Detecting and Monitoring Dynamic Content Blocks of a Web Page by Merging its Historical Versions

Shu Tang¹, Zhicheng Dou², Xing Xie², and Jun He³

¹, ³Renmin University of China ²Microsoft Research

¹tangshu927@gmail.com, ²{zhichdou, xingx}@microsoft.com, ³hejun@ruc.edu.cn

ABSTRACT

Nowadays, most people and organizations with websites design their own homepages to facilitate readers’ obtaining information about the entity in question. The content of these homepages is usually divided into different areas, each of which only contains information about one specific aspect. Some of these areas’ pieces of information are updated over time. It would be very convenient for browsers of the site if we can automatically detect dynamic information areas and trace their content. Previous studies have paid little attention to homepages, and have not made full use of pages’ historical information and conducted exploration in the temporal line. We build a merged tree from one page’s historical versions. We then use it to detect dynamic content blocks, and extract and trace their content. Experimental results based on a large number of Web pages from diverse domains show that the proposed technique is able to extract the dynamic content blocks with a high level of accuracy.

1. INTRODUCTION

Information in homepages is often displayed as several content blocks, as shown in the boxes in Figure 1 which is a screenshot of the CNN homepage. There are several common characteristics between these information blocks in this figure. Each block includes content talking about only one aspect. Content in the figure displays as images and news hyperlinks. Different blocks’ content seldom overlap. Additionally, some of these blocks’ contents update over time while others remain the same, as shown in the dashed boxes (updated content) and solid box (unchanged content) in this figure. We call them as dynamic blocks and static blocks respectively. Furthermore, we assume that readers are more likely to take interest in the dynamic blocks’ content while seldom checking the static ones.

Based on these observations, we aim to automatically detect the dynamic blocks by comparing the page’s historical information and help readers to monitor such content. Works on this process have been explored in great depth by numerous researchers. But none of them leverage multiple versions of a page, nor do they focus on detecting and monitoring dynamic blocks in the homepages’ temporal line.

Inspired by the existence of temporal changes, we aggregate historical versions in a temporal line and ultimately summarize the dynamic areas from the dynamic text or image. The temporal structure fluctuation of HTML pages is integrated together, providing a more general matching reference, allowing the corresponding blocks in the future pages to be found with greater accuracy.

More specifically, we first build an aggregated tree which merges the historical DOM trees of the page. Then we use this tree to detect the dynamic content blocks. After the dynamic blocks’ detection, we can recommend several content blocks to readers. For each block that the user specifies, it is marked in the aggregated tree. Then, based on this tree, the specific block’s content could be monitored by a designed strategy.

2. RELATED WORK

The most popular technique of wrapper is to identify the repeat pattern using a tree alignment algorithm. Several adapted tree alignment methods have been proposed. One such method is RTDM [2] (restrict top-down mapping), which works by adding some restrictions on a traditional tree alignment method, detecting the template among multiple pages. Xia et al. [3] relax the restrictions of RTDM and get better results on forum and blog pages. DEPTA [4] and WPC [7] all focus on products list pages, identifying repeat modes in a single page.

RoadRunner [1] works by solving the mismatches between HTML terms to identify the repeat pattern but not tree
alignment. Other particular features either in the HTML terms level or the vision level are studied in [3,13]. They are used to speculate news content or partition the news page.

None of the methods mentioned above leverage multiple versions of a page, nor do they focus on detecting and monitoring dynamic blocks in the homepages’ temporal line.

3. PROBLEM STATEMENT AND SYSTEM OVERVIEW

Since we aim to use a homepage’s historical information to detect and trace future dynamic information, the problem could be depicted as follows: For a given homepage, we have its m historical versions \( <v_1,v_2,\ldots,v_m> \), which are used to build the merged tree, and n versions \( <v'_1,v'_2,\ldots,v'_n> \) whose contents need to be traced and extracted. Let \( t_1,t_2,\ldots \) denote time. \( v_x \) and \( v'_x \) denote the HTML codes we download from the web browser at time \( t_x \) or \( t'_x \). Without loss of generality, we assume that \( t'_0 > t'_1 > \ldots > t'_1 > t_m > t_{m-1} > \ldots > t_1 \). Here “\( > \)” means the order of the time, for example, \( t'_1 > t_m \) denotes time \( t'_1 \) is after time \( t_m \).

Our targets are as follows: First, we integrate all the historical versions into a merged tree, which is called HMT (Historical Merged Tree). Second, by comparing information from different versions, we can detect the dynamic content blocks. Third, for a particular block specified on the HMT, without specifying anything in the current page, we can extract the block’s content by comparing the current page with the HMT. The flowchart is shown in Figure 2.

In the following sections, \( T_x \) denotes the basic tree built from \( v_x \) (one historical version). \( HMT_m \) denotes the HMT built from \( <T_1,T_2,\ldots,T_m> \).

Note that the logical structure of \( T_x \) and \( HMT_m \) is the same. The difference between them is that \( T_x \) only contains one historical version, whereas \( HMT_m \) may contain multiple versions.

4. OUR APPROACH

4.1 Integrating historical versions to an HMT

We begin by building a basic tree \( T_x \) for each page version \( v_x \). In \( T_x \), we add several extra attributes to the traditional DOM node. Three of them are introduced here: TagName, Index, TagAttributes. The TagName denotes the name of an HTML element. The Index denotes the node’s unique key. TagAttributes is a HashTable, each key-value pair of which records one attribute’s name, and the corresponding values of every version in this node. If the node does not contain any attributes, then we add a “vID” attribute to it, whose value is comprised of the versions that contains this node. Note that here the leaf node can only be a TEXT node or an IMG node, where “TEXT” and “IMG” both represent the node’s TagName. The “text” attribute is only contained in the TagAttributes of the TEXT node, and the “src” attribute is only contained in the TagAttributes of the IMG node.

Assumed that we already have \( n \) basic trees \( (T_1,T_2,\ldots,T_n) \), we create the HMT in terms of algorithm A. Basically, we iteratively merge the next version into the existing HMT. The merge method contains two steps: The first step is to adopt the RTDM (restrict top-down mapping) [2] algorithm in [8] to obtain the optimal matching structure of two trees. Second, we adopt the following two strategies to merge \( T_i \) to \( HMT_{i-1} \), and get \( HMT_i \). For a node \( n \) in \( T_i \), 1). If there is a node \( n_1 \) in \( HMT_{i-1} \) which is matched with \( n \), we just add \( n \)’s attributes to the TagAttributes of \( n_1 \). 2). If there are not any nodes in \( HMT_{i-1} \) that are matched with \( n \), we insert \( n \) to \( HMT_{i-1} \) while keeping the order of hierarchy and siblings. Ultimately, there may be some nodes in \( HMT_i \) that do not contain any information of \( T_i \). These are the nodes that exist in the version of \( \{T_1,\ldots,T_{i-1}\} \), but are deleted in the version of \( T_i \). An example of the merged tree is shown in Figure 3.

Figure 3: An example HMT

![Algorithm 1: CreateMergeTree](image)

```
Algorithm 1: CreateMergeTree
Input: BasicTree[=T1,T2,...,Tn]
Output: MergeTree[HMT]
1 HMT_i:=create(T_i);
2 for i \( \leftrightarrow \) 2 to n do
3 | HMT_i = merge(HMT_{i-1},T_i) ;
4 end
5 return HMT_n;
```

There are several differences between RTDM [2] and our method. In ours, let \( M \) be a mapping between two trees [3]. For each pair \((n_x,n_y)\) in \( M \), \( n_x \) and \( n_y \) are with the same TagName, which means there are only node insertion and removal operation, but not nodes replacement in our method. The mapping cost is given by \( c = S_x + I_y + D_x + M_x \), where \( S \) denotes a subset of pairs \((n_x,n_y)\) in \( M \) with distinct TagAttributes, and \( p \) is the cost assigned to the
reduction of TagAttributes. \( v \) is defined as follows: for each pair \((n_x, n_y) \in M, v = 1 - \frac{N_x}{N_{sum}} \cdot \xi \). Here \( N_x \) denotes the number of versions \( n_x \) contains. \( N_{sum} \) denotes the total number of versions already used in the merging step, which is \((i - 1)\) here. \( \xi \) is used to confirm \( v \) is far less than \( p \). \( M_v \) is added to help us to choose one of the minimum cost mapping that contains more previous aligned versions. Note that \( p \) is far less than \( q \) and \( r \), which are respectively the costs assigned to the insertion and removal operations.

4.2 Detecting dynamic content blocks

After finishing the final HMT, we adopt two steps to detect dynamic content blocks. First, we detect dynamic nodes. During the merging step, we merge the nodes which come from multiple historical versions and match with each other. So a node is marked as dynamic if any version is missing in it. What’s more, a node will also be marked as dynamic if it is a leaf node, and its “text” or “src” attribute has different values within the historical versions. Other nodes are marked as static. Second, we detect dynamic content blocks. Assumed that each block in one page can be represented as a sub-tree. For a certain internal node \( n_x \) in HMT, we define it as a block – level – dynamic node, if and only if it meets the following two conditions. 1) \( n_x \) is a static node; 2) \( L_d / L_s \) is larger than a predefined threshold. Here \( L_d \) denotes the sum of text’s length in \( n_x \)’s children which are marked as dynamic or block – level – dynamic, and \( L_s \) denotes the total text length of \( n_x \). By the time constraint, we did not deeply explore the threshold’s value. We will do it in the future. The reason that we did not use the number of nodes to compute the rate is that, the text length of different nodes may be largely different, which would result in large instability of the threshold among different pages.

The sub-trees rooted by the block – level – dynamic nodes serve as dynamic content blocks.

4.3 Dynamic block monitoring and extraction

We assume that one user is usually interested in one or a few dynamic blocks on a page. After detecting all the dynamic content blocks from HMT, the user can simply specify the blocks she wants to monitor. With \( b_v \) denoting a particular block specified by the user, in a new page version, we directly use the same method as what we adopt in the tree building step. The difference is that we just match this version with the HMT, but not merge them. Ultimately, the sub-tree rooted by the node which is aligned with the root of \( b_v \) represents the corresponding block.

Note that, if there is not any node that is aligned with the root of \( b_v \), it may be that there is not \( b_v \) in this new page. However, there is another possibility that the root of \( b_v \) in this new page is aligned with other nodes in HMT, which will result in missing extraction.

5. EVALUATION

In this section, we mainly test the extraction accuracy. We will test the performance of data blocks detection in the future. The experiment data is a set of 69 homepages manually selected from the top 100 websites ranked by Alexa.com. Most of them change frequently. We crawl the pages every 5 hours for 50 versions \( \langle v_1, v_2, \ldots, v_{50}\rangle \). For each URL in the data set, a labeler is asked to freely annotate three content blocks that she likes to follow. All of the labelers are encouraged to label the three blocks in different ranges. The labeler annotates three blocks in the first version, and tracks and annotates corresponding blocks in the later versions. For the versions the block does not exist in, the labeler tags it as missing.

5.1 Experiment with page changes

We use Figure 4 to illustrate two kinds of changes happened over time in homepages: structure changes and content changes. Examples of structure changes are a piece of breaking news is inserted or deleted, or rows are added or deleted from a table. Content changes are the updating of text or image. In this figure, we compute the similarity between \( v_1 \) and \( v_v \) \((x \text{ from 2 to 50})\). We use \( HMT_{1x} \) to denote the tree merged by \( v_1 \) and \( v_v \). The structure similarity is computed as follows: the number of the nodes that both exist in \( v_1 \) and \( v_v \) divided by the number of total nodes in \( HMT_{1x} \). And text similarity is calculated as follows: the number of the leaf nodes whose content from \( v_1 \) and \( v_v \) are the same divided by the total number of the leaf nodes in \( HMT_{1x} \). Figure 5 shows that with the increase of the time interval, the structure similarity changes periodically to some extent. The possible reason may be that some web editors change their pages’ structure periodically. Besides, the text similarity goes down first and then remains stable. The static content may be blocks’ titles and other illustrations that rarely change.

5.2 Experiment with extraction accuracy

We experiment with building the merged tree using different numbers of historical versions (lengths of history). In

![Figure 4: The impact of time interval between two versions on the two versions' similarity](image)

![Figure 5: The impact of the number of historical versions on the extraction accuracy](image)
5.3 Strategies of rebuilding HMTs

Since the extraction accuracy would decline over time based on the previous experiment, we need to update the HMT if we want to achieve a reasonably accurate extraction. The process of updating should be efficient and effective enough. During the process of updating the HMT, built from $v_1, \ldots, v_n$, to the HMT built from $v_1, \ldots, v_{n+1}$, there three ways to update it. The first one is to remove $v_1$ from HMT, and then merge $v_{n+1}$ to HMT; the second way is to merge $v_{n+1}$ to HMT, and then remove $v_1$ from HMT; the third way is to directly re-build the tree from $v_2$ to $v_{n+1}$. During removing, we delete all the information of version 1, and remove the nodes that do not contain any versions. It is obvious that the third way would lead to an enormous cost of time. We compute the F1-value of extraction respectively from these three types HMT. By t-test, when $\alpha$ is 0.05, the three sets of data have no significant difference between each of the two. So both of the first two strategies are recommended to update the HMTs. The average time of clean a historical version and merge a new version is about 0.5 second.

5.4 Comparing with RoadRunner

We compare our method with RoadRunner. We use a set of historical versions $<v_{36-HN}, \ldots, v_{36-1}>$ to generate the wrapper with RoadRunner, and extract the content from $v_{36}$. HN denotes the number of historical versions. The data set here are the same with what we used in Experiment 2. We compare their results in Figure 7. It shows that the F1 of our method is much higher than that of RoadRunner. The possible reasons are shown as follows: First, RoadRunner targets at the sites with a fairly regular structure, but most homepages nowadays do not meet with this requirement. Second, RoadRunner generates the wrapper by solving the mismatches during parsing, while in our method, the overall situation is considered.

6. CONCLUSIONS

In this paper, we focus on building a merged tree from one page’s historical versions, and using it to detect and trace dynamic content blocks. We update the merged tree to achieve better extraction accuracy. Experimental results show that our proposed method is able to accurately monitor the dynamic blocks.

7. REFERENCES