P.Sivakumar, Dr. R.M.S Parvathi / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue 2,Mar-Apr 2012, pp.719-722

Mainly significant Content Mining of Entire Web Page

P.Sivakumar
Department of computer science and engineering,
KSR College of Engineering,
Namakkal, Tamilnadu, India

Dr. R.M.S Parvathi
Department of Computer science and Engineering
Sengunthar College of Engineering,
Tamilnadu, India.

Abstract- User explore for the necessary information with search engines. Search engines crawl and index web pages according to their informative content. User is attracted only in the useful contents and not in non-informative content blocks. Web pages often contain navigation sidebars, advertisements, search blocks, copyright notices, etc which are not content blocks. The information contained in these non-content blocks can harm web mining. So having an algorithm to extracts only most important content could help better quality on web page indexing. Almost all algorithms have been proposed are tag dependent means they could only look for primary content among specific tags such as <TABLE> or <DIV>.The proposed technique is tag free and has two phases to achieve the extraction work. primary it transform contribution DOM tree obtain from input HTML detailed web page into a block tree based on their visual representation and DOM structure in a way that on every node it will have specification vector, then it traverses the obtained small block tree to find main block having dominant computed value in comparison with other block nodes based on its requirement vector values. This introduce technique doesn't have any knowledge phases and can find educational content on any casual input complete web page.

Keywords
Web mining, Noise elimination, Informative content, Information retrieval, Information extraction

1. INTRODUCTION

A web page structure and layout varies depend on different content type it will represent or the tastes of designer styling its content. Thereby main content position or the main tag containing main content differs in variety of websites. Even there might be some content in page view that are besides each other but actually in DOM tree they are not in the same level and same parents, so finding the main content in this area that doesn't follow any specific rules for arranging and positioning elements needs complicated and costly algorithms. Algorithms that could simulate a user visiting a website, in high probability could find informative content as result because in most cases actual users in internet could find the area of the main content. But which specifications and structures could help an algorithm to find main content?

The VIPS algorithm uses obtained content structure and tries to simulate how actual user finds a main content by blocking the page based on structure and visual delimiters. The blocking result is satisfactory but the algorithm does many loops to reach its desire granularity. Content structure and tries to simulate how actual user finds a main content by blocking the page based on structure and visual delimiters. The blocking result is satisfactory but the algorithm does many loops to reach its desire granularity.

CE [7] considers all detailed pages of a website as pages with the same class. It runs a learning phase with two or more pages as its input and finds the blocks that their pattern repeats between input pages and marks them as non-informative blocks then stores them in storage. These non-informative blocks are mostly copyright information, header, footer, sidebars and navigation links. Then when we use CE algorithm in actual world it first eliminate non informative patterns from the structure of its input pages based on the stored patterns in its storage for specific class of input pages. Finally from the remaining blocks in the page it will return the text of block containing the most text length. CE needs a learning phase so it couldn't extract the main content from random one input web page.

FE [7] extracts the text content of a block that has the most probability of having text so it will work fine in web pages that text content of main content dominates other types of content. In addition FE could return just one block of the main content, so [7] proposed K-FE that returns k blocks with high probability of having the main content. Algorithm steps of K-FE and FE are the same except the last part. In K-FE the algorithm

final section, sorts the blocks depends on their probability then it uses k-means clustering and takes high probability clusters.So the proposed paper intends to introduce an algorithm which could extract main content that is not necessarily the dominant content and without any learning phase, with one random page and by using visual cues to simulate user page visit and block the page based on it and gains higher precision.

2. PROPOSED ALGORITHM

The proposed algorithm called Visual Gathering Extractor (VGE). It gets DOM tree of input web page as its input and returns the informative content block as its output.

Algorithm (Visual Gathering Extractor)
Step 1: Read HTML File
Step 2: Convert DOM File of HTML (consider as a input File)
Step 3: Read Function (GetBlockTree (DOMTree ,outGeneralParameters, outBlockTree)
{Final computation of the average of Computations parameters}
Step 4: Check Function (FindMainBlocks (BlockTree ,GeneralParameters, outCandidate Blocks,outMainBlocks)
{ MainBlcok }(output)
Step 5: PrimaryContentBlock (output)

The third step of the algorithm use GetBlockTree sub-algorithm and sends the DOM tree as its input. The GetBlockTree subalgorithm recursively traverses the input DOM tree in pre-order and returns block tree and general parameters as its output. Each node in block tree clusters one or more nodes from DOM tree so the number of elements in block tree is significantly lower than the number of nodes in DOM tree. Each node in block tree has specification vector that we will use it to find the main block later. General parameters contain general total value of specifications for text. Link density, width and height and we can use these values to compute average values before starting to find the main block.

2.1 Constructing Block Tree

Here, process of block tree construction through GetBlockTree algorithm. This algorithm recursively loops through input DOM tree and it produces block tree as final result and meanwhile it flags the blocks such as comment blocks which their patterns repeats in unordinary manner and they should not consider as the main content. In addition besides constructing the block tree, the algorithm append the text of child blocks to their parent so we will have text manipulation just in this section without any need for additional loop for text computation later on the block tree. This subalgorithm contains sub-sections which will introduce in the next parts.

Algorithm: GetBlockFunction(Visual Gathering Extractor cont)
Step 1: GeneralParametters, BlockTree(Input)
Step 2: Updated Root element, Updated GeneralParametters, Updated BlockTree(Output)
Step 3: Recursive (Type)
Step 4:

1) PatternCache ←Null
2) DetectedBlock(RootElement,out BlockTree)
3) AddText(RootElement)
4) IfRootElement has child **then**
5) **Foreach** Child in RootElement's children **do**
6) **If CheckElementIs Valid(**Child) then
   a. **GetBlockTree**(Child,out enteralParametters,outBlockTree)
   b. **Patten Verifier(**RootElement,Out patternCache)
   c. **TextManipulation(**Child**)**
7) **Else**

Step 5:**PattenVerifier** (RootElement ,outPattenCache)
Step 6:**FinalElementManipulation** (RootElement, outGentralParamentter)

2.1.1 Why Block

First I define the concept of block in this paper. A block is a family of elements with same visual styles and order in DOM tree and their conceptual purpose to appear in the page are the same. The algorithm proposed in this paper tries to first divide the page into some blocks. One thing that we should consider here is why we should make block tree in content extraction algorithms? Or further why we should have block? I answer this question with an example. Consider we want to remove the navigation links in sidebar, to accomplish this target we should eliminate the parts of the page which have high link density. If you don't use block then the algorithm will consider each individual node element as a page part and removes all the links in the page even if the links are in the main content, because the link density on each individual link is high. But when you are using block, the algorithm would decide on blocks instead of each individual DOM node. So in our example algorithm will remove the navigation links in side bars and it doesn't remove the links in the main content.

2.1.3 DetectBlock Method

This method will specify either its input node should form a new block or it should use the block of its parent node in DOM tree. This method makes this decision by checking if the visual distance of current node in contrast with its parent is more than a threshold value or not. If its distance was not valid then it would make a new block and the algorithm flags the current DOM node as the parent of new block. If the distance was valid it means we should only add the current element to the block of its DOM parent node. The visual distance is the number of differences on their visual styles which are important in this algorithm, such as width, height, font-size, background color, top and left. Actually these visual styles are the parts of a block specification vector.

2.1.4 AddText Method.

This method will compute the immediate text of current node and immediate link text of current node and add them to the relevant attribute of its block. Other child nodes could add their text to these texts later. Figure 1 shows the immediate text and immediate link text of a DOM node. The text length and link text length are some of the specifications in a block specification vector.

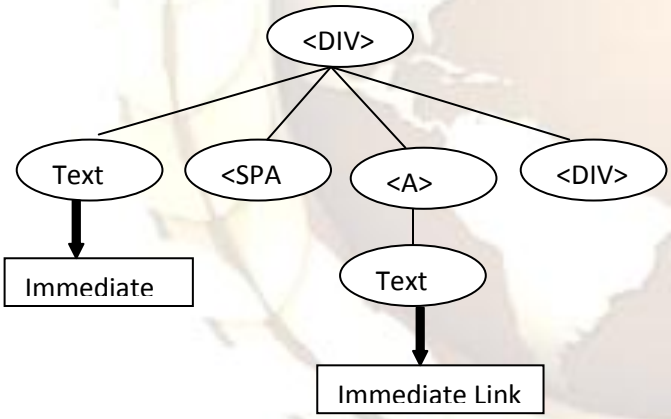

Figure 1. Immediate text and Immediate Link text for a DOM Element

2.1.5 Children Manipulations

For each DOM children of input element first we do the recursive GetBlockTree algorithm. The recursive algorithm make sub block tree for each child node and their subordinating child nodes. Then PatternVerifier algorithm runs for the child block with additional input called PatterCache. PatternVerifier checks if the current block pattern has any same pattern among its siblings. To accomplish this task, it uses PatternCache which contains unique pattern of the current block siblings (Figure 2). No of repetitive patterns is a negative specification in a block specification vector.

Algorithm: PattenVerifier Function (Visual Gathering Extractor cont)

Step 1: Elements, Pattern catch holds its sibling block Patterns (Input)

Step 2: Upatedted Number of repetitive block pattern in the parent of input element and update Pattern catches (output)

Step 3: Block Pattern set Base on root tag types its children blocks

Step 4: Pattern Cache Unique Patterns

Step 5: if input element block root **and**

Step 6: **foreach** pattern in PatternCache **do**

Step 7: if the pattern Block of input element is equal with current pattern **then**

{Ingress the number of reparative Pattern in the parent of input element} break

{ Add Block Pattern of input element to pattern cache if there wasn't any equal pattern in the for each loop for it }

Step 8:end

The last process which operates the block of child node is TextManipulation method that add the text of child node to its parent if the link density of child node was ok depend on a threshold. Figure 6 depicts more detail for TextManipulation method.

Algorithm: FindMainBlock Function (Visual Gathering Extractor cont)

Step 1: Block Tree, Gentral parameters, CandiatateBlocks, MainBlock (Input)

Step 2: Updated candidateBlocks,MainBlock(Output)

Step 3: Recursive (Type)

Step 4: **Foreach** ChildBlock in the root of BlockTree **do**

{ FindMainBlock (BlockTree of ChildBlock, GeneralParameters, CandidateBlocks, MainBlock)

IfGeneralProperties of ChildBlock based on GeneralParameters was ok **then**

{Compute FactorValue for the ChildBlock based on following formula }

FactorValue = Text length + width+ Average font size+Bottom

Link density + (Number of repetitive Patten * Top

{ **Add ChildBlock** to CandidateBlock list.}

If MainBlcok is Null OR

FactorValue→Factorvalue of Main Block **then**

MainBlock←ChildBlock

Step 5: End

Finally FindMainBlock sub-algorithm returns The Main Block which has the highest FactorValue and the content of this block represented in the output.

3. EXPERIMENTAL RESULTS

In this section we evaluate our algorithm with the dataset in [7] which contains over 5000 pages and compare it with K-FE [7] (because it seems to have better result in comparison with other Algorithms), in block-level based on Block-Precision, Block- Recall and Block-F-Measure factors which are introduced in [7], [3] (Table 1).

Web sites	Address	b-F Measure of VCE	b-F ReCall of VCE	b-F Prec of VCE	b-F Measure of K-FE	b-F ReCall of K-FE	b-F Prec of K-FE
ABC	abcnews.com	1	1	1	1	1	1
BBC	bbcnews.com	0.98	1	0.98	1	1	1
CBS	cbsnews.com	1	1	1	0.978	0.77	0.98
CNN	cnn.com	1	1	1	0.98	0.98	0.98
FOX23	fox23news.com	1	1	1	1	1	1
MSNBS	Msnbs.com	1	1	1	0.95	1	0.92

Table 1. Block-Level comparison between VCE (proposed algorithm in this paper) and K-FeatureExtracor [7]

4. Conclusion

We proposed an algorithm called VCE here, which could extract the main content from a random detailed web page. As we saw in section 3 this algorithm gains higher b-Precision, b-Recall and b-F-measure so we gain higher precision in extracted content. The VCE algorithm is not dependant on any tag type and it just has an iteration to block its input page while it doesn't have any learning phase. Furthermore it could detect and eliminate comments from the extracted content.

5. REFERENCES

[1] Alberto H. F. Laender, Berthier A. Ribeiro-Neto, Altigran Soares da Silva, Juliana S. Teixeira: A Brief Survey of Web Data Extraction Tools. SIGMOD Record 31(2): 84-93 (2002).

[2] Deng Cai, Shipeng Yu, Ji-Rong Wen and Wei-Ying Ma: Extracting Content Structure for Web Pages based on Visual Representation. In: The Fifth Asia Pacific Web Conference (APWeb2003), Springer Lecture Notes in Computer Science (2003).

[3] Deng Cai, Xiaofei He, Ji-Rong Wen and Wei-Ying Ma: Block Level Link Analysis. In: Proc. 2004 Int. Conf. on Research and Development in Information Retrieval (SIGIR'04), Sheffield, UK (July 2004).

[4] Jeff Pasternack, Dan Roth: Extracting Article Text from the Web with Maximum Subsequence Segmentation. In: www '09: proceedings of the 18th international conference on World Wide Web, New York, ny, usa, acm, 971—980 (2009).

[5] Lakshmish Ramaswamy, Arun Iyengar, Ling Liu and Fred Douglis: Automatic Detection of Fragments in Dynamically Generated Web Pages. In: 13th International Conference on the World Wide Web (WWW-2004), pp. 443-454 (2004).

[6] Lan Yi, Bing Liu, and Xiao-Li Li: Eliminating Noisy Information in Web Pages for Data Mining. In: Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD-2003), Washington, DC, USA, August 24 – 27 (2003).

[7] Sandip Debnath, Prasenjit Mitra, Nirmal Pal, C. Lee Giles: Automatic Identification of Informative Sections of Web Pages. In: IEEE Transactions on Knowledge and Data Engineering, 17(9): 1233-1246 (2005).

[8] Shian-Hua Lin and Jan-Ming Ho: Discovering informative content blocks from web documents. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pages 588–593 (2002).

[9] Suhit Gupta, Gail Kaiser, David Neistadt, Peter Grimm: DOMbased Content Extraction of HTML Documents. In: 12th International World Wide Web Conference, 12th International World Wide Web Conference (May 2003).