Scuola Superiore Sant'Anna

Ask.com, European Research Center, Pisa

Scuola Superiore Sant'Anna

Classe Accademica di Scienze Sperimentali, Settore di Ingegneria

Corso di Laurea Specialistica in

Ingegneria Informatica

A fast and parameter-free template detection method for web pages and online newspapers

Tesi di Diploma di Licenza Specialistica

Luca Foschini

Tutore:
Prof. Paolo Ancilotti
Relatore:
Dott. Antonino Gullì

Contents

Abstract

The increased use of content-management systems to generate webpages has significantly enriched the browsing experience of end users; the multitude of site navigation links, sidebars, copyright notices, and timestamps provide easy-to-access and often useful information to the users. From an objective standpoint, however, these "template" structures pollute the content by digressing from the main topic of discourse of the webpage. Furthermore, they can cripple the performance of many modules of search engines, including the index, ranking function, summarization, duplicate detection, etc., and can seriously harm Web mining, e.g., clustering and classification.
With templated content currently constituting more than half of all HTML (Hyper Text Markup Language) on the web and growing steadily [GPT, BYR02], it is imperative that search engines develop scalable tools and techniques to reliably detect templates on a webpage.
In this work we propose a novel approach that identifies templates by analyzing a history of data associated to a website and generates a set of regular expressions that match the templated content of that site. Those expressions can be reused verbatim to clean the pages they were extracted from and other belonging to similar sites.
Our method is almost parameter-free, in the sense that the algorithm itself depends on a very limited set of parameters, most of them causing little or no changes in the outcome when varied. Moreover, our method does not require any kind of preprocessing on the HTML pages, therefore, it does not suffer from performance penalties caused by parsing and manipulating HTML. Another point of strength of our method is that it has proven to be particularly well-suited for cleaning web news articles, which are assuming an increasing interest in the web community.
In this work we present our algorithm and discuss a prototype implementation based on suffix array. We then review some algorithms we developed in order to speed up the computation of statistics based on the suffix array. As a result, despite its prototypal nature, our implementation of the template-extractor system has proven to be extremely fast, making the suffix array computation (for which we used a linear-work, state-of-the-art algorithm) the actual bottleneck of our implementation.
In the end, a preliminary evaluation of our system shows how it performs with high accuracy in removing templates in web pages and news articles.

Keywords: template detection, noise removal, suffix array, motif finding.

Chapter 1

Introduction

The rapid expansion of the Internet has made the WWW (Word *Wild* Web) a popular place for disseminating and collecting information. Data mining on the web thus becomes an important task for discovering useful knowledge or information from the web [Cha02, HC02].
Many problems arise when automatic systems try to collect information from the web. In the first place, most of the data present on the WWW are in HTML (Hyper Text Markup Language) format, which is not designed for structured data extraction at the first place, but mainly for data presentation [Myl01]. HTML pages are usually "dirty", i.e., improper closed tags, improper nested tags, and incorrect parameter value of a tag are examples of such problems. In order to reduce parsing errors over ill formed web pages caused by loose HTML standard, WWW Consortium [W3C] recommend stricter standard Markup Languages, such as XHTML (Extensible HyperText Markup Language) and XML (Extensible Markup Language). However, parser of search engine still needs to cope with existing web pages that do not conform to the new standards.
Unfortunately, data on the web are not only affected by problems caused by the misuse of markup language, which can be easily ameliorated using freely available tools [Rag]. A more daunting problem arises from the lack of structure in web pages, which causes useful information on the web being often accompanied by a large amount of noise, such as banner advertisements, navigation bars, copyright notices, etc. Although such information items are functionally useful for human viewers and necessary for the web site owners, they often hamper automated information gathering and web data mining, e.g., web page clustering, classification, information retrieval and information extraction.

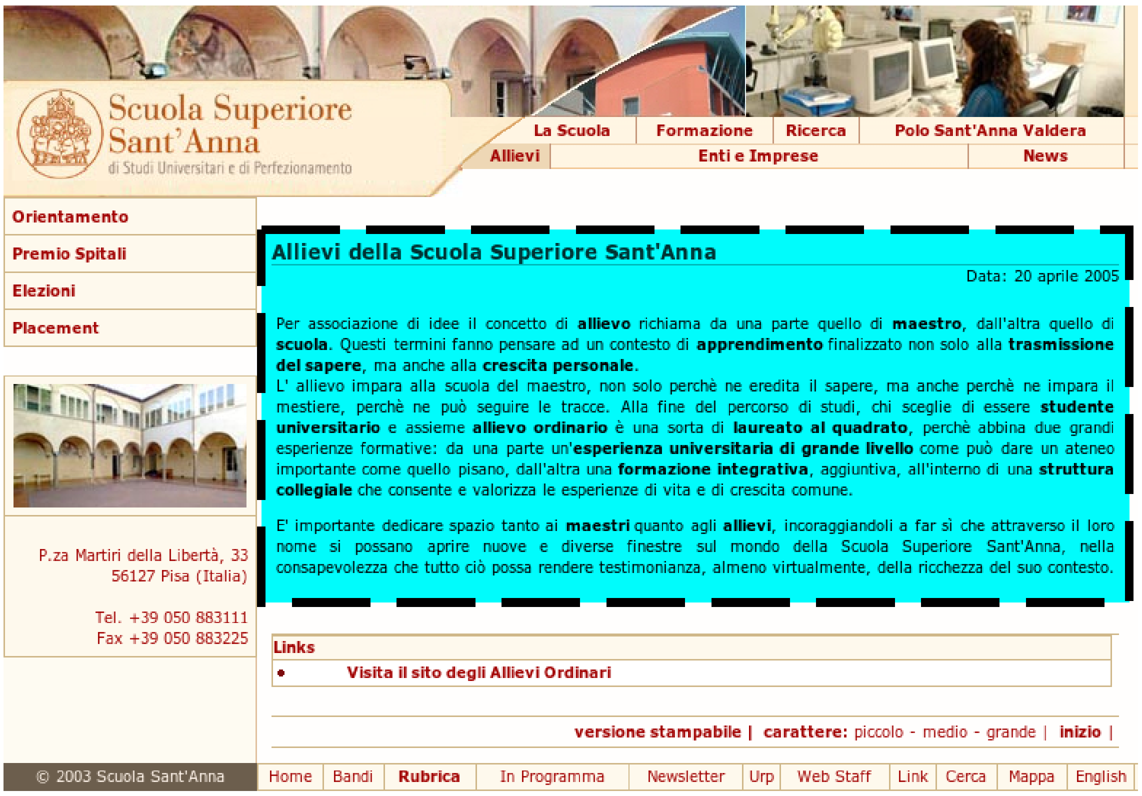
1.1 The Noisy World *Wild* Web

We have just stressed how web data are noisy, in both content and representation. In this work we focus mainly in content-related noise, which can be grouped into two categories according to its granularity.

Global noise is noise with large granularity, which is usually no smaller than individual pages. Global noise include mirror sites, legal/illegal duplicated web pages, old versioned web pages to be deleted, etc.;

Local (intra-page) noise: that is, noisy regions/items within a web page. Local noises are usually incoherent with the main contents of the web page. Such noises include banner, navigational bars and advertisements, which are in fact noise data for information retrieval as they have no direct relevancy with content data.

Local noise, as observed by Ziv Bar-Yossef et al. [BYR02] is mainly caused by templatized pages from professional designed website sharing the same context for browsing.
Templates are common parts shared by many web pages in a web site that usually have the same look, layout and, as mentioned before, typical templates include navigation bars, advertisement, privacy policy and contact information, etc.



Allievi della Scuola Superiore Sant'Anna

Data: 20 aprile 2005

Per associazione di idee il concetto di **allievo** richiama da una parte quello di **maestro**, dall'altra quello di **scuola**. Questi termini fanno pensare ad un contesto di **apprendimento** finalizzato non solo alla **trasmissione del sapere**, ma anche alla **crescita personale**.
L' allievo impara alla scuola del maestro, non solo perchè ne eredita il sapere, ma anche perchè ne impara il mestiere, perchè ne può seguire le tracce. Alla fine del percorso di studi, chi sceglie di essere **studente universitario** e assieme **allievo ordinario** è una sorta di **laureato al quadrato**, perchè abbina due grandi esperienze formative: da una parte un'**esperienza universitaria di grande livello** come può dare un ateneo importante come quello pisano, dall'altra una **formazione integrativa**, aggiuntiva, all'interno di una **struttura collegiale** che consente e valorizza le esperienze di vita e di crescita comune.

E' importante dedicare spazio tanto ai **maestri** quanto agli **allievi**, incoraggiandoli a far sì che attraverso il loro nome si possano aprire nuove e diverse finestre sul mondo della Scuola Superiore Sant'Anna, nella consapevolezza che tutto ciò possa rendere testimonianza, almeno virtualmente, della ricchezza del suo contesto.

Figure 1.1: Web portal of the Scuola Superiore Sant'Anna. The main content (highlighted by the blue dashed rectangle) only occupies a little portion of the original page.

Figure 1.1 gives a sample page from the portal of Scuola Superiore Sant'Anna. As can be seen, the main content (in the dashed-contoured blue-shaded rectangle) only occupies one third of the original web page, and the rest of the page contains navigation links, privacy statements, site headline, etc.. If we perform clustering on a set of product pages like this page, such items are irrelevant and should be removed.

As a matter of facts, although templates are widely used in web site design, they are negative factors for information retrieval. First, the words in template hurt relevance computation by their irrelevant content. Second, the link distribution of web pages is skewed by the duplicated links in templates, as showed by Ziv Bar-Yossef et al., hence reducing precision. Similar concept are stressed in other works, such as [MGCC03, LH02, SR02]. More generally speaking, Ziv Bar-Yossef et al. stated and proved that navigation bar and paid advertisement are in direct violation of Hypertext IR (Information Retrieval) Principles [BYR02].

We have stressed that local noise in web pages can seriously harm the accuracy of data mining. Therefore,we must extract unstructured data from the web page; meanwhile remove irrelevant template navigation bars and advertisements. Cleaning the web pages before mining becomes critical for improving the data mining results. We call this preprocessing step "web page cleaning".

Many methods have been proposed to accomplish such a task. Conventional approach for web page data extraction requires one wrapper for each web site. A wrapper is a program that can extract data from web pages according to the tag structure of the page. Programmers have to manually find the reference point and absolute tag path for the targeted data content. A wrapper is incorrect if it is trained with sample pages missing optional or disjunction elements. A modification to a tag path causes the wrapper to easily break. Therefore, manually coding and maintaining wrapper can be all-consuming work.

A tailored wrapper can extract data from a particular web site because the targeted web pages are usually generated from the same template, and therefore share identical navigation bars.

Some attempt has been done in developing automated template detection systems too. Among those, most existing methods for template detection operate on a per website basis by analyzing several webpages from the site and identifying content and/or structure that repeats across many pages. These "site-level" template detection methods offer a lot of promise, these methods are error prone when the number of pages analyzed from a site is statistically insignificant, either because the site is small, or because a large fraction of the site is yet to be crawled.

An alternative paradigm that avoids many of these pitfalls is to detect templates on per webpage basis, i.e., "page-level" template detection. This is especially attractive since it can be easily deployed as a drop-in module in existing crawler work-flows. A tempting approach to page-level template detection is to extract sufficiently rich features from the DOM (Document Object Model) nodes and train a classifier to assign "templateness" scores to each node in a DOM tree. While this approach is entirely plausible, it has several handicaps. First, for the classifier to have a reasonable performance, both accurate and comprehensive training data is required; this can involve prohibitive human effort. Second, by classifying each DOM node in isolation this approach does not take a global view of the templateness of nodes in the DOM tree. In the next chapter we will review various ideas applied on the two aforementioned baseline and evaluate pros and cons.

In this paper we develop a novel framework for site-level template detection based on statistical analysis of different version of the same page or different pages of the same site. Our system does not consider the format and structure information in web pages and extracts significance information by means of statistics calculated on the suffix array computed on the raw data, interpreted as a string. The system output a series of regular expressions that matches the extracted templated components in the document set analyzed and those expression can then be used verbatim to clean documents coming from the same site. The method is almost parameter free, in the sense that the performance and accuracy of the cleaning depend on few parameters. These characteristics makes our module very flexible and language independent since as long as the suffix array can be computed on the text, the system would be able to provide meaningful results.
More precisely, in this work we make the following contributions:

- we review the state-of-the-art work in template detection;
- we propose a statistical method based on the *frequent item set* principle that automatically extracts templates from a set of document published by the same autority (i.e., coming from the same site or from the same news feed) as a set of regular expressions
- we propose an effective implementation of the aforementioned system based on suffix array and we devise optimization to compute effectively statistics on the documents;
- we propose a method to apply the set regular expressions extracted by the mentioned method in order to remove the templates
- we give some preliminary results on the performance and accuracy of template removal as evaluated by editorial and propose some standard ways to evaluate our system

Chapter 2

Present Work

Although web page cleaning is an important task, relatively little work has been done in this field.
In [LH02], a method is proposed to detect informative blocks in news web pages. However, the work in [LH02] is limited by the following two assumptions: (1) the system knows a prori how a web page can be partitioned into coherent content blocks, and (2) the system knows a priori which blocks are the same blocks in different web pages.
Web page cleaning is also related to feature selection in traditional machine learning (see [YP97]). In feature selection, features are individual words or attributes. However, items in web pages have some structures, which are reflected by their nested HTML tags. Hence, different methods are needed in the context of the web.
Kushmerick in [Kus99] proposes some learning mechanisms to recognize banner ads, redundant and irrelevant links of web pages. However, these techniques are not automatic. They require a large set of manually labeled training data and also domain knowledge to generate classification rules.
Kao et al. [KLHC02] enhances the HITS algorithm [Kle99] by using the entropy of anchor text to evaluate the importance of links. It focuses on improving HITS algorithm to find more informative structures in web sites. Although it segments web pages into content blocks to avoid unnecessary authority and hub propagations, it does not detect or eliminate noisy contents in web pages.
Crescenzi et al. [CMM01] (Road Runner Project) and Arvind Arasu [AGMU03] investigated techniques in recognizing semantic structure of data from large web sites. Helena Ahonen-Myka [AM02] and Florian Beil et al. [BEX02] designed efficient algorithms for frequent term set clustering problem.

2.1 Page Segmentation

Many methods proposed for the template problem relies on the concept of page segmentation. Since it is usually simple to visually separate template content from informative content because they both occur in well defined, non overlapping area, (they are not scat-

tered one into another) a lot of work have been devoted to attempt to segmenting web page into smaller but more cohesive *pagelets*.
The notion of *pagelet* does provide a new angle for the granularity of information retrieval. However, it is difficult to obtain the optimum threshold value k for determining whether a sub parse tree is indeed a *pagelet*. For instance, Xiaoli Li et al. [LPHL02] proposed segmenting web page into topically more focused MIU (Micro Information Unit). Their algorithm employs text font property and term set similarity, building HTML page tag tree, merging heading with text, and adjacent similar paragraphs as MIU.
Bar-Yossef et al. [BYR02] employ a notion of pagelet to segment a web page too. In their case, a pagelet is determined by the number of hyperlinks in a HTML element and web page cleaning is defined as a frequent template detection problem. They propose a frequency based data mining algorithm to detect templates and views those templates as noises. The cleaning method in [BYR02] is not concerned with the context of a web site, which can give useful clues for page cleaning. Moreover, in [BYR02], the partitioning of a web page is pre-fixed by considering the number of hyperlinks that an HTML element has. This partitioning method is simple and useful for a set of web pages from different web sites, while it is not suitable for web pages that are all from the same web site because a web site typically has its own common layouts or presentation styles, which can be exploited to partition web pages and to detect noises A similar method is proposed by Ma et al. in [MGCC03], which partition web pages based on HTML tag `<TABLE>`. Lin et al. [LH02] employ the same partition method as [BYR02] and keywords of each block content are extracted to compute entropy for the block. Blocks with small entropy are identified as templates. The approach faces two problems: (1) its accuracy relies on feature selection, (2) it is time consuming to extract features for all the blocks.
Other related work includes data cleaning for data mining and data Warehousing [LLL00], duplicate records detection in textual databases [NBM02] and data preprocessing for web Usage Mining [CMS99].
In [YLL03] Yi et al. introduce a tree structure, the site style tree (SST) to capture the common presentation style of web pages. Observing that the web pages in a given web site usually share some common layout or presentation styles, they propose the Style Tree to capture those frequent presentation styles and actual contents of the web site. The site style tree (SST) provides rich information for analyzing both the structures and the contents of the web pages. SST is a tree that merges all the DOM trees of all web pages. Entropy of SST element is computed to determine which element is template. To clean a page from a site, they simply map the page to its SST. Their cleaning technique is evaluated using two data mining tasks. Compared with other methods, SST achieves a higher accuracy. However, several drawbacks limit its scalability: (1) when a new DOM Tree is merged, the content of each DOM Tree's element must be compared with all the branches in SST to determine whether it is a new branch or an existed one. This process costs much computation. (2) When dealing with large volume web pages, SST cannot fit into memory. Besides the research on template detection, there are many other related studies: web segmentation [YCWM03], block-leveled search [LPHL02], block-leveled link analysis [CYWM04, CHWM04], and content extraction [RGSL04]. Some studies propose

other methods to solve similar problems, e.g. statistical method [AGMU03] and machine learning [Kus99].

2.1.1 Site-level Template Detection

The problem of template detection and removal was first studied by Bar-Yossef and Rajagopalan [BYR02], who proposed a technique based on segmentation of the DOM tree, followed by the selection of certain segments as candidate templates depending on their content. Vieira et al. [VdSP+06] framed the template detection problem as a problem of mapping identical nodes and subtrees in the DOM trees of two different pages. They proposed performing the expensive task of template detection on a small number of pages, and then removing all instances of these templates from the entire site by a much cheaper approach. Gibson et al. [GPT] conducted a detailed study of templates on the web, demonstrating the prevalence of templated content and its steady growth. All of these methods, however, need multiple pages from the same website to perform template detection, and thus suffer from the problems mentioned in the introduction.

2.1.2 Page-level Template Detection

Some page-level algorithms have also been proposed recently. Kao et al. [KHC05] segment a given webpage using a greedy algorithm operating on features derived from the page. However, their method is not completely page-level; they also use some site-level features such as the number of links between pages on a website. Debnath et al. [DMPG05] also propose a page-level algorithm ("L-Extractor") that applies a classifier to DOM nodes, but only certain nodes are chosen for classification, based on a predefined set of tags. Kao et al. [KLHC02] propose a scheme based on information entropy to focus on the links and pages that are most information-rich, reducing the weights of template material as a by-product. Song et al. [SLWM04] use visual layout features of the webpage to segment it into blocks which are then judged on their salience and quality. Other local algorithms based on machine learning have been proposed to remove certain types of template material. Kushmerick [Kus99] develops a browsing assistant that learns to automatically removes banner advertisements from pages. Another set of papers focus only on segmentation of webpages for the purpose of displaying them on small mobile device screens [Bal06, CXMZ05, YL04]. However, template detection is not their primary focus. Only a subset of DOM nodes ("segments") are operated upon, this subset having been chosen prior to any determination of the templateness of those segments. As a result, should a segment itself be composed of several template and non-template nodes, this would not be detected.

2.2 A Recently Proposed Hybrid Approach

Chakrabarti et al. in [CKP07] present a framework for classifier based page-level template detection that constructs the training data and learns the notion of "templateness" auto-

matically using the site-level template detection approach. They formulated the smoothing of classifier assigned templateness scores as a regularized isotonic regression problem on trees, and presented an efficient algorithm to solve it exactly. Using human-labeled data they empirically validated our system's performance, and showed that template detection at the page-level, when used as a preprocessing step to webmining applications, such as duplicate detection and webpage classification, can boost accuracy significantly.
First, they generate training data by applying the site-level template detection method on several randomly selected sites. Next, they define and extract appropriate features for these site-level templates. Finally, they use this automatically generated data to train a classifier that can assign a templateness score to every node in a DOM tree. They show that the proposed classifier generalizes beyond its site-level training data and can also discover templates that manifest only at the page-level. Their second contribution is the formulation of a global property that relates templateness scores across nodes of the DOM tree. They assert that templateness is a monotone property: a node in the DOM tree is a template if and only if all its children are templates. An appropriate relaxation of this property leads to the following regularized isotonic regression problem: given a tree with classifier scores at each node, find smoothed scores that are not far from the classifier scores, but satisfy the relaxed monotonicity property.
On the whole, their approach eliminates the aforementioned issues with pure classifier-based approaches. An interesting by-product of their framework is that it obtains a sectioning of a page into segments; this is useful in many applications. Chakrabarti et al. performed an extensive set of experiments to validate the proposed framework and algorithm. In terms of detecting content within templates, their algorithm achieves an f-measure in excess of 0.65 for text and 0.75 for links on a human-labeled test set. They highlight the applications of template detection by showing that removing templates as a pre-processing step boosts the accuracy of standard web mining tasks on theier datasets, by as much as 140% on duplicate detection, and 18% on webpage classification. Further, they show that the gains obtained by using their page-level template detection approach are substantially greater than those obtained by using the more expensive site-level approach.

2.3 Open Problems

As stated before, template detection is still a young research field and many challenges are still to be faced. Many HTML web pages incorporate dynamic technique such as javascript and PHP script. For instance, sub menu of navigation bars are rendered by “document.write” method of javascript on the event of mouse over. Image map are also implemented with javascript code. None wants to include any template navigational bar in the indexing phase. However, there are cases when page content are rendered with javascript methods.
Another issue is handling images with descriptive functions in the web pages. It is not good to simply remove any image from the page, as we all know that a picture sometimes worth one thousand words. For example, fashion site like `elle.com` prefers image text rather than

a plain text. Image text certainly facilitates readers' understanding; on the other hand it prohibits data extraction.

Chapter 3

The proposed method

3.1 Sketch of the Algorithm

The system we are about to describe is based on the simple idea that given different observations of a web page coming from a particular editorial authority, this observations should share the same template content but have different informative content. This concept is pervasive in data mining and has been stressed more than once in Chapter 2. The problem of template detection is therefore reduced, with this reasoning, to the problem of finding frequent items into a set. In order to base the "observation history" needed to compute the frequency statistics we need different snapshot of the same website, and this could be different version of the site crawled in different point in time (useful to deal with online newspapers) or a section of different pages of the same website.
Once we have such a collection, we can run the template regular expression extractor on each collection, on a per site basis, (recall that each collection depends on a particular publishing authority supposed to use the same templates across time and space). The results of the extraction for a given collection is then possibly filtered and finally the mined expressions can be applied to effectively clean out all the pages of that publishing authority, even if not used to train the system.

3.2 A closer look at the algorithm

The algorithm takes as input a collection of documents $D = d_1, d_2, \ldots, d_n$ belonging to the same publishing authority (i.e. a website or a news feed) and gives as output a set of regular expression $R = r_1, r_2, \ldots, r_m$ which, applied in turn to documents coming from the same source should remove the templates. The steps of the algorithms are as follows.

1. All the documents in D are then tokenized with respect to a tokenization parameter and tokens are assigned an increasing id number, same tokens have the same ids. In this way, each document's text is represented as a sequence of tokens. From now on we will call "string" a generic sequence of tokens.

2. All the sequences of tokens are then padded with a fictitious symbols with id unused and unique in D and for all the documents in D, all the sequences of tokens are then concatenated to a corpus T. An helper map $T- > D$ associates the positions in T (T is a corpus made of l tokens) to the document which the token in that position in T belongs.

3. The suffix array [MM90] S is computed on the corpus T

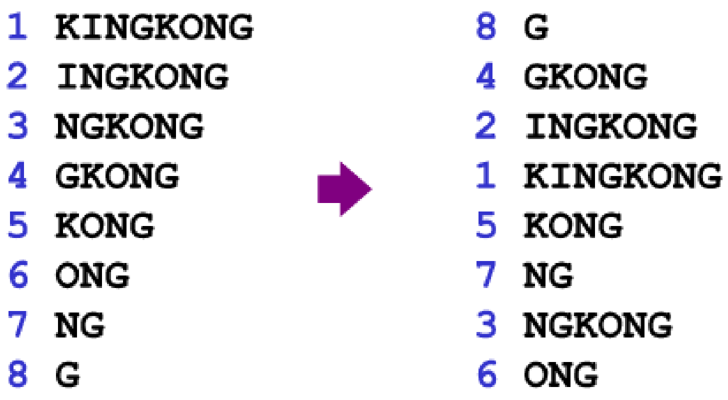


Figure 3.1: Suffix array built on the string “KINGKONG”. On the right all the suffixes of the word are sorted lexicographically. $S[i]$ contains the original position in the text of the $i - th$ suffix considered in lexicographic order.

4. The lcp [KLA+01] vector L is computed on the pair $< S, T >$

5. For each substring s in T, using $< L, S, T, T- > D >$ some statistics are computed, namely:

   (a) the number of occurrence of s in T

   (b) the number of distinct documents of D in which s appears

   The details of how this statistics can be computed in a fast way will be sketched out in Chapter 4. For now it is sufficient to know that computing those is computationally feasible.

6. Only the maximal strings are retained. The suffix array returns all the possible substring of T but, loosely speaking we are interested only in “different instances” of the same string. More formally, if a string s' is a substring of s and appears *only* where s appears, then we are not interested in the statistics of s' since s' probably does not represent a concept *per se'* but only when present as part of s.

   This need can be satisfied in an efficient way while computing the statistics by means of some additional memory used, as it will be explained in chapter 4

7. Some simple heuristics are performed in order to filter-out irrelevant strings, more precisely, strings appearing in a small number of different documents and strings made of less than two tokens are discarded.

8. The histogram of the distribution of substrings with respect the number of distinct documents in which they appear is computed and is shown in Figure 3.2

   Intuitively, this histogram can give us a strong hint on the templateness of the strings we are analyzing. Since we have filtered out strings made of a small number of tokens, thus avoiding to process single stopwords as “and”, “is”, etc., only the strings belonging to templates will likely appear in every document of D. This means that those strings will show up in the right part of the histogram so we can easily find them. The only think we miss now is to set the cut-off point above which a substring is considered a template.

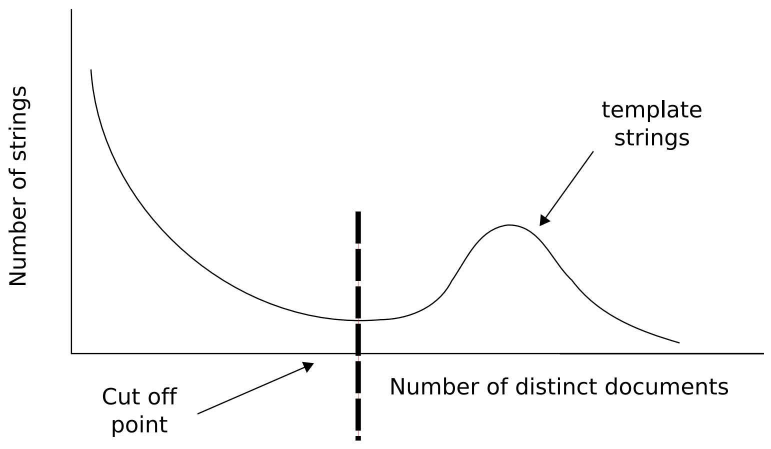


Figure 3.2: Probability density function of number of strings vs number of distinct documents. It can be seen the power-law shaped distribution with another mode for high values of distinct documents, which identifies template strings

9. We have implemented a system that infers the cutoff value automatically. The probability density function associated to the histogram is bi-modal because without templates in the pages, it would be skewed to the left (power law) since there are many strings that appear only in few documents. We just need to recognize where this modes are. More precisely:

   (a) the histogram is binned in 100 bins

   (b) we wants to find a cut-off bin b_c in the histogram that minimize the following function:

$$E(b_c) = \sum_{i=0}^{i<b_c}(i-x)^2 \cdot hi + \sum_{i=b_c}^{i<n}(i-y)^2 \cdot hi \quad , b_c = \frac{x+y}{2} \tag{3.1}$$

Equation 3.1 is a quadratic problem that we can solve via brute-force (we have only 100 bins). $E(b_c)$ resembles an energy function. This method is robust and usually places the cut-off point in the distribution with high accuracy where the human eye would have placed it [1]

3.3 Why this cannot work?

The previously described algorithm has a fundamental flaw: it is based on suffix array, then it find *exact* duplicate strings. Even if a string is part of a template, it could occur in the same template block in different pages slightly varied. For instance, the whole menu block of a page could contain session-based links that changes every time the page is accessed or a date, which is fast-mutable too.
In this case, the strings representing slightly varied templates will be not matched together by the suffix array, and thus will not be identified as a template. In order to avoid such a problem, we need to give some flexibility to our system. We add flexibility by changing the input of the system, and, more precisely:

- Once mutable patterns are recognized (urls, dates, numbers, email addresses) they are replaced in the original text in a "normalized" representation in order to have them looking the same from the suffix array's point of view.

  An example of normalization would be to replace every token entirely made by uppercase characters (thus matched by a regex like u+) with the word "UPCASE".

- For convenience, which we will explain later, we chose to replace a mutable pattern with the strictest regular expressions that match it. The regex we are interested in have to be strict, in the sense that it should match no different patterns from the one it was created for.

- After this pre-processing pass everything works as described before, mutable expression are treated by the suffix array as normal tokens and will be processed just like other tokens.

[1]It has to be notice that naiive methods, such as to set b_c as the minimum value on the histogram .(if the histogram is bimodal, the minimum will be somewhere in the middle of the two modes) will not work because we are dealing with discrete distributions and each method based on differencing is going to be killed by noise in the distribution (even though the histogram always has more or less the same shape, the variance of where the minimum value could fall is very high)

Another subtle point of the algorithm described in session 3.1 is that the cutoff value above which the strings are considered to be part of templates, even though chosen with extreme accuracy, will not lead to a perfect result, for the simple reason that a clean cutoff between what is good and what is bad does not exist. Wherever we put our cutoff we will lose some template strings and find some non-template strings that generally are idiomatic forms such as "as much as" or recurrent passages such as "on monday says" and so on.
In order to avoid this problem we have chosen to be conservative in selecting the cutoff point in order not to have false negative (that is, in order not to lose template strings). We then perform a pass of cleaning on the mined strings following the simple reasoning that template strings will be very frequent within a site but pretty different among different site, unlike recurrent expressions as seen before, which will appear often regardless the site. Following this cue, we have implemented a "second level" cleaning algorithm that takes as input all the strings mined from the collection of documents $D_1, D_2, \ldots, D_z$ and remove string appearing in many different collections.
Now that we finally have the set of strings that describes supposed-to-be templated content in web pages, we can think about how to use them to strip that content from the analyzed sites.

3.4 Cleaning document

We have stated in section 3.1 that our system output a set of regular expression R, but we have only templated strings (sequences of tokens) so far. At this point, the pass to create those expression is simple:

1. the tokens on the strings are mapped back to their original text value they had in the document. In the same way, the ids of input regular expressions representing a mutable pattern are mapped back to the textual representation of the regular expression;

2. textual representation of the tokens in a token sequence are concatenated together with each element separated by the inverse of the pattern used to tokenize.

   For instance, if the input text was tokenized using the expression `\S+` (which means: tokens are sequence of non-space character of non null length) then the templated strings will be concatenated using separator `\s+` (which means: any space characters). This means that, if the input text was "hello! How are you?" it will be tokenized in four tokens : "hello!", "How", "are", "you?". If those tokens will output as a template string, they will be glued together forming the expression "hello!`\s+`How`\s+`are`\s+`you?`\s+`", which matches the original text.

At this point, it is clear that the strings given in output after the aforementioned post-processing are indeed regular expressions, which can match the text they were extracted from.

Mutable pattern will be matched by the expressions they were replaced with during input preprocessing. In this way we have created cleaning expressions from the templated strings for free.
For instance, if the text to be cleaned was "I'm 24 years old" and we had a mutable pattern "[0-9]+", which matches any sequence of digit, the text would be first of all transformed in "I'm [0-9]+ years old", then tokenized with the usual separator, so we obtain tokens: "I'm", "[0-9]+", "years", "old". If those token are identified as a template then in output will the expression "I'm\s+[0-9]+\s+years\s+old" will be returned, which matches the input text and generalizes it since it matches, for instance, "I'm 25 years old" too.

3.4.1 Applying regex to clean texts

Texts can be cleaned by applying the extracted regex to it and remove the matched content. The order in which this regular expressions are applied is very important, if we think to apply the expression r_{i+1} on the text resulting from the application of expression r_i (cascade approach). In this case it would be arguably that it is better to apply longer expressions first, since those will likely cause long matches to happen in a greedy fashion. A shorter expression, if applied before one that would match a superset of its match, could prevent the longer to match, resulting then in a poor performance.
For instance, if we have a text t='aabb' and two replace expressions r_1 = 'aabb' and r_2='ab' if we first apply r_2, then r_1 we will remove only two characters because the modification done by r_2 prevents r_1 to match. If we reverse the order and apply r_1 before r_2 we will remove all the four characters, instead.
In order to avoid such pitfalls, we devised a method for order-invariant matching that works as follows:

1. Given a text W that we want to clean and the set of regular expression $R = r_1, r_2, \ldots, r_m$,
2. for each expression r_i we try to match it on the text, if some matches occur, we store the intervals of start and end of each match in the text $< s_{r_i,1}, e_{r_i,1} >, < s_{r_i,2}, e_{r_i,2} > \ldots$
3. we compute I as the union of all the intervals $< s_{r_1,1}, e_{r_1,1} >, < s_{r_1,2}, e_{r_1,2} > \cdots < s_{r_2,1}, e_{r_2,1} > \ldots$ for all the expressions and remove the text in W that falls into these intervals. Some heuristics is applied in removal, for instance, if two intervals to be removed are separated by less than user-defined number of characters, the intervals are merged and all the text between the start of the leftmost and the end of the rightmost is removed.

Chapter 4

Implementation

As described before, the system is composed of two modules:

- the template expression extractor;
- the actual cleaner module;

In addition to these core components we have developed some utility scripts that performs the postprocessing of templated strings as seen in section 3.4 and a crawler to create a dataset to work on.

4.1 The Crawler

A multithreaded crawler written in python has been developed. The crawler is fed with a list of URL (Universal Resource Locator) and a configurable number of threads take charge of the retrieving of those URLs, in parallel.
The crawler enforces a "politeness" policy which consists in not crawling too often the same domain. More formally, between two consecutive crawls to the same domain the crawler must honor a input-configurable latency (defaulted to 1 sec)
When a crawling is deferred because of the aforementioned constraint another one, that does not violate the constraint, takes place, if an URL which has this property is present in input.
The crawler is fed with a stream of URL so it cannot count on batch processing to work out a perfect scheduling and thus needs a structure in charge of managing the priority.
For this purpose we have added a new functionality to the python *queue* object, which is thread safe, in order to act as a priority queue keyed on the earliest time an URL becomes crawlable given the constraint on politeness. URLs are assigned a minimum crawling time once read from input and put into the priority queue and when the time comes are popped out to be crawled. Since if the crawler works under congested conditions it could be processing accumulated work possibly raising the minimum crawling time for the URL that are already in the queue, the constraint is checked also upon popping URLs out from the

queue. If the constraint is not satisfied the URL will be rescheduled to a later time (that is, re-put into the priority queue with a higher minimum crawling time)

4.2 The Extractor Module

Implementing the extractor module has been the hardest part. The module needs to process a big amount of data in order to achieve high precision, thus resulting in high CPU and memory demand, so we implemented it with efficiency in mind. More precisely.

- Each input file is read all-in-one with `mmap`. Using C `*scanf` function or, even worse, C++ `iostream` resulted in a heavy performance penalty
- Text is tokenized and mutable pattern are replaced using `boost::regex`. This is the only case in our implementation where we preferred to utilize a portable and easy-to-use solution in favor of a more efficient one. `boost::regex` is very flexible and powerful but not very efficient, further releases will employ the `PCRE` C library.
- The algorithm used for suffix array computation is an adaptation (mainly a C++-fication) of the one described by Sanders et a. in their work [KS03]. This algorithm is $O(n)$ (linear) where n is the total number of tokens and is the state-of-the-art of the research in suffix array.
- lcp vector is computed on the suffix using the algorithm by Kasai et al. [KLA+01] which runs in $O(n)$
- The calculation of statistics stated in section 3.1 are performed using a technique inspired to the work of Yamamoto et al. [YC01] and modified to perform non maximal string elimination. The logic of the algorithm is pretty complex and will not be reported here, anyway, it achieves running times empirically close to $O(n)$ even though in the worst case the algorithm would be quadratic.

4.2.1 The cleaner module

The cleaner module implements the interval logic as explained in section 3.4. It uses `boost::regex` as well to apply regular expression.

Chapter 5

System Evaluation

We evaluated our system both in performance and accuracy and both on HTML and raw text data.

5.1 performances

We ran the extractor on 9795 news crawled from Business Week via Yahoo! between February, 14 and May, 29. The news are in HTML format and occupies 220Mb of space on disk. The box on which we run the extractor was a 8-hyperthreaded processors Intel(R) Xeon(TM) CPU 3.00GHz with 4Gb RAM
As expected, the whole input fit in memory and the computation time was 7m29.051s, varying almost linearly with the input size, implicitly confirming our time complexity estimations of the algorithms involved.
The reported case was one of the largest we could try with our system and probably excessively big for the systems' statistical needs. In a separate experiment conducted on raw text news we have seen that a two weeks worth of articles from a moderately prolific source is more than enough.
However, in this way we have demonstrated that the systems is fast and does not suffer from scalability problems, as opposite to other approaches we have reviewed in the literature.

5.2 Accuracy

For what it concerns the accuracy in template removal, we performed two different experiments and we have preliminary results for both of them. More precisely:

- We performed template extraction and cleaning on a set of news in text format, from different sources, parsed from the HTML by inaccurate, template-unaware, parsers.
- We performed template extraction and cleaning on the aforementioned collection of news directly crawled from Business Week via Yahoo in HTML format.

In experiment 5.2, we are given a set of sources and an history of articles from that sources that have been already parsed with low accuracy by template-unaware parsers.
In this case we have news articles like the following (from http://www.sevendaysvt.com/):

```
#  Permanent link | no responses | Published on: Tuesday, 27th
February, 2007 Real Estate developers and mortgage financiers in

[more text omitted]

growth model to be replicated elsewhere." he said. Please login to
leave a reply First budget hotel in Karama, another five planned
for emirate read more Thursday 22nd February read more
```

It is clear in this case that the naive HTML parser employed has simply converted to text what encounter on its way and this resulted on the garbage present in the head and tail of the article, namely the permanent link, publication date and comment list in the head and the links to "read more" in the tail. It is clear that such a poorly parsed articles will hurt any kind of IR task that can be performed on it, such as searching or clustering. In this case, the word "template" acquire a slightly different form with respect to what we have treated so far and assumes its general meaning of "recurrent and non informative pattern". The extractor module has correctly identified the aforementioned templates, as can be seen from an excerpt of the extracted cleaning expressions for the source http://www.sevendaysvt.com/:

```
([\-\s]+|^)Permanent[\-\s]+link[\-\s]+\|[\-\s]+no[\-\s]+responses[\-\s]    \
+\|[\-\s]+Published[\-\s]+on:[\-\s]+(Monday|Tuesday|Wednesday|Thursday|    \
Friday|Saturday|Sunday)[[:punct:]\-=+()_\\]*[\-\s]+\d+(st|nd|rd|th)[\-\s]  \
+(January|February|March|April|May|June|July|August|September|October|     \
November|December)[[:punct:]\-=+()_\\]*[\-\s]+[\d,.]+([\-\s]+|$)           \
...

([\-\s]+|^)(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday)      \
[[:punct:]\-=+()_\\]*[\-\s]+\d+(st|nd|rd|th)([\-\s]+|$)
...
([\-\s]+|^)read[\-\s]+more([\-\s]+|$)
....
```

As can be easily recognized, the first expression (which spans five lines) matches (and thus will be used to remove) the template in the header while the last two expressions, combined together using the interval logic explained in section 3.4 will clean up (part of) the garbled tail.
Those expressions, together with many others, where extracted automatically starting from the input text and a list of regular expressions that matches what we have called the *mutable pattern*, here reported:

```
[[:upper:]]\w{1,4}\.
\d+(st|nd|rd|th)
[\d,.]+
[\d\-\\\/:]+
[$\d,.()\]\[\-\\\/:]+
-+
[[:upper:]()\-_\\]{2,}[[:punct:]\-=+()_\\]*
(Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|Sep|Oct|Nov|Dec)[[:punct:]\-=+()_\\]*
(January|February|March|April|May|June|July|August|September|October|     \
 November|December)[[:punct:]\-=+()_\\]*
(Monday|Tuesday|Wednesday|Thursday|Friday|Saturday|Sunday)[[:punct:]\-=+ \
()_\\]*
(Mon|Tue|Wed|Thu|Fri|Sat|Sun)[[:punct:]\-=+()_\\]*
(hr|day|minute|second)s*
[\w[:punct:]]+@[\w[:punct:]_\\]+
http:\/\/\S*
([[:upper:]]\.){2,}
[bB]y
```

Among those one can recognize expressions that match date formats (with month's full name and abbreviated form), telephone numbers, sequences of uppercase letters, URL and email addresses.
As a matter of facts, with this simple input, the outcome of the cleaner applied on the text shown is the following:

~~# Permanent link | no responses | Published on: Tuesday, 27th February, 2007~~ Real Estate developers and mortgage financiers in

[more text omitted]

growth model to be replicated elsewhere." he said. Please login to leave a reply First budget hotel in Karama, another five planned for emirate ~~read more Thursday 22nd February read more~~

which correspond exactly to what one may want.
The system has shown the same performance in removing template from web page as well, without changing any parameter. This happens because the system is totally oblivious of the structure which subsumes the data it is applied to and treats everything as a string. In this way template detection is performed on a flat view, and is mapped onto extracting frequent substrings from corpora, which is a known problem and for which many efficient solutions are known.
We did not perform an extensive evaluation for the results because there are no standard measure of the goodness of template removal nor a consolidated baseline in the literature to

compare to. Since our system was developed for commercial needs, we decided to evaluate manually the outcome on a sub-sample of the raw text news data collection.
In general editorials report an accurate cleaning for most of the sources. However, some minor problems have been identified, that is:

- New mutable patterns matching expression are needed to catch the complexity of some site.
- The number of false positives (non-templated detected as templates) is not null, albeit small, and this is probably the major concern
- The number of false negatives (templates not correctly identified) is not null, albeit negligible

These evidences suggest that the system should undergone a tuning session with editorial feedback to improve precision.

Chapter 6

Conclusion and Future Work

In this paper we have described the design and implementation of a system for template detection and removal for web pages and online newspapers. The system is based on plain string processing and does not assume any kind of structure of the data to be cleaned but extracts frequent patterns and turn them into regular expressions that can in turn be used to remove those patterns. The system allows for a specification of mutable patterns that matches sets of strings that have to be treated as equivalent and, therefore, processed in the exactly same way by the suffix array logic. This enhancement gives to the system the ability to generalize the cleaning expressions in order to be effectively applied in pages different from the ones of the training set. The system is very fast because it basically performs only simple computation on strings and integer and almost parameter free, in the sense that the result of the cleaning is not affected very much by changes done to the fews input parameters.

6.1 Future work

The first think that this system needs is a solid and in-depth evaluation. Unfortunately, no standard measures have been proposed to evaluate the goodness of such kind of systems and there no baseline in the literature to compare to.
Possible way to evaluate a template detection an removal system could be:

- Inject synthetic templated data into clean web pages and check if the extracted cleaning expressions matched the injected data
- evaluate the performance of some IR tasks, such as clustering, on both unclean and cleaned data for which is known apriori a good clusterization. Given a particular clustering algorithm, we would expect that the standard performance measures for clustering goodness reveal that the clusterization on the cleaned dataset resembles more closely the one known apriori.
- Another possible way to check the module could be to use it to identify mirroring between sites. If a site mirrors another one they share the same content but, with

high probability, they will have different presentation style, and therefore, different templates. A good template removal system applied on both the site and the mirrored one should extract the informative content in a way that could be compared in some almost exact method. The edit distance between the two site, once both cleaned, should be very low, for instance.

- One more possible way of evaluating the goodness of template removal would be to select an RSS that provides clean news data and, in the mean time, crawl its web counterpart. The template removal method applied on the crawled web pages, if properly working, should lead to a content very similar to the one provided by the clean-by-definition RSS.

Not only the evaluation of the system can be improved. The system itself should be tuned to better match the data it is applied on. In the first place more “mutable patterns” must be put into the system to make it more flexible, secondly better locality heuristics on the intervals should be added during the cleaning phase in order to avoid false positive (for instance, a few characters removed in the middle of a clean chunk of text are probably a false positive) and negative (conversely, a little chunk of good text in between two large templated string is probably part of the template too)

Acknowledgements

I thank **ask.com** for having given the possibility to work on this topic and for having provided me with the equipment necessary to perform my tasks.
I'd like to thanks Antonio Gullì, Antonio Savona, Giuseppe Ottaviano and Giovanni Deretta for their valuable help and the time spent in fruitful discussions.

Bibliography

[AGMU03] Arvind Arasu, Hector Garcia-Molina, and Stanford University, Extracting structured data from web pages, SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data (New York, NY, USA), ACM Press, 2003, pp. 337–348.

[AM02] Helena Ahonen-Myka, Discovery of frequent word sequences in text, Lecture notes on computer science **2447** (2002), 180–??

[Bal06] Shumeet Baluja, Browsing on small screens: recasting web-page segmentation into an efficient machine learning framework, WWW '06: Proceedings of the 15th international conference on World Wide Web (New York, NY, USA), ACM Press, 2006, pp. 33–42.

[BEX02] Florian Beil, Martin Ester, and Xiaowei Xu, Frequent term-based text clustering, KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM Press, 2002, pp. 436–442.

[BYR02] Ziv Bar-Yossef and Sridhar Rajagopalan, Template detection via data mining and its applications, WWW '02: Proceedings of the 11th international conference on World Wide Web (New York, NY, USA), ACM Press, 2002, pp. 580–591.

[Cha02] Soumen Chakrabarti, Mining the Web: Discovering knowledge from hypertext data, Morgan-Kauffman, 2002.

[CHWM04] Deng Cai, Xiaofei He, Ji-Rong Wen, and Wei-Ying Ma, Block-level link analysis, SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM Press, 2004, pp. 440–447.

[CKP07] Deepayan Chakrabarti, Ravi Kumar, and Kunal Punera, Page-level template detection via isotonic smoothing, WWW '07: Proceedings of the 16th international conference on World Wide Web (New York, NY, USA), ACM Press, 2007, pp. 61–70.

[CMM01] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo, Roadrunner: Towards automatic data extraction from large web sites, VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 2001, pp. 109–118.

[CMS99] Robert Cooley, Bamshad Mobasher, and Jaideep Srivastava, Data preparation for mining world wide web browsing patterns, Knowledge and Information Systems **1** (1999), no. 1, 5–32.

[CXMZ05] Yu Chen, Xing Xie, Wei-Ying Ma, and Hong-Jiang Zhang, Adapting web pages for small-screen devices, IEEE Internet Computing **9** (2005), no. 1, 50–56.

[CYWM04] Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma, Block-based web search, SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval (New York, NY, USA), ACM Press, 2004, pp. 456–463.

[DMPG05] Sandip Debnath, Prasenjit Mitra, Nirmal Pal, and C. Lee Giles, Automatic identification of informative sections of web pages, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 9, 1233–1246.

[GPT] David Gibson, Kunal Punera, and Andrew Tomkins, The volume and evolution of web page templates.

[HC02] J. Han and K. Chang, Data mining for web intelligence, IEEE Computer (2002), 54+.

[KHC05] Hung-Yu Kao, Jan-Ming Ho, and Ming-Syan Chen, Wisdom: Web intrapage informative structure mining based on document object model, IEEE Transactions on Knowledge and Data Engineering **17** (2005), no. 5, 614–627.

[KLA$^+$01] Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park, Linear-time longest-common-prefix computation in suffix arrays and its applications, CPM '01: Proceedings of the 12th Annual Symposium on Combinatorial Pattern Matching (London, UK), Springer-Verlag, 2001, pp. 181–192.

[Kle99] Jon M. Kleinberg, Authoritative sources in a hyperlinked environment, Journal of the ACM **46** (1999), no. 5, 604–632.

[KLHC02] H. Kao, S. Lin, J. Ho, and M. Chen, Entropy-based link analysis for mining web informative structures, 2002.

[KS03] J. Kärkkäinen and P. Sanders, Simple linear work suffix array construction, Proc. 13th International Conference on Automata, Languages and Programming, Springer, 2003.

[Kus99] Nicholas Kushmerick, Learning to remove internet advertisement, Proceedings of the Third International Conference on Autonomous Agents (Agents'99) (Seattle, WA, USA) (Oren Etzioni, Jörg P. Müller, and Jeffrey M. Bradshaw, eds.), ACM Press, 1999, pp. 175–181.

[LH02] Shian-Hua Lin and Jan-Ming Ho, Discovering informative content blocks from web documents, KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM Press, 2002, pp. 588–593.

[LLL00] Mong-Li Lee, Tok Wang Ling, and Wai Lup Low, Intelliclean: a knowledge-based intelligent data cleaner, Knowledge Discovery and Data Mining, 2000, pp. 290–294.

[LPHL02] Xiaoli Li, Tong-Heng Phang, Minqing Hu, and Bing Liu, Using micro information units for internet search, CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management (New York, NY, USA), ACM Press, 2002, pp. 566–573.

[MGCC03] Ling Ma, Nazli Goharian, Abdur Chowdhury, and Misun Chung, Extracting unstructured data from template generated web documents, CIKM '03: Proceedings of the twelfth international conference on Information and knowledge management (New York, NY, USA), ACM Press, 2003, pp. 512–515.

[MM90] Udi Manber and Gene Myers, Suffix arrays: A new method for on-line string searches, Proc. of the 1st ACM-SIAM Symposium on Discrete Algorithms, 1990, pp. 319–327.

[Myl01] Jussi Myllymaki, Effective web data extraction with standard XML technologies, World Wide Web, 2001, pp. 689–696.

[NBM02] U. Nahm, M. Bilenko, and R. Mooney, Two approaches to handling noisy variation in text mining, 2002.

[Rag] Dave Raggett, Html tidy sourceforge homepage, `http://tidy.sourceforge.net/`.

[RGSL04] D. Reis, P. Golgher, A. Silva, and A. Laender, Automatic web news extraction using tree edit distance, 2004.

[SLWM04] Ruihua Song, Haifeng Liu, Ji-Rong Wen, and Wei-Ying Ma, Learning block importance models for web pages, WWW '04: Proceedings of the 13th international conference on World Wide Web (New York, NY, USA), ACM Press, 2004, pp. 203–211.

[SR02] D. Sanchez and D. Rodriguez, A survey on defacement methods, TEABAG '00: Proceedings of the 6th ATM TEABAG Conference on Thesaurus-enhanced anthologies for bounded abstract groups, 2002, pp. 162–171.

[VdSP$^+$06] Karane Vieira, Altigran S. da Silva, Nick Pinto, Edleno S. de Moura, M. B. Cavalcanti, and Juliana Freire, A fast and robust method for web page template detection and removal, CIKM '06: Proceedings of the 15th ACM international conference on Information and knowledge management (New York, NY, USA), ACM Press, 2006, pp. 258–267.

[W3C] W3C, W3c hypertext markup language homepage, http://www.w3.org/MarkUp/.

[YC01] Mikio Yamamoto and Kenneth W. Church, Using suffix arrays to compute term frequency and document frequency for all substrings in a corpus, Comput. Linguist. **27** (2001), no. 1, 1–30.

[YCWM03] Shipeng Yu, Deng Cai, Ji-Rong Wen, and Wei-Ying Ma, Improving pseudo-relevance feedback in web information retrieval using web page segmentation, WWW '03: Proceedings of the 12th international conference on World Wide Web (New York, NY, USA), ACM Press, 2003, pp. 11–18.

[YL04] Xinyi Yin and Wee Sun Lee, Using link analysis to improve layout on mobile devices, WWW '04: Proceedings of the 13th international conference on World Wide Web (New York, NY, USA), ACM Press, 2004, pp. 338–344.

[YLL03] Lan Yi, Bing Liu, and Xiaoli Li, Eliminating noisy information in web pages for data mining, KDD '03: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining (New York, NY, USA), ACM Press, 2003, pp. 296–305.

[YP97] Yiming Yang and Jan O. Pedersen, A comparative study on feature selection in text categorization, ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning (San Francisco, CA, USA), Morgan Kaufmann Publishers Inc., 1997, pp. 412–420.