm will be able even to learn new patterns on the fly in future developments. Each rule has its own parameters and a degree of affection based on block's relevance.

There are several steps the algorithm goes through (see Fig. 1.). They are:

3.1 Information Request

The aim of this step is to download a single web page with everything included and store that in proper objects. First of all, the source page (main page) is processed. As usually there are many other web documents linked with the main file, in this case these files are also downloaded: frame files (like .php, .html, .inc, etc.), linked CSS files (.css), linked from inside of CSS files, JavaScript code files (.js), picture files (in case of using OCR).

Style sheets that are already inside the main page file (blocks between <style> tags) are considered as file content and also extracted and saved on equal terms with linked CSS files (further on the step of Extraction Preparation they will be eliminated).

Downloading and reviewing Frame files should be done before CSS and JavaScript files, because frame files could potentially include links to other CSS and JavaScript files. CSS files could also include links to other CSS files using @import url("mycss.css").

It is also essential to take into account that a source page of a given URL could be redirected in some way. There are at least three types of redirection from one page to another: header of the HTTP response, meta tag and JavaScript code.

At this step content of all the downloaded files are stored separately, but in Extraction Preparation phase some of them are: merged forever (frames), merged and then removed (JavaScript), not merged at all and removed if exist (CSS).

3.2 Extraction preparation

Firstly, downloaded files should be validated using a validator with the requirements similar to this algorithm's (it could be implemented on web scripting languages, in our case PHP, and run on common web servers; a good candidate could be TWINE¹).

Secondly, all the linked files should be embedded into the main file (except CSS), exactly into the same place where the link to that file was located.

There are lots of web pages, where navigational menu is made using JavaScript (for example with dropdown or popup submenus, etc.). Because of "on the fly" browser interpretation of JavaScript people could use such menus without any problem. However it is hardly possible for a retrieval engine to use immediately this kind of information – it should be interpreted. There is almost no one who actively raised this problem in application to information retrieval (except [14], where only an existence of a problem about JavaScript was mentioned).

After injection of linked files, JavaScript interpreter should run JavaScript code which at this step is entirely inside the main file's HTML code. The aim is to produce as much as possible pure HTML code from the JavaScript. As an interpreter could be used, for example, the "PHP JavaScript Interpreter"² or J4P5³.

The next step is to perform cleansing - trim spaces (if they are not single), line breaks, tabulation between HTML tags and within text, some unused JavaScript parts and style sheets etc., in order to convert the entire file into one long single line (this method is also used in many web document compacting applications). This allows giving an unique address (fix a location relatively the beginning of a web page) for each html tag, so it can be referred to as entry point.

3.3 Data extraction

After extraction preparation, the web page is converted into a single line of text. All the variety of data is being extracted. All "enclosing" tags (like <div></div>, but not like) considered as *text holder blocks*.

¹ TWINE Project – <http://twineproject.sourceforge.net>

² PHP JavaScript Interpreter – <http://phpjs.berlios.de>

³ Java Script for PHP5 – <http://j4p5.sourceforge.net>

Each such block extracted and stored into *block object* has its own properties. Some of them are assigned during the extraction.

Text holder blocks and their properties are the key features of the whole algorithm. This aspect makes it to be similar to the methods used in other works [1, 4, 5]. However, in our work some special properties or metrics of the block and their interpretation are the whole show.

All types of tags are extracted one by one. Here is the list of parameters being extracted. *Entry point* – shows the number of symbols from the beginning of a web page to the symbol where given token (block) starts. *Pure all text* – is the same as *pure own text*, but with length of insider blocks' texts. *Parent block* – is an identification number of *parent block* with the same type as a given block (if we imagine a hierarchy of blocks, this property will point one step upward for the nearest block with the same type; is computed on extraction level). *Global parent* – is the same as *parent block* but refers to any block one level upward in the hierarchy. *Depth* – is an inheritance level in hierarchy of blocks of the same type (within own type). *Global depth* – is the same as *depth*, but shows the level in the hierarchy of all the blocks (not within only own type). *Block length* – is length of *html text* (it is useful, for example, for returning of *entry point* which is right after the given block). *Pure own text extracted* – is extracted *pure own text* for real use. *Width* – is extracted from different sources (HTML, CSS, inline CSS etc.), it is an approximate width of a block. *Height* – is extracted from different sources (HTML, CSS, inline CSS, etc.) approximate height of a block. *Weight* – is evaluated through analysis (depends on many other parameters); it is the main source of recognition. There are separate weight parameters for each logical unit of the page. There is own *weight holder* for navigational menu, main text, header and footer, so they could be compared. *Font size* – is size of *pure own text* font. *Length to next* – is used in hyperlinks (shows number of symbols from *entry point* right after the block to the *entry point* of the next block of the same type).

Some parameters are extracted from CSS, such as *width* and *height* (also text size for *pure own text* could be extracted). There are at least three sources of such information: inline HTML, inline CSS, inside CSS which is rather linked file or between <style> tags in the <head> or in other places of the main document.

Extra texts' and blocks' parameters could be extracted in the same manner. Optionally it is useful to extract text from images using OCR. As a candidate to OCR engine that will meet our algorithm's requirements is phpOCR⁴. After OCR recognizes needed images (if we deal with image menu) texts are compared to captions under the images ("alt" attribute of the tag). Width and height are taken from the image's height and width; *entry point* is the position of tag. Also it is possible to declare tag as information tag which makes the algorithm to retrieve and recognize all pictures. In some existing retrieval algorithms [11] OCR engine is also used, but in order to recognize the entire page (actually the whole algorithm is based on OCR), what is extremely time and computational power demanding.

3.4 Analysis

Analysis is carried out through iterations over the *block array* each time on different types of blocks. First of all, blocks are sorted in *blocks array* by *entry point*. Then some corrections are added to block parameters in order to prevent possible extraction algorithm deviations and side effects, so that blocks will be ready for further analysis and recognition. After that, *global parent*, *global depth*, *pure own text* and *pure all text* are found, analyzed and extracted. *Pure all text* analysis and extraction is similar to *pure own text* extraction, but with recursive inner block's *pure own texts* gathering. Position of *text holder blocks* relatively to the whole page is calculated in *entry point* (text) equivalent.

Different types of statistics are gathered: number of blocks for each block type represented in a web page; mean value of pure own text per block; blocks that are in the first quarter of a web page; blocks that are in the middle of a web page; blocks that are in the last quarter of a web page; mean value of widths and heights of blocks; mean value of depth of hierarchy for leafs of blocks tree; percent rate for text amount per block type.

3.5 Recognition

Knowledge about the web page parts could be gathered rather manually or automatically from different sources [2] and then put into the rules. Example of the rule: "There is a chain of blocks that contain links, each block in the chain has the same *Length to Next*"; parameters: inaccuracy < 3 symbols; degree of affection: 95%; decision basis: statistics. It is possible to create groups of such rules; each group will correspond to some web page part, for example main text or menu.

⁴ phpOCR – <http://www.phpclasses.org/browse/package/2874.html>

Then tuning parameters and degree of affection rather manually or using machine learning leads us to the possibility to extract all those web page parts.

4 Applications of the algorithm

Several fields would benefit from using the algorithm. As the main application of the algorithm could be considered an improvement of existing web and web image search, content matching, summaries making, web graph calculation and etc. engines. Moreover it is very useful as a data provider for classification and data mining [13], especially when information is going to be taken from web databases and searching engines.

There is also an advantage for business field, for instance CMS (Content Management System) providers could make available intelligent engine to simply transfer old client data to a content management system database for a straight-away use.

In addition design companies would benefit, from providing a “fitting-room” for their customers where one can “try on” different web design templates (so a customer can put his/her old web page URL check how the information might look like with different designs).

This algorithm could be used for caching and gathering information from the web (for example, companies could entirely retrieve databases from the web, like product and business contacts catalogs, etc.)

Finally, obtaining a web page template seems to be easier. It is possible just to replace old information of existing web page with a new one, saving the design.

Contributions for new applications of the algorithm are welcome.

5 Empirical Evaluation

The algorithm was implemented in object-oriented PHP, and has a framework style (it is possible to easily tune the algorithm for specific purposes and turn on/off some routines or use modules in other applications). Web based version is available upon request for benchmarking and research purposes.

There are several tests done using the proposed algorithm, both in multipurpose and specific [13] modes. Also the speed of the algorithm was measured and roughly estimated. Small size web server was used for tests.

The aim of the experiment (multipurpose mode) below (see also Fig.2.) is to investigate how the algorithm works in real world scenario with a randomly selected web page. A web page was tested to retrieve main parts using the proposed approach.

5.1 Examples

The source for the example was <http://www.computer.org/tkde/>. Menus on this web site are done using JavaScript. Furthermore, JavaScript is linked from external .js files, thus if one just clicks on “View Source” in a browser, web page menus that are generated/interpreted by a browser could not be seen in HTML source. The algorithm proposed in this work was able to cope with such pages. See Fig.2. for the details about the extraction.

Fig. 2. Extraction from <http://www.computer.org/tkde/>. The web page before the extraction is shown on the left side of the figure, all JavaScript menus are active; extracted parts of the web page are shown in blocks on the right side of a figure. They are: 1- navigational menu, 1a and 1b- submenu, 2- menu, 3- main text, 4- header, 5- footer; (pictures were eliminated from extracted parts for demonstration clearness).

6 Conclusion

The problem of an automatic web page parts extraction was considered in this work. After the literature review and a discussion about several aspects of well-known modern branches of algorithms for content retrieval from HTML page, an algorithm was proposed for resolving various shortcomings of the previous approaches. All the variety of HTML tags, CSS, JavaScript, Frames and Images (with the support of OCR) are analyzed and used in the retrieval process. The algorithm was implemented in object-oriented scripting language PHP and has a framework style; source is available upon request for benchmarking and research purposes.

The presented result demonstrated that main parts of a web page (navigational menus, main text, header and footer) are precisely retrieved, even in the case of multiple menus or menus using JavaScript.

What concerns the error rate, for multipurpose mode it is about 20-25% of wrong recognition (after the future improvements in decision making part of recognition module and development of learning module the accuracy will be increased incredibly), for specific [13] modes it is less than 1%. According to the rough speed tests (even on a small web server) near real-time speed of the algorithm was achieved, less than 10 seconds for the entire job per one complicated web page was needed in the multipurpose mode; less than 15 seconds were needed per 100 thousands words on one HTML page in the specific mode.

7 Acknowledgments

I gratefully acknowledge Innar Liiv for stimulating discussions and supervising.

8 References

- [1] Ageev, M., Vershinnikov, I., Dobrov, B., (2005). Automatic Extraction of Significant Part of Web Pages for Informational Retrieval. Proceedings of the RCDL'2005.
- [2] Bernard, L. M. (2003). Criteria for optimal web design (designing for usability), Software Usability Research Laboratory
- [3] Burget, B. (2005). Visual HTML Document Modeling for Information Extraction. In Proceedings of RAWs 2005, pp. 17-24.
- [4] Buyukkokten, O., Garcia-Molina, H., Paepcke, A. (2001). "Seeing the Whole in Parts: Text Summarization for Web Browsing on Handheld Devices". In Proceedings of the 10th international conference on World Wide Web.
- [5] Cai, D., Yu, S., Wen, J., Ma, W. (2003). VIPS: a Vision-based Page Segmentation Algorithm. Microsoft Technical Report (MSR-TR-2003-79).
- [6] Crescenzi, V., Mecca, G., Merialdo, P. (2001). Towards Automatic Data Extraction from Large Web Sites. Proceedings of the 27th VLDB Conference.
- [7] Gupta, S., Kaiser, G., Neistadt, D., Grimm, P. (2003) DOM-based Content Extraction of HTML Documents. In WWW2003 proceedings of the 12 Web Conference.
- [8] Hu, Y., Xin, G., Song, R., Hu, G., Shi, S., Cao, Y., Li, H. (2005). Title Extraction from Bodies of HTML Documents and its Application to Web Page Retrieval. The 28th Annual International ACM SIGIR Conference (SIGIR'2005).
- [9] Kim, S., Zhang, B. (2003). Genetic Mining of HTML Structures for Effective Web-Documents Retrieval. Applied Intelligence 18, 243–256, 2003.
- [10] Kovačević, M., Diligenti, M., Gori, M., Milutinović, V. (2003). Searching for web information more efficiently using presentational layout analysis. International Journal of Electronic Business 2003 - Vol. 1, No.3 pp. 310-326.
- [11] Krüpl, B., Herzog, M., Gatterbauer, W. (2005). Using Visual Cues for Extraction of Tabular Data from Arbitrary HTML Documents. In Proceedings of the 14th international World Wide Web Conference.
- [12] Lin, S., Ho, J. (2002). Discovering Informative Content Blocks from Web Documents. Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining.
- [13] Liiv, I., Vedeshin, A., Täks, E. (2007). Visualization and Structure Analysis of Legislative Acts: A Case Study on the Law of Obligations. In Proceedings of The 11th International Conference on Artificial Intelligence and Law (ICAIL'07), pp.189-190.
- [14] Myllymaki, J. (2001). Effective Web data extraction with standard XML technologies. In Proceedings of the Tenth International World Wide Web Conference, WWW10, pp 689-696.