Incremental Web Page Template Detection

Yu Wang, Bingxing Fang, Xueqi Cheng, Li Guo, Hongbo Xu
Institute of Computing Technology, Chinese Academy of Sciences
BeiJing, China
wangyu2005@software.ict.ac.cn,bxfang@ict.ac.cn,cxq@ict.ac.cn,
guoli@ict.ac.cn,hbxu@ict.ac.cn

ABSTRACT

Most template detection methods process web pages in batches that a newly crawled page can not be processed until enough pages have been collected. This results in large storage consumption and a huge delay of data refreshing. In this paper, we present an incremental framework to detect templates in which a page is processed as soon as it has been crawled. In this framework, we don't need to cache any web page. Experiments show that our framework consumes less than 7% storage than traditional methods. And also the speed of data refreshing is accelerated because of the incremental manner.

Categories and Subject Descriptors: H.3.3 [Information Systems]: Information Search and Retrieval

General Terms: Experimentation, Algorithms

Keywords: Automatic Template Removal, Web Page Segmentation

1. INTRODUCTION

Template detection is an important technique since templates could heavily cripple the performance of other modules such as page classification modules or index builders.

Most previous approaches [1, 3, 4, 2] utilize content repetition as a hint for template detection. A block which occurs in many pages is considered to be a template block. For the use of content repetition, they all require a lot of web pages as input. So in practice, these methods run in a batch manner. They start working when enough web pages of a site have been gathered. They detect all template blocks and then wait until another dataset is available.

However, caching the web pages often consume lots of storage. And also, since a newly crawled page can not be processed until enough pages are collected, the delay of data refreshing is huge. So the batch manner is not applicable to schemes in which the speed of data refreshing is concerned. These schemes include news search engines, blog search engines, etc.

In this paper, we propose an incremental framework to detect templates in which a page is processed as soon as it has been crawled. The basic idea of the framework is to maintain some information of past web pages, and detect templates with the guidance of past information.

WWW 2008, April 21–25, 2008, Beijing, China.
ACM 978-1-60558-085-2/08/04.

2. THE PROPOSED FRAMEWORK

Our template detection method are based on the repetition of text segments which are text nodes in DOM trees of web pages. We use a data structure called the text segment table to maintain the repetition information, i.e. the contents and DFs of text segments. We should note that when talking about the DFs of text segments, we must first clarify when two text segments are considered to be the same. In our framework, two text segments are same only when their contents are literally equal and they have the same DOM path.

Whenever a new page is available, it will be passed through four steps: 1) page segmentation, 2) text segment table expansion, 3) template detection, and 4) text segment table shrinkage.

2.1 Page Segmentation

The segmentation process contains two steps: 1) A web page is divided into multiple blocks. Currently, we choose some html tags that usually determine the page layout as separators, these html tags are <TABLE>, <DIV>, etc. 2) Then each block is further divided into text segments by html tags, process instructions, and html comments.

2.2 Text Segment Table Expansion

After page segmentation, text segments are used to update the text segment table. If a text segment already exists in the table, the DF of it will be increased by one. Otherwise, the text segment will be inserted into the table and its DF is initialized to one.

2.3 Template Detection

Template detection occurs in block level. We search every text segment of a block in the text segment table, checking whether it is a template segment. We define template segments as text segments whose DFs are larger than or equal to 5. We can then calculate the template ratio of a block:

$$template\ ratio = \frac{\sum lengths\ of\ template\ segments}{\sum lengths\ of\ all\ text\ segments}$$

If the template ratio of a block is larger than 0.7, we label the block as a template block.

2.4 Text Segment Table Shrinkage

The text segment table will consume more and more storage if only the expansion step is applied. To control the storage use, we need to delete some text segments.

The cost of deleting a text segment is defined as the times to classify a template segment as non-template segment be-

cause of the deletion. For example, if a template segment appears after its deletion, it will be recognized as non-template segment. The cost of deleting a text segment is related to the DF and the future occurring times of the text segment.

To minimize the cost of deletion, we allow text segments that have larger DFs to live longer than those with smaller DFs, because the former are more likely to occur in the future. We should note that we don't use the publish times or crawling times as the timestamps of pages and blocks, but we assign every page a page number and use it as the timestamp. The maximum living time of a text segment is then modelled by the logistic function:

$$t = \frac{T_b N}{\left(1 + (N-1) * e^{-(df-1)}\right)}$$

T_b is the maximum living time of a text segment which appears only once. *df* is the past DF of the text segment. $T_b N$ defines the upper bound of maximum living time of a text segment before a new occurrence comes, no matter what *df* is. When a text segment doesn't appear for t, it will be removed from the text segment table.

3. EXPERIMENT

As there is no standard test set, we build a new data set by hand. We pick five popular sites and sample 400 pages from each site. People are asked to label every block in pages as template or not depending on their understanding. All sites are picked from the data set of Ma's work [3]. This guarantees the test set has no prejudice towards our method.

In this section, we compare our template detection method with shingle [3] and SST(Site Style Trees [4]. To our knowledge, SST is currently the best template detection method in the literature. Our method runs in the incremental manner, while shingle and SST run in the batch manner.

Since the precision of template detection is often very high that the precision of all these methods is higher than 98% in our experiment, we don't concern the precision of methods. We first compare the storage consumption of all template detection methods when their recall rates are equal. The expected recall is 80%. To achieve the recall rate, the number of web pages in each batch is adjusted for shingle method and SST method, and the parameters of logistic strategy is adjusted for our method.

In the incremental template detection process, the size of the text segment table changes when it is updated. In the batch template detection process, the size of cached web pages also changes. So we compares the average storage consumption of methods. To be concrete, if the size of the text segment table when inserting the i_{th} page is denoted as s_i, then the average size of the text segment table can be calculated as follow:

$$AverageSize = \sum_{all\ possible\ i} s_i / \sum_{all\ possible\ i} 1$$

In the batch template detection process, the average size of cached web pages is the average size of all batches.

The result is summarized in Table 1. The second column contains the sizes of text segment tables of our method, the third column contains the sizes of web pages of shingle method, and the fourth column contains the sizes of web pages of SST method.

As shown in Table 1, our method with logistic strategy saves 93.81% storage compared with shingle method when recalls of these two methods are equal. In other words, our method only consumes 6.19% storage of that consumed by shingle method. When compared with SST method, our strategy saves 93.22% storage. On the other hand, we expect that our method achieves higher recall than other methods when using the same amount of storage.

Table 1: When the recall rates are equal, our method with logistic strategy saves 93.81% storage compared with shingle method and saves 93.22% storage compared with SST method.

	Our Method	Shingle	SST
amazon.com	419KB	5739KB	4947KB
cnnfn.com	126KB	2754KB	2295KB
ebay.com	139KB	5225KB	4522KB
nba.com	386KB	3073KB	3210KB
yahoo.com	50KB	1294KB	1534KB
Average Size	224KB	3617KB	3302KB
Saving Storage		93.81%	93.22%

In the experiment, the average number of pages in each batch is 24. So in the batch manner, the template detection methods should wait for 24 pages to be collected before they can be run. While in our framework, every page will be processed immediately.

4. CONCLUSION

In this paper, we propose a promising framework to detect template blocks incrementally. Our approach uses the text segment table as a compact representation of past web pages, and uses an effective updating strategy to maintain it. Further more, a new template detection algorithm is proposed based on the text segment table. Experiments on five popular web sites indicate that our framework consumes less than 7% storage than traditional methods. And also the speed of data refreshing is accelerated.

5. REFERENCES

[1] Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *Proceedings of the 11th international conference on World Wide Web*, pages 580–591, 2002.

[2] D. Chakrabarti, R. Kumar, and K. Punera. Page-level template detection via isotonic smoothing. In *Proceedings of the 16th international conference on World Wide Web*, pages 61–70, 2007.

[3] L. Ma, N. Goharian, A. Chowdhury, and M. Chung. Extracting unstructured data from template generated web documents. In *Proceedings of the 12th international conference on Information and knowledge management*, pages 512–515, 2003.

[4] L. Yi, B. Liu, and X. Li. Eliminating noisy information in Web pages for data mining. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 296–305, 2003.